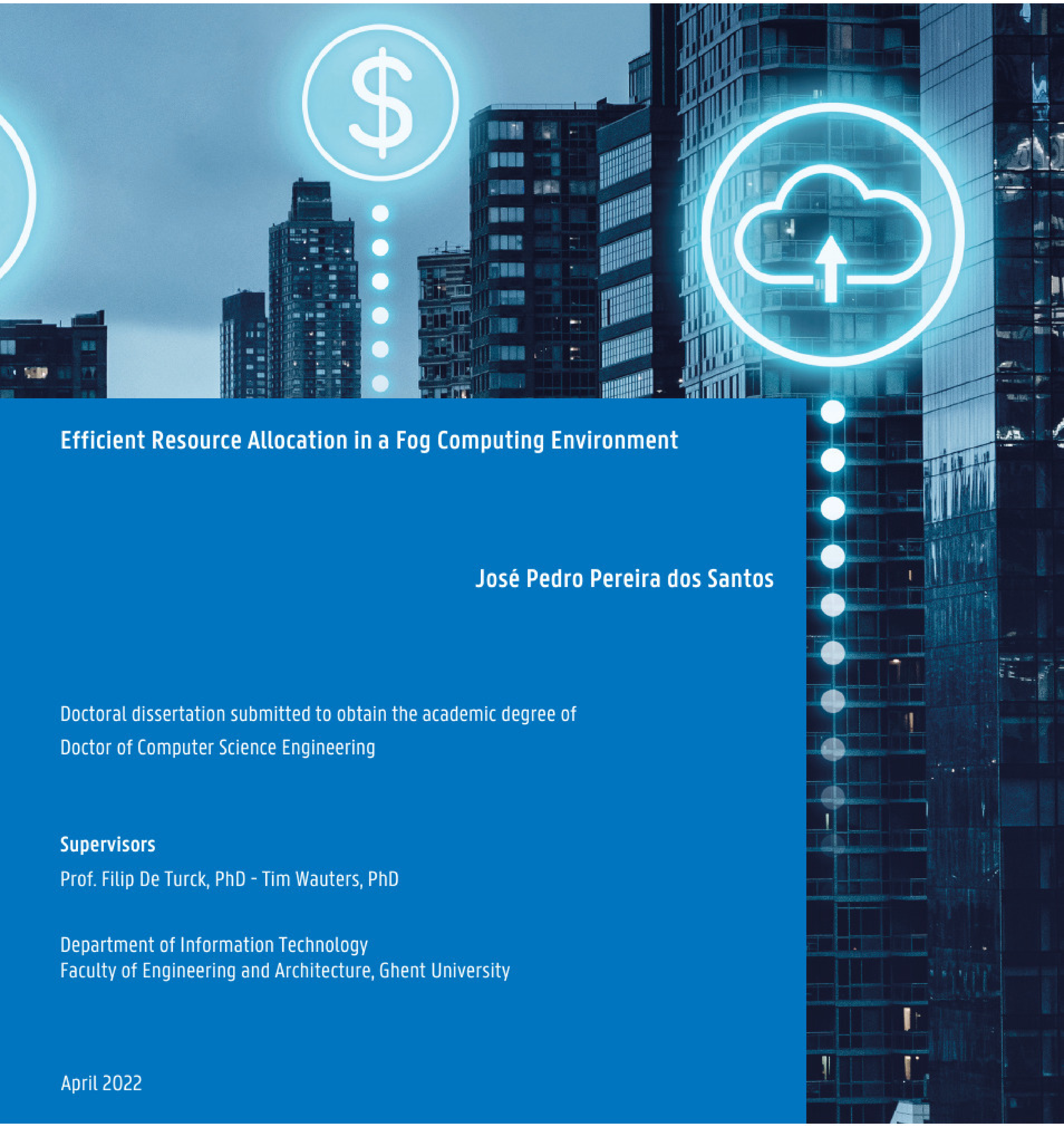
FACULTY OF ENGINEERING
AND ARCHITECTURE



Efficient Resource Allocation in a Fog Computing Environment

José Pedro Pereira dos Santos

Doctoral dissertation submitted to obtain the academic degree of
Doctor of Computer Science Engineering

Supervisors
Prof. Filip De Turck, PhD - Tim Wauters, PhD

Department of Information Technology
Faculty of Engineering and Architecture, Ghent University

April 2022

GHENT
UNIVERSITY

FACULTY OF ENGINEERING
AND ARCHITECTURE

Efficient Resource Allocation in a Fog Computing Environment

José Pedro Pereira dos Santos

Doctoral dissertation submitted to obtain the academic degree of
Doctor of Computer Science Engineering

Supervisors
Prof. Filip De Turck, PhD - Tim Wauters, PhD

Department of Information Technology
Faculty of Engineering and Architecture, Ghent University

April 2022

GHENT
UNIVERSITY

ISBN 978-94-6355-584-5
NUR 986, 982
Wettelijk depot: D/2022/10.500/25

Acknowledgments

"No one who achieves success does so without acknowledging the help of others."

–Alfred North Whitehead (1861 - 1947)

This book represents five incredible years during my doctoral studies. I had several amazing moments that I will always remember during my life: starting living in another country, traveling outside Europe, a lot of brainstorming discussions, hard work, meeting great collaborators, and so much more. All of this would have never been possible without the support and help of several people during my Ph.D. at Ghent University.

First, I would like to thank my supervisors Filip and Tim for their full support and guidance during my doctoral studies. I am extremely grateful for the opportunity you two gave me. In my mind, your advice was key for the several achievements during my Ph.D. and I hope I was able to make you two proud of the decision of accepting me as a Ph.D. candidate. Tim, I really enjoyed our brainstorming sessions on which my initial ideas became so much better after your feedback.

Second, I would like to give a special thanks to all my co-authors who helped me improve the quality of my work. A big thanks to Bruno, Maria, and Jeroen for their availability and feedback when I needed it. Also, I would like to give a special tribute to Chen, who not only allowed me to work under her wing during the last stages of my Ph.D., but also helped me go the extra mile with my work. Your feedback and encouragement allowed me to make my work visible and impactful towards the industry and I am extremely grateful for that. I feel honored to have been able to work with you despite the difficulties created by the 6-hour timezone difference. I would like to thank as well to all the members of the examination board. Bruno, Chen, prof. Dieter, prof. Mays-AL-Naday, and prof. Mohamed Faten Zhani: your feedback contributed to further enhancing the quality of my dissertation. Thank you so much. In addition, I would like to show my sincere appreciation for all the members of the administrative staff at IDLab. A special thanks to Martine, Davinia, Karen, Bernadette, and Mike. Thank you for the support you provide to all PhD students. Also, most of my experiments would have not been possible without the support of the IDLab's A-Team. A big thanks to Brecht and Joeri for creating an amazing testbed infrastructure.

Third, I would like to thank all the great colleagues I met at IDLab during the past five years: Lucas, Rafael, Thomas, Wannes, Jasper, Ankita, Leandro, Jerico, Merlijn, Stefano, Jeroen, Dries, Sarah, Joachim, Pieter-Jan, Maria, Roberto, Hemanth,

Stijn, Philip, Vincent, Patrick, Tom, Miel, Sam: thanks to all of you! I would like already to apologize to the ones I might potentially forget to mention! You all helped me with the new transition in my life and made my reallocation to Ghent easier.
Finally, I would like to thank my family and friends for their support. Em especial, gostaria de agradecer à minha mãe, Helena que me ensinou praticamente tudo o que sei hoje. Sem o teu suporte e apoio isto não seria possível. Amo-te muito mãe. Obrigado por seres quem és. Gostaria de agradecer também ao meu Pai, José que apesar de não estar mais entre nós, esteve sempre a olhar por mim onde quer que ele esteja. Obrigado do fundo do coração. Se completei esta etapa da minha vida, foi grande parte por causa dos meus pais. Gostaria de deixar também uma menção honrosa à minha família e amigos que me ajudaram nesta fase: Tia Maria, Avó Veronica, Tia Rosa, Cassilda, Leopoldina, Júlia, Linda, Júlio, Nuno, Silvia, Ana, Rosalina, Rui, Isabel, Maria João, Alex Silva, Gonçalo, Dulce, Bruno, Fernando, Gabriel, Ana Isabel, Maria, Filipe, Maria João, Pedro, Ricardo, João Fernandes, Rui Carvalho, Luís, Miguel, Bruno, Carlos, Jota, Telmo, Isabel, Eduardo, Pedro, Feng, Aves, Perdiz, Xavier, Brito, Diogo, Joana, Flor, Nina: obrigado a todos pelo vosso apoio apesar de estarem longe a maior parte do tempo. And last but not least, I would like to thank my lovely partner, Catarina. Não tenho palavras para descrever todo o apoio que me deste ao longo destes anos. Obrigado por tudo! Obrigado por todas as nossas conversas, por me ajudares a rever os meus artigos, pelas tuas excelentes ideias para apresentações, por seres a minha companheira! Espero que tenha conseguido retribuir parte do que tu me deste. Se cheguei aqui, foi também muito pela tua ajuda e suporte. Obrigado pela tua paciência ao longo destes anos. Amo-te muito minha pequenina, minha principessa!

Gent, March 2022
José Pedro Pereira dos Santos

Table of Contents

List of Figures

List of Tables

List of Acronyms

0-9

5G-PPP	5G Infrastructure Public Private Partnership
6G	Next-Generation Networks

A

ABFN	Area Border Fog Node
ADR	Adaptive Data Rate
AE	Application Entity
AI	Artificial intelligence
AID	Association Identifier
API	Application Programming Interface
AR	Augmented Reality
AS	Autonomous System

B

BC	BlockChain
BER	Bit Error Rate
BIP	Binary Integer Programming
BSS	Business Support System

C

CLI	Command Line Interface
CN	Cloud Node
CoAP	Constrained Application Protocol

COHERENT	Coordinated control and spectrum management for 5G heterogeneous radio access networks
CoT	City of Things
CS	CyberSecurity
CSE	Common Service Entity
CSMA	Carrier Sense Multiple Access

D

D2D	Device-to-Device
D2F	Device-to-Fog
DC	Data Center
DCN	Data Center Network
DDoS	Distributed Denial of Service
DL	Deep Learning

E

E2E	end-to-end
EC	European Commission
ECU	Electronic Control Unit
eMBB	enhanced Mobile Broadband
EMS	Elemental Management System
ETSI	European Telecommunications Standards Institute

F

FA	Fog Agent
FC	Fog Computing
FD	Fog Decision
FL	Federated Learning
FM	Fog Manager
FN	Fog node
FO	Fog Orchestrator
FoE	Fog of Everything
FPGA	Field-Programmable Gate Array
FVC	Fog Vehicular Computing

G

GPRS	General Packet Radio Service
GUI	Graphical User Interface
GV	Ground Vehicle

H

HCF	High-Capacity Fog
HFC	Hybrid Fog and Cloud
HTC	Holographic Type Communication

I

IaaS	Infrastructure-as-a-Service
IBN	Intent-based Networking
IEEE	Institute of Electrical and Electronics Engineers
ID	Identifier
IDE	Integrated Development Environment
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IF	Isolation Forrest
IIoT	Industrial Internet of Things
ILP	Integer Linear Programming
IoE	Internet of Everything
IoT	Internet of Things
IP	Internet Protocol

K

KS	Kube–Scheduler

L

LCF	Low-Capacity Fog
LOF	Local Outlier Factor
LPWAN	Low Power Wide Area Networks
LSTM	Long-Short Term Memory

LwM2M	Lightweight Machine-to-Machine

M

M2M	Machine-to-Machine
MANO	Management and Orchestration
MDP	Markov Decision Process
ME	Mobile Edge
MEH	Mobile Edge Host
MEC	Mobile Edge Computing
MEC	Multi-access edge computing
MEP	Mobile Edge Platform
MILP	Mixed Integer Linear Programming
MINL	Mixed-Integer Non-Linear
MINLP	Mixed-Integer Nonlinear Programming
ML	Machine Learning
mMTC	massive Machine-Type Communication
mmWave	millimeter-wave
MNO	Mobile Network Operator
MQTT	MQ Telemetry Transport
MPTCP	MultiPath Transmission Control Protocol
MTU	Maximum Transmission Unit

N

NAS	Network-Aware Scheduler
NBI	Northbound Interface
NF	Network Function
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestrator
NN	Neural Network
NS	Network Service
NS	Network Slicing
NSE	Network Service Entity
NSH	Network Service Header

O

OF	OpenFlow

OMA	Open Mobile Alliance
OSS	Operations Support System
OSPF	Open Shortest Path First

P

P2P	Peer-to-Peer
PCC	Point Cloud Compression
PMI	Particle Matter Indicator
PSR	Packet Success Ratio
PVRV	Panoramic Virtual Reality Video

Q

QoE	Quality of Experience
QoS	Quality of Service

R

RAN	Radio Access Network
RAT	Radio Access Technology
RC	Robust Covariance
RDP	Robustly Disjoint Path
REST	Representational State Transfer
RF	Radio Frequency
RL	Reinforcement Learning
RTT	Round Trip Time

S

SaaS	Software-as-a-Service
SDN	Software-Defined Networking
SDX	Software-Defined Internet Exchange Point
SESAME	Small cEllS coordinAtion for Multi-tenancy and Edge Services
SF	Spreading Factor
SFC	Service Function Chaining
SGX	Software Guard Extensions

SINR	Signal-to-Noise-plus-Interference Ratio
SLA	Service Level Agreement
SLO	Service Level Objective
SOA	Service-Oriented Architecture
SONATA	Service Programming and Orchestration for Virtualized Software Networks
SR	Segment Routing
SRD	Segment Routing Domain

T

TC	Trusted Computing
TCP	Transmission Control Protocol
TOPS	Tera Operations per second
TOSCA	Topology and Orchestration Specification for Cloud Applications
TTI	Transmission Time Interval

U

UAV	Unmanned Aerial Vehicle
UDI	Universal Data object Identifier
UE	User Equipment
UI	User Interface
URLLC	Ultra-Reliable Low-Latency Communication

V

V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-Everything
VHD	Virtual Honeypot Device
VIM	Virtualized Infrastructure Manager
VLC	Visible Light Communication
VM	Virtual Machine
VNF	Virtual Network Function
VNFM	Virtualized Network Function Manager
VR	Virtual Reality

W

WSN	Wireless Sensor Networks

X

xMBB	Extreme Mobile Broadband
XR	Extended Reality

Y

YAML	YAML Ain't Markup Language

List of Symbols

A	The set of all IoT applications. Each IoT application is composed of a set of communicating services.
$A_{ed,gw}$	The Association matrix. If $A_{ed,gw} = 1$, the end device ed can associate with the gateway gw.
B_{n_1,n_2}	The Bandwidth matrix between comp. nodes indicates the bandwidth (Mbit/s) available between the comp. node n_1 and the comp. node n_2.
C_{n_1,n_2}	The Communication matrix between comp. nodes indicates the bandwidth (Mbit/s) required between services of an IoT application.
C	Communication range in meters.
D_a	The total number of requests for an application $a \,\varepsilon\, A$.
$D_{gw,ed}$	The Distance matrix indicates the distance (in meters) between a gateway and an end device.
D_{LPWAN}	Download data rate (in Mbps) of the LPWAN technology.
D_{Fog}	Download speed (in Mbps) between a fog node and the cloud node.
$E_{f,l}$	$E_{f,l} = 1$ indicates that fog cloud f is at location l.
$E_{n,f}$	$E_{n,f} = 1$ indicates that comp. node n is managed by fog cloud f.
$E_{n,l}$	$E_{n,l} = 1$ indicates that comp. node n is at location l.
$E_{gw,l}$	$E_{gw,l} = 1$ indicates that gateway gw is at location l.
$E_{ed,l}$	$E_{ed,l} = 1$ indicates that end device ed is at location l.
$F^{a,r}_{s_1,s_2}(n_1, n_2)$	The flow matrix contains the bandwidth belonging to the rth request of IoT application a that is used in the communication between services s_1 and s_2 which are allocated on node n_1 and n_2, respectively.
Γ_n	The total memory capacity (in GB) of the comp. node $n \,\varepsilon\, N_c$.
γ_s	The memory requirement (in GB) of the service $s \,\varepsilon\, S$.
$G_{a,r}$	The acceptance matrix. If $G_{a,r} = 1$, the rth request of IoT application a can be accepted. If $G_{a,r} = 0$, the rth request of IoT application a cannot be accepted.

$H_{n,ed}$	The Hop Count matrix indicates the number of devices between the comp. node n and the end device ed.
$I_{a,s}$	The Instance matrix. If $I_{a,s} = 1$, the communicating service s is part of application a. If $I_{a,s} = 0$, the communicating service s is not part of application a.
L	Set of locations where IoT application requests are generated by end devices.
N_c	The set of comp. nodes on which services are executed.
N_f	The set of fog clouds on the network which manage the comp. nodes.
N_{gw}	The set of wireless gateways on the network.
N_{ed}	The set of end devices on the network.
N	Number of bits in each data sample.
Ω_n	The total CPU capacity (in GHz) of the comp. node $n \varepsilon N_c$.
ω_s	The CPU requirement (in GHz) of the service $s \varepsilon S$.
$\Phi_{a,r,ed}$	The Request matrix. If $\Phi_{a,r,ed} = 1$, the end device ed made the rth request of application a.
$PL_{gw,ed}$	The Path Loss matrix indicates the path loss (in dB) between a gateway and an end device.
$P_{s,n}^{a,r}$	The placement matrix. If $P_{s,n}^{a,r} = 1$, an instance of service s is executed on comp. node n for the rth request of IoT application a.
R_a	The set containing all requests for an application $a \varepsilon A$.
$R_{s,n}$	The Relation matrix. If $R_{s,n} = 1$, the communicating service s can be allocated on node n. If $R_{s,n} = 0$, the communicating service s cannot be instantiated on node n.
R_{ed}	A binary value that indicates if the end device ed sent a request for an IoT application $a \varepsilon A$.
R	Number of data samples to be transmitted.
S	The set of all communicating services.
Θ_{gw}	The total association identifiers available on a gateway $gw \varepsilon N_{gw}$.
θ_{ed}	Each end device $ed \varepsilon N_{ed}$ needs an association identifier to associate with a gateway.
T	Transmission time of a packet.
$U_{s,n}$	The service execution matrix. If $U_{s,n} = 1$, an instance of service s is allocated on comp. node n. If $U_{s,n} = 0$, there is not an instance of service s allocated on comp. node n.
$U_{ed,gw}$	The end device execution matrix. If $U_{ed,gw} = 1$, the

	end device ed is associated with gateway gw.
U_{gw}	The gateway utilization matrix. $U_{gw} = 1$ indicates that there is at least one end device associated with gateway gw.
U_n	The comp. node utilization matrix. $U_n = 1$ indicates that there is at least one service allocated on comp. node n.
U_{LPWAN}	Upload data rate of the LPWAN technology in Mbps.
U_{Fog}	Upload speed between a fog node and the cloud node in Mbps.
$z^{a,r}_{s_1,s_2}$	The service bandwidth matrix contains the amount of bandwidth for every flow in the communication between services s_1 and s_2 for the rth request of IoT application a.

Samenvatting – Summary in Dutch –

In de afgelopen jaren heeft de introductie van clouddiensten een hele revolutie teweeggebracht voor implementaties van applicaties. Cloud computing heeft de laatste tijd een enorme groei doorgemaakt en is de standaard geworden voor het uitvoeren van applicaties. Eindgebruikers en apparaten maken verbinding naar een cloudgebaseerd systeem waar deze applicaties worden ingezet. Toegang tot e-mails of het opslaan van documenten via internet zijn voorbeelden van diensten die momenteel worden uitgevoerd in de cloud. Cloudsystemen verlagen de kosten voor ondernemingen omdat computationele hulpmiddelen worden aangevraagd via een cloudprovider die verantwoordelijk is voor onderhoud en upgrades van het cloudsysteem. Bovendien hebben recente toepassingsdomeinen zoals het Internet der Dingen (IoT) en Slimme Steden nieuwe uitdagingen geïntroduceerd voor traditionele cloud systemen. Een van de uitdagingen die blijft bestaan, is om efficiënte allocatiestrategieën voor hulpbronnen aan te bieden voor dit soort toepassingen. Cloud providers streven ernaar om onnodige implementatiekosten te vermijden en hun hardware-infrastructuur te optimaliseren zonder de met eindgebruikers overeengekomen Service Level Agreements (SLA's) te schenden. Deze toepassingen introduceren echter een nieuwe reeks strenge eisen (b.v. lage latentie, hoge bandbreedte) die traditionele toewijzingsstrategieën niet kunnen inwilligen, aangezien hun primaire focus het optimaliseren van de efficiëntie van hulpbronnen is (b.v. CPU en RAM). Latentiereductie speelt een grote rol voor latentiegevoelige IoT-toepassingen, aangezien zelfs een kleine vertraging hun prestaties drastisch kan beïnvloeden, vooral bij gezondheidsmonitoring en noodhulptoepassingen.

IoT-verkeer is de afgelopen jaren exponentieel gegroeid, waardoor gecentraliseerde cloudsystemen ongeschikt zijn voor IoT. Traditionele clouds kunnen niet omgaan met de strenge vereisten geïntroduceerd door IoT-toepassingen, aangezien hulpmiddelen gelijktijdig door meerdere apparaten op verschillende locaties kunnen worden aangevraagd. De huidige clouds kunnen geen lage latentie handhaven voor al deze apparaten verspreid over het netwerk. Om om te gaan met de beperkingen van gecentraliseerde clouds, wordt het Fog Computing-paradigma geïntroduceerd. Fog Computing is een uitbreiding op het Cloud Computing-paradigma waar computationele hulpmiddelen aan de randen van het netwerk worden geplaatst om de latentie die wordt verwacht door IoT-apparaten te verminderen. Fogknopen vertegenwoordigen een kleine cloud-entiteit die een kleinere set computationele hulpmiddelen biedt in vergelijking met cloudlocaties. IoT-apparaten, voorname-

lijk sensoren en actuatoren, communiceren met foglocaties via draadloze gateways. Niettemin blijven onderzoeksuitdagingen bestaan in Fog Computing omdat het nog geen volwassen concept is. Dit proefschrift behandelt verschillende uitdagingen in het Fog Computing-domein gericht op het bieden van een efficiënte toewijzing van middelen in deze gedistribueerde infrastructuren.
Ten eerste bespreekt dit proefschrift het probleem van het toewijzen van IoT-diensten in Fog Computing. Efficiënte allocatiestrategieën vinden in een gedistribueerde cloudinfrastructuur is de afgelopen jaren een van de grootste uitdagingen in Fog Computing geweest. Verschillende diensten vormen samen een enkele IoT-toepassing, volgens het principe van Dienstgeoriënteerde Architecturen (SOA). De verdeling van computationele hulpmiddelen over het netwerk voegt verdere complexiteit toe aan het toewijzen van IoT-diensten, omdat bronnen overal beschikbaar zijn en aanvragen voor diensten van meerdere locaties kunnen komen. De toewijzingsstrategieën moeten rekening houden met efficiënt gebruik van hulpbronnen en de vermindering van latentie, aangezien dit de belangrijkste vereisten zijn voor vertragingsgevoelige IoT-toepassingen. Het instantiëren van diensten ver van apparaten zou resulteren in een hoge communicatielatentie. Daarom moeten de real-time beperkingen van tijdgevoelige IoT-toepassingen worden erkend in de strategie voor het toewijzen van middelen, terwijl de implementatiekosten worden geminimaliseerd en de Dienstkwaliteit (QoS) gemaximaliseerd. Dit proefschrift presenteert een formulering via Geheeltallig Lineair Programmeren (ILP) voor het plaatsen van IoT-diensten, die rekening houdt met meerdere optimalisatiedoelstellingen zoals latentievermindering en energie-efficiëntie. Het werk dient als benchmark in toekomstig onderzoek met betrekking tot plaatsingsproblemen in Fog Computing aangezien de modelbenadering generiek is en op verschillende use-cases kan worden toegepast. De modelevaluatie spitste zich toe op IoT-toepassingen in het kader van het City of Things testbed in Antwerpen, zonder verlies van algemeenheid.
Ten tweede wordt in dit proefschrift besproken hoe een Fog-computerraamwerk kan worden geleverd met beheer- en orkestratiefunctionaliteiten voor IoT. De afgelopen jaren heeft het Europees Telecommunicatie en Standaardisatie Instituut (ETSI) oneM2M gewerkt aan een end-to-end (E2E) architectuur op hoog niveau voor Machine-to-Machine (M2M) communicatie. Verschillende aspecten worden momenteel onderzocht: beveiliging, gegevensbeheer, apparaatauthenticatie, inschrijven bij M2M-diensten, onder andere. Verschillende werken hebben Fog-computerarchitecturen voor IoT voorgesteld, maar er is nog geen gemeenschappelijke basis gelegd. Zonder een duidelijk pad naar standaardisatie, is het moeilijk om de inspanningen van zowel de academische wereld als de industrie te integreren en te combineren. Dit proefschrift presenteert een Fog Computing-raamwerk dat de richtlijnen van ETSI rond de architectuur voor het beheer en orkestratie van Virtualisatie van Netwerkfuncties (NFV MANO) volgt, inclusief aanvullende softwarecomponenten zoals monitoring en data-analysefuncties. Daarnaast is er een nieuw fogprotocol voor de uitwisseling van dienstinformatie tussen fogknooppunten voorgesteld voor snelle beslissingen rond dienstverlening. Uit de eerste evaluatie ervan blijkt dat het Fog Computing raamwerk voor een aanzienlijke vermindering van het gebruik van netwerkbandbreedte zorgt, vergeleken met traditi-

onele, gecentraliseerde clouds.
Ten derde presenteert dit proefschrift een op Fog Computing gebaseerde anomaliedetectiemethode om te voorzien in efficiënte monitoring en analyse van gedistribueerde gegevens in het IoT-ecosysteem. Toezicht houden op het gedrag van IoT-toepassingen is een belangrijk aspect van het beheer van hun levenscyclus, vooral voor toepassingen rond persoonlijke gezondheidsmonitoring en noodhulp. Traditionele anomaliedetectieschema's zijn niet geschikt voor latentiegevoelige IoT-toepassingen, aangezien deze benaderingen vereisen dat alle gegevens naar een gecentraliseerde locatie worden gestuurd, wat resulteert in een hoge latentie. IoT-apparaten hebben doorgaans lage bandbreedtecapaciteiten, waardoor de verzamelde gegevens niet snel genoeg worden getransporteerd om de nodige anomaliedetectiebewerkingen uit te voeren en ongebruikelijke gebeurtenissen en storingen op voorhand te detecteren. Fog Computing helpt de communicatielatentie te verminderen door services dichter bij deze apparaten te instantiëren. Dit proefschrift presenteert inzichten over een op Fog Computing gebaseerd systeem voor het monitoren en analyseren van gedistribueerde gegevens gericht op een anomaliedetectie-aanpak voor Slimme Steden. Daarbij zijn de meest geschikte Low Power Wide Area Netwerktechnologieën (LPWAN) reeds bestudeerd voor de geëvalueerde use case rond Slimme Steden, op basis van een grote verzamelde dataset.
Ten vierde onderzoekt dit proefschrift netwerkbewuste planningsmethoden, in het bijzonder voor latentiegevoelige IoT-toepassingen. Efficiënte planningsstrategieën moeten rekening houden met bandbreedte en latentie in het planningsproces. Latentiereductie en bandbreedte-optimalisatie zijn primaire doelstellingen voor meerdere IoT-toepassingen. Dit proefschrift bestudeert de bruikbaarheid van het bekende containerorkestratieplatform Kubernetes voor IoT-toepassingen. Het presenteert daarbij ook een Netwerkbewuste Planner (NAS) voor op containers gebaseerde applicaties in Kubernetes. De eerste evaluaties tonen reeds dat de prestaties van de voorgestelde heuristiek in vergelijking met de standaard planningscomponent in Kubernetes een reductie tot 70% met betrekking tot netwerklatentie bereiken in vergelijking met het standaardmechanisme. Vervolgens, geïnspireerd door de principes van Gecombineerde Dienstfuncties (SFC), is het werk uitgebreid door het ontwerpen en implementeren van een netwerkbewust raamwerk, genaamd Diktyo, op basis van de architectuur met planningsplug-ins die is ontwikkeld voor Kubernetes. Het doel is om eindgebruikers een lage latentie te bieden en te zorgen voor bandbreedtereserveringen in applicatieplanning. Simulaties tonen aan dat Diktyo de netwerklatentie minimaliseert voor verschillende infrastructuurtopologieën, aan vergelijkbare uitvoeringstijden als de huidige planningsplug-ins. Ook praktische experimenten met benchmarktoepassingen voor microservices tonen aan dat Diktyo de doorvoer met 22% verhoogt en de latentie tot 45% vermindert in vergelijking met standaard plug-ins.
Tot besluit is dit proefschrift slechts een eerste stap naar een efficiënte allocatie van middelen in Fog Computing-architecturen. Hoewel er de afgelopen jaren op het gebied van Fog Computing aanzienlijke vorderingen zijn gemaakt, zal onderzoek naar de toewijzing van middelen blijven groeien, aangezien opkomende toepas-

singen nog meer complexiteit toevoegen. Toepassingen in de Uitgebreide Realiteit (XR) en autonome auto's behoren tot de meest veeleisende gevallen, waarbij cloud-architecturen verder worden gestimuleerd door nieuwe uitdagingen te presenteren rond het gebruik van computationele hulpmiddelen met betrekking tot betrouwbaarheid, doorvoer en latentie. Daarom is het werk met betrekking tot allocatie van hulpmiddelen en dienstplanning in de toekomstige cloudarchitecturen nog lang niet klaar.

Summary

In recent years, the introduction of cloud services has completely revolutionized application deployments. Cloud computing has seen enormous growth lately and has become the standard for running applications. End-users and devices connect to a cloud-based system where these applications are deployed. Accessing emails or storing documents over the Internet are examples of services currently running in the cloud. Cloud systems reduce costs for enterprises since computational resources are requested via a cloud provider responsible for maintaining and upgrading the cloud system. In addition, recent application domains such as the Internet of Things (IoT) and Smart Cities have introduced novel challenges to traditional cloud systems. One of the challenges that persist is how to provide efficient resource allocation strategies for these types of applications. Cloud providers aim to avoid unnecessary deployment costs and optimize their hardware infrastructure without violating the Service Level Agreements (SLAs) agreed with end-users. However, these applications introduce a new set of stringent requirements (e.g., low latency, high bandwidth) that traditional allocation strategies do not acknowledge since their primary focus is to optimize resource efficiency (e.g., CPU and RAM). Latency reduction plays a major role for latency-sensitive IoT applications since even a small delay can drastically impact their performance, especially for health monitoring and emergency response applications.

IoT traffic has been increasing exponentially in the past few years, making centralized cloud systems inadequate. Traditional clouds cannot cope with the stringent requirements introduced by IoT applications since resources can be requested on-demand simultaneously by multiple devices at different locations. Current clouds cannot maintain low latency for all these devices spread around the network. Thus, to deal with the limitations of centralized clouds, the Fog Computing paradigm has been introduced. Fog Computing is an extension to the Cloud Computing paradigm where computational resources are placed on the edges of the network to decrease the latency expected by IoT devices. Fog nodes or fog locations represent a small cloud entity that provides a smaller set of computational resources compared to cloud locations. IoT devices, mainly sensors and actuators, communicate with fog locations via wireless gateways. Nevertheless, research challenges persist in Fog Computing since it is not a mature concept yet. This dissertation addresses several challenges in the Fog Computing domain focused on providing an efficient resource allocation in these distributed infrastructures.

First, this dissertation discusses the problem of IoT service placement in Fog Computing. Finding efficient allocation strategies in a distributed cloud infrastructure

has been one of the main challenges in Fog Computing in the last few years. Different services compose a single IoT application, following Service-Oriented Architecture (SOA) principles. The distribution of computing resources across the network area adds further complexity to the service placement since resources are available everywhere, and service requests can come from multiple locations. Efficient allocation strategies need to consider resource efficiency as well as latency reduction since this is the main requirement for delay-sensitive IoT applications. Deploying services far from devices would result in high communication latency. Thus, the real-time constraints of time-sensitive IoT applications must be acknowledged in the resource allocation strategy while minimizing deployment costs and maximizing Quality of Service (QoS). This dissertation presents an Integer Linear Programming (ILP) formulation for IoT service placement that considers multiple optimization objectives such as latency reduction and energy efficiency. The work serves as a benchmark in future research related to placement issues in Fog Computing since the model approach is generic and can be applied to several use cases. The model evaluation focused on IoT applications within the scope of Antwerp's City of Things testbed without loss of generality.
Second, this dissertation discusses how to provide a Fog computing framework with management and orchestration functionalities for IoT. In recent years, the European Telecommunications Standards Institute (ETSI) oneM2M has been working towards an end-to-end (E2E) high-level architecture for Machine-to-Machine (M2M) communication. Several aspects are currently being investigated: security, data management, device authentication, M2M service subscription, among others. Several works have proposed Fog computing architectures for IoT, but no common ground has yet been established. Without a clear path to standardization, it is hard to integrate and combine efforts from both academia and industry. This dissertation presents a Fog Computing framework that follows the guidelines of the ETSI Network Function Virtualization (NFV) Management and Orchestration (MANO) architecture, including additional software components, such as monitoring and data analysis functionalities. In addition, a novel fog protocol for the exchange of application service information between fog nodes has been proposed for fast service provisioning decisions. Preliminary evaluations show that the Fog Computing framework achieves a significant reduction in terms of network bandwidth usage compared to traditional centralized clouds.
Third, this dissertation presents a Fog-based anomaly detection approach to provide efficient distributed data monitoring and analysis in the IoT ecosystem. Monitoring the behavior of IoT applications is an important aspect of their life-cycle management, especially for personal health monitoring and emergency response applications. Traditional anomaly detection schemes are not suitable for latency-sensitive IoT applications since these approaches require sending all data to a centralized location, resulting in high latency. IoT devices typically have low bandwidth capacities, meaning that the collected data is not transported quickly enough to run the necessary anomaly detection operations and detect unusual events and malfunctions beforehand. Fog Computing helps to reduce the communication latency by deploying services closer to these devices. This dissertation presents in-

sights on a Fog Computing-based distributed data monitoring and analysis scheme focused on an anomaly detection approach for Smart Cities. In addition, the most appropriate Low Power Wide Area Network (LPWAN) technologies have been studied for the evaluated Smart City use case based on a large collected dataset.

Fourth, this dissertation investigates network-aware scheduling methods, specifically beneficial for latency-sensitive IoT applications. Efficient scheduling strategies need to consider bandwidth and latency in the scheduling process. Latency reduction and bandwidth optimization are primary objectives for multiple IoT applications. This dissertation studies the feasibility of a well-known container orchestration platform named Kubernetes for IoT applications. Also, it presents a Network-Aware Scheduler (NAS) for container-based applications in Kubernetes. Preliminary evaluations show the performance of the proposed heuristic compared to the default scheduling component in Kubernetes. Results demonstrate that NAS achieves a reduction of up to 70% concerning network latency compared to the default mechanism. Then, inspired by Service Function Chaining (SFC) principles, the work has been extended by designing and implementing a network-aware framework named Diktyo based on the scheduling plugins architecture developed for the Kubernetes scheduler. The goal is to deliver low latency to end-users and ensure bandwidth reservations in application scheduling. Simulations show that Diktyo minimizes network latency across different infrastructure topologies while achieving similar execution times as current scheduling plugins. Also, practical experiments with microservice benchmark applications demonstrate that Diktyo increases throughput by 22% and reduces latency up to 45% compared to default plugins.

In conclusion, this dissertation is only a first step towards efficient resource allocation in Fog Computing architectures. Although considerable advancements have been made in the Fog Computing field in recent years, resource allocation research will continue to grow since emerging applications add even more complexity. Extended Reality (XR) and autonomous cars are among the most demanding use cases, pushing cloud architectures further by presenting novel challenges in the resource allocation field concerning reliability, throughput, and latency. Therefore, the work regarding resource allocation and service scheduling in future cloud architectures is far from finished.

1 Introduction

"If we knew what it was we were doing, it would not be called research, would it?"

–Albert Einstein (1879 - 1955)

This chapter situates the conducted research work, summarizes the main contributions, and outlines the structure of this dissertation. It also provides an overview of the publications that were authored during this research period. The thesis addresses the challenges introduced by the Internet of Things (IoT) in traditional cloud platforms. The Fog Computing paradigm, an evolution of Cloud Computing, is studied as a potential solution to deal with the massive growth of connected devices due to the IoT. In addition, the heterogeneity of Low Power Wide Area Network (LPWAN) technologies is evaluated based on several IoT use cases. Also, the concept of Service Function Chaining (SFC) is studied for IoT service placement in the container domain. All these concepts are investigated to achieve efficient resource allocation in a Fog Computing environment.

1.1 The Cloud Computing Revolution

Over the last few years, Cloud computing [1] has seen enormous growth since it has become the *defacto* standard for application deployments. End-users and devices connect to a cloud-based system where current applications are deployed. Accessing emails or sending documents over the Internet are examples of data currently being processed in the cloud. Cloud systems reduce expenses for enterprises since computational resources are requested via a cloud provider responsible for maintaining and upgrading the cloud system. However, challenges persist concerning the orchestration and management of cloud applications. Efficient allocations strategies are missing where cloud providers avoid unnecessary costs and optimize their hardware infrastructure. The aim is to minimize the number of allocated resources in the infrastructure without violating the Service Level Agreements (SLAs) agreed with end-users. In addition, recent application domains such as the Internet of Things (IoT) and Smart Cities have introduced novel challenges in the resource allocation field. Table 1.1 shows that IoT traffic has been increasing exponentially in the last few years, making current clouds inadequate for IoT applications due to their stringent requirements (e.g., low latency, low bandwidth capacity). Latency reduction plays a major role since even a small delay can drastically impact the performance of latency-sensitive applications (e.g., health monitoring or emergency response applications). Current cloud systems cannot maintain low latency for all these devices spread around the network. Therefore, to deal with the limitations of centralized clouds, the Fog Computing paradigm has been introduced.

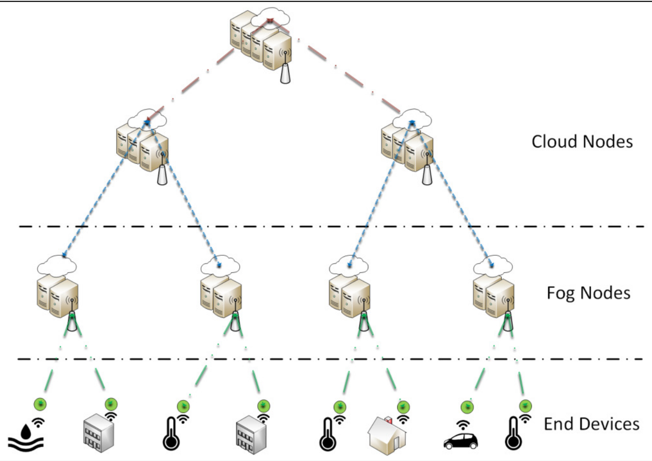
Table 1.1: The number of connections in the IoT ecosystem [2]

	2018	**2019**	**2020**	**2021**	**2022**	**2023**
Billions of connections	6.1	7.4	8.9	10.6	12.5	14.7

1.2 The need for Fog Computing

Fog Computing [3] [4] has emerged as an evolution of the Cloud Computing paradigm where computational resources are placed on the edges of the network to decrease the latency. Fig. 1.1 illustrates a Fog Computing architecture, where IoT devices, mainly sensors and actuators, communicate through wireless gateways, linked with the fog layer through multiple Fog Nodes (FNs). Each FN represents a small cloud entity that provides a given set of computational resources. FNs communicate with the cloud layer through Cloud Nodes (CNs), representing the top-level management entities. Although the theoretical foundations of Fog Computing have already been established, several challenges persist since Fog Computing is not a mature concept yet. One of the main challenges relates to resource allocation. Coming up with efficient allocation strategies in a distributed cloud infrastructure has been the main challenge in Fog Computing in the last few years. The distribution of computing resources adds further complexity to the service placement. Resources are available everywhere across the network area, service requests come from multiple locations, and sensors have limited bandwidth ca-

Figure 1.1 Overview of a Fog Computing architecture.



pacity. Allocating services far from devices would result in high communication latency since end devices and gateways lack in terms of processing power, storage capacity, and memory [5]. The real-time constraints of delay-sensitive IoT applications must be acknowledged in the resource allocation strategy while minimizing deployment costs and maximizing Quality of Service (QoS). Also, the heterogeneity of the hardware resources adds further complexity to the problem since an FN has a lower computing capacity than a CN. Thus, when selecting nodes for service placement, strategies need to consider the hosts' capacity (i.e., computing resources and bandwidth), application requirements, and hosts' location.

Fog Computing and Multi-access Edge Computing (MEC) are close concepts [6]. Fog Computing focuses on IoT and MEC focuses on the mobile network differing in the considered interactions (i.e., between edges and cloud). MEC aims to deploy services close to end-users to reduce latency and avoid congestion in the network core. MEC follows guidelines established by the European Telecommunications Standards Institute (ETSI) Network Function Virtualization (NFV) Management and Orchestration (MANO) [7] while Fog Computing follows architectural principles established by the ETSI Machine-to-Machine (M2M) technical committee [8]. In Fog Computing, bi-directional communications between fog and cloud nodes are crucial due to the hierarchical architecture. For example, a service requiring high computational requirements is allocated in the cloud, but it needs to interact with another service, which may be located in the fog. These interactions need to be considered in the allocation process, leading to complex service dependencies that must be guaranteed.

This dissertation addresses several challenges (e.g., heterogeneity of resources) introduced by Fog Computing concerning resource allocation.

1.3 The impact of Low Power Wide Area Networks

The heterogeneity of Low Power Wide Area Network (LPWAN) technologies introduces a novel set of challenges coming from the wireless domain [9]. LPWANs are used mostly by low-powered devices with low bandwidth capacity. The main advantage offered by these technologies is the long communication range, usually of a few kilometers. LPWANs operate at a lower cost with higher energy efficiency than traditional mobile networks. These solutions can support massive numbers of connected devices over a large area, making them suitable for M2M and IoT use cases. However, to decide which LPWAN technology is the most adequate for a given use case is not a trivial task. Selecting an LPWAN depends on the specific application and its requirements, including minimum communication range, minimum data rate, downlink capacity, security layer enabled, among others. Previous research on resource allocation only addressed constraints coming from the cloud domain. Little attention has been given to requirements stemming from the characteristics of wireless networks. When deploying IoT applications, cloud operators must consider which LPWAN technology will be used by IoT devices to access the deployed services. A smart metering use case or a latency-sensitive air quality application pose different challenges to the cloud system and LPWAN technology. The current LPWAN ecosystem contains a plethora of technologies with diverse characteristics far from mature [10]. The performance and scalability of these technologies are still uncertain since these LPWAN technologies have been only assessed through small-scale experiments and simulations.
This dissertation studies the heterogeneity of LPWAN technologies by designing IoT applications (e.g., air quality monitoring) and evaluating the most appropriate LPWANs for its implementation and deployment in Fog Computing environments.

1.4 The need for efficient Service Function Chaining

Service Function Chaining (SFC) placement [11], [12] has been studied in the network management domain during the last few years. An application is separated into a set of services connected in a specific order forming a service chain. SFC allows mobile operators to benefit from the high flexibility and low operational costs introduced by network softwarization. SFC offers a reliable alternative to today's static network environment. Service chaining has been mostly studied for Software-defined networking (SDN) and NFV use cases, including data center networks [13], carrier networks [14] and edge deployments [15]. The aim is to optimize resource allocation to improve application performance. Service Chaining standards are being developed by several industry research groups. The Internet Engineering Task Force (IETF) has an SFC architecture research group

[16] studying network flow classifications to be used in traffic routing between service functions. Also, ETSI has a research group [17] investigating network forwarding graphs and traffic routing via network service headers. Further, the Open Networking Foundation(ONF) [18] has already proposed an SDN service chaining framework based on the OpenFlow protocol [19] for traffic routing in service chains.
Nevertheless, SFC is still quite unexplored in container placement and in the IoT domain. Most efforts focus on virtual network embedding and Virtual Machine (VM) placement [20], [21]. Further research is needed to determine where to place each service in a fog-cloud infrastructure (i.e., edge, fog, cloud). Deciding where to run each service based on the required resources is crucial to meet the stringent requirements put forward by IoT applications. Service chaining can provide the flexibility and performance needed for emerging applications but its standardization is still ongoing and current deployments are not yet mature.
This dissertation investigates SFC in Fog Computing. IoT use cases have been assessed based on container-based service chains focused on Smart City contexts.

1.5 Challenges

IoT Applications can benefit from the many advantages of Fog Computing since services are deployed at the edge or fog location close to the devices, reducing the communication latency and optimizing the bandwidth usage. Reducing data transportation to the cloud is essential to avoid congestion since these devices have a low bandwidth capacity. Efficient resource allocation is thus important for both the cloud provider and the user. Reducing costs, maximizing QoS, and minimizing latency are examples of goals followed by allocation strategies. This dissertation investigates how computing resources can be efficiently managed in a Fog Computing environment. Both perspectives are considered: the user who wants to maximize its QoS and the cloud provider who intends to reduce its costs regarding the infrastructure without compromising SLAs. In this context, four main challenges have been addressed in this dissertation.
Challenge #1*: provide a benchmark for resource allocation research in Fog Computing.*
Deploying an application in a Fog Computing environment adds further complexity to a traditional deployment in a centralized cloud. Hardware resources, application requirements, fog locations, the bandwidth capacity of the infrastructure are among the constraints addressed in Fog-cloud environments. The performance of allocation strategies should be analyzed by considering different factors, such as latency reduction, energy efficiency, and bandwidth optimization. In recent years, several works have addressed resource allocation in Fog Computing (e.g., [22], [23]), but none of them addressed the real-time requirements of IoT applications.

In addition, the implications of LPWANs should be studied in the allocation process. The aim is to establish a relationship between the cloud and the wireless domain.

Challenge #2*: design a Fog Computing framework supporting autonomous management and orchestration functionalities.*

In recent years, the ETSI oneM2M [24] has been working towards an end-to-end (E2E) high-level architecture for M2M communication. The goal is to establish for M2M what the 3rd Generation Partnership Project (3GPP) realized for mobile networks. Several aspects are currently being addressed: security, data management, device authentication, M2M service subscription, among others. Designing a Fog Computing framework addressing all these aspects would help to standardize Fog Computing architectures. Several works have proposed IoT / Fog computing architectures [25] [26] with several functionalities. However, without a clear path to standardization, it is hard to integrate and combine efforts from both academia and industry.

Challenge #3*: implement efficient distributed data monitoring and analysis in a Fog Computing environment.*

Another important aspect of IoT applications is to monitor their behavior, especially those dealing with personal health monitoring [27], or emergency response services [28] since a small delay can impact their performance and produce severe consequences. Detecting unusual events or malfunctions beforehand is thus an important matter. Traditional anomaly detection approaches are not suitable for delay-sensitive IoT applications since these solutions would require sending all data to a centralized location which would result in high latency [29]. Also, data is not transported quickly enough due to the low bandwidth capacities of IoT devices. Fog Computing helps to reduce the latency since services are deployed close to devices. By identifying unusual events at the Fog level, malfunctions in IoT sensors can be detected, and transmissions of incorrect information are avoided, improving the overall QoS of IoT applications, especially in terms of reliability [30]. Therefore, designing an anomaly detection approach for IoT applications should be based on the advantages of Fog Computing architectures.

Challenge #4*: consider latency and bandwidth in the scheduling process in a container orchestration system.*

Emerging applications such as IoT and video streaming services have revolutionized the cloud computing field. By placing computing resources at the edge or fog, the stringent requirements introduced by these applications can be met, especially in terms of latency and bandwidth. The installation of dedicated hardware close to IoT devices reduces the latency and the consumed bandwidth towards the cloud. Efficient resource allocation strategies need to consider bandwidth and latency in the scheduling process, especially for these applications since its a primary objective. Traditional allocation strategies [31], [32] mostly focus on optimizing resource usages (e.g., CPU and RAM), and only a few consider network-aware or bandwidth-aware algorithms [33], [34]. In addition, containers [35] have been gaining popularity in the virtualization field. Their minimal overhead and high portability compared to traditional VMs have established containers as the main virtualization technology. Previous allocation strategies have been designed for VM allocation and migration, and little attention has been given to containerized applications. Also, SFC is relatively unexplored in the container area, while its major studies focus on VMs. The challenge is how to design efficient allocation strategies that address the latency and bandwidth requirements of these applications while supporting new trends as containerization and service chaining in fog-cloud infrastructures.

1.6 Outline

This dissertation is composed of several publications realized within the scope of this PhD. The selected publications provide an integral and consistent overview of the work performed. Sec. 1.7 details the different research contributions, and the complete list of publications that resulted from this dissertation is presented in Sec. 1.8. This section presents an overview of the remainder of this dissertation and explains how the different chapters are linked together. A schematic overview of how the Chapters (Ch.) and Appendices (App.) are related to each other and to the research contributions is depicted in Fig. 1.2. Table 1.2 illustrates how the chapters of this dissertation relate to the challenges listed in Sec. 1.5.
Chapter 2 (Ch. 2) studies resource allocation in a Fog Computing environment. The chapter presents an Integer Linear Programming (ILP) formulation for IoT application service placement that considers multiple optimization objectives, such as low latency and energy efficiency. The work addresses heterogeneous hardware capacities, different application requirements, and the network bandwidth between cluster nodes. The model is evaluated for IoT applications within the scope of Antwerp's City of Things testbed [36]. The formulation serves as a benchmark in future research related to placement issues of IoT application services in Fog Computing. The model approach is generic and can be applied to several IoT use cases without loss of generality.
Chapter 3 (Ch. 3) focuses on the design of a Fog Computing framework for Smart Cities. A Fog-based framework is proposed that follows the ETSI NFV MANO guidelines. It presents additional software components, such as monitoring and data analysis functionalities. In addition, the chapter describes a novel fog protocol for the exchange of application service information between FNs to provide fast service provisioning decisions for Smart City applications. An anomaly detection use case based on an air quality application has also been evaluated based on the proposed framework. The evaluation shows that the proposed framework

achieves a significant reduction in terms of network bandwidth usage compared to traditional centralized clouds. The approach only requires 1.46% of the available network bandwidth between a fog node and a cloud node.

Chapter 4 (Ch. 4) studies the feasibility of the *de-facto* container management platform named Kubernetes [37] for IoT applications. It extends the work presented in chapter 3 by proposing a Fog Computing framework based on the Kubernetes architectural model. Also, it extends the work in chapter 2 by developing a Network-Aware Scheduler (NAS) for container-based applications in Kubernetes. Evaluations have assessed the performance of the proposed heuristic compared to the ILP formulation presented in chapter 2 and the default scheduling component in Kubernetes. Results show that NAS achieves a reduction of up to 70% concerning network latency compared to the default mechanism.

Chapter 5 (Ch. 5) presents a network-aware framework named Diktyo for the Kubernetes platform inspired by service chaining concepts. The work extends chapter 4 by designing and implementing network-aware algorithms based on the Kubernetes scheduling plugin framework [38]. The work proposes multiple scheduling plugins to optimize the allocation of container-based service chains in Kubernetes focused on low latency and bandwidth optimization. It studies how this combination can approximate the optimal allocation scheme provided by a Mixed-Integer Linear Programming (MILP) model. Application domains such as IoT and video streaming would benefit the most from schedulers that consider latency and bandwidth requirements. Scheduling methods in current cloud platforms focus mostly on resource usage rates (e.g., CPU and RAM), which are inappropriate for applications where latency and bandwidth play a major role. Simulations show that Diktyo can minimize the network latency across different infrastructure topologies while achieving similar execution times as current scheduling plugins. Also, practical experiments with microservice benchmark applications demonstrate that Diktyo increases database throughput by 22% and reduces application response time by up to 45% compared to default plugins.

Appendix A (App. A) presents further insights on distributed data monitoring and analysis in Fog Computing addressed in chapter 3. An anomaly detection approach for Smart City applications has been designed for Antwerp's City of Things use cases. The most appropriate LPWAN technologies are investigated for the Smart City use case based on a large collected dataset: different cars collect air quality metrics consisting of particle matter indicators (e.g., PM1, PM2.5 and PM10) that are annotated with a Global Positioning System (GPS) location.

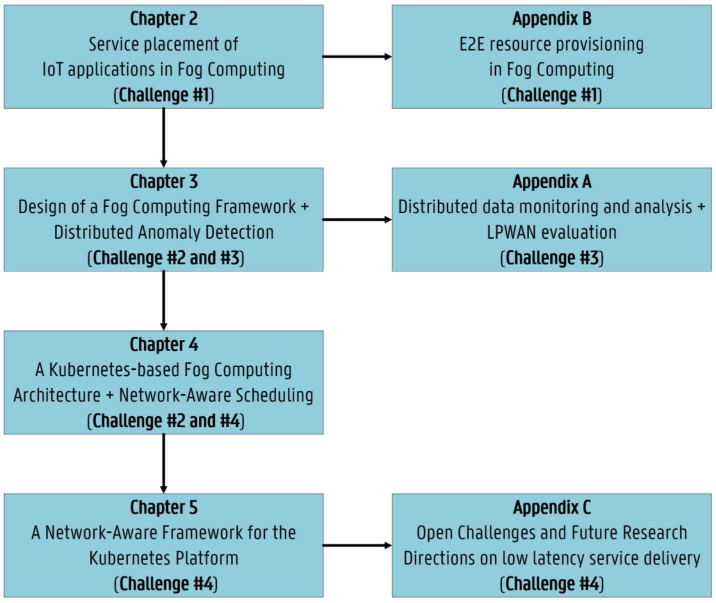
Appendix B (App. B) extends Chapter 2 by proposing a MILP model for the IoT service placement problem that considers service chaining, different LPWAN technologies, service replication, and several optimization objectives. The formulation presents further insights on the complete E2E resource provisioning in Fog-cloud environments. Evaluations show clear trade-offs between the assessed allocation strategies based on Smart City use cases.

Appendix C (App. C) discusses open challenges and future research directions on low-latency service delivery focused on emerging use cases. The advent of softwarized networks has enabled the deployment of service chains on computational resources from the cloud up to the edge, creating a continuum of virtual resources. The next generation of low latency applications (e.g., Virtual Reality (VR), autonomous cars) adds even more stringent requirements to the infrastructure, calling for considerable advancements towards cloud-native architectures.

Table 1.2: An overview of the contributions per chapter in this dissertation.

	Ch.2	Ch.3	Ch.4	Ch.5	App. A	App. B	App. C
Challenge #1	●					●	
Challenge #2		●	●				
Challenge #3		●			●		
Challenge #4			●	●			●

Figure 1.2 Schematic overview of the different chapters in this dissertation.


Chapter 2
Service placement of IoT applications in Fog Computing
(**Challenge #1**)
Appendix B
E2E resource provisioning in Fog Computing
(**Challenge #1**)
Chapter 3
Design of a Fog Computing Framework + Distributed Anomaly Detection
(**Challenge #2 and #3**)
Appendix A
Distributed data monitoring and analysis + LPWAN evaluation
(**Challenge #3**)
Chapter 4
A Kubernetes-based Fog Computing Architecture + Network-Aware Scheduling
(**Challenge #2 and #4**)
Chapter 5
A Network-Aware Framework for the Kubernetes Platform
(**Challenge #4**)
Appendix C
Open Challenges and Future Research Directions on low latency service delivery
(**Challenge #4**)

1.7 Research contributions

Sec 1.5 formulates the problems and challenges for efficient resource allocation in a Fog Computing environment. These challenges are addressed in the remainder of this dissertation for which the outline is given in Sec 1.6. To conclude, an elaborated list of the research contributions within this dissertation is given:

- Address IoT application deployments in Fog Computing. (Ch. 2 and App. B, mainly addressing Challenge #1)
  - Ch. 2 proposes an ILP formulation for IoT service placement in Fog Computing.
  - Evaluations show a clear trade-off between allocation strategies, such as low latency and low energy consumption.
  - The model can serve as a benchmark in future research related to placement issues of IoT services in Fog Computing environments since the model approach is generic and applies to a wide range of IoT applications.
  - App. B extends the work presented in Ch. 2 by focusing on service chaining, different LPWAN technologies, and multiple objectives.
  - Evaluations for Smart City use cases show clear trade-offs between the different provisioning strategies.
- Design a Fog Computing framework with management and orchestration functionalities. (Ch. 3 and Ch. 4, mainly addressing Challenge #2)
  - Ch. 3 proposes a Fog Computing framework that follows the ETSI NFV MANO architecture and the standards of the ETSI oneM2M concerning device management and security.
  - The work proposes an integrated and autonomous solution for efficient resource management and orchestration in Smart Cities.
  - Results show that the approach achieves a substantial reduction regarding bandwidth usage compared to centralized cloud solutions.
  - Ch. 4 extends Ch. 3 by evaluating the feasibility of the Kubernetes platform for IoT services.
  - The work presents a Fog Computing framework based on the Kubernetes architectural model.
- Implement efficient distributed data monitoring and analysis in Fog Computing. (Ch. 3 and App. A, mainly addressing Challenge #3)

  - Ch. 3 proposes an anomaly detection approach for 5G Smart Cities. A use case based on an Air Quality monitoring application shows the benefits of the proposed anomaly detection scheme.
  - App. A presents further details about the anomaly detection approach for Smart City applications in Fog Computing. It has been designed for the Antwerp's City of Things testbed and validated for Smart City use cases.
  - The most appropriate LPWAN technologies are selected for the evaluated use case.

- Consider latency and bandwidth in the scheduling process for application deployments. (Ch. 4, Ch. 5 and App. C, mainly addressing Challenge #4)
  - Ch. 4 presents a Network-Aware Scheduling (NAS) approach for Smart City container-based applications.
  - NAS is an extension to the default scheduling feature available in Kubernetes.
  - Results show that the proposed NAS can significantly improve the service placement concerning latency compared to the Kubernetes default scheduler.
  - The work has been open-sourced [39] and thus available for further experiments by the network management community.
  - Ch. 5 extends the work presented in Ch. 4 by designing a network-aware framework for the Kubernetes platform by collaborating with IBM Research.
  - The network-aware framework has been implemented based on the Kubernetes scheduling framework to ease its addition to the Kubernetes project.
  - App. C discusses open challenges and future directions concerning low latency service delivery for emerging use cases.

1.8 Publications

The results obtained during this PhD research have been published in scientific journals and presented at a series of international conferences. The following list provides an overview of these publications.

1.8.1 Publications in International Journals

[1] **J. Santos**, T. Wauters, B. Volckaert, and F. De Turck. *Fog computing: Enabling the management and orchestration of smart city applications in 5G networks.* Published in Entropy, Volume 20, no. 1, pages 4, MDPI, 2018.

[2] **J. Santos**, T. Vanhove, M. Sebrechts, T. Dupont, W. Kerckhove, B. Braem, G. Van Seghbroeck, T. Wauters, P. Leroux, S. Latre, B. Volckaert, and F. De Turck. *City of things: Enabling resource provisioning in smart cities.* Published in IEEE Communications Magazine, Volume 56, no. 7, pages 177-183, IEEE, 2018.

[3] **J. Santos**, T. Wauters, B. Volckaert, and F. De Turck. *Resource provisioning in fog computing: From theory to practice.* Published in Sensors, Volume 19, no. 10, pages 2238, MDPI, 2019.

[4] **J. Santos**, T. Wauters, B. Volckaert, and F. De Turck. *Towards end-to-end resource provisioning in Fog Computing over Low Power Wide Area Networks.* Published in Journal of Network and Computer Applications, Volume 175, no. 10, pages 102915, Elsevier, 2021.

[5] **J. Santos**, T. Wauters, B. Volckaert, and F. De Turck. *Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and Research Directions.* Published in IEEE Communications Surveys & Tutorials, Volume 23, no. 4, pages 2557-2589, 2021.

1.8.2 Publications in Book Chapters

[1] **J. Santos**, T. Wauters, B. Volckaert, and F. De Turck. *Reinforcement learning for service function chain allocation in fog computing.* Published in Communication Networks and Service Management in the Era of Artificial Intelligence and Machine Learning, pages 147-173, 2021.

1.8.3 Publications in International Conferences

[1] **J. Santos**, T. Wauters, B. Volckaert, and F. De Turck. *Resource provisioning for IoT application services in smart cities.* Published in Proceedings of the 13th International Conference on Network and Service Management (CNSM), 2017, pages 1 - 9, IEEE.

[2] **J. Santos**, P. Leroux, T. Wauters, B. Volckaert, and F. De Turck. *Anomaly detection for smart city applications over 5g low power wide area networks.* Published in Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 1-9, 2018, IEEE.

[3] **J. Santos**, P. Leroux, T. Wauters, B. Volckaert, and F. De Turck. *Towards dynamic fog resource provisioning for smart city applications.* Published in Proceedings of the 14th International Conference on Network and Service Management (CNSM), pages 290-294, 2018, IEEE.

[4] **J. Santos**, T. Wauters, B. Volckaert, and F. De Turck. *Towards network-aware resource provisioning in Kubernetes for fog computing applications.* Published in Proceedings of the IEEE Conference on Network Softwarization (NetSoft), pages 351-359, 2019, IEEE.

[5] **J. Santos**, T. Wauters, B. Volckaert, and F. De Turck. *Towards delay-aware container-based service function chaining in fog computing.* Published in Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 1-9, 2020, IEEE.

[6] **J. Santos**, T. Wauters, B. Volckaert, and F. De Turck. *Live Demonstration of Service Function Chaining allocation in Fog Computing.* Published in Proceedings of the IEEE Conference on Network Softwarization (NetSoft), pages 362-364, 2020, IEEE.

[7] L. N. Vijouyeh, M. Sabaei, **J. Santos**, T. Wauters, B. Volckaert, and F. De Turck. *Efficient Application Deployment in Fog-enabled Infrastructures.* Published in Proceedings of the 16th International Conference on Network and Service Management (CNSM), pages 1-9, 2020, IEEE.

[8] **J. Santos**, J. van der Hooft, M. Torres Vega, T. Wauters, B. Volckaert, and F. De Turck. *SRFog: A flexible architecture for Virtual Reality content delivery through Fog Computing and Segment Routing.* Published in Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 1038-1043, 2021.

[9] **J. Santos**, T. Wauters, B. Volckaert, and F. De Turck. *Resource provisioning in fog computing through deep reinforcement learning.* Published in Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 1-7, 2021.

[10] **J. Santos**, J. van der Hooft, M. Torres Vega, T. Wauters, B. Volckaert, and F. De Turck. *Efficient orchestration of service chains in fog computing for immersive media.* Published in Proceedings of the 17th International Conference on Network and Service Management (CNSM), pages 139-145, 2021.

[11] **J. Santos**, Chen Wang, T. Wauters, and F. De Turck. *Diktyo: Network-aware Scheduling for Container Cloud.* Submitted to the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI), December 2021.

1.9 Code Repositories

During the dissertation, the candidate advocated for open and reproducible research. The following list provides an overview of all public code repositories generated by the candidate during his PhD research.

[1] SFC-controller
URL: `https://github.com/jpedro1992/sfc-controller`.

[2] gym-fog
URL: `https://github.com/jpedro1992/gym-fog`.

[3] Pushing Netperf Metrics to Prometheus
URL: `https://github.com/jpedro1992/pushing-netperf-metrics-to-prometheus`.

[4] Bandwidth as an extended resource
URL: `https://github.com/jpedro1992/bandwidth-as-an-extended-resource`.

[5] Network-aware algorithms for the Scheduler Plugins framework
URL: `https://github.com/jpedro1992/scheduler-plugins`.

Bibliography

[1] Ali Sunyaev. Cloud computing. In *Internet computing*, pages 195–236. Springer, 2020.

[2] Cisco Annual Internet Report, 2018–2023, 2021. URL https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf.

[3] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 workshop on mobile big data*, pages 37–42, 2015.

[4] Ranesh Kumar Naha, Saurabh Garg, Dimitrios Georgakopoulos, Prem Prakash Jayaraman, Longxiang Gao, Yong Xiang, and Rajiv Ranjan. Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE access*, 6:47980–48009, 2018.

[5] Hlabishi I Kobo, Adnan M Abu-Mahfouz, and Gerhard P Hancke. A survey on software-defined wireless sensor networks: Challenges and design requirements. *IEEE access*, 5:1872–1899, 2017.

[6] Paolo Bellavista, Javier Berrocal, Antonio Corradi, Sajal K Das, Luca Foschini, and Alessandro Zanni. A survey on fog computing for the internet of things. *Pervasive and mobile computing*, 52:71–99, 2019.

[7] Mehmet Ersue. Etsi nfv management and orchestration-an overview. *Presentation at the IETF*, 88, 2013.

[8] Jorg Swetina, Guang Lu, Philip Jacobs, Francois Ennesser, and JaeSeung Song. Toward a standardized common m2m service layer platform: Introduction to onem2m. *IEEE Wireless Communications*, 21(3):20–26, 2014.

[9] Bharat S Chaudhari, Marco Zennaro, and Suresh Borkar. Lpwan technologies: Emerging application characteristics, requirements, and design considerations. *Future Internet*, 12(3):46, 2020.

[10] Kais Mekki, Eddy Bajic, Frederic Chaxel, and Fernand Meyer. A comparative study of lpwan technologies for large-scale iot deployment. *ICT express*, 5(1):1–7, 2019.

[11] Deval Bhamare, Raj Jain, Mohammed Samaka, and Aiman Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155, 2016.

[12] Ahmed M Medhat, Tarik Taleb, Asma Elmangoush, Giuseppe A Carella, Stefan Covaci, and Thomas Magedanz. Service function chaining in next generation networks: State of the art and research challenges. *IEEE Communications Magazine*, 55(2):216–223, 2016.

[13] BS Lakshmi and J Lakshmi. Integrating service function chain management into software defined network controller. In *2019 IEEE World Congress on Services (SERVICES)*, volume 2642, pages 160–165. IEEE, 2019.

[14] Long Qu, Maurice Khabbaz, and Chadi Assi. Reliability-aware service chaining in carrier-grade softwarized networks. *IEEE Journal on Selected Areas in Communications*, 36(3):558–573, 2018.

[15] Yeonghun Nam, Sooeun Song, and Jong-Moon Chung. Clustered nfv service chaining optimization in mobile edge clouds. *IEEE Communications Letters*, 21(2):350–353, 2016.

[16] Joel Halpern, Carlos Pignataro, et al. Service function chaining (sfc) architecture. In *RFC 7665*. 2015.

[17] GRNFVIFA ETSI. 024:"network functions virtualisation (nfv) release 3. *Management and Orchestration*.

[18] Cathy Zhang, S Addepalli, N Murthy, L Fourie, M Zarny, and L Dunbar. L4-l7 service function chaining solution architecture. *Open Networking Foundation, ONF TS-027*, 2015.

[19] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74, 2008.

[20] Hassan Hawilo, Manar Jammal, and Abdallah Shami. Network function virtualization-aware orchestrator for service function chaining placement in the cloud. *IEEE Journal on Selected Areas in Communications*, 37(3):643–655, 2019.

[21] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Steven Davy. Design and evaluation of algorithms for mapping and scheduling of virtual network functions. In *Proceedings of the 2015 1st IEEE conference on network softwarization (NetSoft)*, pages 1–9. IEEE, 2015.

[22] Karima Velasquez, David Perez Abreu, Marilia Curado, and Edmundo Monteiro. Service placement for latency reduction in the internet of things. *Annals of Telecommunications*, pages 1–11, 2016.

[23] Chunqiang Tang, Malgorzata Steinder, Michael Spreitzer, and Giovanni Pacifici. A scalable application placement controller for enterprise data centers. In *Proceedings of the 16th international conference on World Wide Web*, pages 331–340. ACM, 2007.

[24] ETSI oneM2M (2016). Etsi technical specification, onem2m functional architecture. onem2m ts-0001 version 2.10.0 release 2., 2017. URL http://www.etsi.org/deliver/etsi_ts/118100_118199/118101/\02.10.00_60/ts_118101v021000p.pdf.

[25] Luis Sánchez, Verónica Gutiérrez, José Antonio Galache, Pablo Sotres, Juan Ramón Santana, Javier Casanueva, and Luis Muñoz. Smartsantander: Experimentation and service provision in the smart city. In *Wireless Personal Multimedia Communications (WPMC), 2013 16th International Symposium on*, pages 1–6. IEEE, 2013.

[26] David Perez Abreu, Karima Velasquez, Marilia Curado, and Edmundo Monteiro. A resilient internet of things architecture for smart cities. *Annals of Telecommunications*, pages 1–12, 2016.

[27] Giada Giorgi, Alessandra Galli, and Claudio Narduzzi. Smartphone-based iot systems for personal health monitoring. *IEEE Instrumentation & Measurement Magazine*, 23(4):41–47, 2020.

[28] Boyi Xu, Li Da Xu, Hongming Cai, Cheng Xie, Jingyuan Hu, and Fenglin Bu. Ubiquitous data accessing method in iot-based information system for emergency medical services. *IEEE Transactions on Industrial informatics*, 10(2):1578–1586, 2014.

[29] Hany F Atlam, Robert J Walters, and Gary B Wills. Fog computing and the internet of things: A review. *big data and cognitive computing*, 2(2):10, 2018.

[30] Lingjuan Lyu, Jiong Jin, Sutharshan Rajasegarar, Xuanli He, and Marimuthu Palaniswami. Fog-empowered anomaly detection in iot using hyperellipsoidal clustering. *IEEE Internet of Things Journal*, 4(5):1174–1184, 2017.

[31] Hendrik Moens, Brecht Hanssens, Bart Dhoedt, and Filip De Turck. Hierarchical network-aware placement of service oriented applications in clouds. In *Network Operations and Management Symposium (NOMS)*, pages 1–8. IEEE, 2014.

[32] Shohreh Ahvar, Hnin Pann Phyu, Sachham Man Buddhacharya, Ehsan Ahvar, Noel Crespi, and Roch Glitho. Ccvp: Cost-efficient centrality-based vnf placement and chaining algorithm for network service provisioning. In *2017*

IEEE Conference on Network Softwarization (NetSoft), pages 1–9. IEEE, 2017.

[33] Leonardo R Rodrigues, Marcelo Pasin, Omir C Alves, Charles C Miers, Mauricio A Pillon, Pascal Felber, and Guilherme P Koslovski. Network-aware container scheduling in multi-tenant data center. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

[34] Amanda Jayanetti and Rajkumar Buyya. J-opt: A joint host and network optimization algorithm for energy-efficient workflow scheduling in cloud data centers. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, pages 199–208, 2019.

[35] Namiot Dmitry and Sneps-Sneppe Manfred. On micro-services architecture. *International Journal of Open Information Technologies*, 2(9), 2014.

[36] Jose Santos, Thomas Vanhove, Merlijn Sebrechts, Thomas Dupont, Wannes Kerckhove, Bart Braem, Gregory Van Seghbroeck, Tim Wauters, Philip Leroux, Steven Latre, et al. City of things: Enabling resource provisioning in smart cities. *IEEE Communications Magazine*, 56(7):177–183, 2018.

[37] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. *Communications of the ACM*, 59(5): 50–57, 2016.

[38] Kubernetes. Scheduling framework. Accessed on 15 October 2021. [Online]. Available: https://kubernetes.io/docs/concepts/scheduling-eviction/scheduling-framework/.

[39] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Sfc controller, an extension to the default scheduler (kube-scheduler) in kubernetes to enable scheduling in terms of latency and bandwidth. Accessed on 15 July 2021. [Online]. Available: https://github.com/jpedro1992/sfc-controller.

2 Resource Provisioning for IoT application services in Smart Cities

"The way to get started is to quit talking and begin doing."

–Walt Disney (1901- 1966)

The second chapter of this dissertation presents an Integer Linear Programming (ILP) formulation for Internet of Things (IoT) service placement discussed in chapter 1. The chapter evaluates the trade-offs between different allocation strategies, such as minimizing latency and maximizing energy efficiency. Latency is related to the network latency in the communication between cluster nodes where services that compose an IoT application are running and the end device that requested the IoT application. Energy efficiency is related to the number of cluster nodes used for the service deployment. The model considers several constraints, including hardware capacities, network bandwidth, and path loss. The model addresses cloud requirements and characteristics stemming from the wireless domain, which have not yet been explored in-depth in literature. The cloud model is based on previous work performed by Moens et al. on network-aware placement of service oriented applications in clouds. Thus, the main contribution of this chapter is an ILP formulation that can serve as a benchmark for resource allocation research in Fog Computing. The model is validated for IoT services without loss of generality.

⋆ ⋆ ⋆

José Santos, Tim Wauters, Bruno Volckaert and Filip De Turck.

Published in IEEE International Conference on Network and Service Management (CNSM), pages 1-9, 2017.

Abstract In the last years, traffic over wireless networks has been increasing exponentially, due to the impact of Internet of Things (IoT) and Smart Cities. Current networks must adapt to and cope with the specific requirements of IoT applications since resources can be requested on-demand simultaneously by multiple devices on different locations. One of these requirements is low latency, since even a small delay for an IoT application such as health monitoring or emergency service can drastically impact their performance. To deal with this limitation, the Fog computing paradigm has been introduced, placing cloud resources on the edges of the network to decrease the latency. However, deciding which edge cloud location and which physical hardware will be used to allocate a specific resource related to an IoT application is not an easy task. Therefore, in this chapter, an Integer Linear Programming (ILP) formulation for the IoT application service placement problem is proposed, which considers multiple optimization objectives such as low latency and energy efficiency. Solutions for the resource provisioning of IoT applications within the scope of Antwerp's City of Things testbed have been obtained. The result of this work can serve as a benchmark in future research related to placement issues of IoT application services in Fog Computing environments since the model approach is generic and applies to a wide range of IoT use cases.

2.1 Introduction

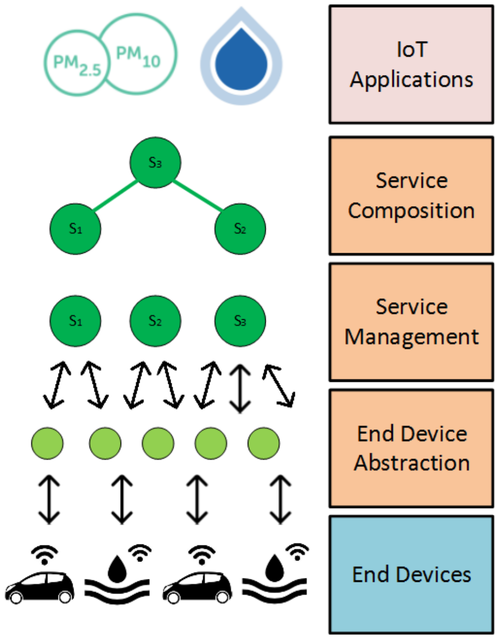
In recent years, the Internet of Things (IoT) has introduced a whole new set of challenges and opportunities by transforming objects of everyday life in communicating devices [1]. Moreover, with the advent of the IoT, the concept of Smart City has become even more popular in the last few years [2]. Smart City applications will transform a wide range of services in different domains of urban life, for instance, by creating intelligent smart grid networks, improving public transportation, developing smart car parking and real-time industrial automation applications and reducing traffic congestion. Essentially, millions of devices will be connected to the network, sending and receiving data to the cloud, which current networks will not be able to support [3]. Therefore, it is necessary to adapt existing cloud and network architectures to future needs and design and develop new management functionalities to help meet the strict requirements of future Smart City IoT applications. Fog Computing extends the Cloud Computing paradigm by bringing cloud services closer to the end devices, thus reducing the communication latency [4], [5]. However, there is still a large number of research challenges associated

with this approach since Fog Computing is in its early stages and needs more time to evolve. One of the main challenges is the proper resource allocation, since services can be placed in a highly congested location, or even further from the end devices, which would result in a higher communication latency because current end devices and gateways are lacking in terms of processing power, storage capacity and memory [6]. Moreover, few resource management strategies are currently addressing the real-time constraints of Smart City IoT applications while minimizing resource costs and maximizing quality of service (QoS). Therefore, efficient resource allocation strategies are needed in order to address all these issues.
This chapter presents an Integer Linear Programming (ILP) formulation for the IoT application service placement problem in order to evaluate resource provisioning in Smart City scenarios. IoT applications have been considered as a set of multiple communicating services, like applications designed in Service-Oriented Architectures (SOA). SOA-based architectures have been used in the last years for IoT [7], [8]. This way, an IoT application can be designed as a coordinated workflow of multiple services which are associated with actions performed by end devices. Research have been carried out to solve the issues of abstracting end device functionalities, trying to provide a suitable architecture with service management and composition capabilities able to link a set of common services in a set of IoT applications. This proposed architecture is presented in Fig. 2.1. Each communicating service can be provided by a Virtual Machine (VM) which may be used by multiple tenants. In a Smart City scenario, when there is a request for an IoT application, resources should be distributed within the network ensuring that the services composing the IoT application are allocated and instantiated close to the end device that made the request. Multiple factors should be taken into account to ensure proper resource allocation such as latency, energy efficiency, bandwidth and cost.
The remainder of the chapter is organized as follows. In the next Section, related work is discussed. Section III introduces the proposed ILP model for the resource provisioning of IoT application services. In Section IV, evaluation scenario is described which is followed by the evaluation results in Section V. Finally, conclusions are presented in Section VI.

2.2 Related Work

In recent years, studies have been carried out in order to deal with application placement issues in IoT. In [9], a model and an architecture have been introduced to deal with resource provisioning in fog computing environments focusing on the reduction of service latency for IoT applications. In [10], a energy management strategy for a Fog Computing platform is presented. Moreover, SmartSantander [11] worked on a suitable architectural model for the IoT and the inherent chal-

Figure 2.1 SOA-based architecture for IoT applications.



lenges of service provisioning in Smart Cities was in their scope [12]. In [13], a resilient IoT architecture for Smart Cities has been presented and in [14] the remaining issues of integrating Cloud Computing and IoT are discussed, where the integration was referred to as Cloud of Things. In [15], a novel scheme for an energy efficient IoT based on Wireless Sensor Networks (WSN) has been introduced focusing on WSN characteristics. Cloud requirements have not been included on the model.

In recent years, research efforts have been carried out to overcome application placement issues mainly focused on cloud environments where IoT or Smart Cities contexts have not been considered. Many works focused only on the allocation of virtual network functions (VNFs) or VMs on clouds [16], [17]. However, recently in [18], a resource aware placement algorithm of IoT applications in Fog Computing environments has been presented focusing on latency, network usage and energy consumption. Only static network topologies have been evaluated and no wireless constraint has been introduced. Nevertheless, a lot of challenges still remain to fully address resource provisioning of Smart City IoT applications, since previous research does not take into account requirements stemming from the characteristics of wireless networks. This way, in this chapter, a resource provisioning ILP model is presented that goes beyond the current state-of-the-art by taking into account not only cloud requirements but also wireless constraints which were, to the best of our knowledge, not yet explored in-depth in literature. The main advantage of ILP is the flexibility to analyze complex problems as the IoT application service issue presented in this chapter. However, ILP models can only be solved if there are clear linear relationships between all the different variables.

2.3 The ILP model

2.3.1 Model Description

The resource provisioning model considers cloud and wireless characteristics. The cloud model is based on the previous work done by Moens et al. [16] on network-aware placement of service oriented applications in clouds. Regarding wireless characteristics, an IEEE 802.11ah [19] Low-Power Wide-Area Network (LPWAN) has been modeled as an ILP formulation. An IoT application is composed of multiple communicating services. End devices send requests for these IoT applications through wireless gateways. These gateways communicate with the fog-cloud infrastructure, managing a set of computational resources. Each service must be allocated and instantiated on a given set of computational resources, subject to multiple constraints [16]:

- Computational resources have limited CPU and memory.
- Communication links between computational resources have limited bandwidth.
- Gateways have limited association identifiers (AIDs) so end devices can associate and send requests for IoT applications.
- IoT application services cannot be instantiated on every computational resource, due to specific hardware or software requirements.

The work in [16] incorporates multiple optimization objectives which have been extended to address the IoT application placement problem identified in this chapter. This way, the model is executed iteratively so that in each iteration a different optimization objective is considered. To retain the objective values obtained in the previous iterations, additional constraints are added to the model. Thus, the solution space continuously decreases since iterations must satisfy the previous optimal solutions. Every iteration refines the previous obtained solution by improving the model with an additional optimization objective. The optimization objectives considered in the model are the following:

1) Maximization of accepted IoT application requests.
2) Maximization of service bandwidth.
3) Minimization of service migrations between iterations.

4) Minimization of number of active comp. nodes.
5) Minimization of the number of active gateways.
6) Minimization of hop count between comp. nodes and end devices.
7) Minimization of path loss.

Optimization objectives from 1) to 4) have been already considered in [16]. The work has been extended with three additional optimization objectives, from 5) to 7), which are related with wireless formulations.

2.3.2 Variables

Table 2.1: Input variables related to the cloud infrastructure.

Symbol	Description
N_c	The set of comp. nodes on which services are executed.
N_f	The set of fog clouds on the network which manage the comp. nodes.
A	The set of all IoT applications. Each IoT application is composed of a set of communicating services.
S	The set of all communicating services.
R_a	The set containing all requests for an application $a \varepsilon A$.
D_a	The total number of requests for an application $a \varepsilon A$.
Ω_n	The total CPU capacity (in GHz) of the comp. node $n \varepsilon N_c$.
Γ_n	The total memory capacity (in GB) of the comp. node $n \varepsilon N_c$.
ω_s	The CPU requirement (in GHz) of the service $s \varepsilon S$.
γ_s	The memory requirement (in GB) of the service $s \varepsilon S$.
$R_{s,n}$	The Relation matrix. If $R_{s,n} = 1$, the communicating service s can be allocated on node n. If $R_{s,n} = 0$, the communicating service s cannot be instantiated on node n.
$I_{a,s}$	The Instance matrix. If $I_{a,s} = 1$, the communicating service s is part of application a. If $I_{a,s} = 0$, the communicating service s is not part of application a.
B_{n_1,n_2}	The Bandwidth matrix between comp. nodes indicates the bandwidth (Mbit/s) available between the comp. node n_1 and the comp. node n_2.
C_{n_1,n_2}	The Communication matrix between comp. nodes indicates the bandwidth (Mbit/s) required between services of an IoT application.
$E_{f,l}$	$E_{f,l} = 1$ indicates that fog cloud f is at location l.
$E_{n,f}$	$E_{n,f} = 1$ indicates that comp. node n is managed by fog cloud f.
$E_{n,l}$	$E_{n,l} = 1$ indicates that comp. node n is at location l.

Table 2.2: Input variables related to the wireless dimensioning.

Symbol	Description
N_{gw}	The set of wireless gateways on the network.
N_{ed}	The set of end devices on the network.
R_{ed}	A binary value that indicates if the end device ed sent a request for an IoT application $a \varepsilon A$.
Θ_{gw}	The total association identifiers available on a gateway $gw \varepsilon N_{gw}$.
θ_{ed}	Each end device $ed \varepsilon N_{ed}$ needs an association identifier to associate with a gateway.
L	Set of locations where IoT application requests are generated by end devices.
$\Phi_{a,r,ed}$	The Request matrix. If $\Phi_{a,r,ed} = 1$, the end device ed made the rth request of application a.
$H_{n,ed}$	The Hop Count matrix indicates the number of devices between the comp. node n and the end device ed.
$D_{gw,ed}$	The Distance matrix indicates the distance (in meters) between a gateway and an end device.
$PL_{gw,ed}$	The Path Loss matrix indicates the path loss (in dB) between a gateway and an end device.
$A_{ed,gw}$	The Association matrix. If $A_{ed,gw} = 1$, the end device ed can associate with the gateway gw. If $A_{ed,gw} = 0$, the end device ed cannot associate with the gateway gw.
$E_{gw,l}$	$E_{gw,l} = 1$ indicates that gateway gw is at location l.
$E_{ed,l}$	$E_{ed,l} = 1$ indicates that end device ed is at location l.

Table 2.3: Decision variables of the ILP model.

Symbol	Description
$G_{a,r}$	The acceptance matrix. If $G_{a,r} = 1$, the rth request of IoT application a is accepted. If $G_{a,r} = 0$, the rth request of IoT application a cannot be accepted.
$P^{a,r}_{s,n}$	The placement matrix. If $P^{a,r}_{s,n} = 1$, an instance of service s is executed on comp. node n for the rth request of IoT application a.
$U_{s,n}$	The service execution matrix. If $U_{s,n} = 1$, an instance of service s is allocated on comp. node n. If $U_{s,n} = 0$, there is not an instance of service s allocated on comp. node n.
$U_{ed,gw}$	The end device execution matrix. If $U_{ed,gw} = 1$, the end device ed is associated with gateway gw.
U_{gw}	The gateway utilization matrix. $U_{gw} = 1$ indicates that there is at least one end device associated with gateway gw.
U_n	The comp. node utilization matrix. $U_n = 1$ indicates that there is at least one service allocated on comp. node n.
$F^{a,r}_{s_1,s_2}(n_1, n_2)$	The flow matrix contains the bandwidth (in Mbit/s) belonging to the rth request of IoT application a that is used in the communication between services s_1 and s_2 which are allocated on node n_1 and n_2, respectively.
$z^{a,r}_{s_1,s_2}$	The service bandwidth matrix contains the amount of bandwidth for every flow in the communication between services s_1 and s_2 for the rth request of IoT application a.

Input variables used in the model are shown in Table 2.1 and in Table 2.2 while decision variables are shown in Table 2.3. All input variables related to the wireless dimensioning have been added to previous work as well as four new variables addressing cloud requirements (17 new variables, representing 57% of the total input variables alongside two new decision variables for the wireless dimensioning). A set of applications A composed of communicating services S are given. The number of requests and the total number of requests for an application $a \varepsilon A$ are given by R_a and D_a respectively. A binary request matrix $\Phi_{a,r,ed}$ indicates if an end device $ed \varepsilon N_{ed}$ made the rth request of application a. Also, a binary instance matrix I indicates if a service $s \varepsilon S$ is part of an application $a \varepsilon A$. Each service s has a CPU and a memory requirement represented by ω_s (in GHz) and γ_s (in GB) respectively. The communicating services must be allocated on computational nodes (comp. nodes) $n \varepsilon N_c$. Each comp. node n has a CPU and a memory capacity represented by Ω_n (in GHz) and Γ_n (in GB), respectively. A binary relation matrix R is used to indicate if an instance of service s could be allocated on a given comp. node $n \varepsilon N_c$. If $R_{s,n} = 1$, the communicating service s can be allocated on node n. Otherwise, due to software or hardware limitations, the service s cannot be instantiated on node n. Moreover, a binary acceptance matrix G is used to indicate if the rth request of application a can be accepted. If $G_{a,r} = 1$, all the services that compose application a are allocated on comp. nodes $n \varepsilon N_c$ and therefore the rth request of application a is accepted. A binary placement matrix P is used to represent in which comp. node n an instance of a service s is allocated. If $P^{a,r}_{s,n} = 1$, an instance of service s is executed on the comp. node n for the rth request of the IoT application a. A set of locations L is used to define where IoT applications requests are generated. Multiple binary matrices E are considered to define in which location fog clouds $f \varepsilon N_f$, end devices $ed \varepsilon N_{ed}$, gateways $gw \varepsilon N_{gw}$ and comp. nodes $n \varepsilon N_c$ are on the network. One additional binary matrix E is considered to indicate if a comp. node n is managed by a fog cloud f.

Regarding wireless formulation, the total AIDs available on a given gateway $gw \varepsilon N_{gw}$ is given by Θ_{gw}. Each end device $ed \varepsilon N_{ed}$ needs an AID to associate with a gateway which is represented by θ_{ed}. Moreover, a distance matrix D indicates the distance (in meters) between a gateway $gw \varepsilon N_{gw}$ and an end device $ed \varepsilon N_{ed}$ while a path loss matrix PL indicates the path loss (in dB) between a gateway $gw \varepsilon N_{gw}$ and an end device $ed \varepsilon N_{ed}$. A R_{ed} binary variable indicates if an end device ed sent requests for an IoT application. If $R_{ed} = 1$, the end device ed sent a request for an IoT application $a \varepsilon A$. Otherwise, $R_{ed} = 0$. An additional binary association matrix A is used to indicate if an end device ed can associate with a gateway gw. This association is based on the distance matrix D. If $D_{gw,ed}$ is less than one thousand meters, the end device ed can associate with gateway gw and then $A_{ed,gw} = 1$. Otherwise, $A_{ed,gw} = 0$. The limit is set to one thousand meters

because this is the maximum coverage range in IEEE 802.11ah networks[20]. A hop count matrix H indicates the number of devices between the comp. node n and the end device ed. Moreover, additional decision execution and utilization matrices U are considered. First, service execution matrix $U_{s,n}$ and end device execution matrix $U_{ed,gw}$ indicate if a service instance s is allocated on comp. node n and if an end device ed is associated with the gateway gw, respectively. Secondly, comp. node utilization matrix U_n and gateway utilization matrix U_{gw} indicate if there is at least a service running on comp. node n and if there is at least an end device associated with gateway gw, respectively.
Then, as mentioned by Moens et al. [16], a two-stage approach has been considered to allocate the bandwidth between communication services s_1 and s_2 that are part of the same IoT application request r. A bandwidth matrix B which indicates the available bandwidth (in Mbit/s) between two comp. nodes is considered. Moreover, a communication matrix C is defined, where C_{s_1,s_2} indicates the needed bandwidth (in Mbit/s) between two communication services. The flow bandwidth between two comp. nodes is defined by the flow matrix F. $F^{a,r}_{s_1,s_2}(n_1, n_2)$ contains the bandwidth (in Mbit/s) belonging to the rth request of IoT application a that is used in the communication between services s_1 and s_2 which are allocated on node n_1 and n_2, respectively. Finally, $z^{a,r}_{s_1,s_2}$ is a decision variable that indicates the percentage of the requested bandwidth between services s_1 and s_2 that is guaranteed for the rth request of the IoT application a.
Each optimization objective is detailed below. The input and decision variables included in the descriptions are all related with novel variables.

2.3.3 Maximizing the Number of Accepted Requests - MAX R

The goal of this optimization is to maximize the number of accepted requests on the network. This objective can be represented as shown in (2.1). This optimization objective is subjected to multiple constraints. Constraints presented in [16] have been considered in the model, which has been extended with additional ones related to the wireless formulation.

$$max \sum_{a \, \varepsilon \, A} \sum_{r \, \varepsilon \, R_a} G_{a,r} \times \left(\sum_{s \, \varepsilon \, S} I_{a,s} \times \omega_s \right) \tag{2.1}$$

In IEEE 802.11ah networks, end devices associate with gateways through an AID, a unique value assigned to an end device by the gateway during association handshake [21]. A gateway cannot have more than 8191 associated stations according to the latest standard. However, the association limitation has been set to 50 since an urban macro deployment with extended range has been considered [21]. This way, with this lower limitation, it has been assumed that good channel conditions are always achieved and that all requests sent for IoT applications can be accepted.

Therefore, a constraint must be added to the model ensuring that the AIDs limit in each gateway is respected. This way, by using the end device execution matrix U, the AIDs limitation can be expressed as shown in (2.2). The total amount of AIDs attributed in a gateway must be less than the total amount of AIDs available.

$$\forall gw \,\varepsilon\, N_{gw} : \sum_{ed \,\varepsilon\, N_{ed}} \theta_{ed} \times U_{ed,gw} \leq \Theta_{gw} \tag{2.2}$$

Secondly, a constraint is added to ensure that end devices are associated with one gateway to be able to send requests for IoT applications. This constraint is represented by (2.3).

$$\forall ed \,\varepsilon\, N_{ed} : \sum_{gw \,\varepsilon\, N_{gw}} U_{ed,gw} \times A_{ed,gw} = 1 \tag{2.3}$$

2.3.4 Maximizing the Satisfied Service Bandwidth Demand - MAX SB

On the previous objective, an IoT application is allocated on the network if at least 80% of the required bandwidth is guaranteed. The goal of this optimization is to further increase the allocated bandwidth, ensuring that the maximum capacity available is allocated to the communicating services that compose the IoT applications requested on the network. This maximization is expressed in (2.4).

$$max \sum_{a \,\varepsilon\, A} \sum_{r \,\varepsilon\, R_a} \sum_{s_1, s_2 \,\varepsilon\, S} z^{a,r}_{s_1,s_2} \tag{2.4}$$

2.3.5 Minimizing Service Migrations On Subsequent Iterations - MIN M

The goal of this optimization is to minimize service migrations between subsequent iterations of the model. Since the model is executed iteratively, the execution matrix from the previous iteration U^{i-1} is added to the model, which is used to compare with the current execution matrix U in order to reduce the service migrations needed to achieve the next optimization objective. Therefore, the minimization of service migrations can be expressed as shown in (2.5). Services may have to be migrated from one comp. node to another in order to find the optimal solution. However, it might be preferable to find a solution where service reallocations are minimized so delay caused by reallocations is kept at a minimum.

$$min \sum_{s\,\varepsilon\,S} \sum_{n\,\varepsilon\,N_c} \mid U_{s,n} - U_{s,n}^{i-1} \mid \tag{2.5}$$

2.3.6 Minimizing the Number of Active Comp. Nodes - MIN N_c

The goal of this optimization is to minimize the number of active comp. nodes in the cloud infrastructure, which results in cost and energy savings. By using the comp. node utilization decision variable U_n, the minimization can be expressed as shown in (2.6).

$$min \sum_{n\,\varepsilon\,N_c} U_n \tag{2.6}$$

The binary decision variable U_n is subject to additional constraints, ensuring that it only takes on value 0 if there is no communicating service allocated on that comp. node [16]. This constraint is expressed in (2.7).

$$\forall n\,\varepsilon\,N_c : \sum_{s\,\varepsilon\,S} U_{s,n} \leq U_n \times \mid S \mid \tag{2.7}$$

2.3.7 Minimizing the Number of Active Gateways - MIN N_{gw}

The goal of this optimization is to minimize the number of active gateways in the network. This minimization results in an improved wireless resource efficiency as well as energy and cost savings. Moreover, since gateways could be placed in a sleep state it contributes to an interference reduction. By using the gateway utilization decision variable U_{gw}, the minimization can be expressed as shown in (2.8).

$$min \sum_{gw\,\varepsilon\,N_{gw}} U_{gw} \tag{2.8}$$

The binary decision variable U_{gw} is subject to additional constraints, ensuring that it only takes on value 0 if there is no end device sending requests for an IoT application through that gateway. This constraint is expressed in (2.9).

$$\forall gw\,\varepsilon\,N_{gw} : \sum_{ed\,\varepsilon\,N_{ed}} U_{ed,gw} \times R_{ed} \leq \sum_{a\,\varepsilon\,A} D_a \times U_{gw} \tag{2.9}$$

2.3.8 Minimizing Hop Count Between Comp. Nodes and End Devices - MIN H

This optimization objective is related to low latency in the communication between comp. nodes where communicating services that compose an IoT application are running and the end device that requested the IoT application. This optimization can be achieved by minimizing the hop count between comp. nodes and end devices. This minimization can be expressed as shown in (2.10) by using the Hop Count matrix H and the placement matrix P.

$$min \sum_{n,ed \,\varepsilon\, N_c,N_{ed}} H_{n,ed} \times (\sum_{a \,\varepsilon\, A} \sum_{r \,\varepsilon\, R_a} \sum_{s \,\varepsilon\, S} P^{a,r}_{s,n} \times \Phi_{a,r,ed}) \tag{2.10}$$

2.3.9 Minimizing Path Loss - MIN PL

The objective of this optimization is to minimize the path loss of the wireless communication links. The path loss matrix PL is calculated based on the path loss formula for IEEE 802.11ah networks, when an urban macro deployment and a central frequency (f_c) of 900 MHz are considered. This formulation can be expressed as in (2.11). The distance (in meters) is given by the Distance matrix D, which indicates the distance between end devices and gateways. This way, by using the path loss matrix PL and the end device execution matrix $U_{ed,gw}$ the minimization objective is given by (2.12).

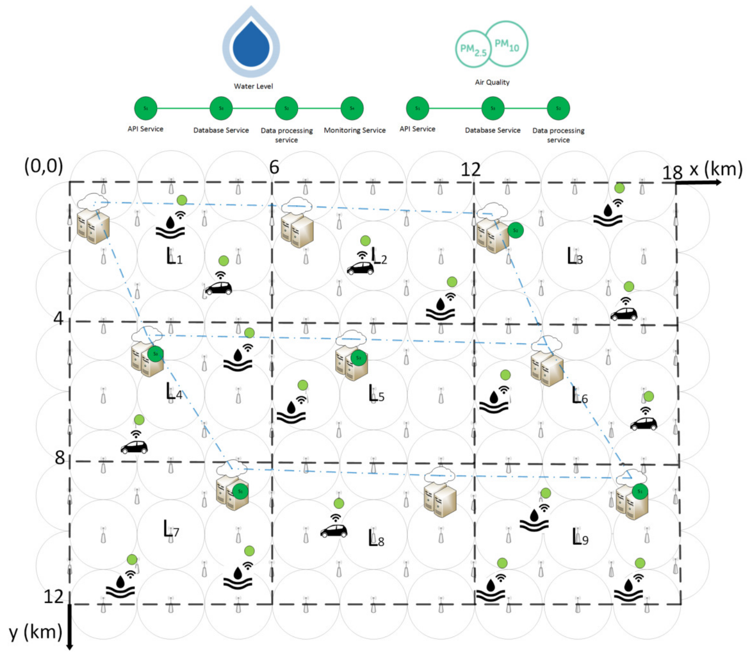
$$PL(dB) = 8 + 37.6 \log_{10}(d) \tag{2.11}$$

$$min \sum_{gw,ed \,\varepsilon\, N_{gw},N_{ed}} PL_{gw,ed} \times U_{ed,gw} \tag{2.12}$$

2.4 Evaluation Scenarios

The evaluation scenarios are based on use cases within the scope of Antwerp's City of Things testbed [22]. A rectangle area of 216 km^2 similar to the area of Antwerp has been considered. Gateways have been strategically placed covering the entire area while minimizing interference due to low coverage overlap between gateways. Two use cases have been evaluated, a static and a dynamic scenario as shown in Fig. 2.2, to demonstrate the wide applicability of the ILP model. It should be noted that nine areas of 24 km^2 are considered as possible locations for fog resources.

Figure 2.2 Evaluation Scenarios.



2.4.1 Static Scenario

As a future use case, water level sensors will be installed in sewers in the city of Antwerp. This use case is a static scenario related to a Water Level IoT application, which is decomposed in four communicating services:

- API service responsible for receiving sensor data.
- Database service for storing information.
- Data processing service for information analysis.
- Monitoring service that supervises all other services.

2.4.2 Dynamic Scenario

As an initial proof of concept of the Antwerp's City of Things architecture, air quality sensors have been installed on cars driving around the city of Antwerp [22]. These sensors send measures of typical gasses and climate data such as temperature and humidity, which are then annotated with GPS locations. This use case is a dynamic scenario related to an Air Quality IoT application, which is decomposed in three services: an API service, a database service and a data processing service.

2.4.3 Evaluation Setup

Table 2.4: Input variables for each Scenario.

Variables	Static	Dynamic
N_{ed}	1000	100
A	1	1
S	4	3
N_c	45	45
N_{gw}	123	123
N_f	9	9

The ILP model presented has been implemented in Java using the IBM ILOG CPLEX ILP solver [23]. Input variable values for each scenario are presented in Table 2.4. Nine fog clouds $f \varepsilon N_f$ each one managing 5 comp. nodes $n \varepsilon N_c$ have been considered. Moreover, 1000 water level sensors randomly distributed in the City of Antwerp were included in the model for the static scenario while for the dynamic scenario, 100 cars driving around the city were considered. In the dynamic scenario, each car is driving at an average velocity of 30km/h and each simulation occurs separated by 5 minutes. Therefore, car positions have been changed 2.5 km between simulations. A new x coordinate is randomly selected and then by solving the quadratic equation two solutions for the new y coordinate are obtained. Constraints have been included in the model to make sure that the new calculated car positions are inside the evaluation area. For both scenarios, the end device ed which makes the rth request of an IoT application is randomly selected from the set N_{ed}. Every comp. node has a CPU and a Memory capacity. The CPU capacity of a comp. node $n \varepsilon N_c$ represents the processing power per core times the number of cores, which is randomly chosen from the set $\{9, 12, 15, 18, 21\, GHz\}$ while the memory capacity is chosen from the set $\{6, 8, 10, 12, 14GB\}$. In the same way, the communicating services have a CPU and a Memory requirement which are chosen from the sets $\{0.5, 0.7, 0.9, 1.1, 1.3\ GHz\}$ and $\{0.5, 0.7, 0.9, 1.1, 1.3, 1.5\ GB\}$, respectively. Each comp. node, gateway, end device and fog cloud has a given location $l \varepsilon L$ associated. A location is one of the nine areas of 24 km^2 previously explained in the Section 2.4. If the location l of a comp. node n and a fog cloud f is the same, it means that the comp. node n is managed by that fog cloud f since there is only one fog cloud for each location $l \varepsilon L$. Moreover, (xy) coordinate positions are randomly attributed to each end device $ed \varepsilon N_{ed}$ while for each gateway $gw \varepsilon N_{gw}$, (xy) coordinate positions are strategically attributed in order to cover the entire evaluation area. Based on these (xy) coordinates, the distance matrix D is calculated by the euclidean distance formula as shown in (2.13).

$$D(gw, ed) = \sqrt{(x_{gw} - x_{ed})^2 + (y_{gw} - y_{ed})^2} \tag{2.13}$$

Then, the path loss matrix PL is calculated based on the path loss formula for the IEEE 802.11ah previously shown in (2.11) by using the calculated distance matrix values. The communication matrix C is a random number between $[0.02, 0.04]$ which represents the bandwidth requirement (in Mbit/s) between two communicating services, s_1 and s_2, respectively. Moreover, the bandwidth matrix B is a random number between $[6, 14]$ which represents the available bandwidth (in Mbit/s) between two comp. nodes, n_1 and n_2, respectively. The hop count matrix H between comp. nodes and end devices is a random number between $[2, 3]$ if the node n and the end device ed are on the same location l or between $[4, 8]$ if the node n and the end device ed are on different locations, l_1 and l_2, respectively. Different sets of model configurations for each scenario have been evaluated. In each iteration of the model, a different optimization objective has been considered. In Table 2.5 and Table 2.6, the model configurations are shown. It should be highlighted that for all model configurations first, the number of accepted IoT application requests is maximized and then the satisfaction of the service bandwidth is maximized, ensuring that the communication requirements between application services are guaranteed as well as possible. In Table 2.5, model configuration A corresponds to a wireless efficiency strategy, since the final objective is the minimization of the number of active gateways on the network. Secondly, model configuration B is related to energy efficiency in the cloud infrastructure, since the final goal of this configuration is the minimization of power consumption of the comp. nodes. Finally, model configuration C corresponds to a low latency strategy based on the minimization of the hop count value between comp. nodes and end devices. IoT application services are placed closer to the end device when this model configuration is executed.

On the other hand, in Table 2.6, four additional model configuration strategies are shown. Both D and E configurations are composed of six optimization objectives. D prioritizes low latency, while E prioritizes energy efficiency. Moreover, both DM and EM configurations are composed of seven optimization objectives since an additional optimization objective is introduced between the 3rd and the 4th iteration of the configurations D and E, related to the minimization of service migrations on subsequent model iterations in order to reduce delay from reallocating IoT services. All model configurations have been evaluated 50 times and confidence intervals of 95% have been considered in the evaluation.

Table 2.5: Model Configurations for the Static Scenario.

A - Wi-Fi Eff.	B - Cloud Energy-Aware	C - Latency
1 - MAX R	1 - MAX R	1 - MAX R
2 - MAX SB	2 - MAX SB	2 - MAX SB
3 - MIN N_{gw}	3 - MIN N_c	3 - MIN H

Table 2.6: Model Configurations for the Dynamic Scenario.

D - Latency	E - Energy Eff.	DM - Latency	EM - Energy Eff.
1 - MAX R	1 - MAX R	1 - MAX R	1 - MAX R
2 - MAX SB	2 - MAX SB	2 - MAX SB	2 - MAX SB
3 - MIN H	3 - MIN N_c	3 - MIN H	3 - MIN N_c
4 - MIN N_c	4 - MIN N_{gw}	4 - MIN M	4 - MIN M
5 - MIN N_{gw}	5 - MIN H	5 - MIN N_c	5 - MIN N_{gw}
6 - MIN PL	6 - MIN PL	6 - MIN N_{gw}	6 - MIN H
		7 - MIN PL	7 - MIN PL

2.5 Evaluation Results

2.5.1 Static Scenario

The model configurations shown in Table 2.5 have been evaluated for the static scenario presented in Section 2.4. In Fig. 2.3, the execution speed of the model configurations is shown. By increasing the number of requests, the execution time of the model configurations increases. For 200 requests, each model configuration requires on average at least 23 minutes to find the optimal solution. In Fig. 2.4, the ratio of active gateways and the ratio of active comp. nodes for each model configuration are illustrated. Regarding gateways, all are active for configurations B and C since no optimization objective is included regarding wireless efficiency. However, for configuration A whose final objective is related to wireless efficiency, the ratio of active gateways slightly increases with the increase of requests in the network. Results show that 50% of the gateways are active for 140 requests and that only for values above 220 requests, the ratio is higher than 60%. This configuration shows a higher wireless efficiency when compared to the other configurations. On the other hand, the ratio of active comp. nodes is 95% independent of the number of requests for configurations A and C due to the lack of an optimization objective regarding energy efficiency in the cloud environment in both configurations. However, for configuration B whose final objective is energy efficiency in the cloud domain, for 20 requests only 9% of the comp. nodes are active. Moreover, only for values above 180 requests, at least 80% of the comp. nodes are active. In Fig. 2.5, the average hop count between comp. nodes and end devices for each model configuration is shown. Configuration C obtained lower hop count values when compared to configurations A and B due to the fact that the final objective of C is low latency. C achieved slightly constant hop count values of 2.2 while A and B achieved hop count values between 4.0 and 5.5, which results in an increased latency since on average two more hops are required. Moreover, it should be noted that the average hop count decreases while requests increase due to the fact that more comp. nodes are needed in these conditions and therefore hop count decreases even if in the model configuration there is no optimization objective related to latency.

Figure 2.3 Execution speed of the model configurations.

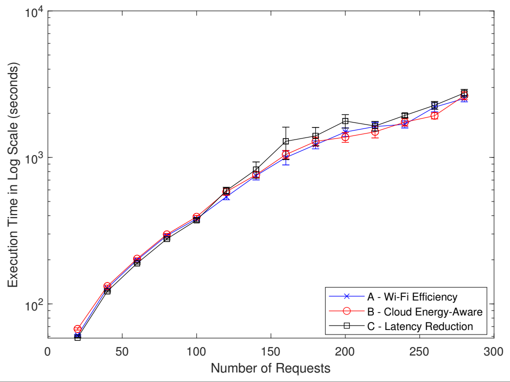
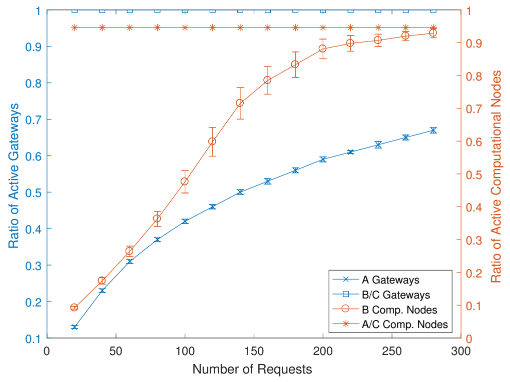


Figure 2.4 Gateway & Comp. Node Activity for each model configuration.



2.5.2 Dynamic Scenario

The four model configurations shown in Table 2.6 have been evaluated using the dynamic scenario presented in Section 2.4. In Fig. 2.6, the ratio of active comp. nodes for each model configuration is shown. Configurations E and EM achieved the same results. Therefore, minimizing migrations in subsequent iterations of the model, when energy efficiency is more important than low latency, does not alter the final solution. The ratio of active comp. nodes is independent of the cars repositioning and remains constant at 16%. However, configurations D and DM obtained different results. For configuration D, where service migrations are not taken into account average values of 51% are obtained while for configuration

Figure 2.5 Average Hop Count value for each model configuration.

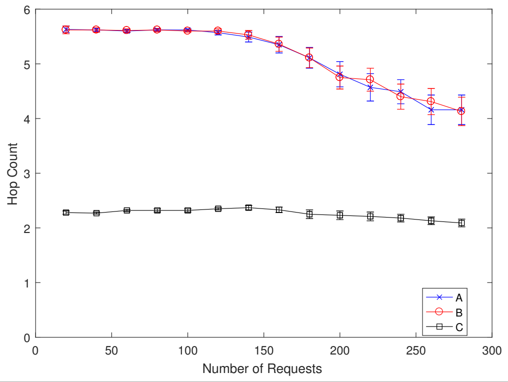


Figure 2.6 Comp. Node Activity for each model configuration.

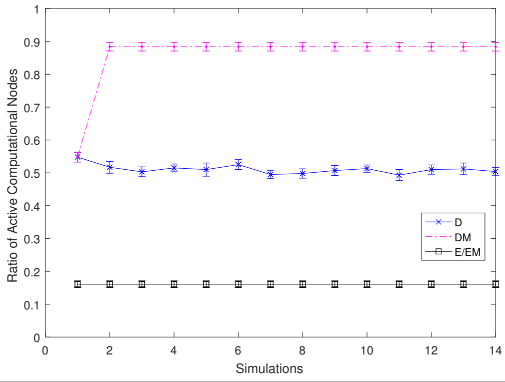


Figure 2.7 Average Hop Count value for each model configuration.

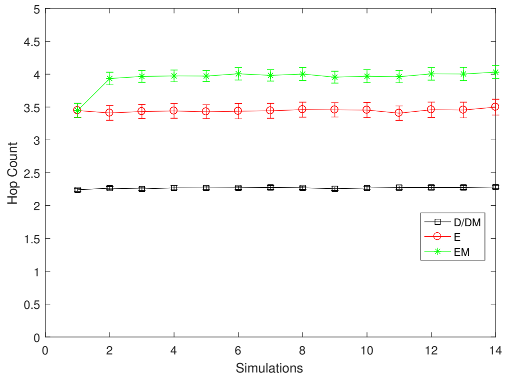
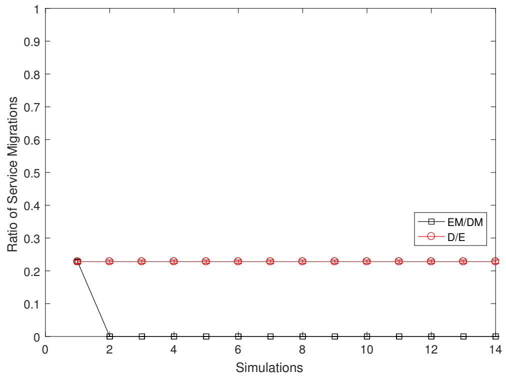


Figure 2.8 Service migrations between 3rd and 4th objective.



DM where service migrations are taken into consideration average values of 88% are achieved. This way, energy consumption cannot be further minimized when reallocations are considered, which will contribute to a higher number of active comp. nodes as it was observed.

In Fig. 2.7, the average hop count between comp. nodes and end devices for each model configuration is shown. Configurations D and DM obtained the same results because the minimization of hop count occurs earlier than the minimization of service migrations and therefore there is no difference in the achieved results for both configurations. Hop count values of 2.25 are then obtained. However, configurations E and EM achieved different results. Hop count values of 3.5 are obtained for model E while for model EM hop count values of 4.0 are achieved. This is due to the fact that when service reallocations are considered, latency cannot be further minimized, which will contribute to an even higher hop count as it was observed. In Fig. 2.8, the ratio of service migrations for each model configuration is illustrated. Values of 0% for configurations DM and EM are achieved while for D and E, average values of 22% are obtained meaning that in order to satisfy the optimal solution, 22% of the communicating services must be reallocated. This way, if both low latency and energy efficiency management strategies are considered, delay caused by service reallocation should be taken into account in the resource provisioning.

2.6 Conclusion

In this chapter, an ILP model for the resource provisioning of IoT application services in Smart Cities has been presented. In the last years, the need for resource management strategies for Smart Cities is increasing due to the deployment of IoT application use cases. Proper resource allocation is required in order to minimize costs and maximize QoS. The model considers not only cloud infrastructure requirements but also characteristics coming from wireless aspects in order to deal with these challenges. The model is executed iteratively since it optimizes multiple objectives, such as latency, service migrations and energy efficiency. Obtained results show that there is a clear trade-off between low latency and low energy consumption. For an IoT application service with real-time constraints low latency may be crucial and therefore a low hop count value between the allocated service and the end device must be achieved. However, another IoT application service without real-time constraints could be allocated far from the end device with the goal of minimizing energy consumption since low latency is not important. The result of this work can serve as a benchmark in research related to placement issues of IoT application services in Fog environments since the model approach is generic and applies to a wide range of IoT use cases. As future work, the ILP model will be validated through realistic evaluations based on real service deployments.

Acknowledgment

This research was performed partially within the projects "Service-oriented management of a virtualised future internet" and "Intelligent DEnse And Longe range IoT networks (IDEAL-IoT)" under Grant Agreement #S004017N, both from the fund for Scientific Research-Flanders (FWO-V). The computational resources (Stevin Supercomputer Infrastructure) and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by Ghent University, FWO and the Flemish Government – department EWI.

References

[1] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014.

[2] Vito Albino, Umberto Berardi, and Rosa Maria Dangelico. Smart cities: Definitions, dimensions, performance, and initiatives. *Journal of Urban Technology*, 22(1):3–21, 2015.

[3] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper, 2017. URL http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf.

[4] Marcelo Yannuzzi, R Milito, René Serral-Gracià, D Montero, and Mario Nemirovsky. Key ingredients in an iot recipe: Fog computing, cloud computing, and more fog computing. In *Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2014 IEEE 19th International Workshop on*, pages 325–329. IEEE, 2014.

[5] Luis M Vaquero and Luis Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014.

[6] Hlabishi I Kobo, Adnan M Abu-Mahfouz, and Gerhard P Hancke. A survey on software-defined wireless sensor networks: Challenges and design requirements. *IEEE Access*, 5:1872–1899, 2017.

[7] Shancang Li, Li Da Xu, and Shanshan Zhao. The internet of things: a survey. *Information Systems Frontiers*, 17(2):243–259, 2015.

[8] Sara Hachem, Animesh Pathak, and Valerie Issarny. Service-oriented middleware for the mobile internet of things: A scalable solution. In *IEEE GLOBECOM: Global Communications Conference*, 2014.

[9] Karima Velasquez, David Perez Abreu, Marilia Curado, and Edmundo Monteiro. Service placement for latency reduction in the internet of things. *Annals of Telecommunications*, pages 1–11, 2016.

[10] Mohammad Abdullah Al Faruque and Korosh Vatanparvar. Energy management-as-a-service over fog computing platform. *IEEE Internet of Things Journal*, 3(2):161–169, 2016.

[11] SmartSantander FP7-ICT-2009-5-257992 Project, 2017. URL http://www.smartsantander.eu/.

[12] Luis Sánchez, Verónica Gutiérrez, José Antonio Galache, Pablo Sotres, Juan Ramón Santana, Javier Casanueva, and Luis Muñoz. Smartsantander: Experimentation and service provision in the smart city. In *Wireless Personal Multimedia Communications (WPMC), 2013 16th International Symposium on*, pages 1–6. IEEE, 2013.

[13] David Perez Abreu, Karima Velasquez, Marilia Curado, and Edmundo Monteiro. A resilient internet of things architecture for smart cities. *Annals of Telecommunications*, pages 1–12, 2016.

[14] Mohammad Aazam, Imran Khan, Aymen Abdullah Alsaffar, and Eui-Nam Huh. Cloud of things: Integrating internet of things and cloud computing and the issues involved. In *Applied Sciences and Technology (IBCAST), 2014 11th International Bhurban Conference on*, pages 414–419. IEEE, 2014.

[15] Shalli Rani, Rajneesh Talwar, Jyoteesh Malhotra, Syed Hassan Ahmed, Mahasweta Sarkar, and Houbing Song. A novel scheme for an energy efficient internet of things based on wireless sensor networks. *Sensors*, 15(11):28603–28626, 2015.

[16] Hendrik Moens, Brecht Hanssens, Bart Dhoedt, and Filip De Turck. Hierarchical network-aware placement of service oriented applications in clouds. In *Network Operations and Management Symposium (NOMS)*, pages 1–8. IEEE, 2014.

[17] Fetahi Wuhib, Rerngvit Yanggratoke, and Rolf Stadler. Allocating compute and network resources under management objectives in large-scale clouds. *Journal of Network and Systems Management*, 23(1):111–136, 2015.

[18] Alan Davy Mohit Taneja. Resource aware placement of iot application modules in fog-cloud computing paradigm. In *IFIP/IEEE International Symposium on Integrated Network Management*, 2017.

[19] Weiping Sun, Munhwan Choi, and Sunghyun Choi. Ieee 802.11 ah: A long range 802.11 wlan at sub 1 ghz. *Journal of ICT Standardization*, 1(1):83–108, 2013.

[20] Stefan Aust and Tetsuya Ito. Sub 1ghz wireless lan propagation path loss models for urban smart grid applications. In *Computing, Networking and Communications (ICNC), 2012 International Conference on*, pages 116–120. IEEE, 2012.

[21] Evgeny Khorov, Andrey Lyakhov, Alexander Krotov, and Andrey Guschin. A survey on ieee 802.11 ah: An enabling networking technology for smart cities. *Computer Communications*, 58:53–69, 2015.

[22] Steven Latre, Philip Leroux, Tanguy Coenen, Bart Braem, Pieter Ballon, and Piet Demeester. City of things: An integrated and multi-technology testbed for iot smart city experiments. In *Smart Cities Conference (ISC2), 2016 IEEE International*, pages 1–8. IEEE, 2016.

[23] IBM CPLEX ILOG Optimization Studio 12.7, 2017. URL http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud.

3
Fog Computing: Enabling the Management and Orchestration of Smart City Applications in 5G Networks

"Talent wins games, but teamwork and intelligence wins championships."

– Michael Jordan, former basketball player

This chapter addresses challenge #2 by investigating novel architectural paradigms for Smart City applications. The chapter proposes a Fog Computing framework with autonomous management and orchestration functionalities for 5G Smart Cities. The chapter also presents a Peer-to-Peer (P2P) fog protocol based on the Open Shortest Path First (OSPF) routing protocol to exchange application service provisioning information. The goal is to enable fast decisions concerning service placement between fog nodes. In addition, the chapter discusses challenge #3 by evaluating an anomaly detection use case based on an air quality monitoring application. Results show that the proposed framework significantly reduces network bandwidth usage and latency compared to centralized clouds. Therefore, the main contribution of this chapter is the design of a fully integrated fog node management system.

⋆⋆⋆

José Santos, Tim Wauters, Bruno Volckaert and Filip De Turck.

Published in MDPI Entropy, Volume 20, Number 1, pages 4, 2018.

Abstract Fog computing extends the cloud computing paradigm by placing resources close to the edges of the network to deal with the upcoming growth of connected devices. Smart city applications, such as health monitoring and predictive maintenance, will introduce a new set of stringent requirements, such as low latency, since resources can be requested on-demand simultaneously by multiple devices at different locations. It is then necessary to adapt existing network technologies to future needs and design new architectural concepts to help meet these strict requirements. This chapter proposes a fog computing framework enabling autonomous management and orchestration functionalities in 5G-enabled smart cities. The approach follows the guidelines of the European Telecommunications Standards Institute (ETSI) NFV MANO architecture extending it with additional software components. The contribution of the chapter is its fully-integrated fog node management system alongside the foreseen application layer Peer-to-Peer (P2P) fog protocol based on the Open Shortest Path First (OSPF) routing protocol for the exchange of application service provisioning information between fog nodes. Evaluations of an anomaly detection use case based on an air monitoring application are presented. Results show that the proposed framework achieves a substantial reduction in network bandwidth usage and in latency when compared to centralized cloud solutions.

3.1 Introduction

In recent years, the Internet of Things (IoT) has transformed objects of everyday life into communicating devices. The number of connected devices will be between 10 and 12 billion by 2021 [1], making it impossible for current network technologies to support this enormous growth. Future networked systems must adapt existing network architectures to future needs and design and develop new management capabilities to help meet the stringent requirements of future use cases. In fact, the upcoming 5G networks aim to tackle these new business opportunities by introducing very high carrier frequencies, an enormous number of antennas and new functionalities, such as Device-to-Device communication (D2D) and fog computing [2]. The fog computing paradigm, which places resources on the edges of the network, extends the cloud computing paradigm to deal with the eminent growth of connected devices [3]. Fog computing brings computing power, storage and memory capacity closer to wireless gateways, sensors and actuators since these devices are currently lacking in terms of such capacities [4]. The so named Fog Nodes (FNs) are cloud entities with a small amount of computational resources distributed across the network that must be able to communicate with a different variety of devices and offer them solutions to gather, process and filter data [3, 5]. These procedures imply a proper resource allocation of multiple services, which is not an easy task. Nowadays, resource provisioning and orchestration is present in different topics of literature related to the life-cycle management of applications and services. Although orchestration is relatively mature in data centers, in fog architectures, there is still a large number of research challenges associated with this approach since fog computing is still in the early stages and needs more time to evolve [6]. In fog architectures, orchestration should be distributed across the multiple FNs, which are then responsible for the local resource provisioning and deployment of applications and services, thereby ensuring the necessary Quality of Service (QoS).

In the last few years, IoT applications have been implemented as a set of small and independent micro-services. The micro-services architecture is a relatively new term in software patterns [7]. The micro-service paradigm is an extension of the traditional Service-Oriented Architecture (SOA) paradigm, where an application is decomposed into a set of fine-grained services. Each service communicates through lightweight communication protocols. Research studies have been carried out to solve the issues of abstracting end device functionalities, trying to provide a suitable architecture with service management and composition capabilities able to link a set of micro-services in a set of IoT applications. Each micro-service can be provided by a lightweight container, which may be used by multiple tenants. In a smart city scenario, resources should be distributed within the network ensuring that the micro-services that make up an application are allocated and instantiated close to the end device that is requesting the IoT application [8]. Multiple factors should be taken into account to ensure proper resource allocation such as latency, bandwidth, energy efficiency and cost.

In this chapter, a fog-based management and orchestration framework is proposed to deal with the application service placement problem in smart cities. The approach follows the guidelines of the European Telecommunications Standards Institute (ETSI) Network Function Virtualization (NFV) Management and Orchestration (MANO) architecture, extending it with additional software components, which will offer not only high computing performance, but also monitoring and data analysis functionalities. Furthermore, based on our expertise in the network management domain, a novel fog protocol is foreseen to enable the exchange of application service information between FNs to provide fast service provisioning decisions for smart city applications. This way, each FN can decide where and when it is more suitable to deploy and instantiate each instance of the micro-services composing a smart city application. Therefore, an application layer Peer-to-Peer (P2P) fog protocol based on the Open Shortest Path First (OSPF) routing protocol is proposed to deal with the exchange of application service information between

FNs and the cloud layer. Finally, the evaluation of an anomaly detection smart city use case based on an air monitoring application is presented. Results show that the proposed fog computing framework achieves a substantial reduction in network bandwidth usage and in latency when compared to centralized cloud solutions.
The remainder of the chapter is organized as follows. In the next section, related work is discussed. Section 3.3 introduces the proposed fog computing management and orchestration framework. Then, in Section 3.4, the evaluation scenario is described, which is followed by the evaluation of the results in Section 3.5. Finally, conclusions are presented in Section 3.6.

3.2 Related Work

With the advent of 5G networks and by exploiting the advantages of new paradigms, such as NFV, Software-Defined Networking (SDN) and Machine-to-Machine (M2M) communication, autonomous network management functionalities become feasible. This section provides a summary of the standardization activities, research projects and open source initiatives relevant for the specification and implementation of a MANO framework for 5G-enabled smart cities.

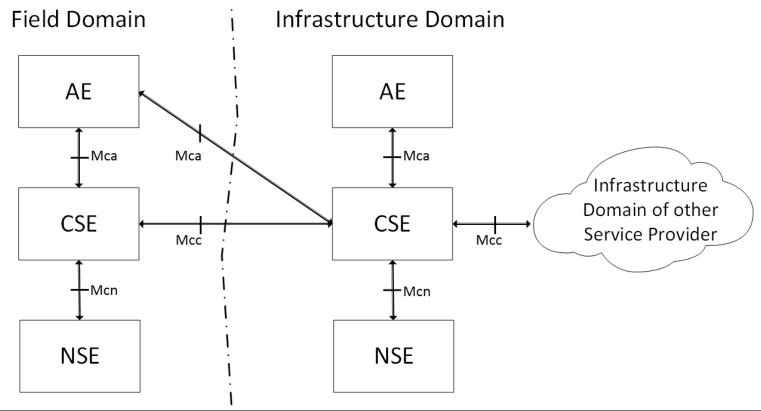
3.2.1 Standardization Activities

This subsection provides a brief summary of the standardization activities relevant in the management and orchestration domain.

3.2.1.1 ETSI oneM2M

In January 2009, the ETSI M2M technical committee was established with the aim to develop an end-to-end high level architecture for M2M. ETSI oneM2M aims to establish for M2M what 3GPP realized for mobile networks. The final release has been available since 2016 [9], and the correspondent architecture is shown in Figure 3.1. The architecture consists of two distinct domains: the field domain and the infrastructure domain. The Application Entity (AE) is responsible for implementing an M2M application service logic in the application layer. Each execution instance is identified with a unique AE Identifier (AE-ID). Examples of the AEs include an instance of a remote blood sugar monitoring application, a power metering application or an emergency response application. The Common Services Entity (CSE) represents an instantiation of a set of common service functions of the M2M environments. These service functions are exposed to other entities through the AE – CSE (Mca) and CSE - CSE (Mcc) reference points. Each CSE is identified with a unique CSE-ID. Examples of service functions offered by CSEs include: data management, device management and M2M service subscription management. Finally, the Network Service Entity (NSE) provides services from the underlying network to the CSEs. Examples of such services include location services and device triggering [9]. The reference point CSE – NSE (Mcn) is used for accessing underlying NSEs. The proposed fog computing framework follows the guidelines of ETSI OneM2M regarding device management and security standards by including the correspondent components in the framework. Nevertheless, the definition of MANO functionalities and components in the ETSI OneM2M reference architecture are still in an early stage.

Figure 3.1 oneM2M functional architecture [9].



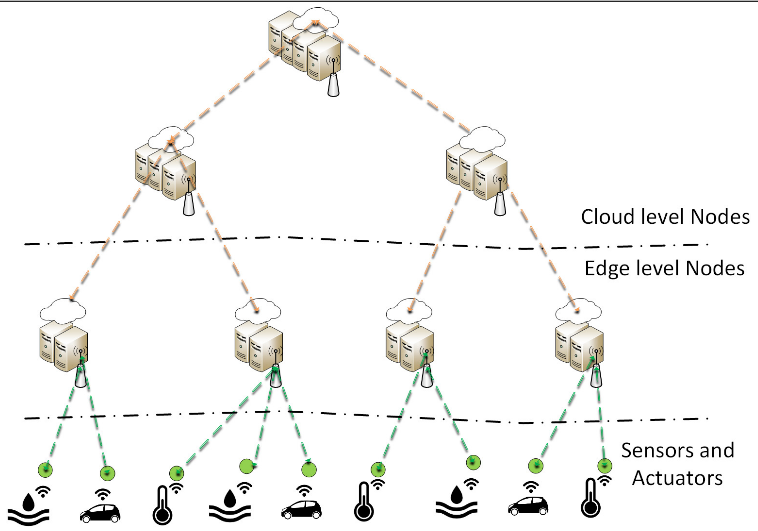
3.2.1.2 OpenFog Consortium

The OpenFog Consortium is working on a framework for efficient and reliable networks between clouds, endpoints and services based on open standard technologies. Recently, OpenFog released a reference architecture [10] that clarifies the characteristics and the requirements for fog computing, focusing primarily on the FN. The OpenFog reference architecture for a multi-tier deployment is shown in Figure 3.2. However, the definition of MANO components in the OpenFog reference architecture are still superficial, and therefore, it is expected that the consortium will work in this direction in future specifications.

3.2.1.3 ETSI MEC

The ETSI Mobile Edge Computing (MEC) technical committee is currently working on a reference architecture [11] for an orchestrator with similar requirements to the orchestrator required by fog computing architectures. MEC is emphasizing the need to consider a set of stringent constraints, such as application instantiations and service reallocations, both very important requirements for fog solutions. The MEC framework is shown in Figure 3.3. MEC is focused on evolving the mobile

Figure 3.2 The OpenFog reference architecture: N-tier fog deployment.



network edges, in order to create a cloud environment close to the Radio Access Network (RAN) that hosts enhanced services provided by the Mobile Network Operator (MNO) or third parties. Mobile Edge (ME) applications run on top of a generic cloud infrastructure located within the RAN, referred to as the mobile edge host. The mobile edge host is an entity containing a mobile edge platform and a virtualization infrastructure, which provides compute, storage and network resources for the purpose of deploying mobile edge applications. The mobile edge platform is the collection of functionalities required to run mobile edge applications on a virtualization infrastructure enabling the provisioning and consumption of mobile edge services [11]. Nevertheless, a detailed and complete specification of the MEC Orchestrator reference architecture is still missing.

3.2.1.4 ETSI NFV MANO

The ETSI NFV MANO technical committee is working on the standardization of management and orchestration frameworks required for the provisioning of Virtual Network Functions (VNFs) and Network Services (NSs). The MANO committee focuses on the definition of a reference architecture to overcome the challenges of the virtualization paradigm by covering operational and management aspects, such as service life-cycle workflows, information elements and interfaces. The final release has been available since 2014 [12], and the correspondent architecture is shown in Figure 3.4. This reference architecture allows network operators to apply a smooth transition to the new management paradigm, where legacy services and functionalities are gradually augmented with SDN and NFV capabilities. The

Figure 3.3 The mobile edge system reference architecture [11].

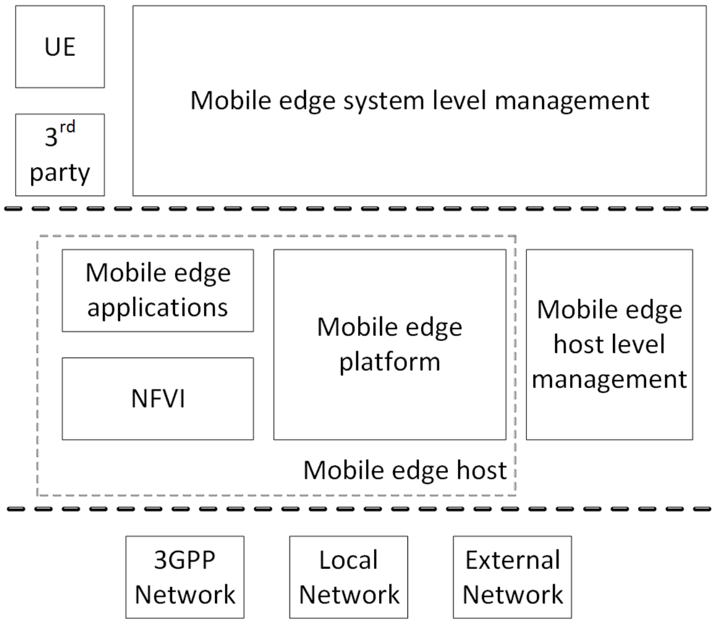
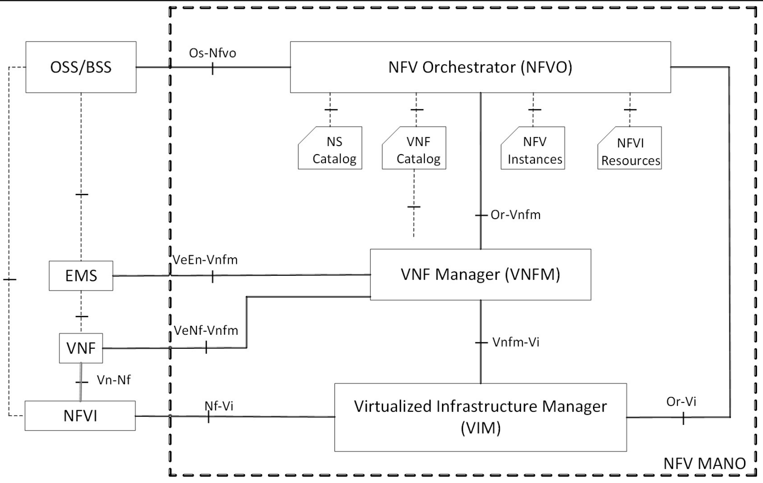


Figure 3.4 The NFV MANO architectural framework with reference points [12].



ETSI NFV MANO architecture is composed of four main functional blocks:

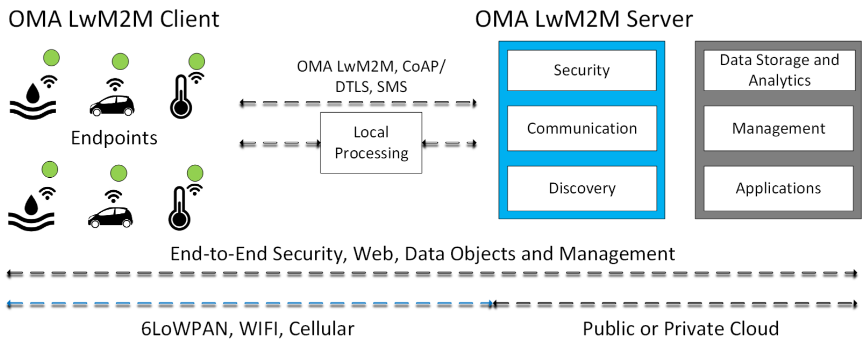
- Operations Support System/Business Support System (OSS/BSS):
  - Responsible for the control of software applications used to support back-office activities;
  - Coordination of business operations, such as billing and customer services.
- NFV Orchestrator (NFVO):
  - Responsible for the registration of new VNFs and NSs;
  - Life-cycle management (including instantiation, scale-out/in, termination).
- VNF Manager (VNFM):
  - Life-cycle management of VNF instances;
  - Coordination of the main configurations between the NFV Infrastructure (NFVI) and the Elemental Management System (EMS).
- Virtualized Infrastructure Manager (VIM):
  - Controlling and managing the NFVI compute, storage and network resources;
  - Collection and forwarding of performance measurements and events;

The proposed fog computing framework in this chapter follows the ETSI NFV MANO reference architecture, extending it with additional software components. The architectural elements share concepts with the MANO architecture; however, the characteristics of fog computing solutions lead us to a different approach, since ETSI NFV MANO has been conceived of to be applied in data centers, while in fog environments, each fog node must be able to manage and control its infrastructure. Management and orchestration are still open issues in the fog computing domain due to the dynamic behavior and distributed management of the network, which makes the investigation of using ETSI NFV MANO standards as the base for a fog-based MANO framework implementation necessary.

3.2.1.5 OMA Lightweight M2M

The Open Mobile Alliance (OMA) designed a device management protocol for sensor networks and M2M environments, which is named Lightweight M2M (LwM2M). LwM2M has been specified by a group of industry experts at the OMAs Device Management Working Group and is based on protocol and security standards from the Internet Engineering Task Force (IETF). OMA LwM2M aims to respond to the demand in the market for a common standard for managing lightweight and low power devices necessary to realize the potential of IoT. The LwM2M protocol builds on an efficient secure data transfer standard named the Constrained Application Protocol (CoAP). The final release has been available since 2014 [13], and the correspondent architecture is shown in Figure 3.5. As mentioned, the proposed fog computing framework follows ETSI's oneM2M device management and security standards, which are the main focus of the LwM2M protocol.

Figure 3.5 The system architecture of OMA LwM2M.



3.2.1.6 TOSCA

In recent years, the Topology and Orchestration Specification for Cloud Applications (TOSCA) [14] has become the standard language for modeling service orchestration in cloud environments in a highly extensible and flexible manner. TOSCA focuses mainly on enhancing the portability and operational management of cloud applications and services across their entire life-cycle, by defining building blocks, requirements and capabilities, which should be taken into consideration in the service orchestration.

3.2.1.7 Cloudlet

The concept of cloudlet was introduced in [15]. A cloudlet is a small-scale data center that is located at the edges of the network. The main purpose of the cloudlet is to provide cloud resources close to mobile devices as fog computing is currently doing for IoT devices. Cloudlet aims to support resource-intensive mobile applications with latency-sensitive requirements that will emerge in the future 5G network.

3.2.1.8 Summary

ETSI MEC has been working on a reference architecture for the evolution of the mobile network edges, in order to create a cloud environment close to the RAN, while ETSI MANO has been developing a MANO framework for the provisioning of VNFs and NSs in data centers. Furthermore, ETSI oneM2M has been designing an end-to-end high level architecture for M2M communications, while OMA LwM2M is working on the same path, but focused on device management and security functionalities. Additionally, the cloudlet paradigm is extending the cloud paradigm by placing resources close to mobile devices, while the OpenFog Consortium by using the fog computing paradigm is placing cloud resources at the edges of the network to meet the strict requirements introduced by IoT use cases. Finally, TOSCA is complementing all these standardization efforts by designing a standard language for modeling service orchestration in cloud environments in a highly extensible and flexible manner.
However, all the cited standardization activities still face industry challenges for the full definition of management and orchestration technologies. Most of the discussed reference architectures are still immature and require further investigation and development before they can be operationalized and used by network operators. Research efforts relevant for the specification and implementation of a MANO framework for 5G-enabled smart cities are detailed next.

3.2.2 Research Projects

In recent years, research projects have been carried out to deal with management and orchestration issues in smart cities. The SmartSantander project [16, 17] worked on a suitable architectural model for IoT and on the inherent challenges of resource provisioning in smart cities. The SmartSantander framework provides a suitable platform for large-scale experimentation and evaluation of a large set of IoT use cases deployed in several urban scenarios. Furthermore, the CityPulse project [18] has been working on a data analytics framework for smart cities. The CityPulse framework integrates powerful data analytics tools, data aggregation and event detection modules and quality assessment algorithms, which aim to support the development of customized smart city applications. The SusCity project [19] is working on a resilient IoT architecture for smart cities focusing on data collection from multiple sources, in order to develop intelligent management solutions that can help the government and citizens to make appropriate decisions. Furthermore, the VITAL project [20] federates heterogeneous IoT platforms via semantics in a cloud-based environment focusing on smart city scenarios.
Additionally, in [21], a big data network composed of SDN technologies and cloud/fog platforms is presented. Their goal is to reduce the large amount of redundant data and the response time in accessing data services. However, their

focus is only on the orchestration of big data services, while the approach is not only concerned with data analytics operations, but also decision making functionalities that can help to autonomously orchestrate smart city applications in a distributed way. Furthermore, their work is based on simulation studies, while the approach is based on an actual deployment within the scope of Antwerp's City of Things testbed. Furthermore, in a recent article about 5G-enabled smart cities [22], an approach for M2M communication in cognitive 5G networks is presented. This paper introduces a novel decentralized multiple gateway assignment protocol based on multi-channel Carrier Sense Multiple Access (CSMA) for M2M communication in 5G networks. Moreover, a low overhead protocol is also proposed, which increases the throughput of the system and minimizes energy consumption by reducing the message header payload. Simulation studies were carried out to show the effectiveness of the proposed schemes in terms of network lifetime and energy consumption.

Regarding European research, the European Commission (EC), alongside industry manufacturers, telecommunications operators, service providers, SMEs and researchers, created the 5G Infrastructure Public Private Partnership (5G-PPP) to advance the research of 5G technologies in Europe and to build global consensus on 5G networks. The projects supported by the 5G-PPP aim to deliver solutions, architectures, technologies and standards for the ubiquitous next generation communication infrastructures of the coming decade. Regarding management and orchestration, the following 5G-PPP projects should be highlighted. First, the Service Programming and Orchestration for Virtualized Software Networks (SONATA) project [23] aims to deliver an agile service development and orchestration in 5G virtualized networks. SONATA targets both the flexible programmability of software networks and the optimization of their deployments. Secondly, the Coordinated control and spectrum management for 5G heterogeneous radio access networks (COHERENT) project [24] will develop and validate a novel control framework for future mobile networks. The key innovation of COHERENT is the unified programmable control framework to coordinate the underlying heterogeneous mobile networks as a whole. Thirdly, the Self-organized Network Management in Virtualized and Software Defined Networks (SELFNET) project [25] will design and implement an autonomic network management framework to achieve self-organizing capabilities, such as self-protection, self-healing and self-optimization functionalities to deal with major network management problems, which are currently still being manually addressed by network operators, thereby significantly reducing operational costs and improving user experience. Fourthly, the End-to-End Cognitive Network Slicing and Slice Management Framework in Virtualised Multi-Domain, Multi-Tenant 5G Networks (SLICENET) project [26] aims to maximize the potential of the future 5G infrastructures and their services based on advanced software networking and cognitive network management in

SDN/NFV-enabled 5G networks. One scenario considered by SLICENET is a smart city use case, the goal of which is to implement a remote water metering and an intelligent public lighting system in the city of Alba Iulia, in Romania. Finally, the Small cEllS coordinAtion for Multi-tenancy and Edge services (SESAME) [27] project' intention is to develop a new multi-operator-enabled small cell that integrates a virtualized execution platform for deploying VNFs and supporting self-management capabilities.
Although the existing and ongoing research projects cited address some of the requirements of MANO functionalities for 5G networks, they have not yet delivered an integrated and autonomous MANO solution. Therefore, in this chapter, a fog-based MANO framework is proposed that goes beyond the current state-of-the-art by introducing a fully-integrated and autonomous FN management system, which combines monitoring and data analysis operations alongside management and orchestration decisions for the resource provisioning issue in smart cities.

3.2.3 Open Source Initiatives

This subsection provides a brief summary of the most popular open source activities related to the development of NFV, SDN and M2M technologies.

3.2.3.1 NFV Open Source Projects

Nowadays, there are several open source projects related to the control and management of VNFs that follow the reference standards provided by ETSI. In Table 3.1, the most popular NFV open source projects today are shown.

3.2.3.2 SDN Open Source Projects

With the growing development of NFV and SDN technologies, the number of open source projects related to SDN controllers is rapidly increasing. In Table 3.2, the working groups related to SDN controllers are shown.

3.2.3.3 M2M Open Source Projects

Currently, with the growing amount of connected devices, the number of open source initiatives related with M2M management issues is rapidly increasing. In Table 3.3, the most popular M2M standardization efforts are shown.

3.2.4 Ambition

In this chapter, a fog computing framework is proposed to enable autonomous MANO functionalities for smart city applications in 5G Networks. This work

Table 3.1: NFV open source projects.

Project	Description
OpenStack [28]	OpenStack allows the management of virtualized resources such as virtual disks, computational nodes and networks. OpenStack addresses the management paradigm of the ETSI MANO NFV architecture.
OpenBaton [29]	OpenBaton is an ETSI NFV compliant NFVO. OpenBaton has a modular implementation allowing the integration of other software modules without having to modify or understand its core implementation. OpenBaton supports a plugin system to incorporate virtual infrastructure managers, such as Openstack.
OpenMano [30]	OpenMANO is an approach to the management and orchestration of ETSI NFV standardization. However, although OpenMano allows the definition of network services and their instantiation and deletion, it does not provide a virtual network function manager.

Table 3.2: SDN open source projects.

Project	Description
OpenDayLight [31]	OpenDayLight is an open source project hosted by The Linux Foundation working on a platform to accelerate the adoption of SDN technologies. OpenDayLight aims to solve a wide range of common network problems in an automatic manner by making networks more programmable and intelligent.
ONOS [32]	ONOS is an open source SDN networking operating system that aims to provide a scalable and modular platform that eases the development of SDN applications and services.
Ryu [33]	Ryu is a component-based SDN framework. Ryu provides software components with well defined APIs to create new network management and control applications.

Table 3.3: M2M open source projects.

Project	Description
OMA LwM2M [34]	OMA LwM2M is a device management protocol designed for IoT networks and the demands of M2M environments.
Leshan [35]	Leshan is a LwM2M server and client Java implementation, which provides software libraries to help the development of other LwM2M applications.
5G-EmPOWER [36]	5G-EmPOWER is a multi-access edge computing operating system supporting heterogeneous radio access technologies. 5G-EmPOWER aims to provide full visibility of the network state and allow the dynamic deployment and orchestration of NSs.

extends the current state-of-the-art within fog computing, NFV and MANO paradigms by introducing a fully-integrated and autonomous fog node management system, which combines monitoring and data analysis operations alongside management and orchestration decisions. The approach follows the guidelines of the ETSI NFV MANO architecture and the standards of ETSI oneM2M in terms of device management and security. Furthermore, a novel fog protocol is introduced, which enables the exchange of application service information between FNs and the cloud layer, improving the performance of application-to-resource provisioning results.

In summary, this work looks beyond existing and ongoing research projects that individually address some of the requirements, but have not yet delivered an integrated and autonomous solution for the resource management and orchestration issue in smart cities. Smart city applications will introduce a set of stringent requirements, such as low latency and high mobility, since services can be requested on-demand simultaneously by multiple devices on different locations. To deal with these limitations, efficient resource provisioning is needed in order to address these constraints introduced by smart city applications while minimizing resource costs. Therefore, in this chapter, a fog-based management and orchestration framework is proposed to deal with the application service placement problem in smart cities. The framework is presented in the next section.

3.3 A Fog-Based Management and Orchestration Framework for 5G Smart Cities

In this section, a fog computing framework is presented for the management and orchestration of smart city applications. First, architectural challenges introduced by the fog computing paradigm are presented. Then, a system overview of the proposed framework is detailed. Furthermore, a fog protocol is introduced to provide exchange of application service provisioning information between multiple FNs. Finally, a hierarchical and distributed data monitoring and analysis approach based on the framework is detailed.

3.3.1 Challenges of a Fog-Based MANO Architecture

To deal with diverse IoT requirements, such as latency, energy efficiency and mobility, the fog computing paradigm has been introduced. Centralized cloud solutions are not suitable for future IoT applications, with real-time constraints and enormous volumes of data to be transported in the network. This way, fog computing architectures can provide effective ways to overcome many limitations of the existing network architectures that rely only on computing resources in the cloud and on end-user devices.

3.3.1.1 Latency Constraints

Fog computing will allow real-time processing and data analytics at the edges of the network, which will enable the deployment of delay-sensitive applications and the control of time-sensitive network functionalities close to end devices. This will allow coping with the strict requirements of the future smart city applications. This way, fog computing will help to reduce the service provisioning delay for most applications.

3.3.1.2 Security Challenges

Existing security solutions are designed for protecting enterprise networks and data centers by providing perimeter-based protections. These security services are no longer adequate for addressing the new security challenges in the emerging IoT systems. Fog computing architectures will enable a wide range of security functions, such as distributed malware monitoring for the multiple IoT devices to compensate these devices' limited security and primarily take advantage of the gathering of local information in order to detect threats and attacks in a timely manner.

3.3.1.3 Network Bandwidth Constraints

Centralized solutions are not suitable for smart city scenarios since sending all the data collected by IoT devices to the cloud layer will require an enormous amount of network bandwidth, which centralized solutions and Low Power Wide Area

Network (LPWAN) technologies cannot support. Fog computing enables data processing and analytics operations locally, which drastically reduces the amount of data that needs to be sent to the cloud layer.

3.3.1.4 Resource-Constrained IoT Devices

IoT devices have limited resources (battery, computational power, storage and memory capacity). It is not feasible to rely only on these devices to fulfill all the needed computational operations. FNs will carry out most of the computational tasks on behalf of resource-constrained IoT devices, hence reducing these devices' complexity, deployment costs and energy consumption.

3.3.1.5 High Dynamicity

Mobility is an important requirement of IoT scenarios since resources can be requested on-demand simultaneously by multiple devices at different locations. Furthermore, IoT introduces serious challenges on how to address seamless mobility in heterogeneous environments, since devices are constantly moving and accessing the medium through different technologies. To deal with these inherent mobility challenges, the fog computing paradigm has been introduced to provide resources and services on the edges of the network to effectively handle handover procedures.

3.3.1.6 Distributed Data Analysis

As IoT devices can send their data samples to local FNs, monitoring and anomaly detection operations can be performed in a distributed way. If unusual events or abnormal behaviors are detected in the data, faster response times can be achieved. This way, malfunctions in IoT devices can be detected and transmissions of incorrect information can be avoided in a timely manner, which can improve the network reliability.

3.3.1.7 Local Autonomous Operations

By enabling decisions and autonomous operations locally, it is possible to reduce the amount of data that needs to be sent to the cloud layer, decreasing the latency in the communication and improving the response time in case network failures are detected.

3.3.2 System Architecture Overview

The approach follows the guidelines of the ETSI NFV MANO architecture for infrastructure management and orchestration, defined in the context of NFV technologies, extending it with additional software components, which will offer not only high computing performance and intelligence in 5G smart cities, but also monitoring and data analysis functionalities to provide secure and reliable communications. Moreover, the solution incorporates the standards of ETSI oneM2M regarding device management and security.

3.3.2.1 Global Overview of the Fog-Based MANO Framework

Figure 3.6 provides an overview of the proposed fog computing framework. IoT devices, mainly sensors and actuators, communicate with LPWAN gateways, which are linked with the fog layer through multiple FNs. Each FN is an autonomous system managing a given set of computational resources. FNs communicate with the cloud layer through a Cloud Node (CN), which is the top level management entity. The CN is responsible for the global management and control operations in the network. FNs will be able to communicate with other FNs and with the CN based on an application layer P2P fog protocol, which enables the exchange of application service provisioning information between fog nodes and the cloud layer, improving the decision making process related to the resource provisioning in 5G smart cities.

3.3.2.2 Fog Node System Architecture

Figure 3.7 presents the detailed architecture of the FN autonomous management system. Some FN architectural components share concepts with the ETSI NFV MANO architecture; however, the nature and behavior of the FNs in the infrastructure lead us to a different approach, since ETSI NFV MANO has been conceived to be applied in data centers, and the proposal is aligned with fog computing architectures. The FN must setup and manage its infrastructure and associated devices in an autonomous manner. Therefore, each FN is managing a set of computational resources by using a virtualization layer residing over a physical layer, offering virtualization of the main network functionalities. The VIM performs the life-cycle management of the multiple network functions deployed on the network.
The next upper software module, is the Fog Manager (FM) component, which is responsible for managing the attached IoT devices through a Fog Agent (FA). The FM addresses the device management and M2M security guidelines defined by ETSI oneM2M. The FM module is mainly responsible for:

- Device discovery operations;
- Updating the devices' configurations through the FA;
- Keeping track of the devices' mobility;
- The security in M2M communications;

Figure 3.6 Overview of the proposed fog-based MANO framework for 5G-enabled smart cities.

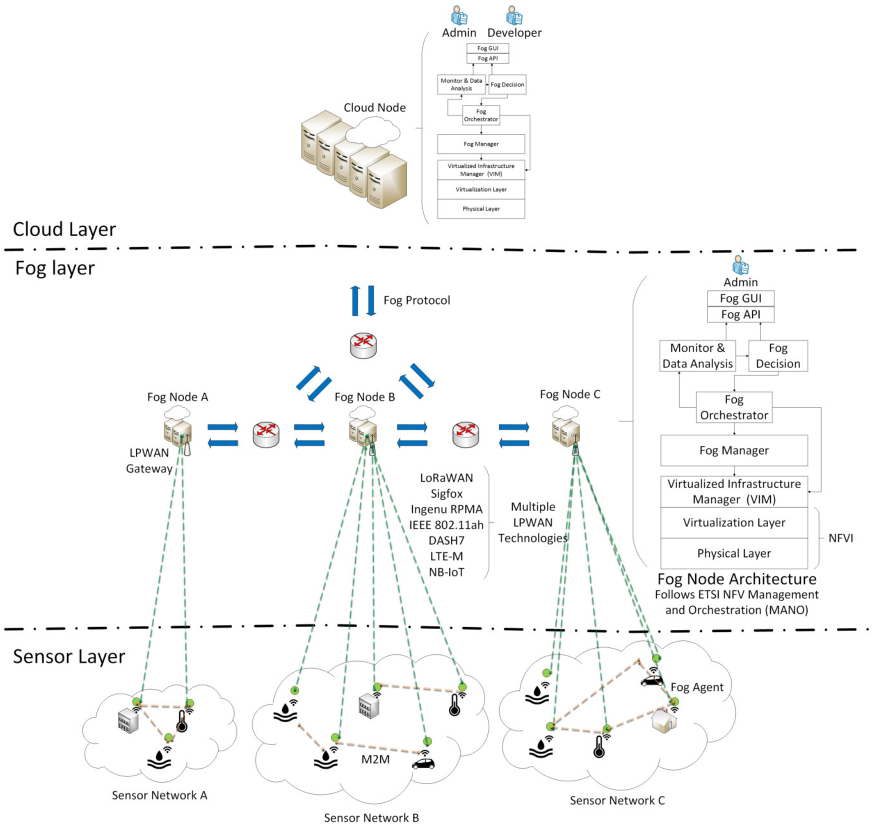
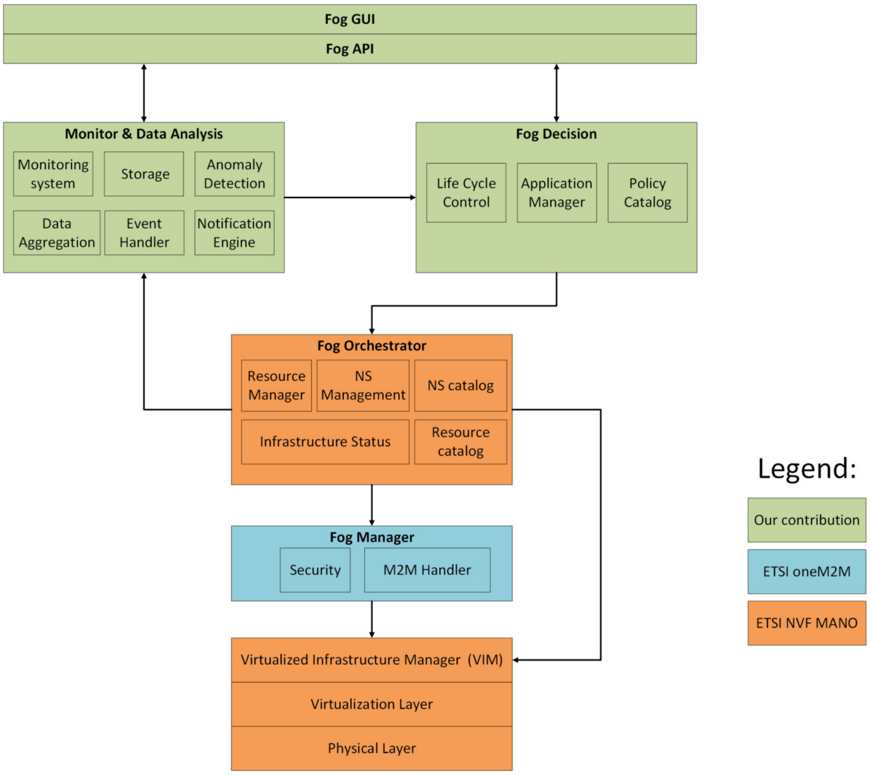


Figure 3.7 Detailed overview of the FN architecture.



Furthermore, each FN has its own instance of a Fog Orchestrator (FO) component. The FO module is mainly responsible for the following operations:

- Life-cycle management of micro-services (including instantiation, scale-out / in, termination);
- Interface with the monitoring and data analysis system;
- Interface with the Fog Decision (FD) module;
- Responsible for the registration of new VNFs and NSs;

On the one hand, a monitor and data analysis component is responsible for gathering information about the current state of the computational resources, such as CPU, memory and storage and collecting data samples from the different IoT sensors through the M2M handler. This module is composed of databases of measurements, events, warnings and notifications, which will help to identify network failures, policy violations and security threats on the network in a timely manner. In fact, this component keeps an updated status of the overall FN system alongside the connected IoT devices. On the other hand, the FN system architecture foresees a fog decision module, which is the component housing the intelligence, mainly responsible for:

- Life-cycle control and management decisions;
- Self-configuration functionalities;
- Applying the network behavior desired by network administrators;
- Providing autonomous responses to unknown situations detected by the monitoring and data analysis module;

The FD module will use different sets of machine learning algorithms to provide self-management responses to unusual events or malfunctions that can be detected in the network. The FD module will apply network strategies indicated in the policy catalog. Moreover, application and service migration requests will be handled by the FD module. The main outcome of the FD module is a set of actions that need to be deployed in the network, which will be done by the FO.
Each FN is composed by a northbound Application Programming Interface (API) defining a standardized entry point to the FN autonomous system. This interface defines the interaction between network administrators and FNs. This API will provide access to all configuration, management and reporting capabilities provided by FNs.
Finally, each FN also provides a Fog Graphical User Interface (GUI), where network administrators can interact and configure each FN. Furthermore, network administrators can obtain a broader view of the behavior of the network, in terms of resource usage, application service deployments, among others, and present this

status in a command and control center. Both the fog GUI and fog API will provide the necessary tools for network administrators to stop or manually enforce any kind of action on an FN despite the fact that each FN is executing a significant part of the processes in an autonomous way.
In summary, FNs are fully-integrated and autonomous management systems with data analysis and decision making functionalities. Essentially, an FN manages a small set of computational resources such as in small cloud environments. This way, an FN can be considered a small cloud entity. The goal is to distribute these entities across the network in order to provide autonomous distributed operations and decisions in future 5G smart cities.

3.3.2.3 Cloud Node System Architecture

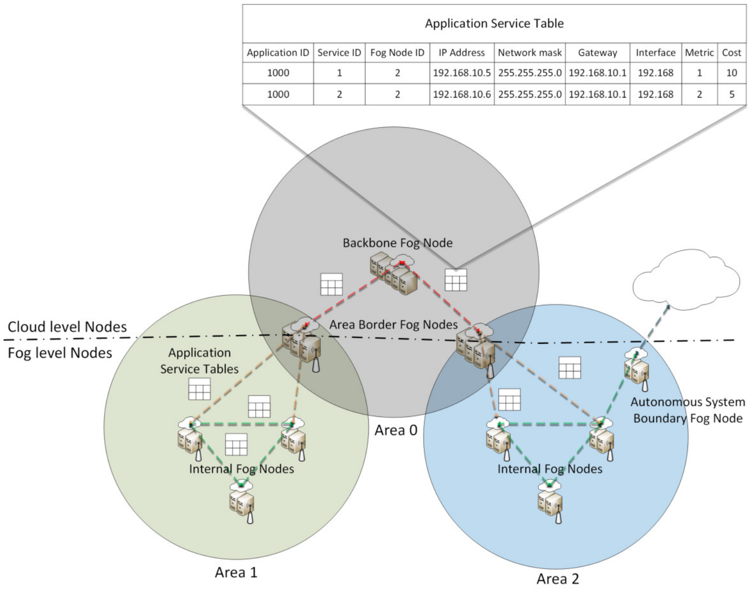
The architecture of the CN is essentially the same of the FN. However, CN functionalities are different since the CN is responsible for the global management and control operations in the network. The CN is mainly responsible for the following operations:

- Software updates of VNFs or NSs on the FNs;
- Coordination and control of FNs;
- Global data analysis operations;
- Overall QoS and Service Level Agreements (SLAs) monitoring;
- Applying the global network behavior indicated by network administrators;

3.3.3 Fog Protocol: Enabling Exchange of Application Service Information between Fog Nodes and the Cloud Layer

In the presented framework, each FN will communicate with the other FNs and with the CN in the cloud layer in order to exchange resource provisioning information. To the best of our knowledge, no suitable way to provide the exchange of this kind of information between the fog and the cloud layer is available in literature. This is still a key research challenge in the fog computing domain. By taking into account the main advantages of fog computing architectures, which are minimized latency and reduced bandwidth use, a suitable fog protocol should be lightweight, transport agnostic and customizable, since QoS parameters must be selected according to the requested application. This way, the fog protocol should be able to cope with the requirements introduced by the different IoT use cases, such as low-power communication devices, low bandwidth data links, low latency communications and high mobility scenarios.
The allocation of smart city application composed of micro-services with delay-sensitive requirements must be reactive if the device that is requesting the application is moving through the network area. This will imply a need for fast service

Figure 3.8 P2P fog protocol: based on a lightweight version of OSPF to exchange application service provisioning information between fog nodes and the cloud layer.



Application Service Table

Application ID	Service ID	Fog Node ID	IP Address	Network mask	Gateway	Interface	Metric	Cost
1000	1	2	192.168.10.5	255.255.255.0	192.168.10.1	192.168	1	10
1000	2	2	192.168.10.6	255.255.255.0	192.168.10.1	192.168	2	5

migrations, which can be achieved if messages are exchanged between the multiple FNs with information regarding services and applications. This way, each FD module can decide where and when it is more suitable to deploy and instantiate each instance of the micro-services composing the smart city application since the application service topology information of the network will be known by the different FNs. Therefore, in the presented framework, an application layer fog protocol based on a lightweight version of the OSPF [37] routing protocol for the exchange of application service information between FNs and the cloud layer is proposed. OSPF provides routing functionality within the same domain. It was designed to optimize the propagation of topological changes in the network. The goal is to use a lightweight application layer protocol based on the advantages of OSPF to propagate the application service logic information in the network, such as micro-service allocations and migrations, so that each FN knows exactly the application service topology, i.e., what and where each micro-service is allocated. This way, application service information can be shared among FNs similar to P2P networks. In fact, the foreseen fog protocol is a hierarchical application layer P2P networking protocol based on the advantages of OSPF.

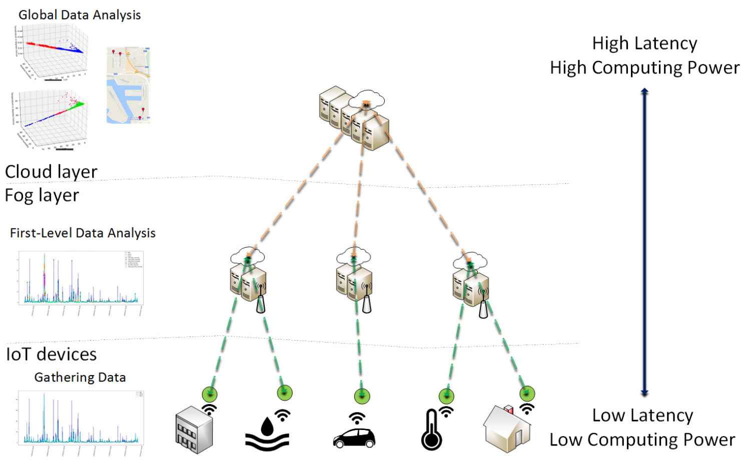
In Figure 3.8, a topology scenario of the fog protocol is presented. By using a fog protocol based on the advantages of OSPF, it will be possible to achieve rapid decisions related to the resource provisioning in smart cities, based on the exchange of application service tables. Each instance of applications and services have a unique ID allowing one to know exactly which instance is allocated to a certain FN. Each FN has an Internet Protocol (IP) address, IPv4 or IPv6, and also a unique ID. These attributes are transmitted as headers on the application service tables. These application service tables enable the exchange of resource operational information allowing the FD module to provide fast decisions on the resource provisioning because the overall network service allocation will be known by each fog node system, and therefore, decisions on service migrations can be achieved in a timely manner.

The fog protocol will split the network domain into multiple areas as in OSPF. FNs will only need to know the application service logic of the areas to which they are connected. Furthermore, FNs are classified according to the functions they will perform. As in OSPF, internal FNs are those connected to networks belonging to the same area, while Area Border Fog Nodes (ABFNs) are those connected to networks belonging to multiple adjacent areas. ABFNs will summarize the application service logic information of their attached areas for distribution through the backbone area, the cloud layer. Backbone FNs are those connected to the backbone area, including ABFNs. Autonomous System Boundary Fog Nodes (ASBFNs) are those that exchange application service information with FNs belonging to other service provider domains.

3.3.4 Distributed Data Monitoring and Analysis for 5G Smart Cities

The future 5G network architecture will support the integration of a massive number of infrastructure components and application services in smart cities. Regarding 5G-enabled applications, one of the remaining challenges is how to provide efficient resource allocation operations, because of the stringent requirements introduced by these type of applications: extremely low latencies, ultra-reliable communications and massive M2M communications. For smart city applications, services can be placed in a highly congested area, which would result in a higher latency in the communication between the fog nodes and the IoT device. This is unfeasible for delay-sensitive applications, since these require very low latencies, meaning these applications must be allocated on fog nodes close to the IoT sensor enabling the control of time-sensitive network functionalities close to the device. Furthermore, to provide a secure and reliable communication, it is necessary to perform data processing and analysis operations in a distributed way in order to detect anomalous behaviors and abnormal events in a timely manner so that appropriate actions can be performed in real time. In traditional centralized cloud solutions, all data are sent to the cloud layer, and then, monitoring and analysis operations are executed, which is not suitable for IoT scenarios. As mentioned before, in the proposed architecture, a monitoring and analysis component is present in the architecture of the FN. The proposed monitoring and data analysis approach is shown in Figure 3.9. This way, data will be processed near the end devices at the edge providing low-latency and scalable anomaly detection solutions, which will provide appropriate responses to protect infrastructure components, as well as application-level communication. First, IoT sensors will be distributed at various public infrastructures to monitor their condition variations over time. Then, these sensors will forward the raw data into the fog layer, where each FN will process the data samples associated with a local group of sensors and perform data analysis operations in a timely manner.

Figure 3.9 Overview of the distributed data analysis approach for 5G smart cities.



The outcomes of this first-level analysis will be processed by an event handler and a notification engine. If anomalous events are already detected on the data, notifications and alerts are generated and sent to the cloud layer, the CN and also to the IoT devices. Meanwhile, the unusual data samples that generate the alarms are also reported to the cloud layer in order to perform global behavior analysis based on machine learning techniques that require a higher computational power. Therefore, the CN will provide a broader view of the behavior of the network. Moreover, in-depth pattern recognition and event detection operations can be performed at the CN in order to support the FD module to apply more appropriate reactive and proactive responses.

3.3.5 Summary

The framework presented in this section enables autonomous MANO functionalities for smart city applications in diverse IoT scenarios. First, architectural challenges of the fog computing paradigm have been presented. Then, a system overview of the proposed framework has been detailed. The work provides MANO capabilities to 5G networks, which according to our knowledge are still not yet fully explored in literature since previous and ongoing research has not yet delivered an integrated and autonomous solution for the resource management and orchestration issue in smart cities. The approach follows the guidelines of the ETSI NFV MANO architecture and the standards of ETSI oneM2M regarding device management and security aspects, by extending it with additional software components that will offer not only high computing performance, but also monitoring and data analysis functionalities to provide secure and reliable communications in smart cities. Then, a novel application layer P2P fog protocol based on the advantages of the OSPF routing protocol has been introduced to provide exchange of application service provisioning information between the multiple FNs in the fog layer and the CN in the cloud layer. Instead of forwarding routing tables across the network, the objective of the foreseen fog protocol is to forward application service information, such as micro-service allocations and migration requests, enabling fast decisions related to the resource provisioning in smart cities. Finally, a hierarchical and distributed data monitoring and analysis approach based on the fog computing framework has been detailed. The proposed framework enables low-latency and scalable data analysis operations in order to detect abnormal events in a timely manner to apply appropriate actions in real time. In the next section, an anomaly detection use case is presented, which has been used to validate the distributed data analysis approach in 5G smart cities for the proposed fog-based MANO framework.

3.4 Evaluation Use Case

In this section, the evaluation scenario is introduced. Then, the datasets are presented. Finally, the data analysis operations used in the evaluation are described.

3.4.1 Scenario Description: Air Monitoring Application

The evaluation scenario is based on a use case within the scope of Antwerp's City of Things testbed [38]. The objective of the air monitoring application is to show the current status of the environment in the City of Antwerp and alert citizens of ambient pollution through a notification system in near real time. Regarding the current literature in air monitoring applications for smart cities, the proposed approach in [39] must be highlighted. In the article, a novel cloud-based approach for air quality monitoring is discussed. The approach is based on the design of two sensor front ends: a stationary air quality sensor that connects to the cloud via Ethernet and General Packet Radio Service (GPRS) and a portable sensor that connects to the smartphone via Bluetooth 4.0.

In contrast, the approach is based on the deployment of a set of air quality sensors mounted on the roofs of bpost's (Belgian postal services) delivery cars as an initial proof of concept. These sensors send measures of typical gases and climate data, such as temperature and humidity, which are then annotated with GPS locations. Furthermore, these sensors allow gathering real-time air quality information with broad city coverage, since each car is continuously driving around in the city.

3.4.2 Datasets

A summary of the characteristics of the datasets gathered for the evaluation is shown in Table 3.4. The two datasets come from two different bpost cars and consist of Particle Matter indicators (PM1, PM2.5 and PM10), temperature and humidity values that are annotated with a GPS location. The datasets have been collected by our research group between 9 May 2017 and 29 June 2017. Temperature and relative humidity values collected by bpost Car 1 and bpost Car 2 are shown in Figure 3.10a and Figure 3.10b, respectively.

Table 3.4: Evaluation datasets. bpost, Belgian postal services.

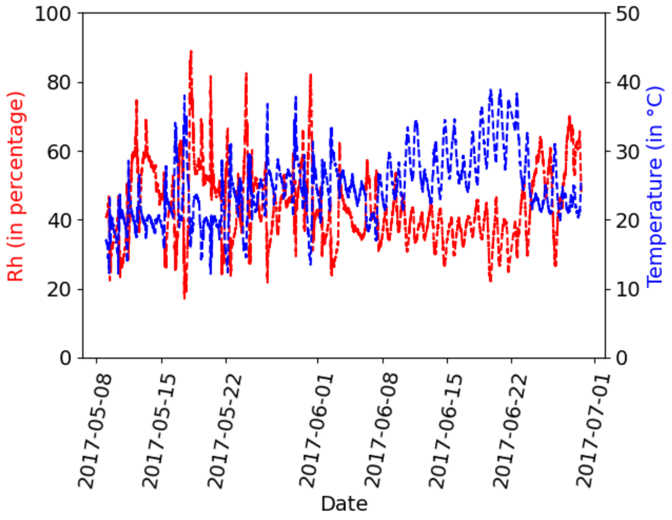
Dataset Name	No. of Records	Description
bpost 1	70636	Particle matter indicators (PM1, PM2.5, PM10), Temperature, Humidity and GPS locations from bpost car 1 between 9 May 2017 and 29 June 2017
bpost 2	70640	Particle matter indicators (PM1, PM2.5, PM10), Temperature, Humidity and GPS locations from bpost car 2 between 9 May 2017 and 29 June 2017

3.4.3 Data Analysis Operations

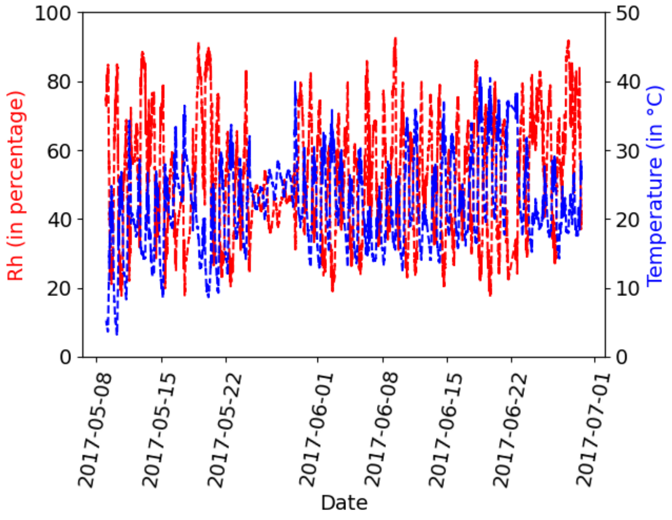
In this subsection, the data analysis operations used in the evaluation are introduced.

Figure 3.10 Temperature and relative humidity values collected from two bpost cars.

(a) bpost Car 1



(b) bpost Car 2



3.4.3.1 First-Level Analysis: Unsupervised Outlier Detection Algorithms

Outlier detection is related to the identification of unusual data samples when compared to the rest of the dataset. The Robust Covariance (RC) and the Isolation Forrest (IF) algorithms have been employed as a first-level analysis to be performed by FNs in the fog layer. These algorithms have been evaluated by using Scikit-Learn [40], a machine learning library written in Python.

3.4.3.2 Global Analysis: GPS Locations of Outliers

The unusual measurements detected by FNs through outlier detection must be further analyzed by application experts in order to extract more information from them. As such, FNs will trigger appropriate alarms and notifications, which will be forwarded through the network to the CN. Then, these unusual samples will be sent to the CN so that more complex and resource-consuming operations can be performed, e.g., for root cause analysis. In the evaluation, the outliers have been compared with the annotated GPS locations as a global analysis operation.

3.4.4 Fog-Cloud Infrastructure Dimensioning

Nowadays, low power wireless technologies and fog computing architectures have gained tremendous emphasis due to the massive growth of connected devices in the network. The need for connecting simple IoT devices, such as sensors and actuators, is increasing rapidly. Variables used in the fog-cloud infrastructure dimensioning are shown in Table 3.5.

Table 3.5: Variables of the fog-cloud infrastructure dimensioning.

Symbol	Description
C	Communication range in meters.
U_{LPWAN}	Upload data rate of the LPWAN technology in Mbps.
D_{LPWAN}	Download data rate of the LPWAN technology in Mbps.
U_{Fog}	Upload speed between a fog node and the cloud node in Mbps.
D_{Fog}	Download speed between a fog node and the cloud node in Mbps.
R	Number of data samples to be transmitted.
N	Number of bits in each data sample.
T	Transmission time of a packet.

In fog computing architectures, fog nodes are usually located within one hop from the IoT sensors. The variable C is used to indicate the communication range in meters between a fog node and an IoT sensor. Two variables, U_{LPWAN} and D_{LPWAN}, are used to indicate the upload and the download data rate of the LPWAN technology, respectively. Then, the number of bits in each data sample is given by N. The number of data samples to be transmitted is given by R. This way, the upload and the download transmission time of a packet between an IoT sensor and a fog node can be expressed as shown in (3.1) and in (3.2), respectively.

$$T(U_{LPWAN}) = \frac{N \times R}{U} \tag{3.1}$$

$$T(D_{LPWAN}) = \frac{N \times R}{D} \tag{3.2}$$

Moreover, the communication between a fog node and the cloud node is via the Internet, where the upload speed is given by U_{Fog} and the download speed is given by D_{Fog}.

3.5 Evaluation Results

3.5.1 Unsupervised Outlier Detection

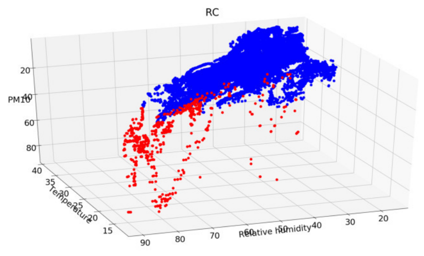
In Figure 3.11a,b, the outcomes of the RC algorithm for the three dimensions regarding PM10, temperature and relative humidity for the bpost Car 1 are shown, with a contamination of 1.0% indicating that the RC algorithm intends to find the 1.0% of samples, which can be considered as abnormal. Similarly, in Figure 3.11c,d, the results obtained for the IF outlier detection algorithm with a contamination of 1.0% for the bpost Car 1 are shown. In addition, in Figure 3.12a,b, the outcomes of the RC algorithm for the bpost Car 2 are presented, while in Figure 3.12c,d, the results obtained from the IF algorithm are shown. As can be observed, clear similarities are present in the results obtained by both algorithms for the two bpost cars. PM10 values above 30 ppm collected by both cars are marked as outliers by both algorithms, which indicates that these samples can be considered as unusual measurements.

3.5.2 Global Analysis: GPS Locations of Outliers

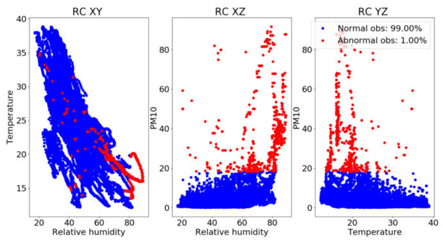
As previously mentioned, as a global analysis operation, the outliers detected by the RC and IF algorithms have been linked with the GPS locations available in the datasets. Only measurements marked as outliers by both algorithms have been considered. In Figure 3.13, the GPS locations where PM10 values above 70 ppm have been collected by the bpost cars, which have been marked as outliers by both RC and IF outlier detection algorithms, are shown on a map of the city. Regarding bpost Car 1 measurements, most of the data samples marked as outliers have been collected in the warehouse (area highlighted in orange in Figure 3.13), where the bpost cars usually stay at night. These high values of PM10 can be related to organic compounds, which were inside the warehouse at the time of the measurements. On the other hand, the outliers measured by bpost Car 2 have been collected

Figure 3.11 Outcomes of RC and IF algorithms (blue color: normal samples, red color: abnormal samples) for PM10, temperature and relative humidity values collected by the bpost Car 1.

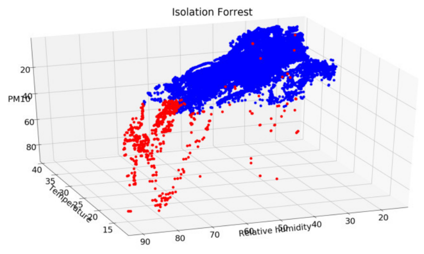
(a) RC with 1.0% (3D perspective).



(b) RC with 1.0% (3D planes).



(c) IF with 1.0% (3D perspective).



(d) IF with 1.0% (3D planes).

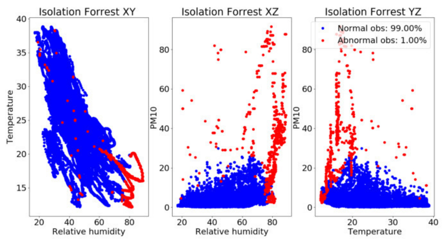
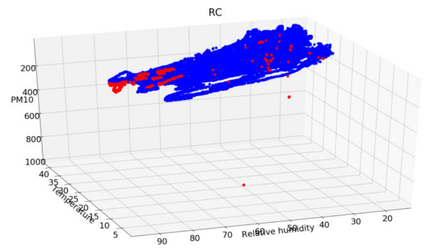
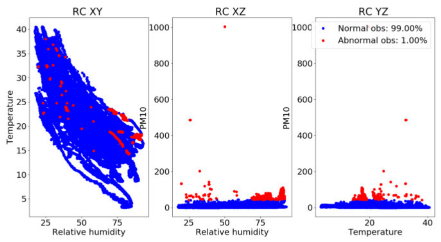


Figure 3.12 Outcomes of RC and IF algorithms (blue color: normal samples, red color: abnormal samples) for PM10, temperature and relative humidity values collected by the bpost Car 2.

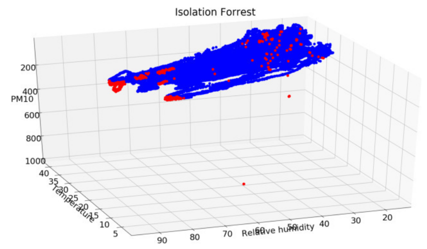
(a) RC with 1.0% (3D perspective).



(b) RC with 1.0% (3D planes).



(c) IF with 1.0% (3D perspective).



(d) IF with 1.0% (3D planes).

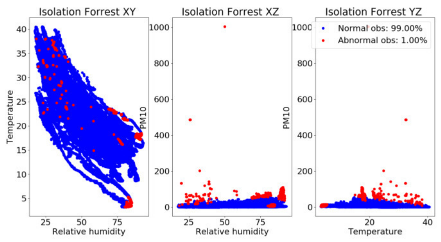
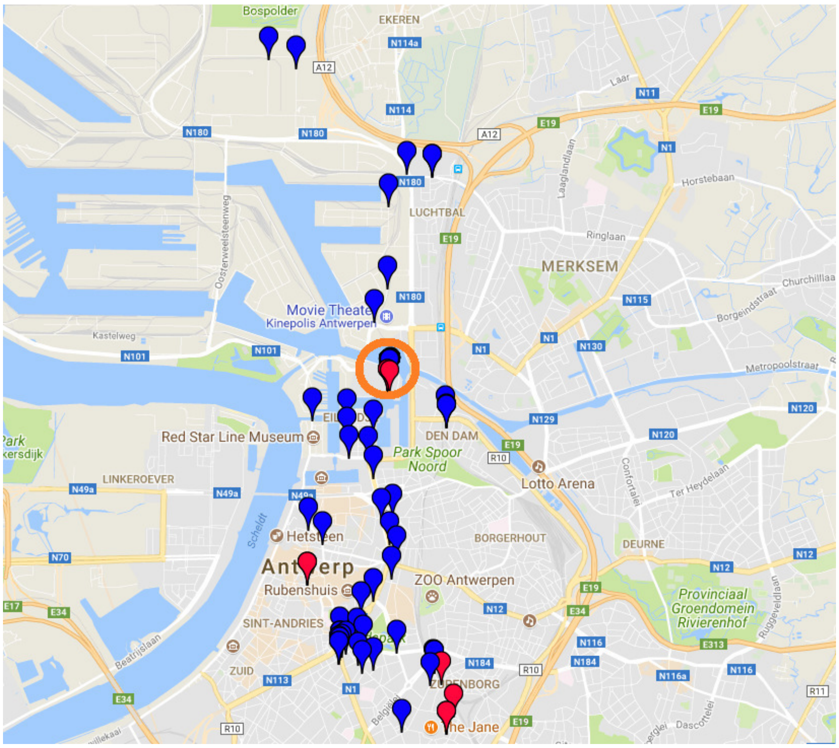


Figure 3.13 GPS locations collected by the bpost cars (bpost1, red/bpost2, blue). These values have been marked as outliers by both outlier detection algorithms considered in the evaluation.



across the city of Antwerp. These values can be related to high traffic volumes in the city at the time of the measurements. Furthermore, a large number of samples collected by bpost Car 2 have been marked as outliers, when the car stayed in the warehouse. These outliers can be explained by the high relative humidity and the high values of the PM10 indicator.

Global analysis operations should be conducted in the cloud layer so that unusual measurements can be understood. This way, citizens and government agencies can receive live input of the city traffic, which should be used to temporarily modify traffic rules, e.g., reduce the maximum driving speed on certain highways. Furthermore, in the long term, these global operations can provide an effective way on how city policies should be changed.

3.5.3 Fog-Cloud Infrastructure Analysis

In the fog-cloud infrastructure evaluation, the communication between a fog node and the cloud node is via the Internet, where the upload speed is 5 Mbps and the download speed is 2.5 Mbps. Furthermore, a 500-m communication range has been considered between a fog node and an IoT sensor. The communication between a fog node and an IoT sensor is performed by a LPWAN technology. An IEEE 802.11ah [41] wireless network has been considered as the LPWAN technology, because it is one of the most promising LPWAN technologies with very high data rates. High data rates are an important requirement of the evaluation use case, because timely alerts need to be sent to the IoT sensor if a malfunction is already detected on a fog node. This way, an upload data rate of 2 Mbps and a download data rate of 1 Mbps have been considered based on IEEE 802.11ah. Although the maximum throughput of IEEE 802.11ah is higher, with this lower limitation, it has been assumed that good channel conditions are always achieved and that all IoT sensors can communicate at these data rates. Other LPWAN technologies, such as LoRaWAN [42] and Sigfox [43], have not been considered because very low data rates and duty cycle restrictions make it impossible for these LPWAN technologies to send a data sample every minute.

Considering that for the use case, each upload message is composed of a string of 12 chars (GPS location, geohash) equal to 12 bytes, a 32 bit integer (timestamp) equal to 4 bytes and 5 floating point 64-bit numbers (particle matter indicators, temperature and humidity) equal to 40 bytes, the total number of payload bytes to be transmitted per minute from the IoT sensor to the fog node is 56 bytes. On the other hand, each download message to be transmitted from the fog node to the IoT sensor in case of unusual behavior or malfunction is composed of a string of 12 chars (GPS location, geohash) equal to 12 bytes and a byte defined by 3 alarm bits and 5 bits for 32 types of predefined messages. Therefore, each upload message is transmitted with at least 56 bytes, which is equal to 448 bits and each download message with 13 bytes, which is equal to 104 bits. Moreover, the fog node will send 1% of the total data samples to the cloud node for global analysis operations. However, in a traditional centralized cloud solution, all data samples need to be transported from the IoT sensor to the cloud node. Based on these assumptions, the fog-based framework has been evaluated and compared with a traditional centralized cloud solution in terms of response time in case abnormal samples are detected and in terms of network bandwidth usage. The comparison is presented in Table 3.6.

Table 3.6: Performance evaluation of the fog-cloud infrastructure.

Variable	**Fog-Cloud**	**Centralized Cloud**
C	500 m	
N_{Upload}	448 bits	
$N_{Download}$	104 bits	
U_{LPWAN}	2 Mbps	
D_{LPWAN}	1 Mbps	
$T_{Upload:Sensor-Fog}$	0.224 ms	
$T_{Download:Sensor-Fog}$	0.104 ms	
U_{Fog}	5 Mbps	
D_{Fog}	2.5 Mbps	
$T_{Upload:Fog-Cloud}$	0.090 ms	
$T_{Download:Fog-Cloud}$	0.041 ms	
$R_{Sensor-Fog}$	70,000 data samples	
$R_{Fog-Cloud}$	700 data samples (1%)	70,000 data samples (100%)
$N_{Fog-Cloud}$	72.8 Kbps	31.36 Mbps
Transporting Data Samples to the Cloud	14.56 ms	6.27 s
Network Bandwidth Usage (%)	1.46%	627.2%
End-to-End delay for Alert/Notification	0.33 ms	0.46 ms

Results show that to send all data samples to the CN, 6.27 seconds are needed, while in the fog computing approach, only 14.56 milliseconds are required since only 1% of the data is transmitted to the cloud node. Moreover, the proposal only requires 1.46% of the available bandwidth between the fog node and the cloud node, while in a traditional centralized solution, on average, six-times the available bandwidth is needed (627.2%) only for a single fog node to transmit all data samples from only one IoT sensor, which makes it impossible for centralized cloud solutions to comply with smart city use cases. The end-to-end delay in case an abnormal sample is detected is lower in the proposed solution since the first-level analysis is executed on fog nodes. On average, it needs 0.33 milliseconds to send an alarm or notification to the IoT sensor, while a traditional approach requires 0.46 milliseconds, but with an enormous amount of network bandwidth.

3.6 Conclusions

In this chapter, a fog computing framework for the management and orchestration of smart city applications in 5G wireless networks is presented. Fog computing has been introduced to deal with the growing amount of connected devices in the upcoming years, by placing computational resources on the edges of the network. This way, fog computing solutions can provide effective ways to overcome the strict requirements introduced by IoT scenarios, such as low latency, high energy efficiency and high mobility. The proposed framework extends the current state-of-the-art within fog computing, NFV and MANO paradigms by providing autonomous capabilities for the resource provisioning in 5G-enabled smart cities. The main contribution of the chapter is its fully-integrated and autonomous fog node management system with data analysis and decision making functionalities alongside a foreseen novel application layer P2P fog protocol based on the OSPF routing protocol enabling the exchange of application service provisioning information between fog nodes and the cloud layer. Moreover, an anomaly detection use case has been evaluated focusing on the proposed framework. The evaluation results show that the presented approach achieves a substantial reduction in terms of network bandwidth usage when compared with traditional centralized cloud solutions. The approach only requires 1.46% of the available network bandwidth between a fog node and a cloud node. Moreover, the proposed framework can send timely alerts to IoT sensors in case abnormal samples are detected, since a first-level analysis is performed in fog nodes. As future work, prototypes and proof-of-concepts of the fog computing framework will be implemented. Moreover, simulation studies will be carried out to evaluate the operational aspects of the foreseen fog protocol.

Acknowledgment

This research was performed partially within the projects “Service-oriented management of a virtualized future internet” and “Intelligent DEnse And Long range IoT networks (IDEAL-IoT)” under Grant Agreement #S004017N, both from the fund for Scientific Research-Flanders Fonds Wetenschappelijk Onderzoek - Vlaanderen (FWO-V).

Addendum

Note on Table 3.6: A mathematical error can be found on the original manuscript this chapter is based upon on Table 3.6. Table 3.7 below provides the corrected Table 3.6.

Table 3.7: Performance evaluation of the fog-cloud infrastructure.

Variable	Fog-Cloud	Centralized Cloud
C	500 m	
N_{Upload}	448 bits	
$N_{Download}$	104 bits	
U_{LPWAN}	2 Mbps	
D_{LPWAN}	1 Mbps	
$T_{Upload:Sensor-Fog}$	0.224 ms	
$T_{Download:Sensor-Fog}$	0.104 ms	
U_{Fog}	5 Mbps	
D_{Fog}	2.5 Mbps	
$T_{Upload:Fog-Cloud}$	0.090 ms	
$T_{Download:Fog-Cloud}$	0.041 ms	
$R_{Sensor-Fog}$	70,000 data samples	
$R_{Fog-Cloud}$	700 data samples (1%)	70,000 data samples (100%)
$N_{Fog-Cloud}$	313.6 Kbps	31.36 Mbps
Transporting Data Samples to the Cloud	62.72 ms	6.27 s
Network Bandwidth Usage (%)	6.27%	627.2%
End-to-End delay for Alert/Notification	0.33 ms	0.46 ms

References

[1] Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021 white paper, 2017. URL http://www.cisco.com/c/en/us/solutions/collateral/service-provider/\visual-networking-index-vni/mobile-white-paper-c11-520862.pdf.

[2] Akhil Gupta and Rakesh Kumar Jha. A survey of 5g network: Architecture and emerging technologies. *IEEE access*, 3:1206–1232, 2015.

[3] Luis M Vaquero and Luis Rodero-Merino. Finding your way in the fog:

Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014.

[4] Hlabishi I Kobo, Adnan M Abu-Mahfouz, and Gerhard P Hancke. A survey on software-defined wireless sensor networks: Challenges and design requirements. *IEEE access*, 5:1872–1899, 2017.

[5] Mung Chiang and Tao Zhang. Fog and iot: An overview of research opportunities. *IEEE Internet of things journal*, 3(6):854–864, 2016.

[6] Mathias Santos de Brito, Saiful Hoque, Thomas Magedanz, Ronald Steinke, Alexander Willner, Daniel Nehls, Oliver Keils, and Florian Schreiner. A service orchestration architecture for fog-enabled infrastructures. In *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 127–132. IEEE, 2017.

[7] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, pages 195–216, 2017.

[8] Charith Perera, Yongrui Qin, Julio C Estrella, Stephan Reiff-Marganiec, and Athanasios V Vasilakos. Fog computing for sustainable smart cities: A survey. *ACM Computing Surveys (CSUR)*, 50(3):1–43, 2017.

[9] ETSI oneM2M (2016). Etsi technical specification, onem2m functional architecture. onem2m ts-0001 version 2.10.0 release 2., 2017. URL http://www.etsi.org/deliver/etsi_ts/118100_118199/118101/\02.10.00_60/ts_118101v021000p.pdf.

[10] OpenFog Consortium (2017). Openfog consortium, openfog reference architecture for fog computing., 2017. URL https://www.openfogconsortium.org/wp-content/uploads/\OpenFog_Reference_Architecture_2_09_17-FINAL.pdf.

[11] ETSI MEC (2016). Etsi technical specification, mobile edge computing (mec): Framework and reference architecture., 2017. URL http://www.etsi.org/deliver/etsi_gs/MEC/\001_099/003/01.01.01_60/gs_MEC003v010101p.pdf.

[12] ETSI NFV MANO (2014). Etsi technical specification, network functions virtualisation (nfv): Management and orchestration., 2017. URL http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/\gs_NFV-MAN001v010101p.pdf.

[13] OMA LwM2M (2014). Oma white paper, lightweight m2m: Enabling device management and applications for the internet of thing., 2017. URL http://cdn2.hubspot.net/hub/183757/file-610591431-pdf/docs/\OMA_Whitepaper_Lightweight_M2M_3-14%5b5%5d.pdf.

[14] TOSCA (2017). Topology and orchestration specification for cloud applications (tosca), oasis., 2017. URL https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca.

[15] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4):14–23, 2009.

[16] Luis Sánchez, Verónica Gutiérrez, Jose Antonio Galache, Pablo Sotres, Juan Ramón Santana, Javier Casanueva, and Luis Muñoz. Smartsantander: Experimentation and service provision in the smart city. In *2013 16th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 1–6. IEEE, 2013.

[17] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, et al. Smartsantander: Iot experimentation over a smart city testbed. *Computer Networks*, 61:217–238, 2014.

[18] Dan Puiu, Payam Barnaghi, Ralf Tönjes, Daniel Kümper, Muhammad Intizar Ali, Alessandra Mileo, Josiane Xavier Parreira, Marten Fischer, Sefki Kolozali, Nazli Farajidavar, et al. Citypulse: Large scale data analytics framework for smart cities. *IEEE Access*, 4:1086–1108, 2016.

[19] David Perez Abreu, Karima Velasquez, Marilia Curado, and Edmundo Monteiro. A resilient internet of things architecture for smart cities. *Annals of Telecommunications*, 72(1-2):19–30, 2017.

[20] Riccardo Petrolo, Valeria Loscri, and Nathalie Mitton. Towards a smart city based on cloud of things. In *Proceedings of the 2014 ACM international workshop on Wireless and mobile technologies for smart cities*, pages 61–66, 2014.

[21] Xiao Liu, Yuxin Liu, Houbing Song, and Anfeng Liu. Big data orchestration as a service network. *IEEE Communications Magazine*, 55(9):94–101, 2017.

[22] Muhammad Naeem, Waleed Ejaz, Lutful Karim, Syed Hassan Ahmed, Alagan Anpalagan, Minho Jo, and Houbing Song. Distributed gateway selection for m2m communication in cognitive 5g networks. *IEEE Network*, 31(6): 94–100, 2017.

[23] SONATA (2017). Sonata nfv: Agile service development and orchestration in 5g virtualized networks., 2017. URL http://sonata-nfv.eu/.

[24] COHERENT (2017). Coherent, a unified programmable control framework for 5g heterogeneous radio access networks., 2017. URL http://www.ict-coherent.eu/.

[25] SELFNET (2017). Selfnet, framework for self-organized network management in virtualized and software defined networks., 2017. URL https://selfnet-5g.eu/.

[26] SLICENET (2017). Slicenet: End-to-end cognitive network slicing and slice management framework in virtualised multi-domain, multi-tenant 5g networks., 2017. URL https://5g-ppp.eu/slicenet/.

[27] SESAME (2017). Sesame: Small cells coordination for multi-tenancy and edge services., 2017. URL http://www.sesame-h2020-5g-ppp.eu/.

[28] Openstack (2017). Openstack, open source software for creating private and public clouds., 2017. URL https://www.openstack.org/.

[29] OpenBaton (2017). Openbaton, an extensible and customizable nfv mano-compliant framework., 2017. URL http://openbaton.github.io/.

[30] OpenMano (2017). Openmano, an open source project providing a practical implementation of the reference architecture for management and orchestration under standardization at etsi nfv isg., 2017. URL https://github.com/nfvlabs/openmano.

[31] OpenDayLight (2017). Opendaylight, a modular open platform for customizing and automating sdn networks of any size and any scale., 2017. URL https://www.opendaylight.org/.

[32] ONOS (2017). Onos, an open source sdn network operating system designed for building next-generation sdn/nfv solutions., 2017. URL http://onosproject.org/.

[33] Ryu (2017). Ryu, component-based sdn framework., 2017. URL https://osrg.github.io/ryu/.

[34] LwM2M (2017). Oma lightweightm2m specifications and lwm2m developer tool kit., 2017. URL https://github.com/OpenMobileAlliance/OMA_LwM2M_for_Developers.

[35] Leshan (2017). Leshan: Oma lightweight m2m server and client java implementation., 2017. URL https://github.com/eclipse/leshan.

[36] 5G-EmPOWER (2017). 5g-empower: Multi-access edge computing operating system supporting lightweight virtualization and heterogeneous radio access technologies., 2017. URL http://empower.create-net.org/.

[37] OSPF (1997). Moy, j. ospf version 2. RFC Editor, Standards Track, 1997, RFC 2178., 2017. URL https://tools.ietf.org/html/rfc2178.

[38] Steven Latre, Philip Leroux, Tanguy Coenen, Bart Braem, Pieter Ballon, and Piet Demeester. City of things: An integrated and multi-technology testbed for iot smart city experiments. In *2016 IEEE international smart cities conference (ISC2)*, pages 1–8. IEEE, 2016.

[39] Xiaofan Jiang. Large scale air-quality monitoring in smart and sustainable cities. pages 725–753, 2017.

[40] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[41] Weiping Sun, Munhwan Choi, and Sunghyun Choi. Ieee 802.11 ah: A long range 802.11 wlan at sub 1 ghz. *Journal of ICT standardization*, 1(1):83–108, 2013.

[42] Ferran Adelantado, Xavier Vilajosana, Pere Tuset-Peiro, Borja Martinez, Joan Melia-Segui, and Thomas Watteyne. Understanding the limits of lorawan. *IEEE Communications magazine*, 55(9):34–40, 2017.

[43] Juan Carlos Zuniga and Benoit Ponsard. Sigfox system description. *LPWAN@ IETF97, Nov. 14th*, 25, 2016.

4 Resource Provisioning in Fog Computing: From Theory to Practice

"Before anything else, preparation is the key to success."

–Alexander Graham Bell (1847 - 1922)

This chapter extends the work presented in chapter 3 by evaluating the feasibility of a popular orchestration platform named Kubernetes for IoT services (challenge #2). The chapter proposes a Fog Computing architecture based on the Kubernetes architectural model. Also, the chapter addresses challenge #4 by studying efficient allocation strategies for IoT applications. It presents a network-aware scheduler for the Kubernetes platform focused on minimizing latency and optimizing bandwidth. Evaluations have assessed the performance of the proposed approach compared to the Kubernetes default scheduling feature and the Integer Linear Programming (ILP) formulation presented in chapter 2. The ILP model has been containerized to compare the performance of the proposed approach with an optimal scheme. Results show that the network-aware scheduling approach achieves reductions of up to 70% concerning network latency compared to the default scheduling mechanism. Thus, the main contribution of this chapter is the development of a network-aware scheduling approach focused on IoT use cases, validated in Kubernetes against theoretical formulations and default scheduling algorithms.

⋆⋆⋆

José Santos, Tim Wauters, Bruno Volckaert and Filip De Turck.

Published in Sensors, Volume 19, no. 10, pages 2238, MDPI, 2019.

Abstract The Internet-of-Things (IoT) and Smart Cities continue to expand at enormous rates. Centralized Cloud architectures cannot sustain the requirements imposed by IoT services. Enormous traffic demands and low latency constraints are among the strictest requirements, making cloud solutions impractical. As an answer, Fog Computing has been introduced to tackle this trend. However, only theoretical foundations have been established and the acceptance of its concepts is still in its early stages. Intelligent allocation decisions would provide proper resource provisioning in Fog environments. In this chapter, a Fog architecture based on Kubernetes, an open source container orchestration platform, is proposed to solve this challenge. Additionally, a network-aware scheduling approach for container-based applications in Smart City deployments has been implemented as an extension to the default scheduling mechanism available in Kubernetes. Last but not least, an optimization formulation for the IoT service problem has been validated as a container-based application in Kubernetes showing the full applicability of theoretical approaches in practical service deployments. Evaluations have been performed to compare the proposed approaches with the Kubernetes standard scheduling feature. Results show that the proposed approaches achieve reductions of 70% in terms of network latency when compared to the default scheduling mechanism.

4.1 Introduction

In recent years, the Internet-of-Things (IoT) rapidly started gaining popularity due to the wide adoption of virtualization and cloud technologies. IoT services have been introducing a whole new set of challenges by transforming everyday life objects into smart connected devices [1]. With the advent of IoT, Smart Cities [2] have become an even more attractive business opportunity. Smart Cities aim to reshape different domains of urban life, such as waste management, public transportation and street lightning. According to [3], by 2022, nearly three-quarters of all connected devices in the mobile network are expected to be smart devices. Additionally, the share of Low-Power Wide-Area Network (LPWAN) connections is expected to grow from about 2 percent in 2017 to 14 percent by 2022, from 130 million devices in 2017 to 1.8 billion devices by 2022. LPWANs are low-power wireless connectivity solutions specifically meant for Machine-to-Machine (M2M) use cases requiring wide geographic coverage and low bandwidth. Nowadays, the centralized structure of cloud computing is facing tremendous scalability challenges to meet the decentralized nature of IoT services due to the enormous bandwidth demands, high mobility coverage and low latency requirements [4]. As an answer, Fog Computing [5, 6] has emerged as an extension to the Cloud Computing paradigm by distributing resources on the edges of the network close to end devices, thus, helping to meet the demanding constraints introduced by IoT services. Waste management platforms, Augmented Reality applications, video streaming services and smart transportation systems are already envisioned Smart City use cases for Fog Computing, which will benefit from the nearby real-time processing and data storage operations to overcome the limitations of traditional cloud architectures [7]. Although the theoretical foundations of Fog Computing have already been established, the adoption of its concepts is in early stages. Practical implementations of Fog Computing solutions are scarce. Additionally, research challenges in terms of resource provisioning and service scheduling still persist. In fact, setting up a proper Fog-based architecture to support millions of devices and their high demand heterogeneous applications without dismissing the importance of network latency, bandwidth usage and geographic coverage is still a challenge to be addressed in Fog Computing [8].

Nowadays, container-based micro-services are revolutionizing software development [9]. Micro-services represent an architectural style inspired by service-oriented computing that has recently started gaining popularity. An application is decomposed in a set of lightweight autonomous containers deployed across a large number of servers instead of the traditional single monolithic application [10]. Each micro-service is developed and deployed separately, without compromising the application life-cycle. Currently, containers are the *de facto* alternative to the conventional Virtual Machine (VM), due to their high and rapid scalability and their low resource consumption. In fact, due to their broad acceptance, several research projects are being conducted on container technologies by IT companies and open-source communities. The most popular among them is named Kubernetes [11]. Kubernetes is an open-source container management platform originally developed by Google. Kubernetes simplifies the deployment of reliable, scalable distributed systems by managing the complete orchestration life-cycle of containerized applications. Although containers already provide a high level of abstraction, they still need to be properly managed, specifically in terms of resource consumption, load balancing and server distribution, and this is where integrated solutions like Kubernetes come into their own [12]. Therefore, in this chapter, a Fog Computing architecture based on the Kubernetes platform for Smart City deployments is presented. The proposed architecture has been designed for Antwerp's City of Things testbed [13]. Furthermore, intelligent allocation decisions are crucial for proper resource provisioning in Fog environments. Multiple factors should be taken into account, such as response time, energy consumption, network latency, reliability, bandwidth usage and mobility [14]. Although Kubernetes already provides provisioning functionalities, the scheduling feature merely

takes into account the number of requested resources (CPU, RAM) on each host, which is rather limited when dealing with IoT services.
Thus, a network-aware scheduling approach presented in [15] has been implemented as an extension to the default scheduling feature available in Kubernetes to enable resource allocation decisions based on the current status of the network infrastructure. Last but not least, an Integer Linear programming (ILP) formulation for the IoT service placement problem presented in [16] has been deployed on the Kubernetes container orchestration platform, showing the full applicability of theoretical approaches in practical service deployments. Finally, evaluations based on Smart City container-based applications have been performed to compare the performance of the proposed provisioning mechanisms with the standard scheduling feature present in Kubernetes.
The remainder of the chapter is organized as follows. In the next Section, related work is discussed. Then, in Section 4.3, the importance of proper resource provisioning in Fog Computing is highlighted. Section 4.4 introduces the proposed Fog-based Kubernetes architecture for the resource provisioning of container-based services in Smart City deployments and its scheduling features. In Section 4.5, the proposed scheduling extensions in Kubernetes are discussed. Then, in Section 4.6, the evaluation setup is described which is followed by the evaluation results in Section 4.7. Finally, conclusions are presented in Section 4.8.

4.2 Related Work

In recent years, several studies have been carried out to deal with resource provisioning issues in Smart City deployments specifically tailored to IoT services. In [17], a reference Fog-based architecture has been presented. Their approach focused on implementing a Software Defined Resource Management layer at the Fog layer to locally serve IoT requests. Among different functionalities, a resource provisioning module has been included which is responsible for making allocation decisions based on metrics gathered by a monitoring module. In [18] both architectural and resource allocation concepts have been tackled. The authors proposed a provisioning algorithm focused on service elasticity and on the number of available resources by using virtualization technologies. Simulation results have shown that the proposed algorithm efficiently schedules resources while minimizing the response time and maximizing the throughput, without any consideration to the overall cost. Furthermore, in [19], a resource scheduling approach based on demand predictions has been presented. Their work focuses on allocating resources based on users' demand fluctuations by using cost functions, different types of services and pricing models for new and existing customers. The model achieves a fair performance by preallocating resources based on user behavior and future usage predictions.

Additionally, in [20], the IoT resource provisioning issue has been modeled as an optimization problem. The model considered the maximization of Fog resources and the minimization of overall network delay. Their work has been extended in [21], where application Quality of Service (QoS) metrics and deadlines for the provisioning of each type of application have been taken into account. In [22], a hybrid approach for service orchestration in Fog environments is introduced. The solution encompasses two stages. On one hand, at the IoT and South-Bound Fog Levels, distributed management is proposed, which applies choreography techniques to enable automated fast decision making. On the other hand, centralized orchestration is suggested at the North-Bound Fog and Cloud Levels. In [23], an algorithm for workload management in Fog infrastructures has been presented. Their work focuses on task distribution at the Fog layer while minimizing response time based on resources demanded by these tasks. However, specific QoS requirements have not been considered in their approach. In [24], a service provisioning approach for combined fog-cloud architectures has been formulated as an optimization problem. Their model focuses on the minimization of network latency while guaranteeing proper service operation. Furthermore, in [25], an optimization formulation for the service deployment of IoT applications in Fog scenarios has been proposed and implemented as a prototype named FogTorch. Their work focused not only on hardware and software demands but also on QoS requirements, such as network latency and bandwidth.

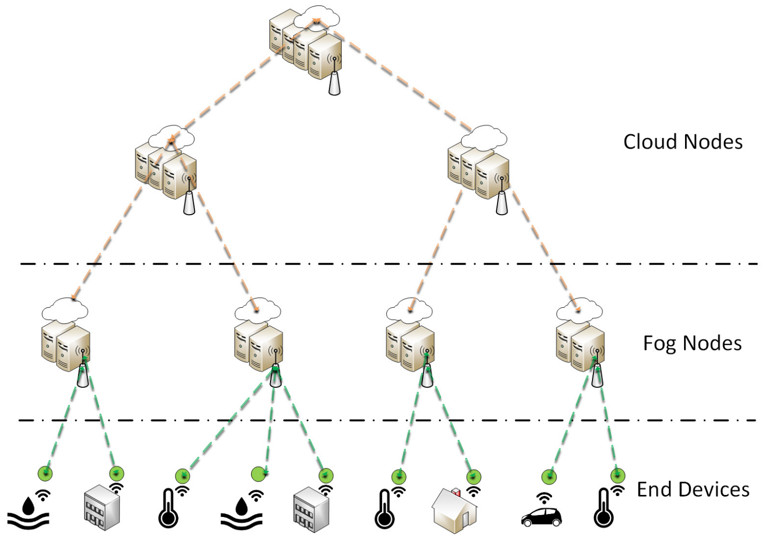
In summary, this work advances beyond existing and ongoing studies that individually address some of the remaining challenges, but have not yet delivered an autonomous and complete solution for proper resource provisioning in Fog Computing. In this chapter, a Fog-based Kubernetes architecture is proposed to enable the deployment of Smart City container-based services, while increasing the performance over existing network infrastructure to fully maximize the potential of new business opportunities triggered by IoT and Smart City use cases. It combines Fog Computing concepts alongside the flexible and powerful Kubernetes platform to improve the performance of application-to-resource provisioning schemes. By combining powerful container orchestration technologies as Kubernetes and Fog Computing concepts, the proposed approach paves the way towards a proper resource provisioning in the Smart City ecosystem.

4.3 Open Challenge: Resource Provisioning in Fog Computing

This section highlights the importance of proper resource provisioning in Fog environments.

4.3.1 Relevance of Proper Resource Provisioning

Figure 4.1 High-level view of Fog Computing.



Resource provisioning is related to the allocation of computing, network and storage resources needed to instantiate and deploy applications and services requested by clients and devices over the Internet. Fog Computing has been introduced to address the inherent challenges of computing resource allocation for IoT services in Smart City deployments. Services can be provisioned in a highly congested location, or even further from sensors, which would result in a higher communication latency since current sensors and gateways are lacking in terms of processing power, storage capacity and memory [26]. Centralized solutions are not suitable for IoT since sending all the collected data to the cloud is unfeasible due to the high bandwidth requests. Fog Computing provides data processing and analytics operations locally, which drastically reduces the amount of data needed to transport to the cloud [27]. Furthermore, appropriate responses to protect infrastructure components as well as application level communication can be executed in a timely manner if malfunctions or abnormal events are detected in the data.

Figure 4.1 presents a high-level view of the Fog Computing architecture. Opposed to a centralized cloud solution, end devices, sensors and actuators mainly communicate through wireless gateways, which are linked with a Fog layer through multiple Fog Nodes (FNs). The communication with the Cloud layer is then performed through Cloud Nodes (CNs). Nevertheless, as previously mentioned, concrete implementations of Fog Computing concepts are still in early stages and several issues still remain unresolved in resource provisioning for Fog Computing architectures:

- **Latency:** IoT services are highly challenging in terms of latency demands, since delay-sensitive applications, such as connected vehicles and health-care monitoring services, require low latencies in the order of milliseconds. If the latency threshold is exceeded, the service can become unstable, action commands may arrive too late and control over the service is potentially lost. Fog Computing is essential to provide low latencies to these delay-sensitive services.
- **Bandwidth:** The available bandwidth between sensors and the cloud is a serious constraint in Smart Cities. Datasets are so huge that the amount of bandwidth needed to transport all the data to the cloud is unacceptable. For instance, considering a video surveillance use case, where a video camera requires a connection of 10 Mb/s. Continuously sending the data from the video camera to the cloud translates into approximately 3.24 TB/monthly for a single camera. It is therefore essential to adopt Fog Computing concepts to perform data analysis operations locally, thus, reducing the amount of data transferred to the cloud.
- **Energy efficiency:** IoT devices are usually resource-constrained regarding their battery capacity, computational power, size and weight. For instance, considering a smart lightning use case, where thousands of sensors are measuring and controlling the light intensity of street lampposts. These sensors periodically wake up, measure values, send data samples to the network and then enter a sleep mode. Then, FNs perform the required computational operations on behalf of the sensors on the data collected to ensure an extension of the devices' lifetime.
- **Programmability:** Fog Computing solutions are currently being designed as software driven technologies [28]. A Fog service provider will own a set of distributed Fog and Cloud Nodes where all hierarchical levels are simple to access and the whole software stack is easy to setup and maintain. Thus, the economic value of IoT is in the application software and the infrastructure running it. In fact, software modules are needed for life-cycle management and orchestration of Smart City services, including resource provisioning mechanisms.
- **Reliability:** Emergency and fire rescue services have extremely demanding availability requirements. In case of malfunctions or failures on a given FN, nearby FNs must be able to allocate the necessary resources to keep the provisioned Smart City services running properly. The hierarchical nature of Fog Computing architectures can improve the networks' reliability by allowing distributed and local decision making in terms of resource provisioning.

- **Mobility:** Several IoT use cases have demanding mobility requirements. For instance, consider a connected waste management fleet, where trucks are continuously moving through the City. Messages must be sent to trucks alerting for possible accidents or roadblocks that may occur on their predefined route. However, due to interference, network overload or even dead coverage spots the connectivity between the FN and the truck may be lost. Therefore, FNs must work together to find the best solution for the allocation of each service instance being requested by a moving device to ensure adequate service operation at all times. High mobility services require the deployment of Fog Computing technologies since centralized management approaches cannot fully satisfy the dynamic demands of these type of services. Thus, Fog Computing is essential to rapidly modify the allocation of services according to highly variable demand patterns.

- **Decentralized Management:** The available computing resources must be distributed towards the edge of the network closer to end devices and users [29]. The so named FNs provide local operations towards improving the response time in terms of resource allocation by efficiently scheduling all the necessary operations in the provisioning workflow. FNs should be aware of the network status and possible anomalies and malfunctions that may occur to react accordingly and keep all the services operating properly. Fog Computing brings intelligence and processing power close to end devices, which increases the networks' robustness and reliability.

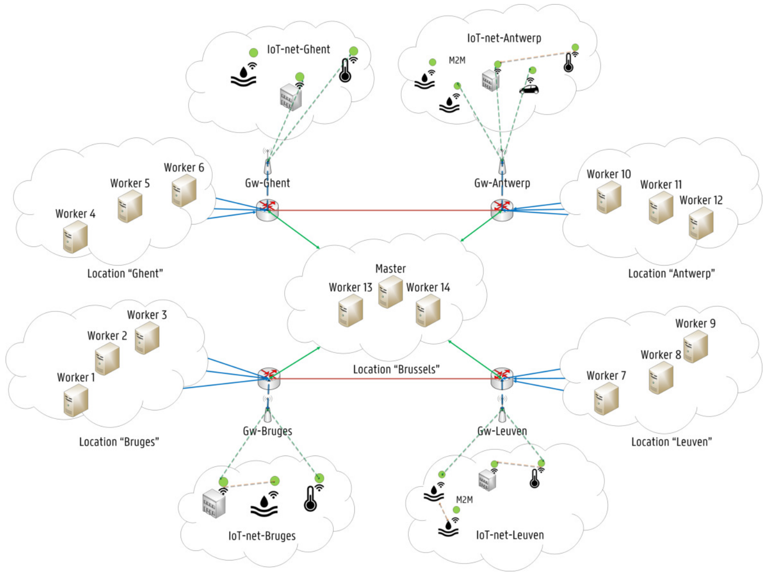
- **Scalability:** Fog Computing has to accommodate different IoT use cases and must possess adequate capacity to deal with growing service demands. IoT services must run without disruption and support millions of devices. Fog Computing architectures must be designed with scalability constraints in mind. FNs require modular software platforms where updates and modifications can be easily made without service interruptions. As network demands increase, FNs can receive additional software modules providing functionalities to deal with the growing service usage.

4.4 Fog-Based Kubernetes Architecture for Smart City Deployments

This section introduces a Fog Computing architecture based on the Kubernetes platform. First, a system overview of the proposed architecture is detailed, followed by the presentation of its main concepts. Then, the scheduling feature of Kubernetes is discussed.

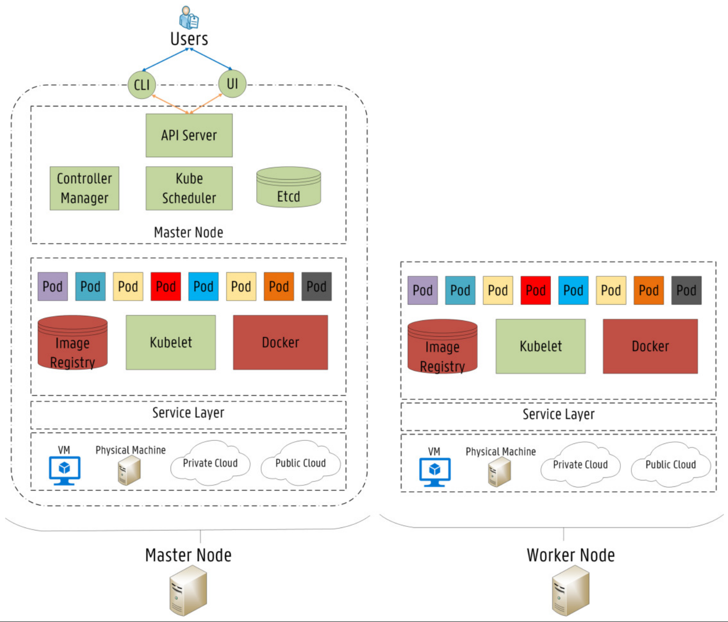
4.4.1 Kubernetes: Empowering Self-Driving Orchestration of Smart City Container-Based Applications

Figure 4.2 High-level view of the proposed Fog Computing infrastructure based on the Kubernetes platform.



The concept of Self-driving Orchestration has been introduced in [30] where it has been used to describe networks capable to measure, analyze and control themselves in an automated manner when reacting to changes in their environment. Kubernetes open source community is working towards a complete self-driving platform, aiming to simplify management and orchestration of scalable distributed systems across a wide range of environments and cloud providers for containerized applications. Kubernetes already provides orchestration features, which can be used to build reliable distributed systems with a high degree of decentralization in terms of the service life-cycle management, which is needed to fully leverage on Fog Computing architectures [31]. The proposed Fog-based Kubernetes architecture is shown in Figure 4.2. Several IoT networks are connected through wireless gateways to each of the represented locations. The architecture follows the master-slave model, where at least one master node manages Docker [32] containers across multiple worker nodes (slaves). End devices such as sensors are considered neither as master nor worker nodes. The proposed architecture follows the FN approach, where each FN is considered as a small cloud entity. The detailed architecture of the master and the slave nodes is shown in Figure 4.3. Nodes can be local physical servers and VMs or even public and private clouds. The Master is responsible for exposing the Application Program Interface (API) server, the scheduling of service deployments and managing the overall cluster life-cycle. Users interact with Kubernetes by communicating with the API server, which provides an entry point to control the entire cluster. Users can also send commands to the API server through the built-in Kubernetes Command Line Interface (CLI), known as Kubectl or even by accessing a web-based Kubernetes User Interface (UI). Another fundamental component is Etcd. Etcd is a lightweight key-value pair distributed data storage. Namespaces, scheduled jobs, deployed micro-services are examples of data stored in Etcd allowing other components to synchronize themselves based on the desired state of the cluster [33]. Furthermore, the main contact point for each cluster node is a service named Kubelet. Kubelet is responsible for recognizing discrepancies between the desired state and the actual state of the cluster. When this happens, Kubelet launches or terminates the necessary containers to reach the desired state described by the API server. Then, the Controller Manager is responsible for monitoring Etcd and the overall state of the cluster. If the state changes, the desired modifications are communicated through the API server. The Controller Manager is also responsible for the overall control of the runtime environment, including the creation and termination of containers.

Figure 4.3 Detailed Architecture of the Master and the Worker Node in the Kubernetes Cluster [15].



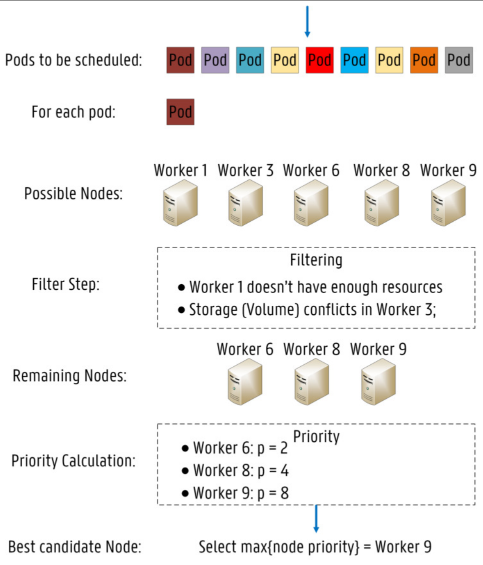
Although Kubernetes makes use of containers as the underlying mechanism to deploy micro-services, additional layers of abstraction exist over the container runtime environment to enable scalable life-cycle orchestration features. In Kubernetes, micro-services are often tightly coupled together forming a group of containers. This is the smallest working unit in Kubernetes, which is named a pod [12]. A pod represents the collection of containers and storage (volumes) running in the same execution environment. The containers inside a pod share the same IP Address, volumes and port space (namespace), while containers in different pods are isolated from one another, since they own different IP addresses and different hostnames. The main limitation is that two services listening on the same port cannot be deployed inside the same pod. Based on the available resources, the component that actually assigns pods to specific nodes in the cluster is named Kube–Scheduler (KS). The KS is the default scheduling feature in the Kubernetes platform, which is responsible for monitoring the available resources in the infrastructure and deciding on which adequate nodes pods should be placed. The selected node then pulls the required container images from the Image Registry and coordinates the necessary operations to launch the pod. The KS mechanisms are further detailed in the next section.

4.4.2 Resource Scheduling in Kubernetes: The Kube–Scheduler (KS)

The KS decision making process is illustrated in Figure 4.4. Every pod requiring allocation is added to a waiting queue, which is continuously monitored by the KS. If a pod is added to the waiting queue, the KS searches for an adequate node for the provisioning based on a two step procedure. The first step is named node filtering, where the KS verifies which nodes are capable of running the pod by applying a set of filters, also known as predicates. The purpose of filtering is to solely consider nodes meeting all specific pod requirements further in the scheduling process. The second operation is named node priority calculation, where the KS ranks each remaining node to find the best fit for the pod provisioning based on one or more scheduling algorithms, also named priorities. The KS supports the following predicates [15, 33]:

- **Check Node Memory Pressure:** This predicate checks if a pod can be allocated on a node reporting memory pressure condition. Currently, Best Effort pods should not be placed on nodes under memory pressure, since they are automatically deassigned from the node.
- **Check Node Disk Pressure:** This predicate evaluates if a pod can be scheduled on a node reporting disk pressure condition. Pods can currently not be deployed on nodes under disk pressure, since they are automatically deassigned.
- **Host Name:** This predicate filters out all nodes, except the one specified in the Spec's NodeName field of the pod configuration file.

Figure 4.4 Sample of detailed scheduling operations of the Kube–Scheduler.



- **Match Node Selector (Affinity/Anti-Affinity):** By using node selectors (labels), it is possible to define that a given pod can only run on a particular set of nodes with an exact label value (node-affinity), or even that a pod should avoid being allocated on a node that has already certain pods deployed (pod-anti-affinity). These rules can be created by declaring Tolerations in the pod configuration files to match specific node Taints. Essentially, affinity rules are properties of pods that attract them to a set of nodes or pods, while taints allow nodes to repel a given set of pods. Taints and tolerations ensure that pods are not deployed onto inappropriate nodes. Both are important mechanisms to fine-tune the scheduling behavior of Kubernetes. Node selectors provide a flexible set of rules, on which the KS bases its scheduling decision by filtering specific nodes (node affinity/anti-affinity), by preferring to deploy certain pods close or even far away from other pods (pod affinity/anti-affinity), or just on node labels favored by the pod (taints and tolerations).

- **No Disk Conflict:** This predicate evaluates if a pod can fit due to the storage (volume) it requests, and those that are already mounted.

- **No Volume Zone Conflict:** This predicate checks if the volumes a pod requests are available through a given node due to possible zone restrictions.

- **Pod Fits Host Ports:** For instance, if the pod requires to bind to the host port 80, but another pod is already using that port on the node, this node will not be a possible candidate to run the pod and, therefore, it will be disqualified.

- **Pod Fits Resources:** If the free amount of resources (CPU and memory) on a given node is smaller than the one required by the pod, the node must not be further considered in the scheduling process. Therefore, the node is disqualified.

The KS knows in advance which nodes are not suitable for the pod deployment by applying these predicates. Inadequate nodes are removed from the list of possible candidates. On one hand, after completion of the filtering process, finding no capable nodes for the pod deployment is always a possibility. In that case, the pod remains unscheduled and the KS triggers an event stating the reason for the failed deployment. On the other hand, if several candidates are retrieved after completion of the filtering operation, the KS triggers the node priority calculation. The node priority calculation is based on a set of priorities, where each remaining node is given a score between 0 and 10, 10 representing "perfect fit" and 0 meaning "worst fit". Then, each priority is weighted by a positive number, depending on the importance of each algorithm, and the final score of each node is calculated by adding up all the weighted scores [33]. The highest scoring node is selected to run the pod. If more than one node is classified as the highest scoring node, then one of them is randomly chosen. The KS supports the following priorities [15]:

- **Balanced Resource Allocation:** This priority function ranks nodes based on the cluster CPU and Memory usage rate. The purpose is to balance the resource allocation after the pod provisioning.
- **Calculate AntiAffinity Priority:** This priority function scores nodes based on anti-affinity rules. For instance, spreading pods in the cluster by reducing the same number of pods belonging to the same service on nodes with a particular label.
- **Inter Pod Affinity Priority:** This priority algorithm ranks nodes based on pod affinity rules. For example, nodes with certain pods already allocated are scored higher, since it is preferred to deploy the given pod close to these pods.
- **Image Locality Priority:** Remaining nodes are ranked according to the location of the requested pod container images. Nodes already having the requested containers installed are scored higher.
- **Least Requested Priority:** The node is ranked according to the fraction of CPU and memory (free/allocated). The node with the highest free fraction is the most preferred for the deployment. This priority function spreads the pods across the cluster based on resource consumption.

- **Most Requested Priority:** This priority algorithm is the opposite of the one above. The node with the highest allocated fraction of CPU and memory is the most preferred for the deployment.
- **Node Affinity Priority:** In this case, nodes are scored according to node-affinity rules. For instance, nodes with a certain label are ranked higher than others.
- **Selector Spread Priority:** This priority algorithm tries to minimize the number of deployed pods belonging to the same service on the same node or on the same zone/rack.
- **Taint Toleration Priority:** This priority function scores nodes based on their taints and the correspondent tolerations declared in the pod configuration file. Remaining nodes are preferred according to the number of intolerable taints on them for the given pod. An intolerable taint is specified by the "Prefer No Schedule" key.

Predicates are evaluated to dismiss nodes that are incapable of running the given pod while priorities are designed to score all the remaining nodes that can deploy the pod. For example, a given node would be scored lower for the Selector Spread Priority if an instance of the requested pod is already allocated on that node. However, if a pod affinity rule is specified in the pod configuration file for the service, the node would be scored higher for the Inter Pod Affinity Priority since it is preferred to deploy the given pods close to each other. Furthermore, if a given pod requires a core CPU (1.0), the Pod Fits Resources predicate returns "False" for a node that only has 800 millicpu free. Additionally, for the same pod, the Most Requested Priority ranks a node that has only 200 millicpu free higher than one with 3.5 cores CPU left, even though both nodes can accommodate the pod (assuming they have the same CPU capacity). It should be noted that the KS searches for a suitable node for each pod, one at a time. The KS does not take the remaining pods waiting for deployment into account in the scheduling process. When the allocation decision is made, the KS informs the API server indicating where the pod must be scheduled. This operation is named Binding.
Another aspect worth mentioning of the Kubernetes provisioning life-cycle is named resource requests and limits. Developers can specify resource requests and limits on the pod configuration files. A resource request is the minimum amount of resources (CPU and/or memory) required by all containers in the pod while a resource limit is the maximum amount of resources that can be allocated for the containers in a pod. Pods can be categorized in three QoS classes depending on resource requests and limits:

- **Best Effort (lowest priority):** A Best Effort pod has neither resource requests or limits on its configuration files for each of its containers. These pods are the first ones to be terminated in case the system runs out of memory.

- **Burstable:** A Burstable pod has all containers with resource requests lower than their resource limits. If a container needs more resources than the ones requested, the container can use them as long as they are free.
- **Guaranteed (highest priority):** A guaranteed pod has resource requests for all its containers equal to the maximum resource needs that the system will allow the container to use (resource limit).

If resource requests are specified, the KS can provide better allocation decisions. Similarly, if resource limits are described, resource contention can be handled properly [34]. When several containers are running on the same node, they compete for the available resources. Since container abstraction provides less isolation than VMs, sharing physical resources might lead to a performance degradation named resource contention. Resource requests and limits enable Kubernetes to properly manage the allocation of resources. Nevertheless, developers still need to accurately set up these requests and limitations, because containers often do not use the entire amount of resources requested which could lead to wasted resources. For example, two pods have been deployed and each one is requesting 4 Gb of RAM in a node with 8GB RAM capacity, but each pod is only using 1 GB of RAM. The KS could allocate more pods onto that node, however, due to the incorrect specification in terms of resource requests, the KS will never schedule additional pods onto that node.

4.5 Resource Scheduling extension in Kubernetes

This section introduces the proposed extensions to the default scheduling mechanism available in Kubernetes. First, a network-aware scheduling approach is detailed. Then, the ILP formulation implemented as a container-based application is discussed.

4.5.1 Network-Aware Scheduler (NAS) implementation in Kubernetes

Although the KS provides flexible and powerful features, the metrics applied in the decision making process are rather limited. Only CPU and RAM usage rates are considered in the service scheduling while latency or bandwidth usage rates are not considered at all. A suitable scheduling approach for Fog Computing environments must consider multiple factors, such as the applications' specific requirements (CPU, memory, minimum bandwidth), the state of the infrastructure (hardware and software), the network status (link bandwidth and latency), among others. Therefore, this chapter presents a Network-Aware Scheduler (NAS) extension to Kubernetes, which enables Kubernetes to make scheduling decisions based on up-to-date information about the current status of the network infrastructure. Kubernetes describes three ways of extending the KS:

- Adding new predicates and/or priorities to the KS and recompiling it.
- Implementing a specific scheduler process that can run instead of or alongside the KS.
- Implementing a "scheduler extender" process that the default KS calls out as a final step when making scheduling decisions.

The third approach is particularly suitable for use cases where scheduling decisions need to be made on resources not directly managed by the standard KS. The proposed NAS has been implemented based on this third approach, since information on the current status of the network infrastructure is not available throughout the scheduling process of the KS. The proposed NAS has been implemented in Go and deployed in the Kubernetes cluster as a pod. The pod architecture of the NAS is illustrated in Figure 4.5. Additionally, the pod configuration file for the NAS is shown in Figure 4.6a, while the scheduling policy configuration file for the NAS is presented in Figure 4.6b. As shown, the pod is composed of two containers: the extender and the NAS. The extender is responsible for performing the proposed scheduling operation, while the NAS is in fact the actual KS. A specific scheduler policy configuration file has to be defined to instruct the KS how to reach the extender and which predicates should be used to filter the nodes as a first step in the scheduling process. Essentially, when the KS tries to schedule a pod, the extender call allows an external process to filter the remaining nodes (second step). The arguments passed on to the "Filter Verb" endpoint consists of the set of nodes filtered through the KS predicates and the given pod. This second step is used to further refine the list of possible nodes.

Figure 4.5 The detailed Pod architecture of the Network-Aware Scheduler (NAS).

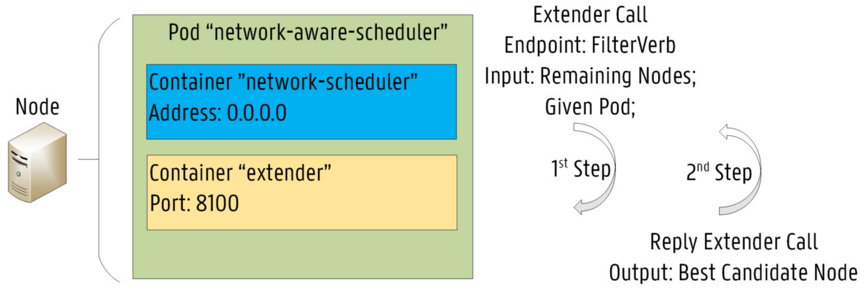
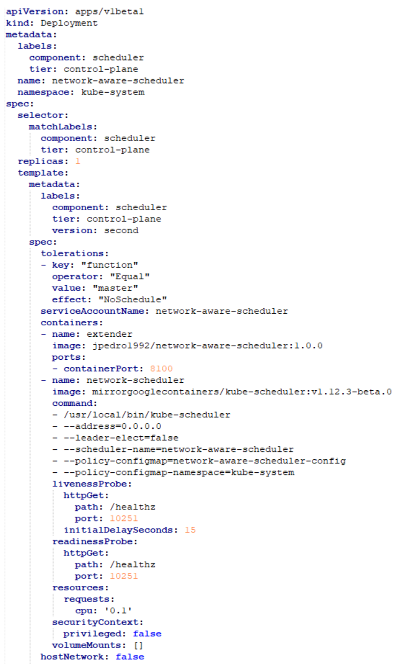


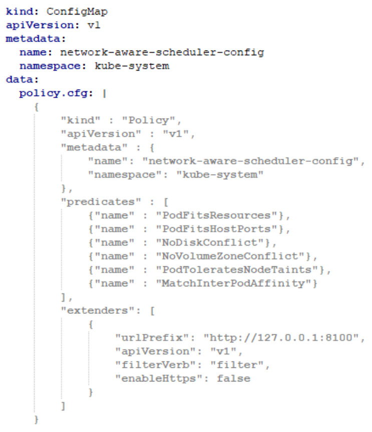
Figure 4.6 The configuration files required for the NAS.

(a) The pod deployment configuration file.



```yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  labels:
    component: scheduler
    tier: control-plane
  name: network-aware-scheduler
  namespace: kube-system
spec:
  selector:
    matchLabels:
      component: scheduler
      tier: control-plane
  replicas: 1
  template:
    metadata:
      labels:
        component: scheduler
        tier: control-plane
        version: second
    spec:
      tolerations:
      - key: "function"
        operator: "Equal"
        value: "master"
        effect: "NoSchedule"
      serviceAccountName: network-aware-scheduler
      containers:
      - name: extender
        image: jpedro1992/network-aware-scheduler:1.0.0
        ports:
        - containerPort: 8100
      - name: network-scheduler
        image: mirrorgooglecontainers/kube-scheduler:v1.12.3-beta.0
        command:
        - /usr/local/bin/kube-scheduler
        - --address=0.0.0.0
        - --leader-elect=false
        - --scheduler-name=network-aware-scheduler
        - --policy-configmap=network-aware-scheduler-config
        - --policy-configmap-namespace=kube-system
        livenessProbe:
          httpGet:
            path: /healthz
            port: 10251
          initialDelaySeconds: 15
        readinessProbe:
          httpGet:
            path: /healthz
            port: 10251
        resources:
          requests:
            cpu: '0.1'
        securityContext:
          privileged: false
        volumeMounts: []
      hostNetwork: false
```

(b) The scheduling policy configuration file.



```yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: network-aware-scheduler-config
  namespace: kube-system
data:
  policy.cfg: |
    {
        "kind" : "Policy",
        "apiVersion" : "v1",
        "metadata" : {
            "name": "network-aware-scheduler-config",
            "namespace": "kube-system"
        },
        "predicates" : [
            {"name" : "PodFitsResources"},
            {"name" : "PodFitsHostPorts"},
            {"name" : "NoDiskConflict"},
            {"name" : "NoVolumeZoneConflict"},
            {"name" : "PodToleratesNodeTaints"},
            {"name" : "MatchInterPodAffinity"}
        ],
        "extenders": [
            {
                "urlPrefix": "http://127.0.0.1:8100",
                "apiVersion": "v1",
                "filterVerb": "filter",
                "enableHttps": false
            }
        ]
    }
```

A complete labeling of the Fog Computing infrastructure previously shown has been conducted based on Affinity/Anti-Affinity rules and node labels mentioned in Section 4.4.2. As illustrated, the infrastructure is composed of a Kubernetes cluster with 15 nodes (1 master node and 14 worker nodes). Nodes have been classified with labels “Min, Med, High” for keywords “CPU, RAM”, depending on their resource capacity. Additionally, nodes have been classified in terms of device type, by classifying them with taints “Cloud, Fog” for the keyword “Device Type” and according to their geographic distribution. Round Trip Time (RTT) values have been assigned to each node as a label so that delay constraints can be considered in the scheduling process. The labels of each node are listed in Table 4.1. These labels enable the placement of services in specific zones or certain nodes based on the location delay. All these rules are important to fine-tune the scheduling behavior of Kubernetes, in particular, to help the scheduler make more informed decisions at the filtering step by removing inappropriate nodes.

The proposed NAS makes use of these strategically placed RTT labels to decide where it is suitable to deploy a specific service based on the target location specified in the pod configuration file. In fact, the node selection is based on the minimization of the RTT depending on the target location for the service after the completion of the filtering step. Additionally, in terms of bandwidth, NAS checks if the best candidate node has enough bandwidth to support the given service based on the pod bandwidth requirement. If the bandwidth request is not specified in the pod configuration file, a default value of 250 Kbit/s is considered during the scheduling phase. After completion of the scheduling request, the available bandwidth is updated on the corresponding node label. The NAS Algorithm is shown in Algorithm 1.

In summary, the proposed NAS approach filters the infrastructure nodes based on KS predicates and then makes use of the implemented RTT location labels to choose the best candidate node from the filtered ones to the desired service location.

4.5.2 From Theory to Practice: ILP model implementation in Kubernetes as a Container-based application

Lastly, an ILP model for the IoT service placement problem has been designed as a container-based application. The ILP model has been implemented in Java using the IBM ILOG CPLEX ILP solver [35] and the Spring Framework [36]. The class diagram of the implementation is shown in Figure 4.7. The class diagram has been generated with IntelliJ IDEA, a Java Integrated Development Environment (IDE) developed by Jetbrains [37]. The proposed ILP container application has been designed as a Representational State Transfer (REST) API. Simulation entities can be created or deleted through the Simulation Controller. ILP solutions

Table 4.1: The implemented node labels in the Kubernetes cluster.

Node	Device Type	CPU	RAM	Bandwidth	RTT Ghent	RTT Antwerp	RTT Bruges	RTT Leuven	RTT Brussels
Master	Cloud	High	High	10.0 Mbit/s	32.0 ms	32.0 ms	32.0 ms	32.0 ms	4.0 ms
Worker 1	Fog	Min	Min	10.0 Mbit/s	64.0 ms	64.0 ms	4.0 ms	14.0 ms	32.0 ms
Worker 2	Fog	Med	Med	10.0 Mbit/s	64.0 ms	64.0 ms	4.0 ms	14.0 ms	32.0 ms
Worker 3	Fog	Min	Min	10.0 Mbit/s	64.0 ms	64.0 ms	4.0 ms	14.0 ms	32.0 ms
Worker 4	Fog	Min	Min	10.0 Mbit/s	4.0 ms	14.0 ms	64.0 ms	64.0 ms	32.0 ms
Worker 5	Fog	Med	Med	10.0 Mbit/s	4.0 ms	14.0 ms	64.0 ms	64.0 ms	32.0 ms
Worker 6	Fog	Med	Med	10.0 Mbit/s	4.0 ms	14.0 ms	64.0 ms	64.0 ms	32.0 ms
Worker 7	Fog	Min	Min	10.0 Mbit/s	64.0 ms	64.0 ms	14.0 ms	4.0 ms	32.0 ms
Worker 8	Fog	Med	Med	10.0 Mbit/s	64.0 ms	64.0 ms	14.0 ms	4.0 ms	32.0 ms
Worker 9	Fog	Min	Min	10.0 Mbit/s	64.0 ms	64.0 ms	14.0 ms	4.0 ms	32.0 ms
Worker 10	Fog	Med	Med	10.0 Mbit/s	14.0 ms	4.0 ms	64.0 ms	64.0 ms	32.0 ms
Worker 11	Fog	Med	Med	10.0 Mbit/s	14.0 ms	4.0 ms	64.0 ms	64.0 ms	32.0 ms
Worker 12	Fog	Min	Min	10.0 Mbit/s	14.0 ms	4.0 ms	64.0 ms	64.0 ms	32.0 ms
Worker 13	Cloud	Min	Min	10.0 Mbit/s	32.0 ms	32.0 ms	32.0 ms	32.0 ms	4.0 ms
Worker 14	Cloud	Med	Med	10.0 Mbit/s	32.0 ms	32.0 ms	32.0 ms	32.0 ms	4.0 ms

Algorithm 1 NAS Algorithm

Input: Remaining Nodes after Filtering Process in

Output: Node for the service placement out

```
 1: handler(http.Request){ //Handle a provisioning request
 2:     $receivedNodes = decode(http.Request)$;
 3:     $receivedPod = decodePod(http.Request)$;
 4:     $node$ = selectNode(receivedNodes, receivedPod);
 5:     return $node$
 6: }
 7: //Return the best candidate Node (recursive)
 8: selectNode(receivedNodes, receivedPod){
 9:     $targetLocation = getLocation(receivedPod)$;
10:     $minBandwidth = getBandwidth(receivedPod)$;
11:     $min = math.MaxFloat64$;
12:     $copyReceivedNodes = receivedNodes$;
13:     for $node$ in range $receivedNodes$ //find min RTT
14:         $rtt = getRTT(node, targetLocation)$;
15:         $min = math.Min(min, rtt)$;
16:     //find best Node based on RTT and minBandwidth
17:     for $node$ in range $receivedNodes$
18:         if $min == getRTT(node, targetLocation)$
19:             if $minBandwidth \leq getAvBandwidth(node)$
20:                 return $node$;
21:             else
22:                 $copyReceivedNodes =$
23:                     $= removeNode(copyReceivedNodes, node)$;
24:     // Available min RTT Nodes are full in terms of Network Bandwidth!
25:     // Repeat the Process (Recursive)!
26:     // First: Check if copy is not empty
27:     if $copyReceivedNodes == null$
28:         return $null$, $Error$("No suitable nodes found!");
29:     else return selectNode(copyReceivedNodes, receivedPod);
30: }
```

can be obtained by issuing a GET Request for all Simulation entities available or by just sending a GET request for a specific Simulation entity. The ILP formulation incorporates multiple optimization objectives. Users can specify the desired optimization objectives and the amount of requests for each specific application when creating Simulation entities through a PUT request. In fact, the model is executed iteratively so that in each iteration a different optimization objective can be considered. To retain the solutions obtained in previous iterations, additional constraints are added to the model. Thus, the solution space continuously decreases since iterations must satisfy the previous optimal solutions. Every iteration refines the previously obtained solution by improving the model with an additional optimization objective.

The main advantage of ILP is the flexibility to analyze complex problems such as the resource provisioning in Fog Computing presented in this paper. However, theoretical studies lack practical implementations, which limit their applicability to real deployments. Therefore, the proposed ILP REST API has been deployed and validated on the Kubernetes platform showing the full applicability of theoretical approaches in practical service deployments. In Figure 4.8, the proposed service architecture is shown. Two YAML Ain't Markup Language (YAML) files are used to deploy the ILP REST API. Firstly, the ilp-rest-api.yaml is responsible for creating the deployment of the ILP REST API. The deployment is composed of three replicated ilp-rest-api Pods, indicated by the replicas field. This is the desired number of replicas. Additionally, for each pod, a core CPU (1.0) and 2 Gb of RAM are requested (resource requests), which can be increased to four cores and 8 Gb, respectively (resource limits). The service is listening on port 8080. Secondly, the svc-ilp-rest-api.yaml creates a Kubernetes Service named svc-ilp-rest-api. A Kubernetes Service is a flexible abstraction that provides a reliable manner to access a logical set of pods. The set of pods exposed are determined by a label selector, which in this case, corresponds to *"app: ilp-rest-api"*. Services make pods consistently accessible. Pods can be created, updated or even terminated so that the service will know exactly how many replicas are running, where pods are located and which IP addresses are being used. Essentially, services enable automatic load-balancing across several pods. The ClusterIP service type (default) has been used to provide internal access to the ilp-rest-api by exposing it on an internal IP in the cluster. Thus, the ilp-rest-api service is only reachable from within the cluster.

The proposed ILP REST API has been evaluated on the Kubernetes platform to compare the performance of the theoretical formulation with the implemented NAS approach and the standard scheduling feature available in Kubernetes. The evaluation use case is presented next.

Figure 4.7 The class diagram of the ILP REST API generated with IntelliJ IDEA.

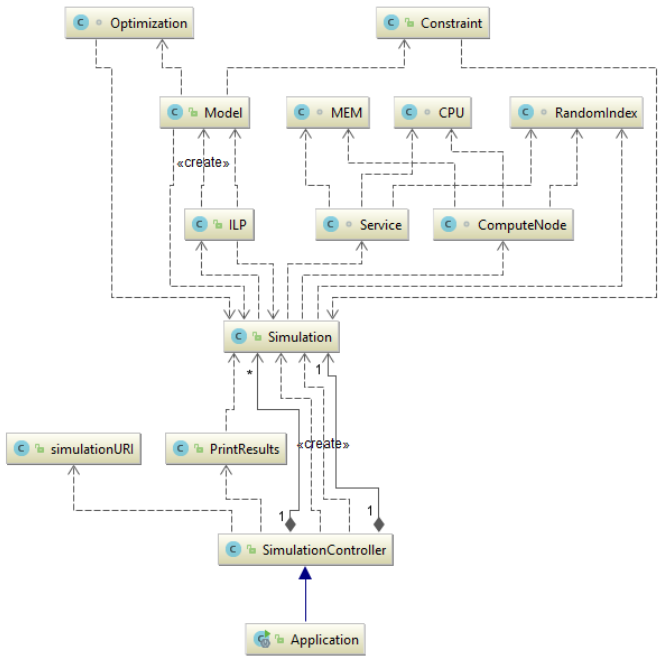
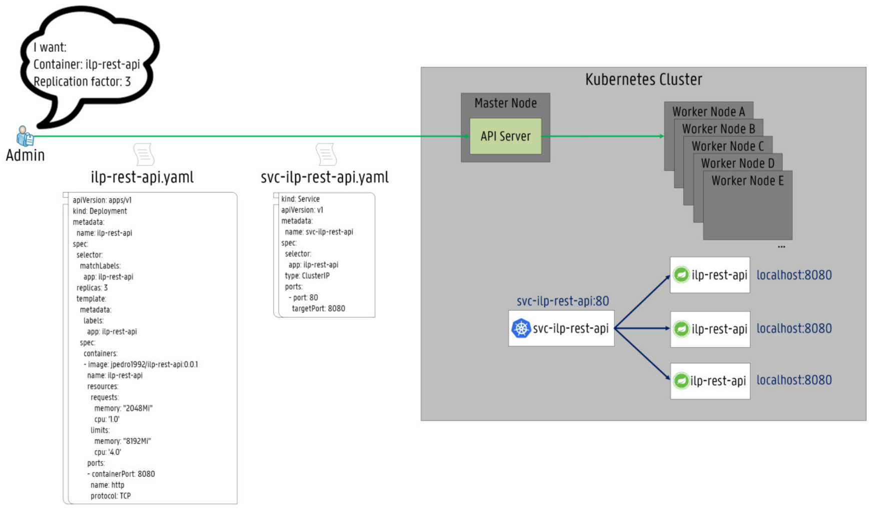


Figure 4.8 The detailed service scheme of the ILP REST API in the Kubernetes platform.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ilp-rest-api
spec:
  selector:
    matchLabels:
      app: ilp-rest-api
  replicas: 3
  template:
    metadata:
      labels:
        app: ilp-rest-api
    spec:
      containers:
      - image: jpedro1992/ilp-rest-api:0.0.1
        name: ilp-rest-api
        resources:
          requests:
            memory: "2048Mi"
            cpu: '1.0'
          limits:
            memory: "8192Mi"
            cpu: '4.0'
        ports:
        - containerPort: 8080
          name: http
          protocol: TCP
```

```
kind: Service
apiVersion: v1
metadata:
  name: svc-ilp-rest-api
spec:
  selector:
    app: ilp-rest-api
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 8080
```

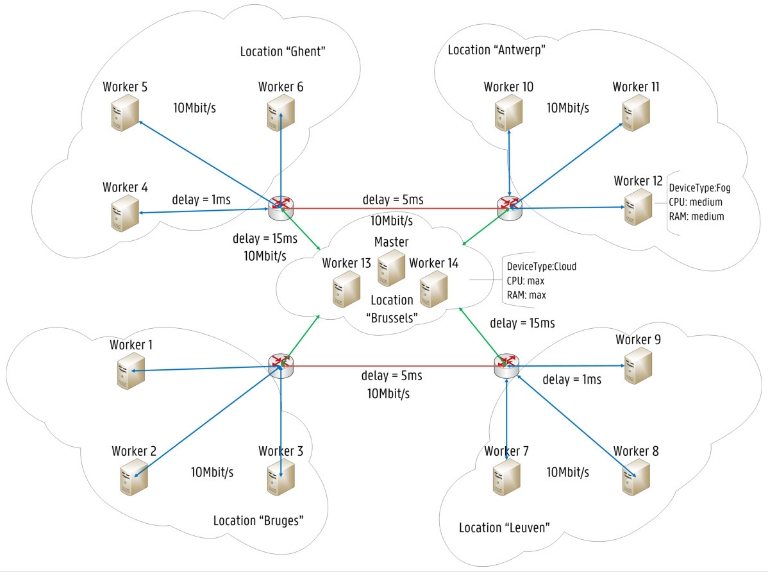
4.6 Evaluation Use Case

In this section, the evaluation setup is detailed. Then, the evaluation scenario is described.

4.6.1 Evaluation Setup

The Kubernetes cluster has been set up on the imec Virtual Wall infrastructure [38] at IDLab, Belgium. The Fog Computing infrastructure illustrated in Figure 4.9 has been implemented with Kubeadm [39]. The cluster node hardware configurations are shown in Table 4.2. Furthermore, the software versions used to implement the Kubernetes cluster are listed in Table 4.3.

Figure 4.9 A Fog Computing infrastructure based on the Kubernetes platform.



4.6.2 Scenario Description: Air Monitoring Service

The evaluation use case is based on an Air Monitoring Service performing unsupervised anomaly detection. This scenario has been previously presented in [40], where a novel anomaly detection solution has been proposed for Smart City applications in Smart Cities based on the advantages of Fog Computing architectures. The purpose of this use case is to collect air quality data in the City of Antwerp to detect high amounts of organic compounds in the atmosphere based on outlier detection and clustering algorithms. Clustering allows the detection of patterns in unlabeled data while outlier detection is related to the identification of unusual

Table 4.2: The Hardware Configuration of each Cluster Node.

Node	CPU	RAM
Worker 1	2x Quad core Intel E5520 (2.2 GHz)	12 Gb
Worker 2	1x 4 core E3-1220v3 (3.1 GHz)	16 Gb
Worker 3	2x Quad core Intel E5520 (2.2 GHz)	12 Gb
Worker 4	2x Quad core Intel E5520 (2.2 GHz)	12 Gb
Worker 5	2x Hexacore Intel E5620 (2.4 GHz)	24 Gb
Worker 6	2x Hexacore Intel E5620 (2.4 GHz)	24 Gb
Worker 7	2x Dual core AMD opteron 2212 (2.0 GHz)	8Gb
Worker 8	2x Hexacore Intel E5620 (2.4 GHz)	24 Gb
Worker 9	2x Dual core AMD opteron 2212 (2.0 GHz)	8Gb
Worker 10	2x Hexacore Intel E5620 (2.4 GHz)	24 Gb
Worker 11	2x Hexacore Intel E5620 (2.4 GHz)	24 Gb
Worker 12	2x Quad core Intel E5520 (2.2 GHz)	12 Gb
Worker 13	2x Dual core AMD opteron 2212 (2.0 GHz)	8Gb
Worker 14	2x Hexacore Intel E5620 (2.4 GHz)	24 Gb
Master	2x 8core Intel E5-2650v2 (2.6 GHz)	48 Gb

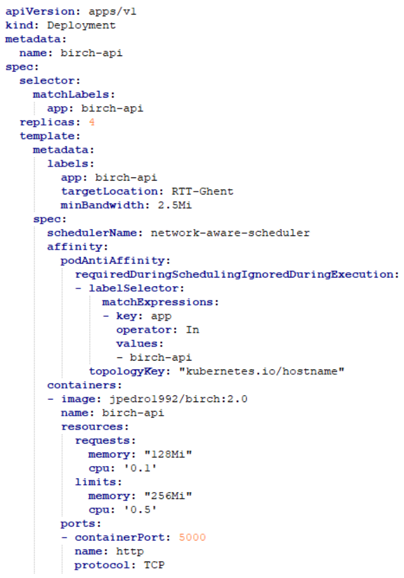
Table 4.3: Software Versions of the Evaluation Setup.

Software	Version
Kubeadm	v1.13.0
Kubectl	v1.13.0
Go	go1.11.2
Docker	docker://17.3.2
Linux Kernel	4.4.0-34-generic
Operating System	Ubuntu 16.04.1 LTS

data samples when compared to the rest of the dataset. In this chapter, the anomaly detection algorithms have been implemented as container APIs and then deployed as pods in the Kubernetes cluster. Regarding clustering, the Birch and the Kmeans algorithms have been evaluated while for outlier detection, the Robust Covariance and the Isolation Forrest have been assessed. The deployment properties of each service are shown in Table 4.4. In Figure 4.10, the pod configuration files for the deployment of the Birch service are presented. As shown, the service chain of the Birch service is composed of two pods, the API and the corresponding database. The desired location for the allocation of the service is expressed by the “targetLocation" label. Furthermore, the minimum required bandwidth per service is expressed by the “minBandwidth" label. As illustrated previously, the available bandwidth per node is 10.0 Mbit/s. Additionally, pod anti-affinity rules have been added to each service so that pods belonging to the same service chain are not deployed together, meaning that a node can only allocate one instance of a certain pod for a particular service. For instance, for the Birch service, the birch-api and the birch-cassandra pods cannot be deployed together. All pods have also been categorized as Burstable, since their containers have resource requests lower than their resource limits. The deployment of these services has been performed to compare the performance of the implemented approaches with the default KS.

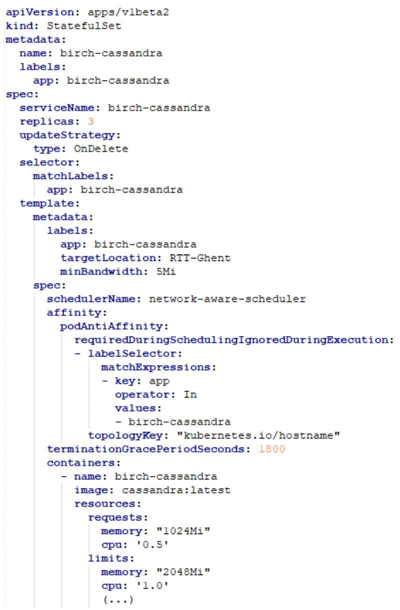
Figure 4.10 The pod configuration files for the Birch Service

(a) birch-api service.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: birch-api
spec:
  selector:
    matchLabels:
      app: birch-api
  replicas: 4
  template:
    metadata:
      labels:
        app: birch-api
        targetLocation: RTT-Ghent
        minBandwidth: 2.5Mi
    spec:
      schedulerName: network-aware-scheduler
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - birch-api
            topologyKey: "kubernetes.io/hostname"
      containers:
      - image: jpedro1992/birch:2.0
        name: birch-api
        resources:
          requests:
            memory: "128Mi"
            cpu: '0.1'
          limits:
            memory: "256Mi"
            cpu: '0.5'
        ports:
        - containerPort: 5000
          name: http
          protocol: TCP
```

(b) birch-cassandra service.



```yaml
apiVersion: apps/v1beta2
kind: StatefulSet
metadata:
  name: birch-cassandra
  labels:
    app: birch-cassandra
spec:
  serviceName: birch-cassandra
  replicas: 3
  updateStrategy:
    type: OnDelete
  selector:
    matchLabels:
      app: birch-cassandra
  template:
    metadata:
      labels:
        app: birch-cassandra
        targetLocation: RTT-Ghent
        minBandwidth: 5Mi
    spec:
      schedulerName: network-aware-scheduler
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - birch-cassandra
            topologyKey: "kubernetes.io/hostname"
      terminationGracePeriodSeconds: 1800
      containers:
        - name: birch-cassandra
          image: cassandra:latest
          resources:
            requests:
              memory: "1024Mi"
              cpu: '0.5'
            limits:
              memory: "2048Mi"
              cpu: '1.0'
              (...)
```

Table 4.4: Deployment properties of each service.

Service Name	Pod Name	CPU Req/Lim (m)	RAM Req/Lim (Mi)	Min. Bandwidth (Mbit/s)	Rep. Factor	Target Location	Dependencies
Birch	birch-api	100/500	128/256	2.5	4	Ghent	birch-cassandra
	birch-cassandra	500/1000	1024/2048	5			birch-api
Robust	robust-api	200/500	256/512	2	4	Antwerp	robust-cassandra
	robust-cassandra	500/1000	1024/2048	5			robust-api
Kmeans	kmeans-api	100/500	128/256	2.5	2	Bruges	kmeans-cassandra
	kmeans-cassandra	500/1000	1024/2048	5			kmeans-api
Isolation	isolation-api	200/500	256/512	1	2	Leuven	isolation-cassandra
	isolation-cassandra	500/1000	1024/2048	5			isolation-api

4.6.3 ILP model configurations

In Table 4.5, the evaluated ILP model configurations are shown. First, for all model configurations, the number of accepted service requests is maximized in the first iteration. Then, on one hand, the ILP-A configuration corresponds to the minimization of service latency based on the service target location. On the other hand, the ILP-B configuration is related to the infrastructure's energy efficiency, since the final goal of this configuration is to minimize the number of nodes used during the service provisioning. Finally, the ILP-C configuration corresponds to a joint optimization of latency and energy, where minimization of latency and the minimization of nodes correspond to the second and third iteration, respectively.

Table 4.5: The evaluated ILP model configurations.

	ILP configurations		
Iteration	ILP-A (Latency)	ILP-B (Energy)	ILP-C (Latency and Energy)
1st	MAX Requests	MAX Requests	MAX Requests
2nd	MIN Latency	MIN Nodes	MIN Latency
3rd	-	-	MIN Nodes

4.7 Evaluation Results

In this section, the evaluation results are detailed. First, the execution time of the different scheduling approaches is presented, followed by the correspondent scheduler resource consumption. Then, the allocation schemes for each of the schedulers is detailed. Finally, the average RTT per service and the expected service bandwidth per node for the different scheduling approaches are shown.

4.7.1 Scheduler Execution Time

In Table 4.6, the execution time of the different scheduler approaches is presented. Each evaluation run considered 24 pods as shown in Table 4.4 previously. The execution time has been evaluated 10 times. The scheduling decision of the default KS is made after on average 4.20 ms per pod, while the NAS requires on average 5.42 ms, due to the extender call procedure. The total execution time of the KS and the NAS is 126.08 ms and 162.74 ms, respectively. Additionally, the three ILP configurations previously presented have been requested to the ILP REST API. Firstly, the execution time of the ILP-A configuration is 1.82 s (first iteration: 0.86 s, second iteration: 0.96 s). Secondly, the execution time of the ILP-B configuration is 6.30 s (first iteration: 0.79 s, second iteration: 5.51 s). Thirdly, three objectives are considered in the ILP-C configuration. The ILP-C is a refined solution of the ILP-A configuration, since the minimization of the number of allocated nodes is considered as a third optimization objective, resulting in a higher execution time of 4.20 s (first iteration: 0.87 s, second iteration: 0.95 s, third iteration: 2.38 s). The higher execution time of the ILP-B configuration is due to the high potential solution space for minimizing energy in this scenario. Regarding the pod startup time, the KS and the NAS require on average 2 seconds to allocate and initialize the required containers while the ILP configurations need between 4 and 8 seconds due to the higher decision time. By comparing both the KS and the NAS with the three ILP formulations, it can be seen that heuristics can significantly reduce the execution time of ILP models.

Table 4.6: The execution time of the different schedulers.

Scheduler	Avg. Scheduling Decision (per pod)	Total Execution Time	Pod Startup Time
KS	4.20 ms	126.08 ms	2.04 s
NAS	5.42 ms	162.74 ms	2.13 s
ILP-A	-	1.82 s	3.97 s
ILP-B	-	6.30 s	8.45 s
ILP-C	-	4.20 s	6.35 s

4.7.2 Scheduler Resource Consumption

In Table 4.7, the resource consumption (CPU and RAM) of the different scheduler approaches is shown. As expected, the ILP REST API requires more resources than the other two scheduling mechanisms. Traditionally, ILP solvers need a high amount of resources and require high execution times to find optimal solutions to the given problem. Nevertheless, ILP techniques could improve the quality of the decision-making process by linearizing the problem and by only considering concrete objectives. The KS and the NAS have a similar resource consumption, since both schedulers are based on the default scheduling mechanism available in Kubernetes.

Table 4.7: The resource consumption of the different schedulers.

Scheduler	Used CPU (m)	Used RAM (Mi)
KS	102	41.93
NAS	102	56.67
ILP-A	233	452.36
ILP-B	639	630.06
ILP-C	438	636.71

4.7.3 Allocation Scheme

In Figure 4.11, the different allocation schemes for each of the schedulers are illustrated. As expected, the KS deployment scheme is not optimized for the service's desired location, since no considerations are made about network latency in its scheduling algorithm. For instance, the KS allocation scheme of the Isolation service is fairly poor since both pods, isolation-api and isolation-cassandra, are not deployed in the desired location (Leuven). Furthermore, the ILP-B configuration is also not optimized for service latency, since the objective of the ILP is to minimize the energy consumption by just considering bandwidth constraints between services allocated on different nodes.

4.7.4 Network Latency and Bandwidth

The differences in the average RTT per scheduler are detailed in Figure 4.12. As shown, the proposed NAS achieves significantly lower RTTs for each of the deployed services when compared with the default KS. Both NAS and ILP-A configuration achieve similar results in terms of the overall RTT. However, clear differences exist in the Birch and Robust service. RTT values of 6.5 ms and 23.0 ms are achieved with the NAS, while values of 16.0 ms and 13.5 ms are obtained with ILP-A. This difference occurs because the ILP takes all remaining pods waiting for deployment into account in the service provisioning, while the NAS searches for a suitable node for each pod, one at a time, similar to the KS. Therefore, the NAS optimizes first the birch-api and the robust-api services and just after their deployment, the correspondent birch-cassandra and robust-cassandra services are scheduled. The service provisioning in terms of network latency is highly improved with the NAS and the ILP-A configuration since KS and ILP-B do not consider bandwidth requests in the scheduling process. In this particular allocation scheme, the NAS improves the performance of the default KS by reducing the network latency by 70% while increasing the scheduling decision time by 1.22 ms per pod.

Figure 4.11 The service provisioning schemes of the different schedulers.

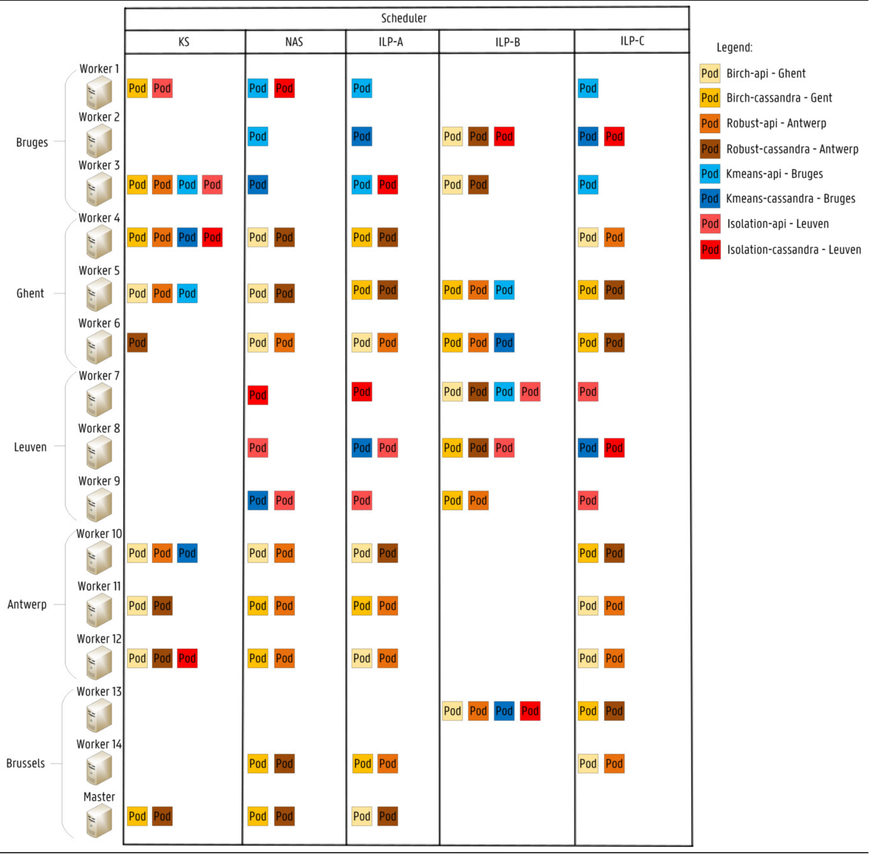
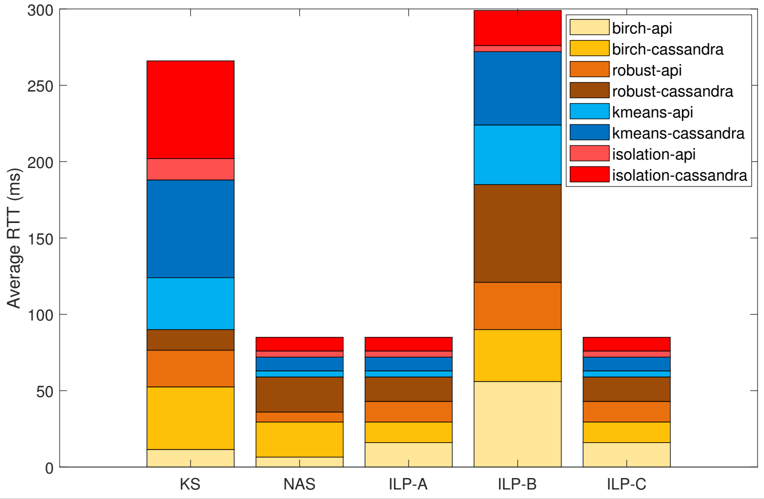


Figure 4.12 Comparison of the average RTT per scheduler for different pod-deployment scheduling strategies in a Smart City air quality monitoring scenario.



In Table 4.8, the expected service bandwidth per node for the different scheduling approaches is presented. Both KS and the ILP-B configuration allocate pods on nodes already compromised in terms of network bandwidth. For instance, KS overloads worker 4 and 12 by allocating to them at least 3 pods leading to service bandwidths of 17.0 Mbit/s and 12.5 Mbit/s for the workers 4 and 12, respectively, which surpasses the available bandwidth of 10.0 Mbit/s. This allocation scheme may lead to service disruptions due to bandwidth fluctuations. Furthermore, the ILP-B configuration overloads 5 worker nodes to reduce the number of active nodes to solely 8, meaning that the remaining 7 are not used in the service provisioning. This occurs due to the selected optimization objective (MIN Nodes). Additionally, it should be highlighted that, although ILP-A and ILP-C achieve the exact same values of RTT for each of the deployed services, their allocation scheme is quite different. ILP-C refines the solution obtained by ILP-A by trying to further optimize the solution space by considering the minimization of nodes as a third optimization objective while maintaining the same RTT values. As shown for this configuration, several nodes can be considered full in terms of network bandwidth since service bandwidths of 10 Mbit/s are expected, which is the limit in the network. Therefore, the ILP-C solution provides us a more efficient usage of the infrastructure by reducing the fraction of free resources per node.

In summary, the proposed NAS optimizes the resource provisioning in Kubernetes according to network latency and bandwidth, which is currently not supported by the default KS. An ILP REST API has been also validated as a container-based application to evaluate the performance of theoretical formulations in real service deployments. As shown, the execution time of ILP models is higher than heuristic mechanisms (KS and NAS). Nevertheless, ILP models obtain the optimal solution for the given problem based on a set of objectives. The evaluated ILP formulations improve the resource provisioning performance of the default KS in terms of latency or energy efficiency and even can refine the allocation scheme of the proposed NAS, while increasing the pod startup time on average by 4 seconds. It should be noted that a dynamic mechanism suitable for dealing with bandwidth fluctuations and delay changes is required, however, it is out of the scope of this chapter.

Table 4.8: The expected service bandwidth per node for the different scheduling strategies.

	Schedulers				
Node	KS	NAS	ILP-A	ILP-B	ILP-C
Worker 1	6.0 Mbit/s	7.5 Mbit/s	2.5 Mbit/s	-	2.5 Mbit/s
Worker 2	-	2.5 Mbit/s	5.0 Mbit/s	**12.5 Mbit/s**	10.0 Mbit/s
Worker 3	**10.5 Mbit/s**	5.0 Mbit/s	7.5 Mbit/s	7.5 Mbit/s	2.5 Mbit/s
Worker 4	**17.0 Mbit/s**	7.5 Mbit/s	10.0 Mbit/s	-	4.5 Mbit/s
Worker 5	7.0 Mbit/s	7.5 Mbit/s	10.0 Mbit/s	9.5 Mbit/s	10.0 Mbit/s
Worker 6	5.0 Mbit/s	4.5 Mbit/s	4.5 Mbit/s	**12.0 Mbit/s**	10.0 Mbit/s
Worker 7	-	5.0 Mbit/s	5.0 Mbit/s	**10.5 Mbit/s**	1.0 Mbit/s
Worker 8	-	1.0 Mbit/s	6.0 Mbit/s	**11.0 Mbit/s**	10.0 Mbit/s
Worker 9	-	6.0 Mbit/s	1.0 Mbit/s	7.0 Mbit/s	1.0 Mbit/s
Worker 10	9.5 Mbit/s	4.5 Mbit/s	7.5 Mbit/s	-	10.0 Mbit/s
Worker 11	7.5 Mbit/s	7.0 Mbit/s	7.0 Mbit/s	-	4.5 Mbit/s
Worker 12	**12.5 Mbit/s**	7.0 Mbit/s	4.5 Mbit/s	-	4.5 Mbit/s
Worker 13	-	-	-	**14.5 Mbit/s**	10.0 Mbit/s
Worker 14	-	10.0 Mbit/s	7.0 Mbit/s	-	4.5 Mbit/s
Master	10.0 Mbit/s	10.0 Mbit/s	7.5 Mbit/s	-	-

4.8 Conclusions

In this chapter, a Fog Computing architecture is proposed for the proper resource provisioning of Smart City container-based applications. Fog Computing has been introduced to manage the growing amount of connected devices in the upcoming years, by placing computational resources on the edges of the network. This trend has encouraged the development of scalable orchestration mechanisms to guarantee the smooth performance of IoT services. Fog Computing provides effective ways to overcome the high demanding requirements introduced by IoT use cases, such as low latency, high energy efficiency and high mobility. The popular open-source project Kubernetes has been used to validate the proposed solution. The scalable design of Kubernetes provides flexible abstractions between the micro-services and the underlying infrastructure. In this chapter, a network-aware scheduling approach is proposed, which enables allocation decisions based on the current status of the network infrastructure. Additionally, an ILP formulation for the IoT service placement problem has been designed as a container-based application and then validated on the Kubernetes platform showing the full applicability of theoretical approaches in real service deployments. Evaluations have been performed to compare the proposed scheduling mechanisms. Results show that the proposed NAS can significantly improve the service provisioning of the default KS by achieving a reduction of 70% in network latency, while increasing the scheduling decision time by only 1.22 ms per pod. Theoretical approaches can demonstrate their full applicability when applied to real service deployments as shown by the validated ILP REST API.

Acknowledgment

This research was funded partially within the projects "Service-oriented management of a virtualised future internet" and "Intelligent DEnse And Longe range IoT networks (IDEAL-IoT)" under Grant Agreement #S004017N, both from the fund for Scientific Research-Flanders "Fonds Wetenschappelijk Onderzoek" (FWO-V).

References

[1] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014.

[2] H Arasteh, V Hosseinnezhad, V Loia, A Tommasetti, O Troisi, M Shafie-Khah, and P Siano. Iot-based smart cities: a survey. In *2016 IEEE 16th Inter-*

national Conference on Environment and Electrical Engineering (EEEIC), pages 1–6. IEEE, 2016.

[3] Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021 white paper, 2019. URL http://www.cisco.com/c/en/us/solutions/collateral/service-provider/\visual-networking-index-vni/mobile-white-paper-c11-520862.pdf.

[4] Mung Chiang and Tao Zhang. Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, 2016.

[5] Amir Vahid Dastjerdi and Rajkumar Buyya. Fog computing: Helping the internet of things realize its potential. *Computer*, 49(8):112–116, 2016.

[6] Subhadeep Sarkar, Subarna Chatterjee, and Sudip Misra. Assessment of the suitability of fog computing in the context of internet of things. *IEEE Transactions on Cloud Computing*, 6(1):46–59, 2018.

[7] Charith Perera, Yongrui Qin, Julio C Estrella, Stephan Reiff-Marganiec, and Athanasios V Vasilakos. Fog computing for sustainable smart cities: A survey. *ACM Computing Surveys (CSUR)*, 50(3):32, 2017.

[8] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H Glitho, Monique J Morrow, and Paul A Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 20(1):416–464, 2018.

[9] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*, pages 195–216. Springer, 2017.

[10] Sam Newman. *Building microservices: designing fine-grained systems.* " O'Reilly Media, Inc.", 2015.

[11] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. 2016.

[12] Kelsey Hightower, Brendan Burns, and Joe Beda. *Kubernetes: Up and Running: Dive Into the Future of Infrastructure.* " O'Reilly Media, Inc.", 2017.

[13] Jose Santos, Thomas Vanhove, Merlijn Sebrechts, Thomas Dupont, Wannes Kerckhove, Bart Braem, Gregory Van Seghbroeck, Tim Wauters, Philip Leroux, Steven Latre, et al. City of things: Enabling resource provisioning in smart cities. *IEEE Communications Magazine*, 56(7):177–183, 2018.

[14] Marcelo Yannuzzi, Frank van Lingen, Anuj Jain, Oriol Lluch Parellada, Manel Mendoza Flores, David Carrera, Juan Luis Pérez, Diego Montero, Pablo Chacin, Angelo Corsaro, et al. A new era for cities with fog computing. *IEEE Internet Computing*, 21(2):54–67, 2017.

[15] Jose Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Towards network-aware resource provisioning in kubernetes for fog computing applications. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 351–359. IEEE, 2019.

[16] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Resource provisioning for iot application services in smart cities. In *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2017.

[17] Amir Vahid Dastjerdi, Harshit Gupta, Rodrigo N Calheiros, Soumya K Ghosh, and Rajkumar Buyya. Fog computing: Principles, architectures, and applications. In *Internet of Things*, pages 61–75. Elsevier, 2016.

[18] Swati Agarwal, Shashank Yadav, and Arun Kumar Yadav. An efficient architecture and algorithm for resource provisioning in fog computing. *International Journal of Information Engineering and Electronic Business*, 8(1):48, 2016.

[19] Mohammad Aazam and Eui-Nam Huh. Dynamic resource provisioning through fog micro datacenter. In *2015 IEEE international conference on pervasive computing and communication workshops (PerCom workshops)*, pages 105–110. IEEE, 2015.

[20] Olena Skarlat, Stefan Schulte, Michael Borkowski, and Philipp Leitner. Resource provisioning for iot services in the fog. In *2016 IEEE 9th international conference on service-oriented computing and applications (SOCA)*, pages 32–39. IEEE, 2016.

[21] Olena Skarlat, Matteo Nardelli, Stefan Schulte, and Schahram Dustdar. Towards qos-aware fog service placement. In *2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC)*, pages 89–96. IEEE, 2017.

[22] Karima Velasquez, David Perez Abreu, Diogo Gonçalves, Luiz Bittencourt, Marilia Curado, Edmundo Monteiro, and Edmundo Madeira. Service orchestration in fog environments. In *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 329–336. IEEE, 2017.

[23] Deze Zeng, Lin Gu, Song Guo, Zixue Cheng, and Shui Yu. Joint optimization of task scheduling and image placement in fog computing supported

software-defined embedded system. *IEEE Transactions on Computers*, 65 (12):3702–3712, 2016.

[24] Vitor Barbosa C Souza, Wilson Ramírez, Xavier Masip-Bruin, Eva Marín-Tordera, G Ren, and Ghazal Tashakor. Handling service allocation in combined fog-cloud scenarios. In *2016 IEEE international conference on communications (ICC)*, pages 1–5. IEEE, 2016.

[25] Antonio Brogi and Stefano Forti. Qos-aware deployment of iot applications through the fog. *IEEE Internet of Things Journal*, 4(5):1185–1192, 2017.

[26] Hlabishi I Kobo, Adnan M Abu-Mahfouz, and Gerhard P Hancke. A survey on software-defined wireless sensor networks: Challenges and design requirements. *IEEE access*, 5:1872–1899, 2017.

[27] Farzad Samie, Vasileios Tsoutsouras, Lars Bauer, Sotirios Xydis, Dimitrios Soudris, and Jörg Henkel. Computation offloading and resource allocation for low-power iot edge devices. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 7–12. IEEE, 2016.

[28] Charles C Byers. Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for fog-enabled iot networks. *IEEE Communications Magazine*, 55(8):14–20, 2017.

[29] Cheol-Ho Hong and Blesson Varghese. Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms. *ACM Computing Surveys (CSUR)*, 52(5):1–37, 2019.

[30] Patrick Kalmbach, Johannes Zerwas, Péter Babarczi, Andreas Blenk, Wolfgang Kellerer, and Stefan Schmid. Empowering self-driving networks. In *Proceedings of the Afternoon Workshop on Self-Driving Networks*, pages 8–14. ACM, 2018.

[31] Genc Tato, Marin Bertier, and Cédric Tedeschi. Designing overlay networks for decentralized clouds. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 391–396. IEEE, 2017.

[32] Charles Anderson. Docker [software engineering]. *IEEE Software*, 32(3): 102–c3, 2015.

[33] Kubernetes. Kubernetes, automated container deployment, scaling, and management, 2019. URL https://kubernetes.io/.

[34] Víctor Medel, Rafael Tolosana-Calasanz, José Ángel Bañares, Unai Arronategui, and Omer F Rana. Characterising resource management performance in kubernetes. *Computers & Electrical Engineering*, 68:286–297, 2018.

[35] IBM. Ibm ilog cplex optimization studio, 2019. URL https://www.ibm.com/products/ilog-cplex-optimization-studio.

[36] Spring. Spring: the source for modern java, 2019. URL https://spring.io/.

[37] IntelliJ IDEA. Intellij idea, capable and ergonomic ide for jvm, 2019. URL https://www.jetbrains.com/idea/.

[38] The virtual wall emulation environment, 2019. URL https://doc.ilabt.imec.be/ilabt-documentation/index.html.

[39] Overview of kubeadm, 2019. URL https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm/.

[40] José Santos, Philip Leroux, Tim Wauters, Bruno Volckaert, and Filip De Turck. Anomaly detection for smart city applications over 5g low power wide area networks. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2018.

5
Diktyo: Network-aware Scheduling for Container Clouds

"Patience is a key element of success."

–Bill Gates, former Microsoft CEO

This chapter extends chapter 4 by designing a network-aware framework named Diktyo integrated into the Kubernetes scheduling plugins project. Diktyo proposes three additional plugins to optimize the allocation of service chains in Kubernetes focused on low latency and bandwidth optimization. The approach in Chapter 4 relied on an extender call not fully integrated with the Kubernetes project. Latency-sensitive applications demand lower latency between the microservices in the application. Current scheduling methods focus on reducing deployment costs, insufficient for applications where end-to-end latency reduction is a primary objective. Simulations show that Diktyo can minimize network latency across different infrastructure topologies while achieving similar execution times as current scheduling plugins. Practical experiments in a Kubernetes cluster show that Diktyo increases throughput by 22% and reduces latency by 45% by placing dependent microservices close to each other. The main contribution of this chapter is the development of the Diktyo framework for the Kubernetes scheduling plugins project. This chapter has been submitted for publication in the OSDI 2022 proceedings and represents a collaboration with IBM TJ Watson, NYC, USA.

⋆⋆⋆

José Santos, Chen Wang, Tim Wauters and Filip De Turck.

Submitted to USENIX Symposium on Operating Systems Design and Implementation (OSDI), December 2021.

Abstract Recent applications are becoming more and more delay-sensitive, demanding lower latency between the microservices in the application. Current scheduling policies aim to reduce costs or increase resource efficiency, which is insufficient for applications where end-to-end latency reduction becomes a primary objective. Application domains such as multi-tier web services and highly-available databases would benefit the most from network-aware scheduling policies, which consider both latency and bandwidth requirements. This paper proposes a fast and scalable network-aware scheduling framework for the Kubernetes platform named Diktyo, which determines the placement of containerized applications at run time to guarantee low latency and necessary bandwidth reservations. Simulations show that Diktyo can significantly reduce network latency for various applications across different infrastructure topologies. Scalability benchmarks of Diktyo prove that its execution times are similar to production-ready scheduling plugins and are scalable over 10k nodes/pods. Further application experiments in a distributed Kubernetes cluster demonstrate that Diktyo scheduling on average can increase database throughput by 22% and reduce the application's response time up to 45%.

5.1 Introduction

Next-generation applications are pushing cloud infrastructures further by demanding even lower latency between their microservices. Application domains such as the Internet of Things (IoT) [1], multi-tier web services [2], databases [3], and video streaming services [4] are latency-sensitive, requiring sub-millisecond end-to-end (E2E) latency for their proper operation. Current scheduling policies in popular orchestration platforms (e.g., Kubernetes [5], Amazon AWS [6]) focus mostly on optimizing resource utilization in the infrastructure (e.g., CPU and RAM), which is not enough to meet the stringent requirements of these applications. Kubernetes is currently the most popular container orchestration platform. It automates several processes through the containerized applications' life-cycle, including deployment and scaling [7]. Nevertheless, network-aware scheduling policies are missing to enable the network-aware placement of application microservices in container clouds.

Latency reduction is a primary objective since users usually face latency issues when using multi-tier applications, affecting the overall application performance [8]. These applications typically include tens to hundreds of microservices with complex inter-dependencies. Distance between servers is usually the primary culprit as these microservices are scheduled in the infrastructure without latency awareness. Currently, the best strategy is to reduce the latency between chained microservices of an application, according to previous works on Service Function Chaining (SFC) [9, 10]. **Bandwidth** optimization also plays a key role, especially for those applications with high volumes of data transfers among microservices. For example, multiple replicas in a database application may require frequent copies to ensure data consistency. Spark jobs [11] may have regular data transfers between mappers and reducers. Insufficient network capacity on links between nodes would lead to increasing delay or packet drops, which would further degrade the Quality of Service (QoS) for applications.

These applications would benefit the most from network-aware scheduling policies, considering latency and bandwidth in addition to computing resources (e.g., CPU and RAM) used by the default scheduling mechanism. Network latency and available bandwidth on links between cluster nodes can vary according to their locations in the underlying infrastructure. Deploying microservices on different sets of nodes can impact the application's response time and overall QoS. For example, latency and bandwidth requirements are critical for the Redis cluster application [12] (Fig. 5.1a), where master nodes need to synchronize data with slave nodes regularly. Dependencies between the masters and the slaves need to be considered in the scheduling process. High latency or low bandwidth between masters and slaves can lead to slow Create, Read, Update, and Delete (CRUD) operations [13]. Similarly, dependencies exist between microservices for multi-tier web applications, e.g. the Online Boutique e-commerce application [14] (Fig. 5.1b). Placing a single microservice far away from others can impact the overall performance of the service chain and its E2E latency. Previous works on network-aware scheduling focus mostly on theoretical formulations (e.g., [15–17]) or heuristic-based solutions (e.g.,[18, 19]) that usually are assessed via simulations, which limits their applicability in production systems.

This paper presents a network-aware scheduling framework, named Diktyo [1], for the Kubernetes platform inspired by its recent scheduling plugins architecture [20]. Diktyo places application microservices on cluster nodes by applying network-aware mechanisms. It schedules dependent microservices on nodes with links of sufficient capacity to reserve the required bandwidth. Also, it minimizes the SFC latency by selecting nodes with low network costs based on dependencies of chained microservices. All types of infrastructures can benefit from Diktyo, as network conditions between cluster nodes vary according to their locations in the topology. Common topologies include Multi-Region (MR) geo-distributed scenarios, Data Centers (DCs) with fat-tree topology or even a highly available multi-zone cluster topology. Further details about the framework are given in Section 5.5.

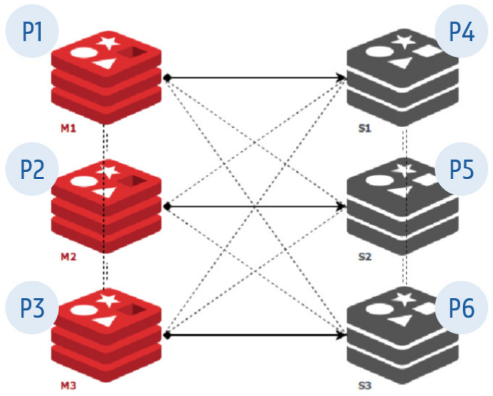
[1]Diktyo means "network" in Greek, as an analogy of Kubernetes, which means "pilot" in Greek.

To the best of our knowledge, Diktyo goes beyond the current state-of-the-art. It is the first attempt toward efficient service chain placement in future cloud-native architectures. As the first network-aware scheduling framework on Kubernetes, Diktyo provides near-optimal container placement, considering both the application microservice dependencies and the underlying cluster topology in a scalable manner. The main contributions are the following:

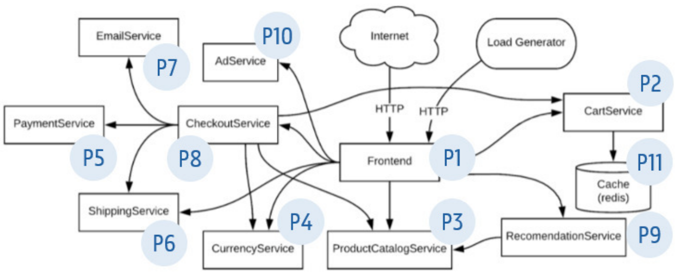
- **Diktyo framework**: The design and implementation of a network-aware scheduling mechanism that separates the control plane (the scheduling logic) from the data plane (e.g., the application microservice dependencies and the cluster network topology). Several scheduling plugins are designed to determine the container placement algorithm. Two asynchronous controllers manage two Custom Resources (CRs) defined as Custom Resource Definitions (CRDs) [21] in Kubernetes: **AppGroup** CRD establishes service chain dependencies; **NetworkTopology** CRD caches and updates network costs between regions and zones for the underlying cluster topology.
- **Mixed-Integer Linear Programming (MILP) modeling**: The formulation of a MILP model for the container scheduling problem, where containers are chained in an application dependency graph, and the underlying cluster topology has links with varying network latency and capacity. The optimal solution from the MILP model serves as an ideal baseline to compare Diktyo and other heuristic scheduling algorithms. Simulations show that Diktyo significantly outperforms current production-ready scheduling plugins in terms of reducing application E2E latency (Sec. 5.6.1).
- **Plugin implementation**: The design of three plugins for the Diktyo together approximate a near-optimal scheduling solution that minimizes the application E2E latency in a scalable manner. These plugins include a **TopologicalSort** plugin that sorts microservices based on topology information, a **NodeNetworkCostFit** plugin that filters out candidate nodes based on the microservice's AppGroup requirements and a **NetworkMinCost** plugin that scores nodes based on network weights ensuring low SFC latency for microservices in AppGroups. The time complexity of these plugins is logarithmic over the number of pods and nodes.
- **Evaluations in Practical Use Cases**: Diktyo is evaluated on real-world applications, including a multi-tier web application benchmark (Online Boutique [14]) and a database application (Redis cluster [12]). Experiments in a distributed Kubernetes cluster show that Diktyo increases throughput by 22% for the Redis database and reduces the application response time up to 45% for the Online Boutique application (Sec. 5.6.3).

Figure 5.1 Illustration of microservice dependencies [12, 14].

(a) Redis Cluster application.



(b) Online Boutique application.



5.2 Related Work

This section addresses current literature on topology-aware and network-aware scheduling. Section 5.2.1 presents prior works related to topology-aware scheduling. Section 5.2.2 introduces works studying network-aware solutions for application deployment in cloud platforms, and Section 5.2.3 reviews recent research on container-based scheduling, including open-source plugins.

5.2.1 Dependency/Topology-aware scheduling

Microservice inter-dependencies have been mainly addressed in the field of batch job scheduling systems [22–25]. Most efforts focus on improving resource efficiency [25–27], characterizing inter-job dependencies [28], reducing the makespan of jobs [22, 24, 29] or improving the system's throughput [23, 30, 31]. The considered dependencies usually are temporal [27, 28] dependencies, meaning later running jobs depend on successful completion of earlier ones. However, the microservice dependencies considered in this paper are spatial, meaning all containers need to communicate with others and run as a whole for the application. The placement of these dependent containers can impact the performance of the whole application.

Topology-aware scheduling allows the placement to be aware of the underlying cluster topology information. An example is the Topology-aware [32] scheduler plugin for Kubernetes. The node topology is specified in *NodeResourceTopology CRDs*. Its placement overcomes performance issues concerning memory, and CPU accesses in Non-Uniform Memory Access (NUMA) nodes [33]. Also, the heterogeneity of resources has been studied for performance improvement in [31]. However, these works do not address the network topology of the cluster. In contrast, the Diktyo framework considers both the microservice dependencies and the underlying cluster network topology to enable network-aware scheduling in Kubernetes.

5.2.2 Network-aware scheduling

Theoretical formulations have been the most applied method to solve network-aware allocation in the last few years [15–19, 34]. Typically, these proposals focus on Integer Linear Programming (ILP) and MILP models to find the optimal allocation scheme based on a particular objective. The main drawback of these modeling approaches is that they cannot find a feasible solution within an acceptable time, thus limiting their applicability in operational environments. However, the modeling can always provide an optimal benchmark for heuristic-based algorithms.

Data Center is an important scenario where network-aware scheduling has been studied in recent years [16, 17, 19, 35–38]. These works focus on optimizing network bandwidth to reduce traffic congestion [36] or reducing the average task completion time [35]. However, practical implementations of these methods are missing since most network-aware algorithms are evaluated via simulations. In addition, most efforts focus on Virtual Machine (VM) placement and only a few address container allocation [19]. Nevertheless, network latency is usually overlooked in current network-aware approaches for DC topologies since network bandwidth is their primary focus.

Geo-distributed clouds also impose several challenges toward network-aware deployments [34, 39–44]. Prior works [45, 46] model application microservice dependencies as service chains to optimize service scheduling in Fog Computing [47]. High latency is a concern in these topologies, especially for IoT and video streaming applications. These efforts have demonstrated the benefits of using network-aware placement solutions concerning the application's response time and throughput.

5.2.3 Container-based Scheduling

Containerized applications have been recently studied [19, 43, 45, 46, 48–51]. Authors aim mostly to reduce the network latency in container deployments by efficiently resizing containers [50]. In addition, resource fragmentation [19] and multi-tenant fair scheduling [49] have also been addressed lately. Furthermore, a few scheduling plugins are already available for Kubernetes in open-source that consider service dependencies. The Volcano project [52] provides several plugins, such as Gang [53], Task Topology [54], Binpack [55] and Dominant Resource Fairness (DRF) [56]. The Gang scheduling plugin considers several tasks running in different containers as a group. It ensures that a minimum number of containers in the group can be deployed as a whole based on cluster resource availability. The Task Topology plugin groups containers into buckets based on task affinities and anti-affinities. The goal is to minimize data transmission between tasks in the same bucket, thus decreasing transmission delay in the overall job execution time. However, latency is not clearly defined as an objective because tasks are placed according to the created buckets. The Binpack and DRF plugins are additional plugins in Volcano to work with Gang and Task Topology to improve node resource utilization and prevent small job starvation [57]. Another plugin that considers the relationship between microservices in an application is the CoScheduling [58]. The CoScheduling plugin operates similarly to the Gang plugin provided by Volcano. It ensures that a minimum number of pods belonging to the same *PodGroup* are scheduled as a whole. The main difference between Diktyo and existing plugins is that current plugins cannot handle complex dependencies between heterogeneous pods.

In summary, Table 5.1 shows a comparison of all plugins listed above with the proposed Diktyo framework in terms of plugin extension points, placement optimization goals, and their network awareness. The proposed approach goes beyond the current literature since Diktyo is the only one that considers application service dependencies and the underlying network topology to optimize the SFC latency with the necessary bandwidth reservation. Prior works focus on theoretical models and heuristics evaluated via simulations or small testbed experiments, limiting their applicability in large-scale production clusters.

5.3 Scheduling in Kubernetes

Microservices in Kubernetes are often tightly coupled into a group of containers named a *pod*. A pod is the smallest working unit in Kubernetes representing the collection of containers and volumes (storage) running in the same execution environment [5]. Kubernetes schedules a given pod on a certain node based on the pod deployment requirements and the cluster's available resources. The selected node

Table 5.1: Comparison among different scheduling plugins.

Plugin / Framework	Project	Extension	Main focus	Service Topology	SFC	Latency	Bandwidth
Gang	V	PF	AP & R	✗	✗	✗	✗
Task Topology	V	QS & PF & S	T & R	✓	✗	✗	✗
Binpack	V	S	R	✗	✗	✗	✗
Dominant Resource Fairness (DRF)	V	QS & PF	R	✗	✗	✗	✗
CoScheduling	S	QS & PF	AP	✓	✗	✗	✗
Topology-aware	S	F	P & T	✓	✗	✗	✗
Diktyo	W	QS & F & S	L & B	✓	✓	✓	✓

Project: V = Volcano, S = Scheduler Plugins, W = Proposed Approach.
Extension: QS= QueueSort, PF = PreFilter, F = Filter, S = Score.
Main Focus: R = Resources, AP = App. Dependency, T = Topology-aware, P= Performance, L = Latency, B = Bandwidth.
Service Topology, SFC, Latency, Bandwidth: ✓= addressed, ✗= not considered.

pulls the container images from the image registry if needed and coordinates the necessary operations to launch the pod. The component responsible for scheduling operations is named Kube-Scheduler (KS), the default scheduler in Kubernetes.
The KS chooses a node for the pod deployment based on a two-step operation. Firstly, the KS checks if nodes can run the pod based on a set of filters, also known as predicates. These filters mostly focus on the pod's resource requirements (e.g., CPU and RAM) and check if the node has enough capacity to meet those. Secondly, the KS applies node priority calculation by ranking each remaining node based on a set of scoring algorithms, also named priorities. Then, the highest-scoring node is selected by KS for the pod deployment. To ease the development of further filter and scoring functions, Kubernetes released a scheduling framework [59] so that developers could implement their algorithms and contribute to the Kubernetes project.
The Kubernetes scheduling framework implements several extension points for the KS. The framework allows developers to implement their algorithms as plugins without interfering with the main scheduling components. The framework currently exposes the following main extension points, responsible for:

- **QueueSort**: sort pods in the scheduling queue.
- **Filter**: filter out nodes that cannot run the pod.
- **Score**: rank nodes that have passed the filtering phase.
- **NormalizeScore**: modify scores before the final ranking of nodes.

The Diktyo framework makes use of these extension points to include bandwidth and latency in the scheduling process in Kubernetes. The framework is detailed in Sec. 5.5. The next section presents a MILP model for the Kubernetes deployment scheme, where pods are allocated on nodes based on a given objective. The model provides a benchmark for the network-aware framework and existing plugins to compare their sub-optimal allocation schemes with an optimal solution.

5.4 Mixed-Integer Linear Programming (MILP) Approach

This section presents the MILP formulation for the network-aware Kubernetes scheduler, which places application pods on cluster nodes to both maximize the total number of applications admitted and minimize the application's SFC latency. The main advantage of MILP models is the flexibility to analyze NP-hard problems [60] as the Kubernetes application deployment model and provide a benchmark for developed heuristics [61]. The MILP model formulates the pod placement problem in Kubernetes as an optimization problem subject to several constraints, where

Table 5.2: Input variables of the MILP model.

Symbol	Description
N	The set of nodes on which pod replicas are executed.
A	The set of all applications. Each application consists of a set of different pods ordered in a chain.
P	The set of all pods.
P^a	The set of all pods p belonging to the application a.
P_r^a	The replica set r of pods of the same pod type belonging to the application a.
Z	The set of zones where applications can be deployed.
$I_{a,p}$	The Instance matrix I. If the element $I_{a,p} = 1$, the pod p belongs to application a.
$\Omega_n[c]$	The capacity vector of node n. c denotes resources as CPU (in vcpu), memory (in Mi), and bandwidth (in Mbps).
$\omega_p[r]$	The demand vector of pod p. r denotes resources as CPU (in vcpu), memory (in Mi), and bandwidth (in Mbps).
B_{n_1,n_2}	The network bandwidth matrix B. B_{n_1,n_2} indicates the bandwidth capacity (in Mbps) between node n_1 and node n_2.
τ_{n_1,n_2}	The network latency matrix τ. The element τ_{n_1,n_2} indicates the latency (in ms) between the node n_1 and the node n_2.
C_{p_i,p_j}	The pod communication matrix C. The element C_{p_i,p_j} indicates the minimum bandwidth (in Mbps) demand between the pod p_i and p_j. The pod p_i is the source of the network flow while p_j is the sink.
$E_{n,z}$	The zone matrix E. If the element $E_{n,z} = 1$, the node n is at zone z.
α_{p_i,p_j}	The SFC matrix α. If the element $\alpha_{p_i,p_j} = 1$, the flow bandwidth between the pods p_i and p_j must be guaranteed.

Table 5.3: Decision variables of the MILP model.

Symbol	Description
G_a	The acceptance matrix G_A. If the element $G_a = 1$, the application a is deployed.
$G_{a,p}$	The pod acceptance matrix $G_{A\times P}$. If the element $G_{a,p} = 1$, the pod p for the application a is allocated.
$P_{p,n}$	The placement matrix P. If the element $P_{p,n}$ = 1, the pod p is executed on node n.
$F_{p_j,p_i}(n_1, n_2)$	The binary flow matrix indicates that pod p_i and pod p_j are allocated on node n_1 and n_2, respectively.
Λ_a	The SFC latency matrix Λ. The element Λ_a indicates the end-to-end response time (in ms) of requests traversing the service chain belonging to application a.

input variables are listed in Table 5.2 and decision variables as the application acceptance matrix G_A and the placement matrix P are shown in Table 2.3.
The objectives considered in the model are the following:

1) Maximization of Application Requests (MAX AR).
2) Minimization of the Network Latency (MIN NL).

These two objectives are executed iteratively, meaning that a different optimization is applied in each iteration. First, the acceptance of application requests is maximized, and then, in the second iteration, the network latency is minimized. Additional constraints are added to the model to retain the objective value obtained in the previous iteration. Thus, the solution space continuously decreases since iterations must satisfy the previous optimal solutions. Every iteration refines the previously obtained solution by improving the model with an additional objective.
The maximization of application deployments related to the first objective is expressed in (5.1) by using the acceptance matrix G_A. The second objective relates to the latency reduction in the service chain between pods from the same application. The model determines an allocation scheme based on the service dependency graph where pods with an established connection are allocated close to each other to reduce the expected network latency while respecting all constraints. This objective is expressed as shown in (5.2), where the SFC latency (in ms) of each application can be derived from the application flow matrix F as shown in (5.3).

$$max \sum_{a \,\varepsilon\, A} G_a \tag{5.1}$$

$$min \sum_{a \,\varepsilon\, A} \Lambda_a \tag{5.2}$$

$$\forall a \,\varepsilon\, A : \Lambda_a = \sum_{p_i, p_j \,\varepsilon\, P^a} \sum_{n_1, n_2 \,\varepsilon\, N} \tau_{n_1,n_2} \times F_{p_j,p_i}(n_1, n_2) \tag{5.3}$$

The application flow matrix F is subjected to various constraints to accurately represent network flows. A chain path is expressed by the Flow Factor $\Upsilon p_i, p_j$ as shown in (5.4) by using the SFC matrix α_{p_i,p_j}. Bandwidth capacity limitations (5.5) ensure the infrastructure capacity is respected, while flow conservation constraints (5.6) and (5.7) ensure no flow is lost within the network.

$$\Upsilon_{p_i,p_j} = I_{a,p_i} \times I_{a,p_j} \times \alpha_{p_i,p_j} \tag{5.4}$$

$$\forall n_1, n_2 \,\varepsilon\, N :$$
$$\sum_{a \,\varepsilon\, A} \sum_{p_i, p_j \,\varepsilon\, P^a} \Upsilon_{p_i,p_j} \times C_{p_i,p_j} \times F_{p_j,p_i}(n_1, n_2) \leq B_{n_1,n_2} \tag{5.5}$$

$$\begin{aligned}&\forall a \,\varepsilon\, A, \forall p_i, p_j \,\varepsilon\, P^a, \forall n_1, n_2 \,\varepsilon\, N : \\ &F_{p_j,p_i}(n_1, n_2) = 0 \quad \text{if } P_{p_i,n} = 0 \vee P_{p_j,n} = 0\end{aligned} \tag{5.6}$$

$$\sum_{a \,\varepsilon\, A} \sum_{p_i,p_j \,\varepsilon\, P^a} \sum_{n_1,n_2 \,\varepsilon\, N} F_{p_j,p_i}(n_1, n_2) = \sum_{p_i,p_j \,\varepsilon\, P^a} \alpha_{p_i,p_j} \tag{5.7}$$

Also, the placement of all application pods needs to satisfy multiple constraints. First, all pods in the application need to be deployed as a whole. An indicator constraint (5.8) states that an application a is deployed (i.e., $G_a = 1$) if all pods belonging to the application have been deployed. Further, pod placements must ensure nodes to be scheduled on have enough resources for allocation represented in (5.9).

$$\forall a \,\varepsilon\, A : \sum_{p \,\varepsilon\, P^a} G_{a,p} = \sum_{p \,\varepsilon\, P^a} I_{a,p} \quad \textbf{if } G_a = 1 \tag{5.8}$$

$$\forall n \,\varepsilon\, N : \sum_{a \,\varepsilon\, A} \sum_{p \,\varepsilon\, P^a} P_{p,n} \times \omega_p[r] \leq \Omega_n[c] \tag{5.9}$$

In addition, two constraints have been added to represent topology preferences [62]. Pod anti-affinity rules (5.10) ensure replicas of the same pod are allocated on different nodes, and zone anti-affinity rules (5.11) spread pod replicas across different zones in the cluster.

$$\forall n \,\varepsilon\, N, \forall a \,\varepsilon\, A : \sum_{p \,\varepsilon\, P_r^a} P_{p,n} \leq 1 \tag{5.10}$$

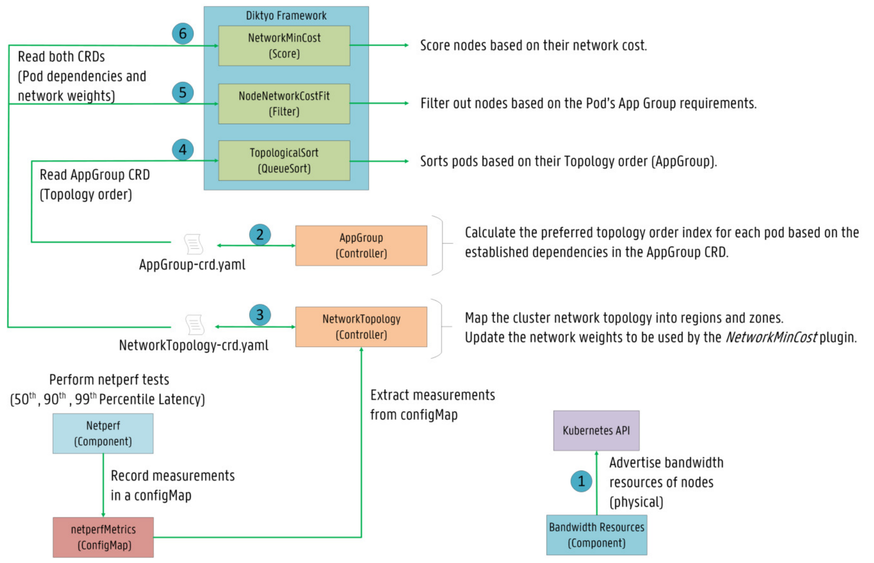
$$\forall a \,\varepsilon\, A, \forall z \,\varepsilon\, Z : \sum_{p \,\varepsilon\, P_r^a} \sum_{n \,\varepsilon\, N} P_{p,n} \times E_{n,z} \leq 1 \tag{5.11}$$

5.5 Diktyo framework: System Design

5.5.1 System Overview

Fig. 5.2 shows an overview of the Diktyo framework. Bandwidth resources are advertised to the Kubernetes API to consider the node's available bandwidth capacity in the scheduling process ①. The framework also introduces two CRs: **AppGroup** and **NetworkTopology** to maintain both the service dependency information ② and the infrastructure network topology ③. The **NetworkTopology** controller obtains network weights between nodes across regions and zones.

Figure 5.2 Illustration of the Diktyo framework.


Diktyo Framework
NetworkMinCost (Score)
NodeNetworkCostFit (Filter)
TopologicalSort (QueueSort)
Score nodes based on their network cost.
Filter out nodes based on the Pod's App Group requirements.
Sorts pods based on their Topology order (AppGroup).
Read both CRDs (Pod dependencies and network weights)
Read AppGroup CRD (Topology order)
6
5
4
2
3
1
AppGroup-crd.yaml
AppGroup (Controller)
Calculate the preferred topology order index for each pod based on the established dependencies in the AppGroup CRD.
NetworkTopology-crd.yaml
NetworkTopology (Controller)
Map the cluster network topology into regions and zones.
Update the network weights to be used by the NetworkMinCost plugin.
Perform netperf tests (50th, 90th, 99th Percentile Latency)
Netperf (Component)
Record measurements in a configMap
netperfMetrics (ConfigMap)
Extract measurements from configMap
Kubernetes API
Advertise bandwidth resources of nodes (physical)
Bandwidth Resources (Component)

Thus, Diktyo is aware of both application and infrastructure network topology. The network-aware algorithms are implemented via the Kubernetes Scheduler plugin framework [59], and it consists of three scheduling plugins: **TopologicalSort**, **NodeNetworkCostFit** and **NetworkMinCost**. First, pods are sorted based on their established dependencies ④. Then, nodes are filtered out based on the pod's AppGroup requirements ⑤, and lastly, nodes are scored based on network weights focusing on reducing the network latency between chained microservices ⑥. Further explanations are given below on how these plugins interact with both CRs.

5.5.2 Bandwidth Enforcement

In Kubernetes, bandwidth resources can be defined for pods and nodes and used by the KS via extended resources [63]. The node's (physical) bandwidth capacity can be advertised in the cluster through a bandwidth component [64] developed to send HTTP requests to the Kubernetes API server. The label *network.aware.com/bandwidth* specifies bandwidth capacities. The bandwidth CNI plugin [65] can enforce network bandwidth allocations for pods. It supports ingress and egress traffic shaping to limit the pod bandwidth. Pods share the host network bandwidth when deployed on the same node. Limiting Pod bandwidth can prevent mutual interference and improve network stability [66]. The addition of *kubernetes.io/ingress-bandwidth* and *kubernetes.io/egress-bandwidth* annotations to the pod configuration file ensures bandwidth limitations are respected. This allows to perform filter and score algorithms based on bandwidth resources (e.g., *MostRequested*, *BalancedAllocation*).

5.5.3 Application Group Custom Resource Definition (CRD) & Controller

Figure 5.3 Example of a service chain in Kubernetes.

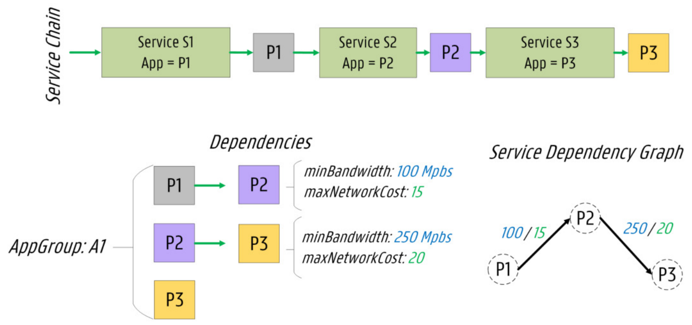


Table 5.4: Topological sorting for Online Boutique.

Algorithm	Topological order
Kahn	[P1, P10, P9, P8, P7, P6, P5, P4, P3, P2, P11]
Alt. Kahn	[P1, P11, P10, P2, P9, P3, P8, P4, P7, P5, P6]
Rev. Kahn	[P11, P2, P3, P4, P5, P6, P7, P8, P9, P10, P1]
Tarjan	[P1, P8, P7, P5, P4, P2, P11, P9, P10, P6, P3]
Alt. Tarj.	[P1, P3, P8, P6, P7, P10, P5, P9, P4, P11, P2]
Rev. Tarj.	[P3, P6, P10, P9, P11, P2, P4, P5, P7, P8, P1]

An AppGroup CRD [67] is defined to describe the microservice dependencies in service chains. Fig. 5.3 shows an example of a service chain in Kubernetes and the corresponding service dependency graph. The *minbandwidth* requirement defines the minimum bandwidth between two pods of the same AppGroup. Thus, pods cannot be scheduled on nodes with the connection not having enough capacity to meet the specified bandwidth requirement. The *maxNetworkCost* requirement determines the maximum network cost between two pods. If the network cost between two nodes is higher than the defined *maxNetworkCost*, then Diktyo cannot place these two pods on these nodes.
An application consists of several pods with dependencies. The tighter constraints a pod has, the more likely the pod cannot find a node that satisfies all constraints. Scheduling the pod with tighter constraints earlier would be preferred, so it would

not be blocked later, leading to starvation. However, it is not straightforward to determine which pod has tighter constraints. Therefore, the **AppGroup controller** [68] calculates the scheduling order of all pods in an application via six heuristic topological sorting algorithms [69]: *Kahn* [70], *Tarjan* [71], *AlternateKahn*, *AlternateTarjan*, *ReverseKahn*, *ReverseTarjan*. Alternate Kahn modifies the order given by Kahn by selecting the first element of Kahn as its first element, the last of Kahn as its second, the second of Kahn as its third, and so on. AlternateTarjan follows the same pattern of AlternateKahn and modifies the order of Tarjan. ReverseKahn and ReverseTarjan essentially reverse the order given by Kahn and Tarjan, respectively. For the service chain example, the preferred topology order would be **P1, P2, P3** for Kahn and **P3, P2, P1** for ReverseKahn. Furthermore, a complex AppGroup is the Online Boutique application previously shown in Fig. 5.1b. It consists of eleven microservices, named from P1 - P11. Table 5.4 presents the preferred order for all sorting algorithms. As shown, significant differences in the ordering of pods are obtained depending on which topology algorithm is selected.

5.5.4 Network Topology Custom Resource Definition (CRD) & Controller

A networkTopology CRD [72] defines a networkTopology CR to store and update network costs between all pair-wise nodes in the cluster based on their zones and regions. As an initial design, network weights can be manually defined in a single NetworkTopology CR where network costs between zones and between regions are specified. In addition, to accurately measure the latency in the cluster, a **netperf component** [73] has been developed for the Diktyo framework. Netperf tests [74] are executed based on the nodes available in the infrastructure, allowing to estimate the latency between nodes in the cluster, especially different latency percentiles (i.e., 50th, 90th, and 99th percentile).These measurements are recorded in a configmap [75] as key-value pairs with *origin* and *destination* as labels. Then, the **networkTopology controller** [76] accesses the configmap to extract the netperf measurements and calculates accurate network costs across regions and zones in the cluster. The networkTopology controller can then dynamically update the CR accordingly, so scheduling plugins always use the updated network weights instead of the one-time manually configured weights. The periodical probing of the network latency via the netperf component is only necessary for one pair of nodes between zones/regions. One-time probing between a single pair of nodes is sufficient if nodes within a zone have similar connections. Therefore, the probing is limited, avoiding significant overhead for large-scale clusters. Furthermore, the networkTopology controller can work with any customized software components that update the configmap. Thus, cloud administrators can apply various methods to update the network costs according to their preferences. Also, the networkTopology controller maintains the available bandwidth capacity (i.e., *bandwidthAllocatable*) between regions and zones in the cluster. Pod allocations and corresponding dependencies are read from an AppGroup lister [77], and bandwidth reservations are saved in the networkTopology CR based on pod deployments.

5.5.5 Network-aware Scheduling Plugins

This section presents further details on the scheduler plugins designed and implemented for the Diktyo framework.

TopologicalSort QueueSort Plugin [78] sorts pods belonging to an AppGroup based on the topology order, which are calculated by the AppGroup controller using the specified sorting algorithm. It prioritizes pods based on the sorted order. If pods do not belong to an AppGroup or belong to different AppGroups, the plugin follows the strategy provided by the QoS plugin [79].

NodeNetworkCostFit Filter Plugin [80] respects pod dependencies established in the AppGroup CR. The implementation currently focuses on *maxNetworkCost* requirements, filtering out nodes that would generate higher network costs than the specified threshold. Since applications usually include multiple pods with inter-dependencies, the pods deployed later may have constraints not fully respected. Thus, the *NodeNetworkCostFit* plugin filters out nodes that cannot support the majority of the requirements required by the dependencies of pods already deployed to reduce the number of nodes to score.

NetworkMinCost Score Plugin [81] favors the node with the lowest aggregated network cost to nodes where pods are already deployed. The aggregated network cost is calculated based on all pods' dependencies maintained by the AppGroup. The cost per dependency is the network weight between nodes of scheduled pods. Pods scheduled in an AppGroup are retrieved via a pod lister [82] from the Kubernetes API. Thus, the plugin calculates the aggregated cost to schedule a pod on a particular node based on previous pod placements. Also, network weights between regions and zones of the cluster are available in the NetworkTopology CR. The plugin favors all candidate nodes equally for the first pod scheduled in the AppGroup. Lastly, the *NetworkMinCost* plugin normalizes scores between 0 and 100 based on all candidate nodes' maximum and minimum costs. Nodes with lower costs are favored since it also corresponds to lower latency. The *NetworkMinCost* plugin can be combined with other scoring functions (e.g., *BalancedAllocation*, *LeastRequestedPriority*). However, a higher weight should be attributed to the *NetworkMinCost* plugin to prefer the latency-aware scheduling policy.

5.6 Evaluations

This section presents the experiments conducted to evaluate the performance of the Diktyo framework. Sec. 5.6.1 presents a simulation environment used to validate the proposed approach, and Sec. 5.6.2 shows the scalability benchmarks of the implemented plugins. Lastly, Sec. 5.6.3 presents the testbed experiments concerning the implemented framework.

5.6.1 Simulation Environment

5.6.1.1 Infrastructure Topologies & Input Variables

Table 5.5: The hardware configuration of each node based on Amazon EC2 On-Demand Pricing [83].

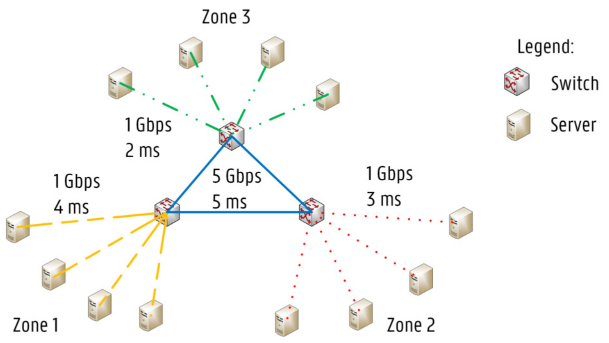
Topology	Node	Amazon Image	CPU (cpu)	RAM (Mi)	Band. (Gbps)
MR	C	4XL	16.0	32.0	40.0
	FT2	2XL	8.0	16.0	10.0
	FT1	XL	4.0	8.0	5.0
	E	L	2.0	4.0	1.0
C	FT1	XL	4.0	8.0	1.0
DC	FT1	XL	4.0	8.0	1.0

Topology: MR= Multi-Region, C= Cluster, DC= Data center.
Node: C= Cloud, FT2= Fog Tier 2, FT1= Fog Tier 1, E=Edge.
Amazon Image: 4XL= a1.4xlarge, 2XL= a1.2xlarge, XL= a1.xlarge, L= a1.large.

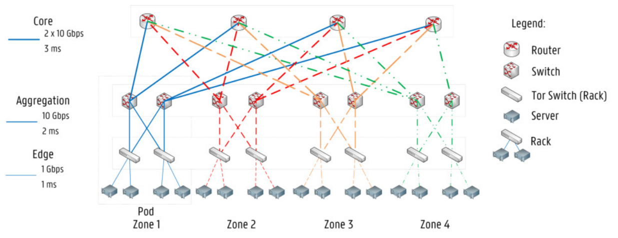
Fig. 5.4 shows the three typical infrastructure topologies to be evaluated. Fig. 5.4a represents a highly available cluster with nodes deployed across zones. Fig. 5.4b illustrates a DC topology with nodes connected via a fat tree network [84]. Fig. 5.4c shows a typical edge MR cluster with nodes distributed across several zones and regions. Nodes provide computing resources to allocate pods in the infrastructure. Table 5.5 presents the nodes' hardware configurations for each topology. The bandwidth matrix B_{n_1,n_2} is based on the available bandwidth capacity and the latency matrix τ_{n_1,n_2} is calculated based on the shown latency values. The described MILP model has been implemented in Python using the IBM ILOG CPLEX ILP solver [85]. The model considered two consecutive objective functions: first, the acceptance of applications requests is maximized (i.e., MAX AR), and then the network latency is minimized (i.e., MIN NL). As stated, the MILP model is executed iteratively, and to retain the objective value of the first objective, an additional constraint is added to the model. The model has been executed

Figure 5.4 Illustration of the three evaluated topologies.

(a) Cluster infrastructure.



(b) Data Center (DC) fat tree topology.



(c) Multi-Region (MR) scenario.

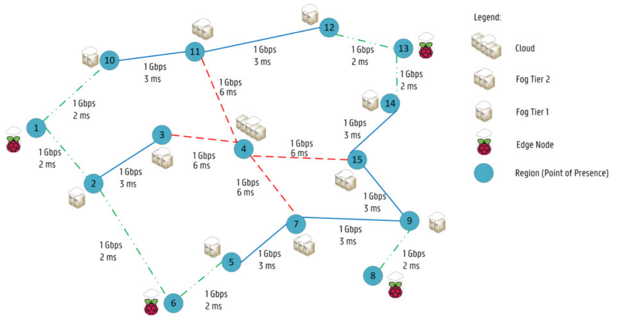


Table 5.6: Deployment properties of the evaluated applications.

App.	Pods & Number of Replicas	CPU (cpu)	RAM (Mi)	Band. (Mbps)
Basic (a_1)	P1 ($p_1^{a_1}$): $\{1,..,5\}$ P2 ($p_2^{a_1}$): $\{1,..,5\}$ P3 ($p_3^{a_1}$): $\{1,..,5\}$	1.0	1.0	500
Redis Cluster (a_2)	Master 1 ($p_1^{a_2}$): $\{1\}$ Master 2 ($p_2^{a_2}$): $\{1\}$ Master 3 ($p_3^{a_2}$): $\{1\}$ Slave 1 ($p_4^{a_2}$): $\{1,..,5\}$ Slave 2 ($p_5^{a_2}$): $\{1,..,5\}$ Slave 3 ($p_6^{a_2}$): $\{1,..,5\}$	1.0	1.0	250
Online Bout. (a_3)	Frontend ($p_1^{a_3}$): $\{1,2\}$ Cart ($p_2^{a_3}$): $\{1,2\}$ Prod. ($p_3^{a_3}$): $\{1,2\}$ Currency ($p_4^{a_3}$): $\{1,2\}$ Payment ($p_5^{a_3}$): $\{1,2\}$ Shipping ($p_6^{a_3}$): $\{1,2\}$ Email ($p_7^{a_3}$): $\{1,2\}$ Checkout ($p_8^{a_3}$): $\{1,2\}$ Recom. ($p_9^{a_3}$): $\{1,2\}$ Ad ($p_{10}^{a_3}$): $\{1,2\}$ Redis ($p_{11}^{a_3}$): $\{1,2\}$	1.0	1.0	250

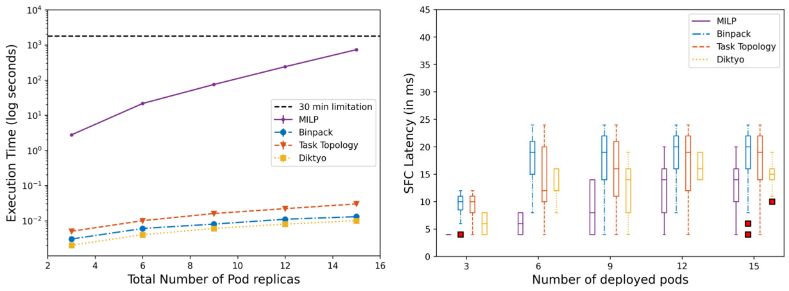
on a 6-core Intel i7-9850H CPU @ 2.6 GHz processor with 16 GB of memory. The model and the scheduling algorithms are evaluated 50 times and the results are shown with a 95% confidence interval.

5.6.1.2 Applications & Scheduling algorithms

Table 5.6 shows the experimental settings for the three evaluated applications. The **Basic** (a_1) application provides a naive service chain example previously shown in Fig. 5.3 to demonstrate the viability of the MILP model. Then, two practical use cases are assessed: 1) a **Redis Cluster** (a_2) database and 2) an **Online Boutique** (a_3) application as shown in Fig. 5.1. The simulation compares four scheduling algorithms: the optimal allocation scheme provided by the MILP model, the Volcano Binpack scheduling plugin, the Volcano Task Topology scheduling plugin, and the Diktyo framework.

Figure 5.5 The MILP model requires 12 and 30 minutes for 15 pods in the cluster and MR infrastructures for the basic application. The Diktyo framework reduces the expected SFC latency by up to 34% compared to existing plugin algorithms.

(a) Execution time (Cluster). **(b)** Network latency (Cluster).



(c) Execution time (MR). **(d)** Network latency (MR).

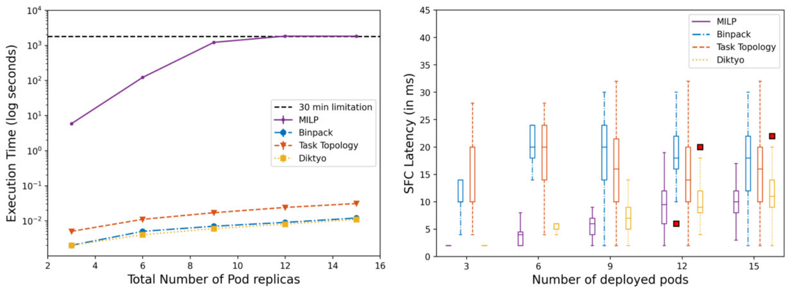


Figure 5.6 Deployment of the Redis cluster application (12 pods) with different topology constraints.

(a) Network Latency (Cluster). **(b)** Network Latency (DC).

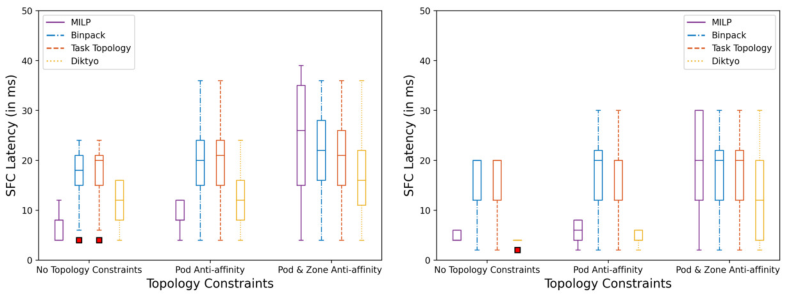
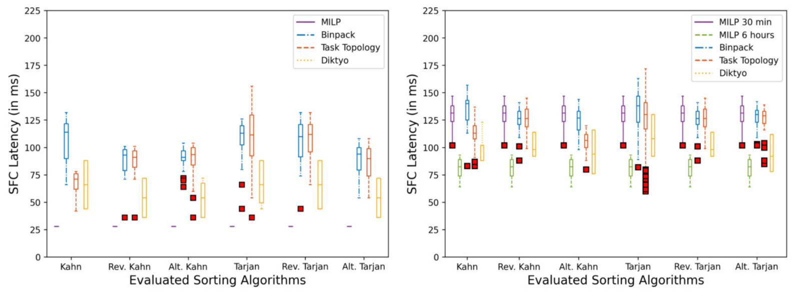


Figure 5.7 Deployment of the Online Boutique application (cluster topology) for different sorting algorithms.

(a) Network Latency (11 pods). **(b)** Network Latency (22 pods).



5.6.1.3 Simulation Results

Fig. 5.5a and Fig. 5.5c compare the execution time of the four scheduling methods to obtain the placement solution on both the multi-zone Cluster topology and the MR edge scenario. The MILP model provides an ideal optimal solution that typically cannot be applied in practice, as its execution time increases significantly as the number of pods in the application increase. As it is not acceptable to run scheduling for a long time in production systems, a 30-minute limitation has been added to the MILP model. Though the execution time of all algorithms increases due to the increasing number of pods, other heuristic scheduling algorithms only increase the execution time slightly compared to the MILP model. The MILP model requires 12 minutes and over 30 minutes to schedule 15 pods in the basic application on the cluster and MR topologies, respectively. In contrast, Diktyo needs only 0.01 seconds for both topologies, similar to the execution times of the Binpack and Task Topology algorithms. This occurs because all network weights are pre-calculated beforehand, and no recalculation is needed in the Diktyo plugins. Regarding network latency, Diktyo considerably reduces the expected SFC latency compared to the Binpack and Task Topology algorithms as shown in (Fig. 5.5b and Fig. 5.5d). It achieves reductions of up to 34% and 26% for the MR scenario with 15 pods compared to Binpack and Task Topology, respectively. Compared to the ideal optimal solution of the MILP model, Diktyo only increases the average SFC latency by 20% and 15% for 9 and 15 pods, respectively. However, it provides a scalable solution due to its lower execution time.

Fig. 5.6 evaluates the SFC latency in box plots for the Redis Cluster application under different topology constraints. The zone/region anti-affinity constraints are usually applied to provide high availability for Redis clusters. No particular topology sorting algorithm is applied in this scenario since all pods depend on all others in the Redis Cluster application. As shown, latency reductions by the Diktyo framework are more noticeable when pod and zone anti-affinity rules are not considered. Diktyo can reduce the latency up to 34% in the cluster (Fig. 5.6a) and up to 77% in the DC topology (Fig. 5.6b) compared to Binpack and Task Topology. Compared to the MILP model, Diktyo increases the average SFC latency by 37% in the cluster topology and reduces the expected latency by 22% in the DC topology as the MILP model fails to find the optimal solution within 30 minutes. Furthermore, even with additional topology constraints, the proposed framework can still reduce the latency up to 20% in the cluster topology and up to 38% in the DC topology compared to Binpack and Task Topology algorithms. The MILP model cannot find the optimal solution for both scenarios and obtains similar results to the Binpack algorithm due to the increased complexity of additional pod and zone topology constraints.

Fig. 5.7 evaluates the impact of different topology sorting algorithms for scheduling the Online Boutique application. The first scenario (Fig. 5.7a) considers the scheduling of 11 pods, in which the service chain has only one possible path since there is only one replica per pod type. As expected, the placement solution from the MILP model obtains the lowest latency (28 ms) while Diktyo produces a placement scheme 50% worse on average when combined with Alt. Kahn and Alt. Tarjan sorting algorithms. Nevertheless, Diktyo can still reduce the SFC latency up to 34% on average compared to Binpack and Task Topology algorithms. The second scenario (Fig. 5.7b) considers the scheduling of 22 pods, in which the service chain has several possible paths due to two replicas per pod type. Two allocation schemes are obtained from the MILP model in this scenario. The first experiment runs the model up to 30 minutes and the second up to 6 hours to search for the optimal placement scheme. However, the MILP model cannot find the optimal solution within 6 hours due to the highly complex service dependency graph for the Online Boutique application with 22 pods. Diktyo reduces the SFC latency up to 30% compared to the MILP 30-minute model. Also, it significantly reduces the expected latency compared to Binpack and Task Topology algorithms. It only increases the SFC latency by 13% compared to the MILP 6-hour model. Furthermore, Diktyo provides a much faster and more scalable solution as it solves the scenario on average in about 0.015 seconds.

In conclusion, the results show that the Diktyo scheduling can significantly reduce the expected SFC latency in Kubernetes clusters. Performance comparisons with two available scheduling plugins, Binpack and Task Topology, show that these scheduling strategies are not latency-aware. The MILP model shows that network-aware scheduling needs to solve a very complex optimization problem, which requires an unacceptable execution time. Thus, these methods are not practical to use in production systems. However, the Diktyo framework achieves similar execution times to Binpack and Task Topology by pre-calculating all network weights beforehand. On average, Diktyo obtains sub-optimal solutions 10 - 30% worse than the MILP model but provides a much faster and more scalable solution.

5.6.2 Plugin Evaluation

Figure 5.8 Benchmark of the Filter and Score plugins. The execution time of both plugins increases logarithmically over the number of nodes, though it is still below 1 second for 10000 nodes.

(a) Execution Time. **(b)** Number of Operations.

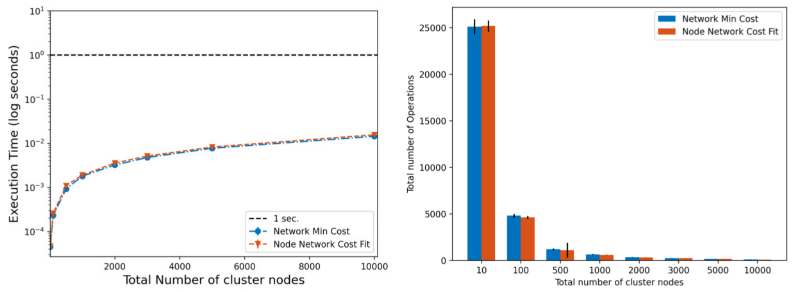
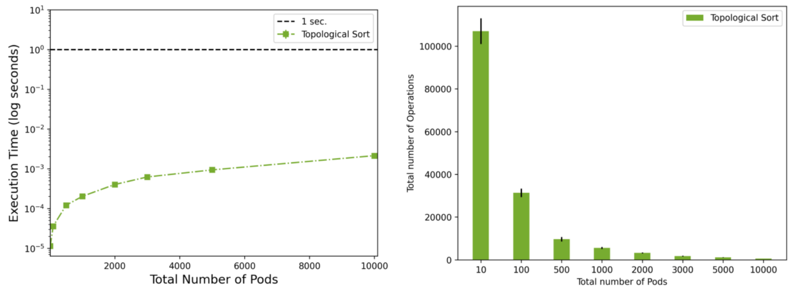


Figure 5.9 Benchmark of the QueueSort plugin. The execution time increases logarithmically over the number of pods.

(a) Execution Time. **(b)** Number of Operations.



The Go [86] testing package provides an integration testing utility that can be used to benchmark the performance of the scheduling plugins. Integration tests are implemented to assess the scalability of the plugins, including *NodeNetworkCostFit* and *NetworkMinCost*. Fig. 5.8 shows the plugins' execution time over the number of cluster nodes. Though the execution time of both plugins increases logarithmically over the number of nodes, the execution time for 10000 nodes is still below 1 second. It confirms that accessing network-aware information via CRs does not add significant overhead in terms of execution time to the Kubernetes scheduling process. Also, the *TopologicalSort* plugin is evaluated based on the number of pods in the sort queue as shown in Fig. 5.9. Results show a similar pattern (i.e., logarithmic time) as the other two plugins. This benchmark highlights that the plugins designed for the Diktyo framework do not introduce significant overhead over the scheduling process and is scalable for clusters with 10K nodes/pods.

5.6.3 Testbed Evaluation

5.6.3.1 Testbed Infrastructure

Figure 5.10 Illustration of the testbed infrastructure.

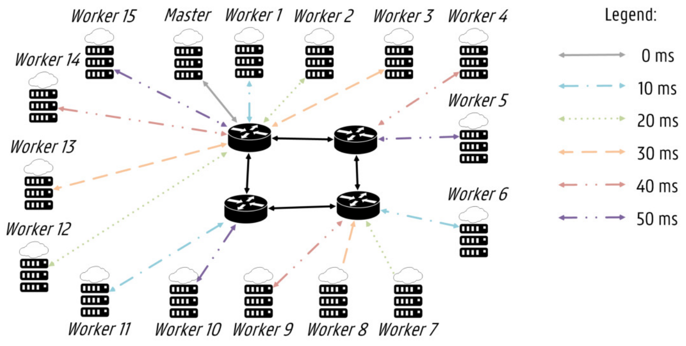


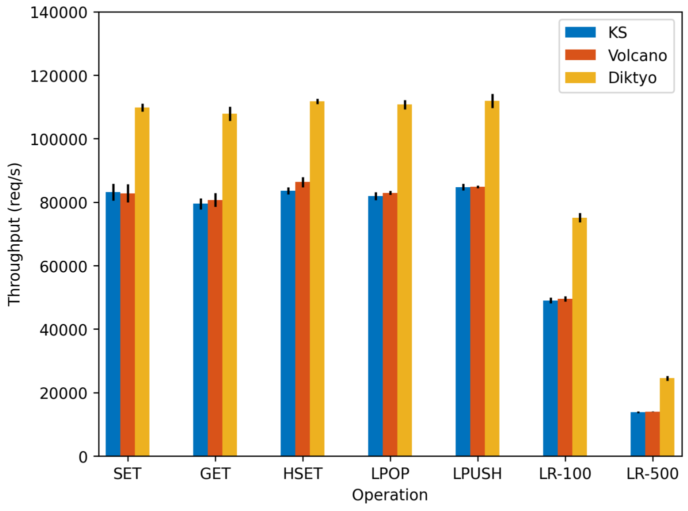
Table 5.7: Software Versions of the Testbed Infrastructure.

Software	**Version**
Kubeadm / Kubectl	v1.22.4
Go	go1.16.10
Docker	docker://20.10.10
Linux Kernel	5.4.0-80-generic
Operating System	Ubuntu 20.04.2 LTS

A Kubernetes cluster is set up using Kubeadm [87] on VMs created with IBM Cloud [88]. Network connections between VMs are illustrated in Fig. 5.10. The cluster consists of 16 nodes (1 master and 15 workers). Nodes are labelled with region (i.e., *topology.kubernetes.io/region*) and zone (i.e., *topology.kubernetes-.io/zone*) labels. All nodes are in the same region (i.e., r1), but each node belongs to a different zone (i.e., master, z1, .., z15). These topology labels are important to evaluate the scheduling behavior of Diktyo. Varying delays are emulated on network connections via Traffic Control (TC) [89]. The netperf component is deployed to estimate the latency between nodes in the cluster and caches the measurements in a configmap object. Then, the networkTopology controller calculates the network weights between zones based on the measurements in the configmap. Therefore, the proposed plugins can apply accurate network weights instead of manually defined weights. Table 5.7 lists the software versions of all the components used to set up the Kubernetes cluster.

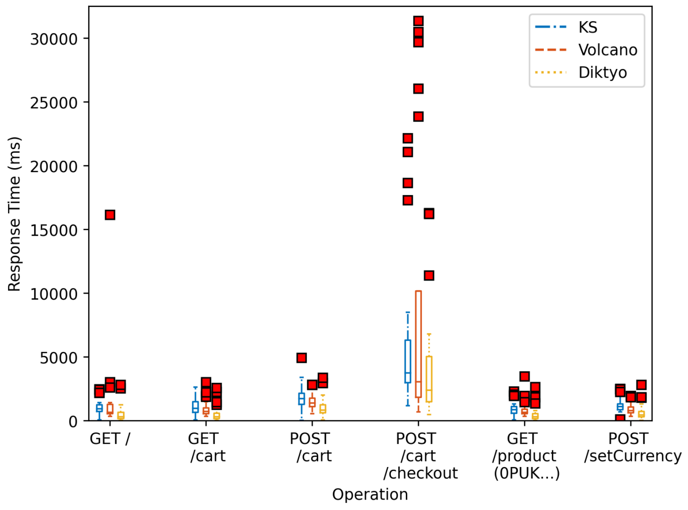
5.6.4 Testbed Results

Figure 5.11 Performance of a Redis Cluster application deployed using different schedulers. Diktyo increases database throughput up to 22% for most operations.



First, a Redis Cluster application is deployed with different schedulers. The deployment consists of five replicas for Redis-master pods and five for Redis-slave pods. To benchmark the performance of the Redis cluster, the Redis-benchmark utility [90] is used to generate a total of 250K database queries from 50 emulated clients. Fig. 5.11 shows the throughput obtained with different schedulers. As shown, the Diktyo framework achieves higher throughput on average by 22% compared to KS and Volcano since it allocates Redis-master and Redis-slave pods close to each other based on the dependencies established in the Redis AppGroup CR [91]. With the specification of pod dependencies, the Diktyo framework can produce an optimized placement for database application microservices than KS and Volcano, leading to higher throughput for various database operations.
Second, the Online Boutique application is deployed, and its load generator based on the locust load tool [92] is used to evaluate the performance concerning the application response time. Different *GET* and *POST* requests are executed as shown in Fig. 5.12. An Online Boutique AppGroup [93] composed with the dependencies shown in Fig. 5.1b is submitted via an AppGroup CR in the Kubernetes clus-

Figure 5.12 The response time for the Online Boutique application, deployed using different schedulers. Diktyo reduces the average response time up to 45% for most requests.



ter to establish all pod dependencies needed by Diktyo to find a near-optimal pod placement scheme. As shown, the Diktyo framework reduces the Online Boutique response time by 15% to 45% for most requests on average. This result highlights the main advantage of the Diktyo framework since it allocates pods with established dependencies close to each other, resulting in lower latency.

In summary, the achieved results show the benefits of the proposed Diktyo framework. By specifying dependencies among pods in Kubernetes clusters, scheduling algorithms can make informed decisions regarding latency and bandwidth. Application developers need to know their applications' microservice dependencies to define their AppGroup properly. Given information on microservice dependencies, the proposed three plugins combined in the Diktyo framework can produce a near-optimal scheduling solution to minimize the SFC latency with complex topology constraints in logarithmic time. The testbed experiments demonstrate that Diktyo increases database applications' throughput and reduces the expected response time for typical web-based applications in a distributed cluster.

5.7 Conclusions

This paper presented a network-aware scheduling approach for the Kubernetes platform based on its recent scheduling plugin architecture. The aim is to tackle the challenge of designing and implementing scalable and efficient network-aware scheduling algorithms capable of delivering low latency to end-users without compromising the system performance. Most efforts available in the literature require an unacceptable execution time to find proper allocation schemes. Results obtained from detailed experiments and realistic scenarios, show the advantages of the Diktyo framework compared to default schedulers, especially in terms of throughput and network latency. Diktyo can increase the throughput by 22% for typical database applications and reduce the expected response time by 45% in web-based applications. The work is an important step toward efficient service chain placement in future cloud-native architectures.

References

[1] Kinza Shafique, Bilal A Khawaja, Farah Sabir, Sameer Qazi, and Muhammad Mustaqim. Internet of things (iot) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5g-iot scenarios. *Ieee Access*, 8:23022–23040, 2020.

[2] Omer Adam, Young Choon Lee, and Albert Y Zomaya. Stochastic resource provisioning for containerized multi-tier web services in clouds. *IEEE Transactions on Parallel and Distributed Systems*, 28(7):2060–2073, 2016.

[3] Stefan Halfpap and Rainer Schlosser. A comparison of allocation algorithms for partially replicated databases. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 2008–2011. IEEE, 2019.

[4] Xiangbo Li, Mahmoud Darwich, Mohsen Amini Salehi, and Magdy Bayoumi. A survey on cloud-based video streaming services. In *Advances in Computers*, volume 123, pages 193–244. Elsevier, 2021.

[5] Brendan Burns, Joe Beda, and Kelsey Hightower. *Kubernetes: up and running: dive into the future of infrastructure*. O'Reilly Media, 2019.

[6] AMAZON AWS. Amazon web services. Accessed on 15 July 2021. [Online]. Available: https://aws.amazon.com/.

[7] Marko Luksa. *Kubernetes in action*. Simon and Schuster, 2017.

[8] Diana Popescu, Noa Zilberman, and Andrew Moore. Characterizing the impact of network latency on cloud-based applications' performance. 2017.

[9] Deval Bhamare, Raj Jain, Mohammed Samaka, and Aiman Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155, 2016.

[10] Boutheina Dab, Ilhem Fajjari, Mathieu Rohon, Cyril Auboin, and Arnaud Diquélou. Cloud-native service function chaining for 5g based on network service mesh. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2020.

[11] Khadija Aziz, Dounia Zaidouni, and Mostafa Bellafkih. Leveraging resource management for efficient performance of apache spark. *Journal of Big Data*, 6(1):1–23, 2019.

[12] Redis. Redis, an open source in-memory data structure store.,. Accessed on 9 September 2021. [Online]. Available: https://redis.io/.

[13] Yaser Mansouri, Victor Prokhorenko, and M Ali Babar. An automated implementation of hybrid cloud for performance evaluation of distributed databases. *Journal of Network and Computer Applications*, 167:102740, 2020.

[14] Online Boutique. Online boutique, a cloud-native microservices demo application. Accessed on 9 September 2021. [Online]. Available: https://github.com/GoogleCloudPlatform/microservices-demo.

[15] Luís Alexandre Rocha and Fábio Luciano Verdi. A network-aware optimization for vm placement. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pages 619–625. IEEE, 2015.

[16] Marwa A Abdelaal, Gamal A Ebrahim, and Wagdy R Anis. Network-aware resource management strategy in cloud computing environments. In *2016 11th International Conference on Computer Engineering & Systems (ICCES)*, pages 26–31. IEEE, 2016.

[17] Federico Larumbe and Brunilde Sansò. Elastic, on-line and network aware virtual machine placement within a data center. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 28–36. IEEE, 2017.

[18] Maryam Barshan, Hendrik Moens, Steven Latre, Bruno Volckaert, and Filip De Turck. Algorithms for network-aware application component placement for cloud resource allocation. *Journal of Communications and Networks*, 19 (5):493–508, 2017.

[19] Leonardo R Rodrigues, Marcelo Pasin, Omir C Alves, Charles C Miers, Mauricio A Pillon, Pascal Felber, and Guilherme P Koslovski. Network-aware container scheduling in multi-tenant data center. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

[20] Kubernetes Scheduler Plugins. Repository for out-of-tree scheduler plugins based on the scheduler framework. Accessed on 15 September 2021. [Online]. Available: https://github.com/kubernetes-sigs/scheduler-plugins.

[21] Kubernetes. Custom resources, . Accessed on 15 September 2021. [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/.

[22] Shivaram Venkataraman, Aurojit Panda, Ganesh Ananthanarayanan, Michael J Franklin, and Ion Stoica. The power of choice in data-aware cluster scheduling. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 301–316, 2014.

[23] Shaoqi Wang, Xiaobo Zhou, Liqiang Zhang, and Changjun Jiang. Network-adaptive scheduling of data-intensive parallel jobs with dependencies in clusters. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 155–160. IEEE, 2017.

[24] Zhiming Hu, James Tu, and Baochun Li. Spear: Optimized dependency-aware task scheduling with deep reinforcement learning. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 2037–2046. IEEE, 2019.

[25] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 285–300, 2014.

[26] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 595–610, 2018.

[27] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R Ganger, and Eric P Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, 2021.

[28] Andrew Chung, Subru Krishnan, Konstantinos Karanasos, Carlo Curino, and Gregory R Ganger. Unearthing inter-job dependencies for better cluster scheduling. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pages 1205–1223, 2020.

[29] Ionel Gog, Malte Schwarzkopf, Adam Gleave, Robert NM Watson, and Steven Hand. Firmament: Fast, centralized cluster scheduling at scale. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 99–115, 2016.

[30] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. {GRAPHENE}: Packing and dependency-aware scheduling for data-parallel clusters. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 81–97, 2016.

[31] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pages 481–498, 2020.

[32] Kubernetes Topology-aware Plugin. Topology-aware scheduler plugin implementations. Accessed on 15 July 2021. [Online]. Available: https://github.com/kubernetes-sigs/scheduler-plugins/tree/master/pkg/noderesourcetopology.

[33] Hamidreza Khaleghzadeh, Ravi Reddy Manumachu, and Alexey Lastovetsky. A hierarchical data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous multi-accelerator numa nodes. *IEEE Access*, 8:7861–7876, 2019.

[34] Alejandro Santoyo-González and Cristina Cervelló-Pastor. Network-aware placement optimization for edge computing infrastructure under 5g. *IEEE access*, 8:56015–56028, 2020.

[35] Fahad R Dogar, Thomas Karagiannis, Hitesh Ballani, and Antony Rowstron. Decentralized task-aware scheduling for data center networks. *ACM SIGCOMM Computer Communication Review*, 44(4):431–442, 2014.

[36] Chuangen Gao, Hua Wang, Linbo Zhai, Yanqing Gao, and Shanwen Yi. An energy-aware ant colony algorithm for network-aware virtual machine placement in cloud computing. In *2016 IEEE 22nd international conference on parallel and distributed systems (ICPADS)*, pages 669–676. IEEE, 2016.

[37] Hang Zhu, Kostis Kaffes, Zixu Chen, Zhenming Liu, Christos Kozyrakis, Ion Stoica, and Xin Jin. Racksched: A microsecond-scale scheduler for rack-scale computers. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pages 1225–1240, 2020.

[38] Amanda Jayanetti and Rajkumar Buyya. J-opt: A joint host and network optimization algorithm for energy-efficient workflow scheduling in cloud data

centers. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, pages 199–208, 2019.

[39] Hangwei Qian and Qixin Wang. Towards proximity-aware application deployment in geo-distributed clouds. *Advances in Computer Science and its Applications*, 2(3):382–386, 2013.

[40] Marzieh Malekimajd, Ali Movaghar, and Seyedmahyar Hosseinimotlagh. Minimizing latency in geo-distributed clouds. *The Journal of Supercomputing*, 71(12):4423–4445, 2015.

[41] Ehsan Ahvar, Shohreh Ahvar, Noel Crespi, Joaquin Garcia-Alfaro, and Zoltán Adám Mann. Nacer: a network-aware cost-efficient resource allocation method for processing-intensive tasks in distributed clouds. In *2015 IEEE 14th International Symposium on Network Computing and Applications*, pages 90–97. IEEE, 2015.

[42] Jad Darrous, Shadi Ibrahim, Amelie Chi Zhou, and Christian Perez. Nitro: Network-aware virtual machine image management in geo-distributed clouds. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 553–562. IEEE, 2018.

[43] Fabiana Rossi, Valeria Cardellini, Francesco Lo Presti, and Matteo Nardelli. Geo-distributed efficient deployment of containers with kubernetes. *Computer Communications*, 159:161–174, 2020.

[44] Dávid Haja, Balázs Vass, and László Toka. Towards making big data applications network-aware in edge-cloud systems. In *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, pages 1–6. IEEE, 2019.

[45] Jose Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Towards network-aware resource provisioning in kubernetes for fog computing applications. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 351–359. IEEE, 2019.

[46] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Towards delay-aware container-based service function chaining in fog computing. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2020.

[47] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[48] Yang Hu, Junchao Wang, Huan Zhou, Paul Martin, Arie Taal, Cees De Laat, and Zhiming Zhao. Deadline-aware deployment for time critical applications in clouds. In *European Conference on Parallel Processing*, pages 345–357. Springer, 2017.

[49] Angel Beltre, Pankaj Saha, and Madhusudhan Govindaraju. Kubesphere: An approach to multi-tenant fair scheduling for kubernetes clusters. In *2019 IEEE Cloud Summit*, pages 14–20. IEEE, 2019.

[50] Ataollah Fatahi Baarzi and George Kesidis. Showar: Right-sizing and efficient scheduling of microservices. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 427–441, 2021.

[51] Daniel Crankshaw, Gur-Eyal Sela, Xiangxi Mo, Corey Zumar, Ion Stoica, Joseph Gonzalez, and Alexey Tumanov. Inferline: latency-aware provisioning and scaling for prediction serving pipelines. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 477–491, 2020.

[52] Cloud Native Computing Foundation (CNCF) Volcano. Volcano, a batch system built on kubernetes, . Accessed on 15 July 2021. [Online]. Available: https://github.com/volcano-sh/volcano.

[53] Cloud Native Computing Foundation (CNCF) Volcano. The gang plugin, . Accessed on 15 October 2021. [Online]. Available: https://github.com/volcano-sh/volcano/tree/master/pkg/scheduler/plugins/gang.

[54] Cloud Native Computing Foundation (CNCF) Volcano. Task topology plugin, . Accessed on 15 October 2021. [Online]. Available: https://github.com/volcano-sh/volcano/tree/master/pkg/scheduler/plugins/task-topology.

[55] Cloud Native Computing Foundation (CNCF) Volcano. The binpack plugin, . Accessed on 15 October 2021. [Online]. Available: https://github.com/volcano-sh/volcano/tree/master/pkg/scheduler/plugins/binpack.

[56] Cloud Native Computing Foundation (CNCF) Volcano. Dominant resource fairness (drf) plugin, . Accessed on 15 October 2021. [Online]. Available: https://github.com/volcano-sh/volcano/tree/master/pkg/scheduler/plugins/drf.

[57] Bilgin Ibryam. Principles of container-based application design. *Redhat Consulting Whitepaper*, 2017.

[58] Kubernetes Cosheduling Plugin. coscheduling plugin implementations based on coscheduling based on podgroup crd. Accessed on 15 July 2021. [Online]. Available: https://github.com/kubernetes-sigs/scheduler-plugins/tree/master/pkg/coscheduling.

[59] Kubernetes. Scheduling framework., . Accessed on 15 October 2021. [Online]. Available: https://kubernetes.io/docs/concepts/scheduling-eviction/scheduling-framework/.

[60] Aun Haider, Richard Potter, and Akihiro Nakao. Challenges in resource allocation in network virtualization. In *20th ITC specialist seminar*, volume 18. ITC, 2009.

[61] Claudia D'Ambrosio, Andrea Lodi, and Silvano Martello. Piecewise linear approximation of functions of two variables in milp models. *Operations Research Letters*, 38(1):39–46, 2010.

[62] Kubernetes. Assigning pods to nodes, . Accessed on 9 September 2021. [Online]. Available: https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/.

[63] Kubernetes. Advertise extended resources for a node, . Accessed on 15 September 2021. [Online]. Available: https://kubernetes.io/docs/tasks/administer-cluster/extended-resource-node/.

[64] Anonymous authors. Bandwidth resources advertisement via extended resources, . Accessed on 9 December 2021. [Online]. Available: https://anonymous.4open.science/r/bandwidth-component-3DDF/README.md.

[65] Kubernetes. Network plugins, . Accessed on 15 September 2021. [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/.

[66] Xinxin Fan, Bo Lang, Yuan Zhou, and Tiarmmg Zang. Adding network bandwidth resource management to hadoop yarn. In *2017 seventh international conference on information science and technology (ICIST)*, pages 444–449. IEEE, 2017.

[67] Anonymous authors. App group crd yaml file, . Accessed on 9 December 2021. [Online]. Available: https://anonymous.4open.science/r/scheduler-plugins-1F07/manifests/appgroup/crd.yaml.

[68] Anonymous authors. App group controller, . Accessed on 9 December 2021. [Online]. Available: https://anonymous.4open.science/r/scheduler-plugins-1F07/pkg/controller/appgroup.go.

[69] Chaoyi Pang, Junhu Wang, Yu Cheng, Haolan Zhang, and Tongliang Li. Topological sorts on dags. *Information Processing Letters*, 115(2):298–301, 2015.

[70] Arthur B Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.

[71] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[72] Anonymous authors. Network topology crd yaml file, . Accessed on 9 December 2021. [Online]. Available: https://anonymous.4open.science/r/scheduler-plugins-1F07/manifests/networktopology/crd.yaml.

[73] Anonymous authors. The netperf component for the diktyo framework, . Accessed on 9 December 2021. [Online]. Available: https://anonymous.4open.science/anonymize/netperf-component-6C48.

[74] Netperf. a benchmark to measure the performance of many different types of networking. it provides tests for both unidirectional throughput, and end-to-end latency. Accessed on 15 September 2021. [Online]. Available: https://github.com/HewlettPackard/netperf.

[75] Kubernetes. Configmaps, . Accessed on 15 September 2021. [Online]. Available: https://kubernetes.io/docs/concepts/configuration/configmap/.

[76] Anonymous authors. Network topology controller, . Accessed on 9 December 2021. [Online]. Available: https://anonymous.4open.science/r/scheduler-plugins-1F07/pkg/controller/networktopology.go.

[77] Michael Hausenblas and Stefan Schimanski. *Programming Kubernetes: Developing Cloud-Native Applications.* " O'Reilly Media, Inc.", 2019.

[78] Anonymous authors. The topologicalsort plugin, . Accessed on 9 December 2021. [Online]. Available: https://anonymous.4open.science/r/scheduler-plugins-1F07/pkg/networkaware/topologicalsort/topologicalsort.go.

[79] Kubernetes. the qos plugin sorts pods by .spec.priority and breaks ties by the quality of service class., . Accessed on 15 October 2021. [Online]. Available: https://github.com/kubernetes-sigs/scheduler-plugins/tree/master/pkg/qos.

[80] Anonymous authors. The nodenetworkcostfit plugin, . Accessed on 9 December 2021. [Online]. Available: https://anonymous.4open.science/r/scheduler-plugins-1F07/pkg/networkaware/nodenetworkcostfit/nodenetworkcostfit.go.

[81] Anonymous authors. The networkmincost plugin, . Accessed on 9 December 2021. [Online]. Available: https://anonymous.4open.science/r/scheduler-plugins-1F07/pkg/networkaware/networkmincost/networkmincost.go.

[82] Kubernetes. Core v1 pod api., . Accessed on 2 December 2021. [Online]. Available: https://github.com/kubernetes/client-go/blob/master/listers/core/v1/pod.go.

[83] AMAZON EC2. Amazon ec2 on-demand pricing. Accessed on 15 July 2021. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/.

[84] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review*, 38(4):63–74, 2008.

[85] IBM ILOG. Ibm cplex ilog optimization studio. Accessed on 9 September 2021. [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio.

[86] Alan AA Donovan and Brian W Kernighan. *The Go programming language*. Addison-Wesley Professional, 2015.

[87] Kubernetes. Overview of kubeadm, . Accessed on 15 October 2021. [Online]. Available: https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm/.

[88] IBM Cloud. Hybrid. open. resilient. your platform and partner for digital transformation. Accessed on 15 October 2021. [Online]. Available: https://www.ibm.com/cloud.

[89] Alexey N. Kuznetsov. tc(8) — linux manual page. Accessed on 15 October 2021. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html.

[90] Redis. How fast is redis?, . Accessed on 2 December 2021. [Online]. Available: https://redis.io/topics/benchmarks.

[91] Anonymous authors. Redis app group crd yaml file, . Accessed on 9 December 2021. [Online]. Available: https://anonymous.4open.science/r/scheduler-plugins-1F07/manifests/appgroup/redis-appGroup-example.yaml.

[92] Locust. An open source load testing tool. Accessed on 2 December 2021. [Online]. Available: https://locust.io/.

[93] Anonymous authors. Online boutique app group crd yaml file, . Accessed on 9 December 2021. [Online]. Available: https://anonymous.4open.science/r/scheduler-plugins-1F07/manifests/appgroup/onlineBoutique-appGroup-example.yaml.

6 Conclusion

"The true sign of intelligence is not knowledge but imagination"

–Albert Einstein (1879 - 1955)

This dissertation presented several methods for efficient resource allocation in a Fog Computing architecture. Theoretical formulations, architectural paradigms, and practical implementations are among the various proposed approaches for efficient resource allocation of IoT services in Fog Computing. This chapter reviews the addressed challenges in this dissertation and presents future research directions in the resource allocation field, relevant in the coming years. In addition, App. C complements this chapter by discussing open challenges and research directions concerning low latency service delivery in next-generation networks focused on emerging use cases such as Extended Reality (XR).

⋆ ⋆ ⋆

6.1 Review of the Addressed Challenges

This section briefly summarizes the most important findings of this dissertation.

***Challenge #1**: provide a benchmark for resource allocation research in Fog Computing.*

To tackle this challenge, **Chapter 2** presented an Integer Linear Programming (ILP) formulation for IoT application service placement that considered multiple optimization objectives such as low latency and energy efficiency. Fog Computing adds further complexity to application deployments in a traditional cloud system. The heterogeneity of hardware resources, their availability at different levels in the network area, and the network bandwidth capacity are among the constraints considered in the model. Multiple factors have been considered in the performance analysis of resource allocation strategies, such as latency and deployment costs. Also, the model considered the implications of Low Power Wide Area Network (LPWAN) technologies in the allocation process to establish a relationship between the cloud and the wireless domain. The model has been validated for specific IoT applications within the scope of Antwerp's City of Things testbed. The evaluation showed clear trade-offs between the assessed allocation strategies.

Appendix B extends the ILP formulation presented in Chapter 2 by proposing a Mixed Integer Linear Programming (MILP) model for the IoT service placement problem that considers service chaining, different LPWAN technologies, service replication, and several optimization objectives. The formulation addresses recent advancements concerning Service Function Chaining (SFC) and micro-service deployments. The formulation presents further insights on the complete end-to-end (E2E) resource provisioning in Fog-cloud environments. Evaluations focused on Smart City use cases showed clear trade-offs between the assessed allocation strategies.

***Challenge #2**: design a Fog Computing framework supporting autonomous management and orchestration functionalities.*

Chapter 3 addressed this challenge by designing a Fog Computing framework for Smart Cities that followed the guidelines of the European Telecommunications Standards Institute (ETSI) Network Function Virtualization (NFV) Management and Orchestration (MANO) architecture. The approach proposed additional software components to provide a complete fog node management system to enable the life-cycle management of Smart City applications. Also, the chapter described a fog protocol for the exchange of application service information between FNs to provide fast service provisioning decisions.

Chapter 4 studied the feasibility of a well-known container management platform named Kubernetes for IoT deployments. The chapter proposed a Fog-based framework based on the Kubernetes architectural model.

***Challenge #3**: implement efficient distributed data monitoring and analysis in a*

Fog Computing environment.
Chapter 3 proposed an anomaly detection approach for 5G Smart Cities alongside the Fog Computing framework. The evaluation has shown how an anomaly detection use case for IoT could benefit from Fog Computing architectures. Results have demonstrated that the proposed framework achieves a significant reduction in terms of network bandwidth usage compared to traditional clouds.
Appendix A presented further insights on distributed data monitoring and analysis in Fog Computing. Popular LPWAN technologies have been investigated for the evaluated use case (i.e., air quality monitoring application), leading to a suitable set of LPWAN technologies: IEEE 802.11ah, DASH7, and LTE-M.
***Challenge #4**: consider latency and bandwidth in the scheduling process in a container orchestration system.*
IoT Applications would benefit from scheduling algorithms considering latency and bandwidth requirements. Current scheduling methods in popular cloud platforms focus mostly on optimizing resources (e.g., CPU and RAM), insufficient for applications where latency reduction and bandwidth optimization play a major role. **Chapter 4** addressed this challenge by developing a Network-Aware Scheduler (NAS) for container-based applications in Kubernetes. Evaluations have assessed the performance of the proposed scheduler compared to the ILP formulation presented in chapter 2 and the default scheduling component in Kubernetes. Results show that NAS achieves a reduction of up to 70% concerning network latency compared to the default mechanism. The work has been open-sourced and thus available for further experiments by the network management community.
Chapter 5 presented a complete network-aware framework for the Kubernetes platform inspired by service chaining concepts. The work extended Chapter 4 by designing and implementing a fast and scalable network-aware approach based on the Kubernetes scheduling plugin framework. Simulations show that the proposed framework can minimize network latency across different infrastructure topologies while achieving similar execution times as current scheduling plugins. Experiments with microservice benchmark applications in Kubernetes clusters show that the network-aware framework can increase database throughput by 22% and reduce the application's response time up to 45%. The work attracted attention from the Kubernetes scheduling community and is currently being integrated as part of their open-source project.

6.2 Future Challenges on Resource Allocation

As discussed above, this dissertation presented several improvements to optimize resource allocation in a Fog Computing environment. Nevertheless, novel challenges are emerging, which will need to be addressed by the network management community. This section briefly discusses four main research directions in the resource allocation field, which will become even more important in the next few years.

6.2.1 Towards cloud-native architectures

The advent of novel architectural paradigms enabled the deployment of service chains on computational resources from the cloud up to the edge, creating a continuum of virtual resources. The ambition is to set up a cloud infrastructure across the entire network area by placing resources at the edge and fog, enabling flexible deployments [1]. Architectural concepts for next-generation networks have been studied in Appendix C. The following paradigms will be of utmost importance: Multi-access Edge Computing (MEC) [2], Fog Computing [3], micro-services [4], and hardware acceleration [5].

Regarding networking advancements, SFC research will continue to evolve. This dissertation already addressed service chaining in several chapters by adding SFC concepts to the presented network-aware scheduling methods. In addition, Segment Routing (SR) [6] and Intent-based networking (IBN) [7] are interesting research directions to pursue towards truly cloud-native architectures. SR provides scalable and flexible routing mechanisms, simplifying traffic engineering. In fact, the combination of SFC and SR has already been proposed in the literature to obtain further flexibility in application deployments and optimize traffic flow in future cloud infrastructures [8, 9]. In addition, IBN has been conceptualized to communicate intents to the network. Enforcing rules without detailing the system how it should perform is its main goal. IBN can help to achieve the needed levels of reliability and scalability for emerging use cases. Also, another interesting direction is to study efficient auto-scaling [10] methods to guarantee that deployed services are sufficient to handle current network demand. SFC concepts in auto-scaling research are still quite unexplored.

In summary, the combination of these concepts is needed to obtain a more integrated solution concerning resource allocation in future distributed infrastructures.

6.2.2 Machine Learning-based and Artificial Intelligence-based orchestration

Machine Learning (ML) and Artificial Intelligence (AI) have positioned themselves as crucial enablers of autonomous networks [11]. ML and AI will automate several tasks currently being solved via human intervention. ML and AI-based methods will continue to evolve, especially in the areas of resource allocation [12, 13], network slicing [14], and privacy preservation [15]. Appendix C provides a comprehensive review on three ML research fields: Deep Learning (DL) [16], Reinforcement Learning (RL) [17] and Federated Learning (FL) [18]. DL and RL have already been applied for resource allocation focused on SFC placement [19], [20]. These methods have demonstrated their potential applicability in resource allocation due to their performance and scalability. However, DL and RL research is still far from finished. The applicability of DL and RL algorithms in operational environments is still quite limited since these methods require a high execution time for model training (offline), and their usage without training (online) is still not mature. Therefore, developing RL systems capable of reallocating services in the infrastructure by reacting to sudden network demands is a major research question in network management. Also, applying DL methods capable of identifying patterns on historical data based on previous service demands is another research direction.

In summary, all these ML domains will help achieve higher levels of independence in next-generation networks. The integration of these novel trends can lead to fully automated networks with minimum human intervention, providing self-configuration and self-repairing features that will strongly impact the performance of emerging use cases.

6.2.3 Low-Latency and High-Mobility service delivery

A plethora of allocation methods is proposed in this dissertation to tackle the stringent requirements of IoT applications. Chapter 2 presented a theoretical formulation for the IoT service placement problem, while chapters 4 and 5 proposed network-aware scheduling approaches for container-based service chains in the Kubernetes platform. However, emerging use cases will add further complexity to resource allocation due to their diverse and even more stringent requirements. Extended Reality (XR) [21] requires throughput levels above 1Tbps, while their interactive experiences need sub-millisecond latency. In contrast, autonomous cars [22] demand high mobility and at least seven levels of reliability without necessarily requiring higher throughput. Also, Industrial IoT (IIoT) [23] presents several challenges regarding latency and resilience guarantees. A sub-millisecond latency for these applications may not be enough to detect failures in manufacturing processes.

In summary, resource allocation research will continue to evolve to meet the demanding requirements of all these novel applications. Recent technologies such as IBN and SR in combination with SFC will help to achieve higher levels of flexibility and the referenced sub-millisecond latency.

6.2.4 Distributed and Decentralized Resource Allocation

In the last few years, several studies have addressed resource allocation or service scheduling in current orchestration systems [24]. However, few works have addressed multi-domain scenarios [25] or decentralized approaches [26]. Recent works have been studying how to optimize SFC orchestration across several domains [27]. Academia and Industry working in the network management field agree that a single centralized entity responsible for optimizing the life-cycle orchestration and management of SFCs across several domains might be difficult to converge at optimal strategies due to the increased complexity of the service placement problem. Thus, distributed and decentralized approaches are getting significant attention lately, where several entities cooperate to achieve efficient allocation strategies for service deployments.

In summary, distributed and decentralized approaches in the resource allocation field will become even more important in the next few years. Recent orchestration systems are developing multi-cluster management functionalities capable of efficiently managing service deployments across several clusters and domains.

References

[1] Shihabur Rahman Chowdhury, Mohammad A Salahuddin, Noura Limam, and Raouf Boutaba. Re-architecting nfv ecosystem with microservices: State of the art and research challenges. *IEEE Network*, 33(3):168–176, 2019.

[2] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[3] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H Glitho, Monique J Morrow, and Paul A Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 20(1):416–464, 2017.

[4] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, and Mike Amundsen. *Microservice architecture: aligning principles, practices, and culture.* " O'Reilly Media, Inc.", 2016.

[5] Leonardo Linguaglossa, Stanislav Lange, Salvatore Pontarelli, Gábor Rétvári, Dario Rossi, Thomas Zinner, Roberto Bifulco, Michael Jarschel, and Giuseppe Bianchi. Survey of performance acceleration techniques for network function virtualization. *Proceedings of the IEEE*, 107(4):746–764, 2019.

[6] Zahraa N Abdullah, Imtiaz Ahmad, and Iftekhar Hussain. Segment routing in software defined networks: A survey. *IEEE Communications Surveys & Tutorials*, 21(1):464–486, 2018.

[7] Engin Zeydan and Yekta Turk. Recent advances in intent-based networking: A survey. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5. IEEE, 2020.

[8] Francesco Paolucci. Network service chaining using segment routing in multi-layer networks. *Journal of Optical Communications and Networking*, 10(6):582–592, 2018.

[9] Francois Clad, Xiaoahu Xu, Alibaba Clarence Filsfils, Cisco Daniel Bernier, et al. Segment routing for service chaining. *Work in progress, IETF, Internet-Draft*, 2018.

[10] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of grid computing*, 12(4):559–592, 2014.

[11] Stephen S Mwanje and Christian Mannweiler. Towards cognitive autonomous networks in 5g. In *2018 ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K)*, pages 1–8. IEEE, 2018.

[12] Zaiwar Ali, Sadia Khaf, Ziaul Haq Abbas, Ghulam Abbas, Fazal Muhammad, and Sunghwan Kim. A deep learning approach for mobility-aware and energy-efficient resource allocation in mec. *IEEE Access*, 8:179530–179546, 2020.

[13] Feibo Jiang, Kezhi Wang, Li Dong, Cunhua Pan, Wei Xu, and Kun Yang. Deep learning based joint resource scheduling algorithms for hybrid mec networks. *IEEE Internet of Things Journal*, 2019.

[14] Giuseppe Faraci, Christian Grasso, and Giovanni Schembra. Design of a 5g network slice extension with mec uavs managed with reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 38(10):2356–2371, 2020.

[15] Chunyi Zhou, Anmin Fu, Shui Yu, Wei Yang, Huaqun Wang, and Yuqing Zhang. Privacy-preserving federated learning in fog computing. *IEEE Internet of Things Journal*, 2020.

[16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[17] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[18] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[19] Jianing Pei, Peilin Hong, and Defang Li. Virtual network function selection and chaining based on deep learning in sdn and nfv-enabled networks. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2018.

[20] Doyoung Lee, Jae-Hyoung Yoo, and James Won-Ki Hong. Deep q-networks based auto-scaling for service function chaining. In *2020 16th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2020.

[21] Åsa Fast-Berglund, Liang Gong, and Dan Li. Testing and validating extended reality (xr) technologies in manufacturing. *Procedia Manufacturing*, 25:31–38, 2018.

[22] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius Brito Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago Meireles Paixão, Filipe Mutz, et al. Self-driving cars: A survey. *Expert Systems with Applications*, page 113816, 2020.

[23] Charith Perera, Chi Harold Liu, Srimal Jayawardena, and Min Chen. A survey on internet of things from industrial market perspective. *IEEE Access*, 2: 1660–1679, 2014.

[24] Song Yang, Fan Li, Stojan Trajanovski, Ramin Yahyapour, and Xiaoming Fu. Recent advances of resource allocation in network function virtualization. *IEEE Transactions on Parallel and Distributed Systems*, 32(2):295–314, 2020.

[25] Panagiotis Karamichailidis, Kostas Choumas, and Thanasis Korakis. Enabling multi-domain orchestration using open source mano, openstack and opendaylight. In *2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–6. IEEE, 2019.

[26] André Pires, José Simão, and Luís Veiga. Distributed and decentralized orchestration of containers on edge clouds. *Journal of Grid Computing*, 19(3): 1–20, 2021.

[27] Nassima Toumi, Olivier Bernier, Djamal-Eddine Meddour, and Adlen Ksentini. On cross-domain service function chain orchestration: An architectural framework. *Computer Networks*, 187:107806, 2021.

A
Anomaly detection for Smart City applications over 5G Low Power Wide Area Networks

This appendix extends the work described in Chapter 3 by focusing on a Fog-based distributed data monitoring and analysis approach. It presents further insights based on an anomaly detection solution for Smart City applications designed for Antwerp's City of Things testbed. Popular LPWAN technologies have been investigated for the anomaly detection approach, and the most appropriate ones are selected based on the evaluated use case. Results show that clustering and outlier detection mechanisms can be performed by fog resources close to IoT sensors and, thus, send timely alerts in case unusual events are detected. Therefore, the main contributions of this appendix are the distributed anomaly detection approach, including its design and implementation and further LPWAN evaluation.

⋆ ⋆ ⋆

José Santos, Philip Leroux, Tim Wauters, Bruno Volckaert and Filip De Turck.

Published in Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 1-9, 2018, IEEE.

Abstract In recent years, the Internet of Things (IoT) has introduced a whole new set of challenges and opportunities in Telecommunications. Traffic over wireless networks has been increasing exponentially since many sensors and everyday devices are being connected. Current networks must therefore adapt to and cope with the specific requirements introduced by IoT. One fundamental need of the next generation networked systems is to monitor IoT applications, especially those dealing with personal health monitoring or emergency response services, which have stringent latency requirements when dealing with malfunctions or unusual events. Traditional anomaly detection approaches are not suitable for delay-sensitive IoT applications since these approaches are significantly impacted by latency. With the advent of 5G networks and by exploiting the advantages of new paradigms, such as Software-Defined Networking (SDN), Network Function Virtualization (NFV) and edge computing, scalable, low-latency anomaly detection becomes feasible. In this appendix, an anomaly detection solution for Smart City applications is presented, focusing on low-power Fog Computing solutions and evaluated within the scope of Antwerp's City of Things testbed. Based on a collected large dataset, the most appropriate Low Power Wide Area Network (LPWAN) technologies for the Smart City use case are investigated.

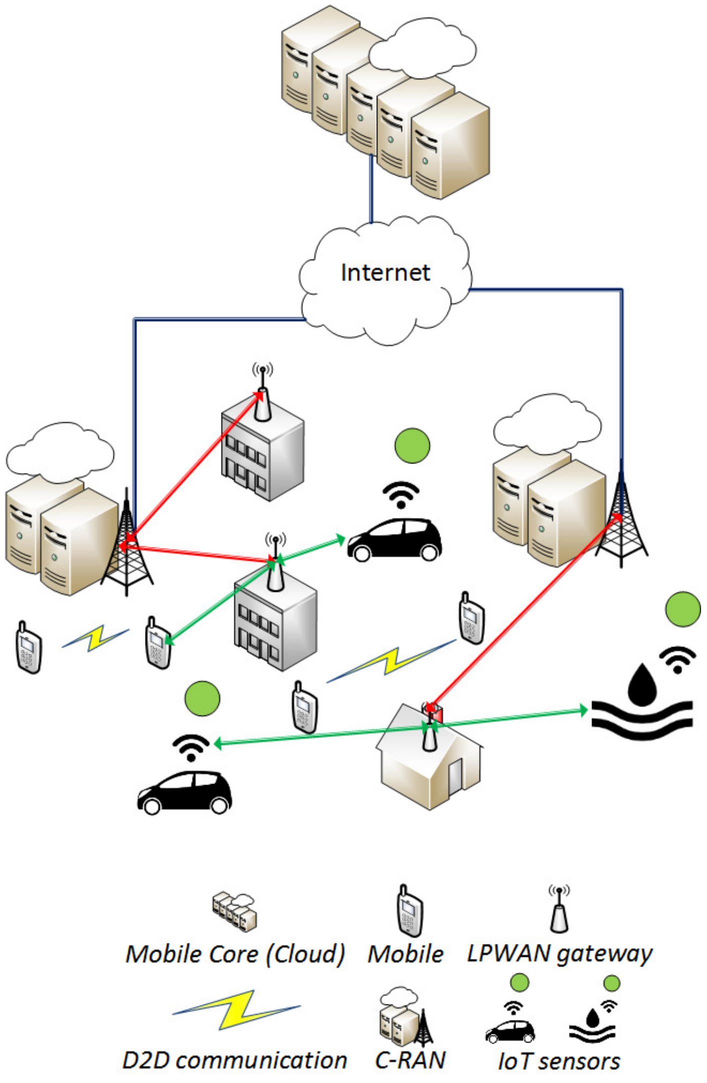
A.1 Introduction

In recent years, with the advent of the Internet of Things (IoT), the concept of Smart Cities has become even more popular [1]. IoT will transform a wide range of services in different domains of urban life, by creating intelligent smart grids, improving public transportation and developing car parking and personal health monitoring applications. In the future network generation, information will be transmitted from different types of devices, over heterogeneous wireless networks with even higher data rates, lower latencies and lower power consumption [2]. Therefore, it will be necessary to adapt existing network architectures to future needs and develop new autonomous management functionalities to help meet the demanding requirements of future 5G use cases. In fact, 5G technologies promise very high carrier frequencies with massive bandwidths, extreme base station densities, an unprecedented large numbers of antennas and new functionalities, such as device-to-device communication (D2D) and Fog Computing [3], [4]. In Fig. A.1, a 5G network architecture is presented in a Smart City context. 5G technologies aim to tackle the new business opportunities created by the stringent requirements of IoT applications. One of the main challenges is how to efficiently handle with the gathering and processing of all data coming from the enormous amount of IoT sensors that will be connected to the network in the next years [5].

The Fog Computing paradigm, which places cloud resources close to the IoT sensors, extends the Cloud Computing paradigm to deal with the eminent growth of connected devices [6]. Nevertheless, Fog Computing is still in its early stages and needs more time to evolve. One of the remaining challenges is how to provide proper resource allocation, since IoT applications and services can be placed in a highly congested area, which would result in a higher latency [7]. Furthermore, current IoT sensors and gateways lack in terms of processing power, battery, memory and storage capacity [8, 9]. IoT applications will be so diverse that they will have different sets of communication requirements. For instance, on one hand, a delay-sensitive IoT application may require very low latencies, meaning this IoT application must be allocated on fog resources close to the sensor enabling the control of time-sensitive network functionalities close to the device [10]. On the other hand, if this requirement is less important, the IoT application could be placed far from the IoT sensor in a central location in order to reduce the number of active fog resources on the network and therefore minimize the total energy consumption in the fog domain. Additionally, it is also important to detect malfunctions and abnormal events in the network. By identifying unusual events, malfunctions in IoT sensors can be detected and transmissions of incorrect information can be avoided, which can improve the overall Quality of Service (QoS) of the IoT application, especially in terms of reliability [9]. Detecting unexpected patterns in the data traffic is known as anomaly detection [11]. Recently, anomaly detection has attracted the attention of the research community in multiple areas, such as intrusion detection, health monitoring, preventive maintenance and fault detection [12]. In this appendix, an anomaly detection approach for IoT applications in 5G Smart Cities based on the advantages of Fog Computing architectures is presented. The proposed architecture has been designed for Antwerp's City of Things testbed [13] and validated for Smart City use cases, in particular for an Air Quality monitoring application. Finally, multiple Low Power Wide Area Network (LPWAN) technologies have been considered for the use case scenario. The evaluation results identify the most adequate LPWAN technologies as wireless communication enablers for the considered IoT application.

The remainder of this appendix is organized as follows. In the next section, related work is discussed. In section III, the anomaly detection approach for smart city environments as well as the LPWAN network dimensioning is presented. Then, in Section IV, the scenario and datasets used in the evaluation are presented, followed by the evaluation results in section V. Finally, conclusions are presented in section VI.

Figure A.1 High-level view of the considered 5G network architecture.



A.2 Related Work

Recently, research studies have been carried out in order to deal with anomaly detection in IoT, Smart City and Industry 4.0 scenarios. In [14], a real-time Intrusion Detection System (IDS) for IoT has been presented. The proposed solution is a novel IDS with an integrated mini-firewall for 6LoWPAN networks in order to detect malicious nodes. Moreover, in [11], an anomaly detection scheme based on sensor data has been proposed to deal with unexpected behaviors in turbomachines in the Petroleum Industry. Furthermore, in [15], a temporal clustering and anomaly detection method has been presented for a car parking IoT application in order to detect unusual events. In [16], a supervised statistical-based anomaly detection method for Smart Grid data has been proposed.

In recent years, research projects have also been focusing on reliable and secure IoT for Smart Cities. In the SOCIOTAL project [17], an anomaly detection method based on hyperellipsoidal models has been used to identify unusual patterns in environmental data collected from IoT sensors [18]. However, a traditional cloud solution has been deployed instead of a Fog Computing approach. Additionally, in [9], a Fog Computing anomaly detection approach for IoT using a hyperellipsoidal clustering algorithm has been proposed to significantly reduce latency and energy consumption in the network when compared to distributed and centralized architectures. Nevertheless, their work is based on simulation studies, while the proposed approach is based on an actual deployment within the scope of Antwerp's City of Things testbed. Finally, in the CityPulse project [19], [20], a complete set of real-time data analytics tools have been presented, such as data aggregation, event detection and decision support.

In summary, in this appendix, a Fog-based anomaly detection approach is proposed. The work takes into account not only the advantages of Fog Computing architectures, which are suitable for IoT applications in Smart City scenarios, but also characteristics stemming from LPWAN technologies, since low power technologies have gained tremendous emphasis due to the enormous growth of connected devices. All Anomaly detection approaches cited are only focused on cloud and data management aspects and no considerations are included about wireless networks. The proposed approach has been implemented on the City of Things platform and evaluated with Unsupervised Clustering and Outlier detection algorithms for the Air Quality monitoring application. Furthermore, the most popular LPWAN technologies today have been assessed based on the requirements of the application.

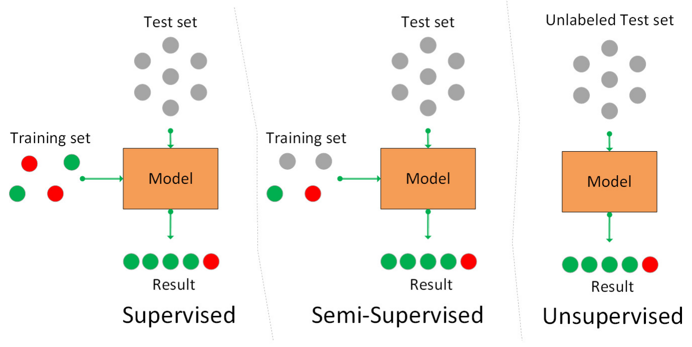
A.3 Anomaly Detection in Smart Cities

This section presents the proposed low-latency anomaly detection approach for Smart Cities on Fog Computing resources interconnected by LPWAN networks.

A.3.1 Anomaly Detection Principles

Anomaly detection or outlier detection is known as the process of detecting unexpected behavior or abnormal patterns in datasets. In the past, anomaly detection was mainly used to remove the outliers from a dataset, which is named data cleansing. However, in recent years, anomaly detection has attracted the attention of the research community because researchers began to get interested in knowing more about the anomalies themselves, since they are usually associated with potentially reoccurring events [21]. There are three main categories of anomaly detection which are shown in Fig. A.2.

Figure A.2 Anomaly Detection Categories: Supervised, Semi-supervised and Unsupervised.



A.3.1.1 Supervised

In supervised anomaly detection methods, a fully labeled training dataset is used, i.e., each sample is considered as normal or abnormal. This category is only used for specific applications where anomalies are known beforehand.

A.3.1.2 Semi-Supervised

In semi-supervised anomaly detection techniques, a training dataset consists of labeled and unlabeled samples. Usually, a small amount of labeled samples is used.

A.3.1.3 Unsupervised

In unsupervised anomaly detection algorithms, there is no training dataset since nothing is known about the samples in advance. Therefore, these methods usually give an estimation of what a normal sample and what an abnormal one is. The approach presented in this appendix makes use of unsupervised techniques since the goal was to see if the selected algorithms could learn the distribution of the data samples without knowing anything beforehand as in most anomaly detection use cases. The objective was that the algorithms themselves could discover and present interesting behaviors in the datasets. Moreover, labeling datasets for anomaly detection is not an easy task.
Many categories of unsupervised anomaly detection algorithms exist, of which the most popular are listed in Table A.1. Many of these algorithms are available in Scikit-Learn [22], a powerful machine learning library written in Python, which has been used to implement the anomaly detection evaluations.

Table A.1: Unsupervised Anomaly Detection Algorithms [23]

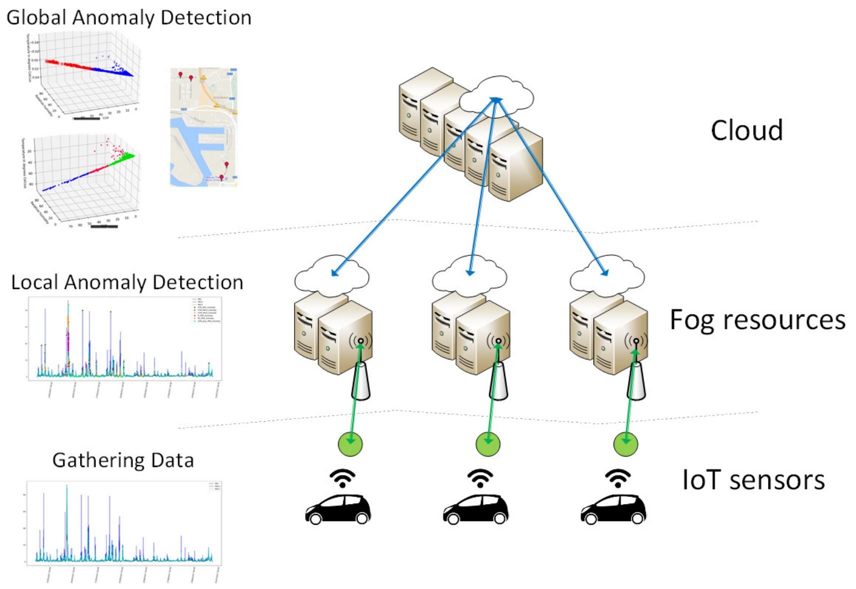
Statistical-based	Univariate and Multivariate Gaussian distribution, Grubbs' test, Likelihood approach
Proximity-based	K Neighbors
Clustering-based	K-Means, MiniBatchKMeans, Birch
Density-based	Local Outlier Factor (LOF)
One-class support vector machines	One Class SVM, Gaussian envelope (Robust Covariance)
Ensemble-based	Isolation Forrest (IF)

A.3.2 Fog-based Anomaly Detection Approach

To deal with the growing amount of connected devices in the network, the Fog Computing paradigm has been introduced to place computational resources on the edges of the network in order to deal with the stringent requirements introduced by IoT use cases, such as low latency and high mobility. Centralized solutions are not suitable for most future IoT applications, since most of them will require real-time communication and generate an enormous volume of data to be transported in the network, which makes it impossible for centralized solutions to comply with these requirements. The IoT sensors communicate with wireless gateways, which are linked with the fog resource layer, managing a set of computational resources. These fog resources can communicate with the cloud layer, which is the top level management entity. Each service or IoT application must be allocated to and instantiated on a given set of computational resources. For instance, for a delay-sensitive IoT application, service allocation must be performed on a fog resource as close as possible to the IoT sensor that runs the client application allowing real-time processing and data analytics at the edges of the network in order to enable the control of time-sensitive network functionalities close to the IoT sensor.

In a traditional centralized approach, all IoT sensors send their data samples to the cloud layer. Then, the anomaly detection algorithms are executed. This approach implies a high bandwidth cost, because all data samples need to be transmitted from the IoT sensors to the cloud layer. The approach presented in Fig. A.3 is based on the advantages of Fog Computing architectures, i.e., anomaly detection operations are performed on fog resources. Every IoT sensor sends its data samples to one of the fog resources. Then, anomaly detection operations are performed in a distributed way. After completion of the anomaly detection tasks, fog resources may send alerts to the cloud layer and to the IoT sensors if unusual events are already detected on the data. This way, faster response times can be achieved if any abnormal behavior is discovered. Moreover, fog resources can send the outcomes of the anomaly operations to the cloud layer to combine results from the different fog resources in order to have a broader view of the behavior of the network. Afterwards, the cloud layer could perform global anomaly detection operations and present the outcomes in a central dashboard in a control room. Then, alerts can be sent to fog resources and IoT sensors in case abnormal patterns or inconsistent events were not detected.

Figure A.3 High-level view of the Fog-based anomaly detection approach.



A.3.3 LPWAN Dimensioning

Nowadays, low power wireless technologies have gained tremendous emphasis due to the massive growth of connected devices in the network. The need for connecting simple IoT devices, such as sensors and actuators, is increasing rapidly. In Table. A.2, the most popular LPWAN technologies and the main differences between them are shown. To select a suitable LPWAN technology for a specific IoT application, an analysis of its requirements in terms of specific parameters such as communication range, upload and download data rate, frequency bands, and latency is needed. In this appendix, multiple LPWAN technologies have been evaluated based on the requirements of the Air Quality monitoring use case presented in Section A.4.1.

Variables used in the LPWAN dimensioning are shown in Table A.3. In Fog Computing architectures, fog resources are usually located within one hop from the IoT sensors. The variable C is used to indicate the communication range in kilometers between a fog resource and an IoT sensor. Two variables, U and D, are used to indicate the upload and the download data rate, respectively. Then, the total number of bits to be transmitted is given by N. This way, the upload and the download transmission time of a packet can be expressed as shown in (A.1) and in (A.2), respectively.

$$T(upload) = \frac{N}{U} \tag{A.1}$$

$$T(download) = \frac{N}{D} \tag{A.2}$$

Moreover, by using the communication range C and the propagation speed P for wireless communications, which is the speed of light (3×10^8), the propagation time is given by (A.3).

$$P = \frac{C}{3 \times 10^8} \tag{A.3}$$

Finally, the total packet delivery time L is given by (A.4).

$$L = T + P \tag{A.4}$$

A.4 Evaluation Scenario

In this section, the evaluation scenario is introduced. Then, the datasets are presented. Finally, the evaluation setup is described.

Table A.2: Comparison between different LPWAN technologies for IoT applications [24], [25], [26], [27]

LPWAN Technology	LoRaWAN	Sigfox	LTE-M	DASH7	IEEE 802.11ah	NB-IoT	Ingenu RPMA
Range urban	2-5 km	3-10 km	2-5 km	5 km	1 km	2-5 km	1-3 km
Range rural	15 km	30-50 km	-	5 km	1 km	-	25-50 km
Data rate	50 kbps	300bps	1 Mbps	166.67 kbps	346.66 Mbps	250 kbps	634 kbps (uplink) 156 kbps (downlink)
Bi-directional	Yes	Limited	Yes	Yes	Yes	Yes	Yes
Freq. band	Unlicensed	Unlicensed	Licensed	Unlicensed	Unlicensed	Licensed	Unlicensed
Power efficiency	Very high	Very high	Medium	Medium	High	Medium	Medium
Security	Medium	Low	High	Medium	Low (In development)	High	High
Mobility	Yes	Limited	Yes	Yes	Yes	Yes	Yes
Proprietary	No	Yes	No	No	No	No	Yes

Table A.3: Variables of the LPWAN dimensioning

Symbol	Description
C	Communication range in kms.
U	Upload Data Rate in kbps.
D	Download Data Rate in kbps.
N	Number of bits to be transmitted.
T	Transmission time of a packet.
P	Propagation time of a packet.
L	Packet Delivery time.

A.4.1 Use Case - Air Quality Monitoring Application

The evaluation scenario is based on a use case within the scope of Antwerp's City of Things testbed. The goal of the Air Quality monitoring application is to detect high amounts of organic compounds in the atmosphere and then alert citizens of air pollution in real-time. As an initial proof of concept, Air Quality sensors have been integrated in collaboration with the Belgian postal services Bpost [13]. For daily mail delivery, Bpost has cars driving around in the city of Antwerp. Therefore, within the City of Things project, a set of Air Quality sensors have been mounted on the roofs of Bpost's delivery cars as shown in Fig. A.4. These sensors send measures of typical gases and climate data such as temperature and humidity, which are then annotated with GPS locations. Moreover, these sensors allow gathering real-time Air Quality information with broad city coverage, since each car is continuously driving around in the city. Currently, the set of Air Quality sensors can communicate via three different LPWAN technologies: LoRaWAN, SigFox and DASH7 [13].

A.4.2 Datasets

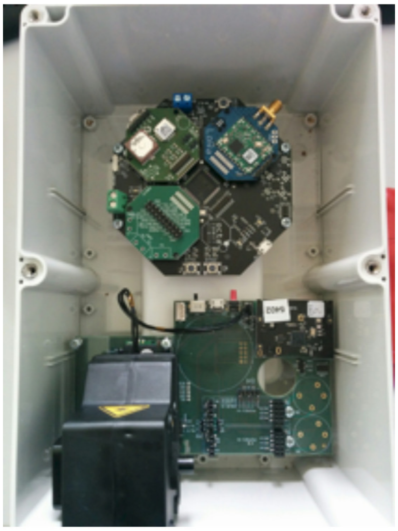
A summary of the characteristics of the datasets gathered for the evaluation is shown in Table A.4. The two datasets come from two different Bpost cars and consist of particle matter indicators (PM1, PM2.5 and PM10) that are annotated with a GPS location. The datasets have been collected by our research group between 2017-05-09 and 2017-06-29. The particle matter indicators are shown in Fig. A.5a and in Fig. 3.10b for Bpost car 1 and Bpost car 2, respectively.

A.4.3 Selected Algorithms

As previously mentioned, the anomaly detection evaluations have been implemented in Python using Scikit-Learn. Unsupervised Clustering and Outlier detection algorithms have been assessed by using the two datasets presented in Section A.4.2. Clustering allows the detection of patterns in unlabeled data with many

Figure A.4 As part of the Antwerp's City of Things testbed, multi-radio Air Quality sensors have been mounted on cars of the Belgian postal service.

(a) Inside view of the multi-radio sensor.



(b) Air Quality sensor mounted on a Bpost car.

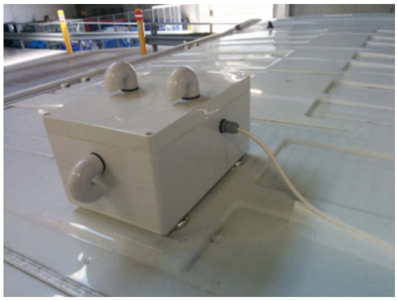
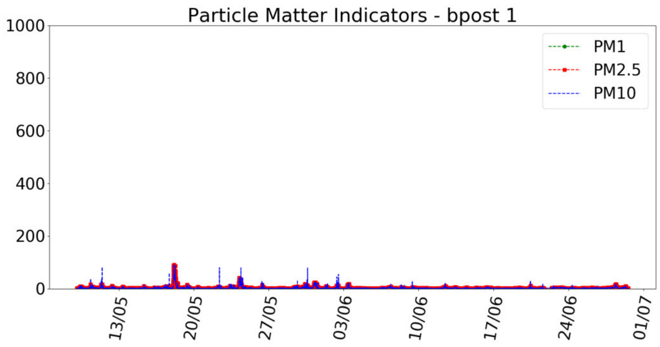


Table A.4: Evaluation datasets

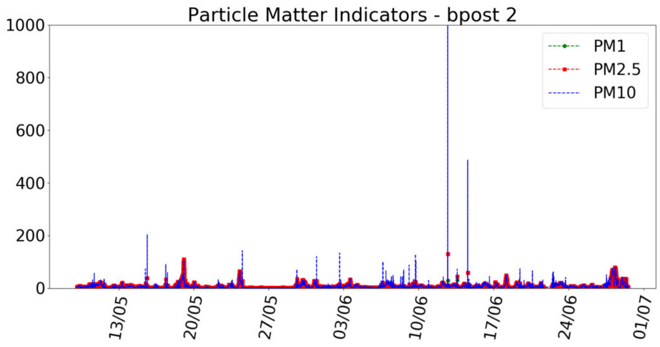
Dataset Name	No. of records	Description
Bpost 1	70636	Particle matter indicators (PM1, PM2.5, PM10) and GPS locations from Bpost car 1 between 2017-05-09 and 2017-06-29
Bpost 2	70640	Particle matter indicators (PM1, PM2.5, PM10) and GPS locations from Bpost car 2 between 2017-05-09 and 2017-06-29

Figure A.5 Particle Matter Indicators (PM1, PM2.5, PM10 in ppm) for two Bpost cars.

(a) Particle Matter Indicators (PM1, PM2.5, PM10) - Bpost car 1.



(b) Particle Matter Indicators (PM1, PM2.5, PM10) - Bpost car 2.



dimensions while Outlier detection is related to the identification of unusual data samples when compared to the rest of the dataset. Regarding Clustering, the Birch algorithm has been evaluated while for Outlier detection Robust Covariance (RC) has been assessed. Birch and RC outcomes have been compared in order to find patterns or unusual events in the datasets. Furthermore, the results have been compared with the correspondent GPS locations to know exactly where in the city of Antwerp each sample has been measured.

A.5 Evaluation Results

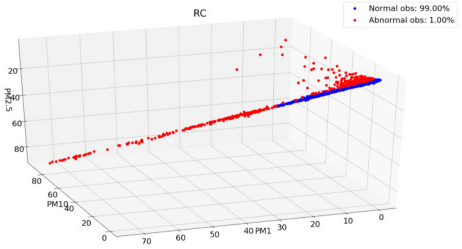
A.5.1 Clustering and Outlier Detection

In Fig. A.6a and in Fig. A.6b, the outcomes of the RC outlier detection algorithm for the three dimensions regarding particle matter indicators for the Bpost car 1 are shown with a contamination of 1.0% indicating that the RC algorithm intends to find the 1.0% of samples which can be considered as abnormal. Moreover, in Fig. A.6c and in Fig. A.6d, the results obtained for the Bpost car 1 by using the Birch clustering algorithm for 5 clusters are illustrated. Regarding outlier detection, values of PM1, PM2.5 and PM10 above 30 ppm collected by Bpost car 1 are marked as outliers meaning that these values can be considered as unusual. Regarding clustering, there are clear similarities between these results and the outcomes obtained by the RC outlier detection algorithm. The first cluster is composed of almost 99.4% of the total data samples indicating that this cluster can be considered as the normal region of values for the three particle matter indicators collected by Bpost car 1. Moreover, the second and the fifth cluster, consist of 0.5% of the data samples which are also detected as outliers by the RC algorithm. These clusters can be considered as unusual regions. Finally, the third and the fourth cluster, are composed of 0.1% of data samples which are also detected as outliers by the RC algorithm. These clusters can therefore be considered as very unusual regions.

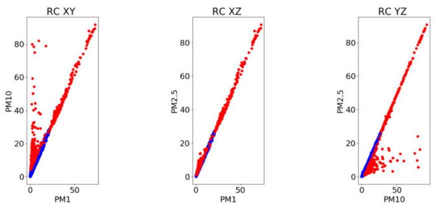
In Fig. A.7a and in Fig. A.7b, the outcomes of the RC algorithm with a contamination of 1.0% for the three dimensions regarding particle matter indicators for the Bpost car 2 are shown. In Fig. A.7c and in Fig. A.7d, the results obtained for the Bpost car 2 by using the birch clustering algorithm for 5 clusters are illustrated. Regarding outlier detection, values of PM1 above 60 ppm, PM2.5 above 70 ppm and values of PM10 above 150 ppm collected by Bpost car 2 are marked as outliers. This way, these values can be considered as unusual data samples. Regarding clustering, there are clear similarities between these results and the outcomes obtained by the RC outlier detection algorithm, as shown for the Bpost car 1.

Figure A.6 Robust Covariance Outlier detection with a contamination of 1.0% (blue color: normal samples, red color: abnormal samples) and Birch Clustering results with 5 clusters (blue, red, green, brown, pink) for Particle Matter Indicators (PM1, PM2.5, PM10) - Bpost car 1.

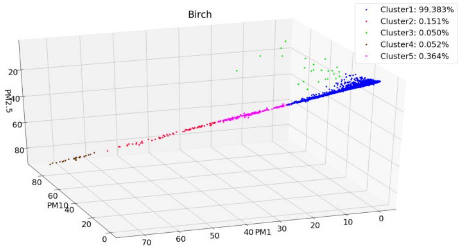
(a) RC with 1.0% - 3D perspective - Bpost car 1



(b) RC with 1.0% - 3D planes - Bpost car 1



(c) Birch: 5 clusters - 3D perspective - Bpost car 1



(d) Birch: 5 clusters - 3D planes - Bpost car 1

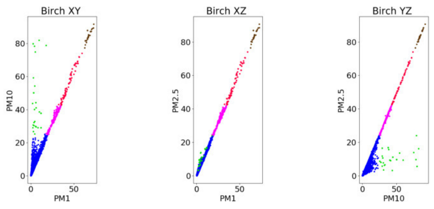
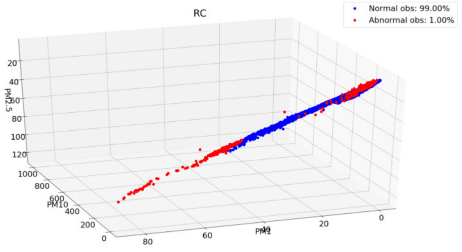
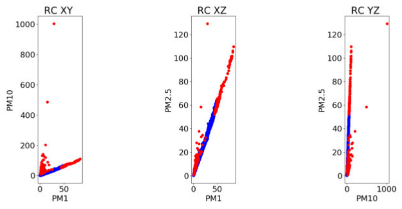


Figure A.7 Robust Covariance Outlier detection with a contamination of 1.0% (blue color: normal samples, red color: abnormal samples) and Birch Clustering results with 5 clusters (blue, red, green, brown, pink) for Particle Matter Indicators (PM1, PM2.5, PM10) - Bpost car 2.

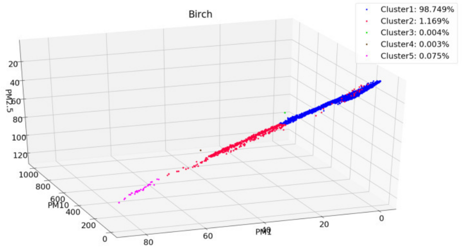
(a) RC with 1.0% - 3D perspective - Bpost car 2



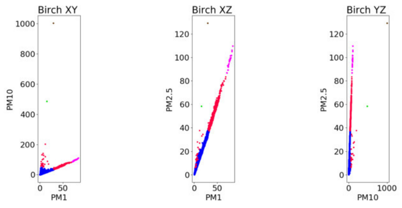
(b) RC with 1.0% - 3D planes - Bpost car 2



(c) Birch: 5 clusters - 3D perspective - Bpost car 2

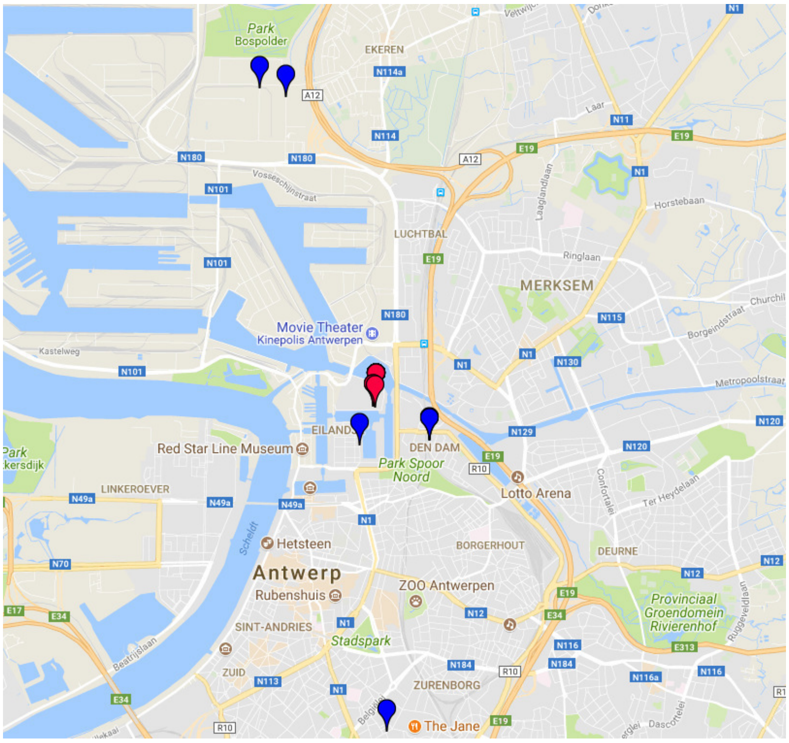


(d) Birch: 5 clusters - 3D planes - Bpost car 2



A.5.2 GPS Locations of outliers

Figure A.8 GPS locations (Bpost car 1 - red / Bpost car 2 - blue) considered as outliers by the RC algorithm.



Outliers must be further analyzed by application experts in order to extract more information from them. In the evaluation, the outliers have been compared with the GPS locations available in the datasets. In Fig. A.8, the GPS locations are shown where PM10 values above 75 ppm and PM2.5 values above 30 ppm have been collected by the Bpost cars, which have been considered as unusual data samples by the RC outlier detection algorithm.

Regarding Bpost car 1 measurements, all the unusual values have been collected in the warehouse where usually the Bpost cars stay at night. In fact, all these values have been collected on a single night between 2:54 am until 6:33 am on 5/18/2017. These high values of PM10 and PM2.5 can be related to dust and organic compounds, which were inside the warehouse at the time of the measurements. On the other hand, the unusual values measured by Bpost car 2 have been collected across the city of Antwerp. These high values of PM10 and PM2.5 can be explained by high traffic volumes in the city at these locations at the time of the measurements.

This way, by conducting anomaly detection operations in fog resources, timely alerts can be transmitted to IoT sensors and to the cloud layer indicating that high values of particle matter indicators have been measured. In doing so, citizens can be alerted of high air pollution levels in real-time.

A.5.3 LPWAN Dimensioning Analysis

Considering that for the use case, each upload message is composed of a String of 12 chars (GPS Location - geohash) equal to 12 bytes, a 32 bit integer (timestamp) equal to 4 bytes and 3 floating point 64 bit numbers (particle matter indicators) equal to 24 bytes, the total number of payload bytes to be transmitted per minute from the IoT sensor to the fog resource is 40 bytes. On the other hand, each download message to be transmitted from the fog resource to the IoT sensor in case of unusual behavior or malfunction is composed of a String of 12 chars (GPS Location - geohash) equal to 12 bytes and a byte defined by 3 alarm bits and 5 bits for 32 types of predefined messages. Furthermore, each message has a header for which the size depends on the LPWAN technology itself. In the evaluation, a general 13 byte header has been considered in each message as in Sigfox and in LoRaWAN technologies. Therefore, each upload message is transmitted with at least 53 bytes which is equal to 424 bits and each download message with 26 bytes which is equal to 208 bits. Moreover, considering that the area of Antwerp is equal to 204.5 km^2, an estimation of the minimum number of gateways required for each LPWAN technology to cover the entire area of the city has been performed. Based on these assumptions, the LPWAN technologies presented in Table A.2 have been evaluated. The comparison is presented in Table A.5 and in Table A.6, a list of pros and cons for the multiple LPWAN technologies is shown. Based on these results and because of the application requirements, Sigfox technology is unfeasible to provide wireless communication, since a single upload message takes more than a second to be transmitted and due to duty cycle regulations, real-time communication is not possible, because sending an upload message every minute implies going against the fairness rules of duty cycle. Moreover, the download capabilities are very limited in Sigfox technology. On the other hand, Ingenu RPMA is not considered as an adequate solution because it operates in the crowded 2.4 GHz band. Nowadays, low frequencies are being considered as optimal to provide wireless communication for IoT solutions. Besides, Ingenu RPMA requires high processing power, which translates into a higher energy consumption.

Regarding licensed LPWANs, LTE-M is the optimal solution to deploy the use case, because it has a higher data rate than NB-IoT making LTE-M more suitable for the application since real-time communication is needed for the scenario. On the other hand, regarding unlicensed LPWANs, IEEE 802.11ah and DASH7 are the most adequate solutions to provide wireless communication between the devices.

Table A.5: Comparison between the different LPWAN technologies based on the requirements of the Air Quality application

LPWAN Technology	LoRaWAN	Sigfox	LTE-M	DASH7	IEEE 802.11ah	NB-IoT	Ingenu RPMA
C	5 km	10 km	5 km	5 km	1 km	5 km	3 km
U/D	50 kbps	300bps	1 Mbps	166.67 kbps	346.66 Mbps	250 kbps	634/156 kbps
P	16.67ms	33.33ms	16.67ms	16.67ms	3.33ms	16.67ms	10ms
N (upload)	At least 53 bytes						
T (upload)	8.480ms	1.413s	0.424ms	2.543ms	1.221μs	1.696ms	0.669ms
L (upload)	25.15ms	1.45s	17.09ms	19.22ms	3.331ms	18.37ms	10.67ms
N (download)	At least 32 bytes						
T (download)	4.16ms	0.69s	0.208ms	1.24ms	0.60μs	0.83ms	1.33ms
L (download)	20.83ms	0.72s	16.88ms	17.91ms	3.331ms	17.5ms	11.33ms
Area of the City of Antwerp	204.5 km^2						
Minimum Number of Gateways	5	2	5	5	117	5	15

Table A.6: List of Pros and Cons of the different LPWAN technologies for the Air Quality application

LPWAN Technology	PROS	CONS
LoRaWAN	Security;	Limited downlink capability;
Sigfox	None;	Duty cycle regulations (transmit time of 36s per 1 hour): impossible to transmit a message every minute for the application; Proprietary protocol; Limited security; Limited downlink;
LTE-M	High data rate; Security;	Under development;
DASH7	High data rate when compared with similar LPWANs;	Open Source Solution; Lack of deployments;
IEEE 802.11ah	Lower transmission time; High data rate;	High number of gateways needed (low range when compared with other LPWANs); Under development;
NB-IoT	High data rate when compared with other LPWANs; Security;	Under development;
Ingenu RPMA	High uplink data rate; High coverage and robustness;	Lower range when compared with other LPWANs; Operates in the crowded 2.4Ghz band; High processing power;

LoRaWAN is not considered as an appropriate solution, because the downlink capacities are very limited. Moreover, LoRaWAN has the lower data rate and the correspondent highest transmission time when compared with IEEE 802.11ah and DASH7. In fact, although IEEE 802.11ah is currently under development, it is one the most promising LPWAN technologies with a very high data rate. However, IEEE 802.11ah deployment will require a very large number of gateways to cover the entire city of Antwerp due to the low communication range. On the other hand, DASH7 supports high data rates when compared to similar LPWAN technologies and it is already deployed in the City of Things testbed. Both technologies meet the application demands, which make them appropriate solutions to provide wireless communication for the use case in the unlicensed spectrum.

A.6 Conclusions

In recent years, the need for management functionalities in Smart Cities is increasing due to the deployment of IoT use cases. Fog Computing provides an efficient manner of dealing with stringent requirements introduced by IoT use cases. It is important to detect malfunctions and abnormal events in IoT devices to provide a secure and reliable communication. Therefore, in this appendix, a low-latency Fog-based anomaly detection approach has been presented to identify unusual events or abnormal patterns in IoT scenarios. The approach has been evaluated for a Smart City use case within the scope of City of Things testbed. Obtained results show that both Birch clustering and RC outlier anomaly detection mechanisms can be performed by fog resources close to IoT sensors and, this way, send timely alerts in case unusual events are detected. Moreover, for multiple criteria, LPWAN technologies have been evaluated for the Air Quality application, leading to a suitable set of LPWAN technologies, IEEE 802.11ah, DASH7 and LTE-M, that can be used as wireless communication enablers for the Smart City use case. As future work, the selected LPWAN technologies will be deployed and techno-economical studies will be performed.

Acknowledgment

This research was performed as part of the "City of Things" project
(Antwerp, Belgium) funded by imec and in the "Intelligent DEnse And Longe range IoT networks (IDEAL-IoT)" project under Grant Agreement #S004017N, from the fund for Scientific Research-Flanders (FWO-V).

References

[1] Vito Albino, Umberto Berardi, and Rosa Maria Dangelico. Smart cities: Definitions, dimensions, performance, and initiatives. *Journal of Urban Technology*, 22(1):3–21, 2015.

[2] Tao Han, Xiaohu Ge, Lijun Wang, Kyung Sup Kwak, Yujie Han, and Xiong Liu. 5g converged cell-less communications in smart cities. *IEEE Communications Magazine*, 55(3):44–50, 2017.

[3] Akhil Gupta and Rakesh Kumar Jha. A survey of 5g network: Architecture and emerging technologies. *IEEE access*, 3:1206–1232, 2015.

[4] Jeffrey G Andrews, Stefano Buzzi, Wan Choi, Stephen V Hanly, Angel Lozano, Anthony CK Soong, and Jianzhong Charlie Zhang. What will 5g be? *IEEE Journal on selected areas in communications*, 32(6):1065–1082, 2014.

[5] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper, 2017. URL http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf.

[6] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[7] Mohit Taneja and Alan Davy. Resource aware placement of iot application modules in fog-cloud computing paradigm. In *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*, pages 1222–1228. IEEE, 2017.

[8] Hlabishi I Kobo, Adnan M Abu-Mahfouz, and Gerhard P Hancke. A survey on software-defined wireless sensor networks: Challenges and design requirements. *IEEE Access*, 5:1872–1899, 2017.

[9] Lingjuan Lyu, Jiong Jin, Sutharshan Rajasegarar, Xuanli He, and Marimuthu Palaniswami. Fog-empowered anomaly detection in internet of things using hyperellipsoidal clustering. *IEEE Internet of Things Journal*, 2017.

[10] Mugen Peng, Shi Yan, Kecheng Zhang, and Chonggang Wang. Fog-computing-based radio access networks: issues and challenges. *IEEE Network*, 30(4):46–53, 2016.

[11] Luis Martí, Nayat Sanchez-Pi, José Manuel Molina, and Ana Cristina Bicharra Garcia. Anomaly detection based on sensor data in petroleum industry applications. *Sensors*, 15(2):2774–2797, 2015.

[12] Monowar H Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal K Kalita. Network anomaly detection: methods, systems and tools. *Ieee communications surveys & tutorials*, 16(1):303–336, 2014.

[13] Steven Latre, Philip Leroux, Tanguy Coenen, Bart Braem, Pieter Ballon, and Piet Demeester. City of things: An integrated and multi-technology testbed for iot smart city experiments. In *Smart Cities Conference (ISC2), 2016 IEEE International*, pages 1–8. IEEE, 2016.

[14] Shahid Raza, Linus Wallgren, and Thiemo Voigt. Svelte: Real-time intrusion detection in the internet of things. *Ad hoc networks*, 11(8):2661–2674, 2013.

[15] Yanxu Zheng, Sutharshan Rajasegarar, Christopher Leckie, and Marimuthu Palaniswami. Smart car parking: temporal clustering and anomaly detection in urban car parking. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*, pages 1–6. IEEE, 2014.

[16] Xiufeng Liu and Per Sieverts Nielsen. Regression-based online anomaly detection for smart grid data. *arXiv preprint arXiv:1606.05781*, 2016.

[17] SOCIOTAL project. An EU FP7 funded STREP project addressing the objective FP7-ICT-2013.1.4 “A reliable, smart and secure Internet of Things for Smart Cities”, 2017. URL http://www.sociotal.eu.

[18] Punit Rathore, Aravinda S Rao, Sutharshan Rajasegarar, Elena Vanz, Jayavardhana Gubbi, and Marimuthu Palaniswami. Real-time urban microclimate analysis using internet of things. *IEEE Internet of Things Journal*, 2017.

[19] CityPulse: Real-Time IoT Stream Processing and Large-scale Data Analytics for Smart City Applications, 2017. URL http://www.ict-citypulse.eu.

[20] Dan Puiu, Payam Barnaghi, Ralf Toenjes, Daniel Kümper, Muhammad Intizar Ali, Alessandra Mileo, Josiane Xavier Parreira, Marten Fischer, Sefki Kolozali, Nazli Farajidavar, et al. Citypulse: Large scale data analytics framework for smart cities. *IEEE Access*, 4:1086–1108, 2016.

[21] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one*, 11(4): e0152173, 2016.

[22] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[23] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[24] Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara. Low power wide area networks: An overview. *IEEE Communications Surveys & Tutorials*, 19 (2):855–873, 2017.

[25] Ramon Sanchez-Iborra and Maria-Dolores Cano. State of the art in lp-wan solutions for industrial iot services. *Sensors*, 16(5):708, 2016.

[26] Dash7 Alliance, 2017. URL http://www.dash7-alliance.org/.

[27] 3GPP Low Power Wide Area Technologies, 2016 GSMA White Paper, 2017. URL https://www.gsma.com/iot/wp-content/uploads/2016/10/3GPP-Low-Power-Wide-Area-Technologies-GSMA-White-Paper.pdf.

B
Towards End-to-End resource provisioning in Fog Computing

This appendix extends the work presented in Chapter 2 by proposing a Mixed Integer Linear Programming (MILP) model for IoT service placement that considers service chaining, different Low Power Wide Area Network (LPWAN) technologies, service replication, and multiple optimization objectives. The model provides several insights on the complete end-to-end (E2E) resource provisioning in Fog Computing environments. Results show clear trade-offs between the evaluated allocation strategies. Therefore, the main contribution of this appendix is the extended MILP model and its considerations concerning service chaining, service replication, and LPWAN interoperability.

⋆⋆⋆

José Santos, Tim Wauters, Bruno Volckaert and Filip De Turck.

Published in Journal of Network and Computer Applications, Volume 175, no. 10, pages 102915, Elsevier, 2021.

Abstract Recently, with the advent of the Internet of Things (IoT), Smart Cities have emerged as a potential business opportunity for most cloud service providers. However, centralized cloud architectures cannot sustain the requirements imposed by many IoT services. High mobility coverage and low latency constraints are among the strictest requirements, making centralized solutions impractical. In response, theoretical foundations of Fog Computing have been introduced to set up a distributed cloud infrastructure by placing computational resources close to end-users. However, the acceptance of its foundational concepts is still in its early stages. A key challenge still to answer is Service Function Chaining (SFC) in Fog Computing, in which services are connected in a specific order forming a service chain to fully leverage on network softwarization. Also, Low Power Wide Area Networks (LPWANs) have been getting significant attention. Opposed to traditional wireless technologies, LPWANs are focused on low bandwidth communications over long ranges. Despite their tremendous potential, many challenges still arise concerning the deployment and management of these technologies, making their wide adoption difficult for most service providers. In this appendix, a Mixed Integer Linear Programming (MILP) formulation for the IoT service allocation problem is proposed, which takes SFC concepts, different LPWAN technologies and multiple optimization objectives into account. To the best of our knowledge, the work goes beyond the current state-of-the-art by providing a complete end-to-end (E2E) resource provisioning in Fog-cloud environments while considering cloud and wireless network requirements. Evaluations have been performed to evaluate in detail the proposed MILP formulation for Smart City use cases. Results show clear trade-offs between the different provisioning strategies. The work can serve as a benchmark for resource provisioning research in Fog-cloud environments since the model approach is generic and can be applied to a wide range of IoT use cases.

B.1 Introduction

In recent years, the Internet of Things (IoT) has introduced a whole new set of challenges and opportunities by converting everyday life objects into smart communicating devices [1]. Due to the advent of IoT and the wide adoption of virtualization and cloud technologies, the concept of Smart Cities has become an even more attractive business opportunity [2]. Smart Cities powered by IoT aim to revolutionize different domains of urban life. For instance, improving public transportation and environmental monitoring. According to Cisco [3], billions of devices will be integrated in the IoT ecosystem in the forthcoming years. All these devices will be connected to the network, sending and receiving data to the cloud, making current centralized cloud solutions impractical. As an answer, Fog Computing [4, 5] has emerged as an extension to the cloud paradigm, by bringing cloud services closer to the end devices, thus, helping to meet the demanding requirements introduced by IoT services (e.g. low latency, high mobility coverage). Waste management platforms and surveillance camera systems are already envisioned Smart City use cases for Fog-cloud infrastructures, which will benefit from the nearby real-time processing, storage procedures and data analytics to overcome the limitations of centralized cloud environments [6].

Furthermore, IoT is pushing for a paradigm shift in terms of connectivity for these so named smart devices. Recently, Low Power Wide Area Networks (LPWANs) have drawn significant attention [7]. These wireless solutions enable low bandwidth communications over long ranges, up to several kilometers, at low power consumption, ensuring a high device lifetime. LoRaWAN [8], Sigfox [9] and the upcoming IEEE 802.11ah standard [10] are among the most popular LPWAN technologies today. In spite of their significant potential to impact the IoT ecosystem, the current market is highly fragmented and many challenges still exist concerning the deployment and management of these technologies, making their wide adoption difficult for most service providers. Additionally, micro-services are currently revolutionizing software development practices [11]. An application is decomposed in a set of small, self-contained containers deployed across a large number of servers, as opposed to the traditional single monolithic application. Containers are currently the most promising alternative to the conventional Virtual Machines (VMs), due to their high scalability and low overhead. Nevertheless, several challenges still prevent cloud providers and end users from fully benefiting from network virtualization and micro-service patterns. For example, users access a database to retrieve collected data, in which data values were already filtered and modified by a machine learning engine. Cloud providers must implement proper allocation strategies to ensure that database services are instantiated close to the users and that machine learning services are allocated nearby database ones to reduce the latency in accessing the inferred results. This key challenge is named Service Function Chaining (SFC) [12, 13], where services must be connected in a specific order, forming a service chain that each request needs to traverse to access a particular Network Service (NS). Thus, NSs are dynamically configured in software without any significant change at the hardware level, which results in optimized network resources and increased application performance. Although SFC provides high flexibility and low operational costs, SFC concepts are still quite unexplored in Fog Computing since most SFC research is focused on Multi-access Edge Computing (MEC) use cases, in which the interactions between fog locations

and the cloud are not considered [14]. In MEC scenarios, all services are preferably allocated at the network edge close to end-users to reduce latency and avoid congestion in the network core while in Fog Computing, services can be placed in a fog node or cloudlet [15], but also in the cloud making the inherent bi-directional communications crucial due to the hierarchical architecture. For example, a service may be allocated in the cloud due to its high computational requirements but needs to interact with another service, which may be located in a Fog location. These interactions (e.g. low latency, minimum available bandwidth) must be guaranteed. These bi-directional communications are currently not being studied in the context of MEC.
Although the theoretical foundations of Fog Computing have been already established, its adoption is in early stages. Research challenges in terms of resource provisioning and service scheduling persist. Therefore, in this appendix, a Mixed-Integer Linear Programming (MILP) formulation for the IoT service allocation problem is proposed, which takes SFC concepts, different LPWAN technologies and multiple optimization objectives into consideration. To the best of our knowledge, the work goes beyond the current state-of-the-art by bridging the gap between the cloud and the wireless domain, since most research only focuses on one of the domains and almost no consideration is given to the joint optimization of both. Finally, evaluations have been performed to validate the proposal, specifically for Smart City use cases. The proposed MILP model optimizes SFC allocation in Fog-cloud environments by not only reducing user latency since services (e.g. databases) are instantiated close to users, but also by decreasing the sensor's data transfer time and taking bandwidth requirements into account. The result of the work can serve as a benchmark in research covering IoT provisioning issues in Fog Computing since the model approach is generic, considers several cloud and wireless aspects and can be applied to a wide range of IoT use cases. Furthermore, the model can be adopted in realistic scenarios as the ones presented in Section B.4 and the performance of future heuristics can be assessed based on the measured results.
The remainder of the appendix is organized as follows. In the next Section, related work is discussed. Section III introduces the proposed MILP model for the IoT service allocation problem. In Section IV, the use cases are described which is followed by the evaluation setup in Section V. Next, in Section VI, results are shown. Finally, conclusions are presented in Section VII.

B.2 Related Work

Resource provisioning or also known as resource allocation has been studied for years in the network management domain [16], [17]. In recent years, resource provisioning and service placement issues gained significant attention in the field of Fog Computing. In [18], an optimization formulation for the service deployment of IoT applications in Fog scenarios has been proposed and implemented as a prototype named FogTorch. Their approach focused not only on hardware and software demands, but also on Quality of Service (QoS) requirements, such as network latency and bandwidth. Results showed that their heuristic algorithm ensures the optimal deployment of services while decreasing hardware capacity and increasing resource demands. Additionally, in [19], the IoT service placement issue has been modeled as an optimization problem. The model focused on the maximization of Fog resources and the minimization of overall network delay. Their work has been extended in [20], where application QoS metrics and deadlines for the provisioning of each type of application have been taken into account in their ILP formulation. In [21] both architectural and resource allocation concepts have been tackled. The authors presented a provisioning algorithm focused on service elasticity and on the optimization of available resources. Simulation results showed that the proposed algorithm efficiently allocates resources while minimizing the response time and maximizing the throughput. Furthermore, in [22], a particle swarm optimization algorithm has been introduced for the resource provisioning in Fog-cloud infrastructures specifically focused on Smart buildings. Results showed that their approach can reduce the response time, the data transfer and the cost of VM allocation. In [23], a provisioning algorithm formulated as a Mixed-Integer Non-Linear (MINL) problem has been presented for VM allocation in IoT networks. Their proposal enables the offloading of computationally intensive and delay-sensitive tasks to Fog nodes connected to IoT gateways. Their goal is to minimize the system cost by meeting QoS requirements. Then, in [24], their work has been extended by evaluating the trade-off between maximizing the reliability and minimizing the overall system cost. A highly computationally complex ILP model has been described followed by a heuristic-based algorithm able to find suboptimal solutions, albeit achieving better time efficiency. Additionally, in [25], a MILP formulation addressing the MEC resource provisioning issue for IoT services has been introduced. Their approach focused on the provisioning of resources (edge servers and applications) as well as the workload assignment while minimizing latency. Moreover, in [26], two placement strategies in Fog Computing based on matching game algorithms have been introduced. The first one is based on SFC concepts, by taking into account the ordered sequence of services requested by each application. The second one overlooks the SFC structure, to lower the computation complexity without losing performance. Comparison results highlighted the increased performance of the stated methods. In [27], the authors proposed a Fog Computing scheme to support IoT-based crowdsensing applications. Their proposal focused on a MINL formulation, which has then been linearized into a MILP model. Results showed that their proposal can outperform traditional cloud infrastructures. Additionally, in [28], a scheduling mechanism for resource provisioning in Fog Computing based on deep reinforcement learning has been presented. The work focuses on minimizing the time consumption of safety-related applications in vehicular fog use cases. Results confirmed that their allocation schemes can reduce time consumption when compared to traditional cloud infrastructures. Recently, in [29], a provisioning approach for Fog Computing based on Bayesian learning techniques has been presented. Their work focuses on the dynamic scaling of resources according to the current network demand. Simulation results have shown that their proposal can reduce costs and delay violations. Furthermore, in [30], a resource allocation approach focused on dynamic deadline-based requirements has been proposed. Their proposal provisions resources in a hierarchical manner by considering dynamic changes in user requirements and the limited available resources in Fog nodes. Simulation results have shown improvements in terms of data processing time, allocation costs and network delay when compared with other approaches.

Although most of the cited research has dealt with provisioning issues in Fog Computing, none of the aforementioned studies considered realistic latency-sensitive services with actual E2E latency demands, or any kind of connectivity constraints coming from the wireless domain. Also, few works considered the strict requirements coming from service chains or container-based applications. Most studies are focused on VM allocations, while container-based provisioning is still a novel research topic. Thus, in this appendix, a MILP formulation has been proposed to tackle the problem of resource provisioning in Fog-cloud environments focused on containerized services. The present work builds further on [31] by considering SFC concepts, different LPWAN technologies and several optimization objectives. By combining characteristics coming from the cloud and the wireless domain, the approach paves the way towards a complete E2E resource provisioning in the Smart City ecosystem.

B.3 The MILP Model: E2E Service Provisioning in Fog-cloud Infrastructures

This section introduces the MILP formulation for the IoT service allocation problem. Then, the variables considered in the model are described, followed by the objectives and respective constraints.

B.3.1 Model Description

The proposed MILP model significantly extends the authors' recent previous work [31] as follows: the previous formulation has already considered cloud and wireless characteristics. Nevertheless, several additions have been made to the model

mainly dealing with service chaining and LPWAN characteristics. The most relevant ones are the limited bandwidth capacity of cloud and Fog nodes and the minimum bandwidth requirement of micro-services. Then, latency metrics have been included instead of hop count. Several SFC and Network Slicing concepts have been added. LoRaWAN has been also included as an LPWAN technology. Then, micro-service replication and the gateway bandwidth factor have been added as a decision variable. Finally, the sensors' data transfer time and the user latency have been included as an optimization objective.
The model decomposes an IoT application in a set of different micro-services, which have a replication factor. Multiple users are expected to access these micro-services. Sensors are spread across the network area to collect data. Each sensor needs to be connected with a wireless gateway to be able to send data to the Fog-cloud infrastructure. The bandwidth available per sensor is affected by a gateway bandwidth factor, which may increase the sensor's data transfer time. The Fog-cloud infrastructure manages a set of nodes, in which micro-service instances must be allocated based on its requirements and subject to multiple constraints:

1. Nodes have limited capacities (e.g. CPU and memory).
2. Micro-service instances cannot be instantiated on every node, due to specific hardware or software requirements.
3. Gateways have limited capacity, based on a maximum number of association identifiers (AIDs).
4. Sensors can only associate with gateways within their communication range.
5. Sensors need to be associated with the gateway slice allocated for the particular application which they are trying to access.
6. The sensors' data transfer time depends on the selected LPWAN technology.
7. All micro-services composing an application must be allocated in the network.
8. Users must access their assigned application.
9. The replication factor of each micro-service depends on the number of user requests.

The model incorporates multiple optimization objectives, which are executed iteratively. In each iteration, a different optimization objective is considered. Additional constraints are added to the model to retain the objective values obtained in previous iterations, imposing an upper or lower bound. Consequently, since iterations must satisfy the previous optimal solutions, the solution space continuously decreases. Every iteration refines the previously obtained solution by improving the model with an additional optimization objective. Sequential objectives have been preferred to multi-criteria objectives to reduce the complexity of the MILP model calculation. Furthermore, opposing allocation policies have been applied in the evaluation, thus, the sequential ordering of objectives eased the shift between strategies and refining the solution space. The objectives considered in the model are the following:

1) Maximization of accepted user requests.
2) Minimization of service migrations between iterations.
3) Minimization of the number of active nodes.
4) Minimization of the number of active gateways.
5) Minimization of the network latency.
6) Minimization of the sensor's data transfer time.
7) Minimization of the user latency.

B.3.2 Variables

Table B.1: Input variables related to the cloud infrastructure

Symbol	Description
N	The set of nodes on which micro-service instances are executed.
A	The set of all IoT applications. Each application is composed of a set of different micro-services.
S	The set of all micro-services.
$\underline{ID}$	The set of SFC identifiers.
$\underline{U}$	The set of users.
L	The set of locations where users can access a given application $a \varepsilon A$.
$\underline{\Phi_{u,a}}$	The user assignment matrix. If $\Phi_{u,a} = 1$, the user u makes use of the application a.
$\underline{\rho_a}$	The maximum number of users that can associate with an application.
$\underline{\rho_s}$	The maximum number of users that can associate with a micro-service instance.
$\underline{\lambda_u}$	The association cost of the user u for the assigned application.
$\underline{\beta}$	The maximum replication factor for each micro-service.
$\underline{\upsilon_s}$	The SFC first position indicator. If $\upsilon_s = 1$, the micro-service s is the first micro-service in the service chain.
$\underline{\iota_s}$	The SFC last position indicator. If $\iota_s = 1$, the micro-service s is the last micro-service in the service chain.
$\underline{\mu}$	The service migration factor represents the maximum allowed percentage of micro-service reallocations.

continued on next page...

Table B.1: Input variables related to the cloud infrastructure (continued)

Symbol	Description
$I_{a,s}$	The Instance matrix. If $I_{a,s} = 1$, the micro-service s is part of application a. Otherwise, the micro-service s is not part of application a.
$R_{s,n}$	The Relation matrix. If $R_{s,n} = 1$, the micro-service s can be allocated on node n. Otherwise, it cannot be instantiated on node n due to hardware or software limitations.
Ω_n	The total CPU capacity (in cpu) of the node $n \varepsilon N$.
Γ_n	The total memory capacity (in GB) of the node $n \varepsilon N$.
Δ_n	The bandwidth capacity (in Mbit/s) of the node $n \varepsilon N$.
ω_s	The CPU requirement (in cpu) of the micro-service $s \varepsilon S$.
γ_s	The memory requirement (in GB) of the micro-service $s \varepsilon S$.
δ_s	The bandwidth requirement (in Mbit/s) of the micro-service $s \varepsilon S$.
B_{n_1,n_2}	The bandwidth matrix indicates the available bandwidth capacity (Mbit/s) between the node n_1 and the node n_2.
τ_{n_1,n_2}	The latency matrix indicates the latency (in ms) between the node n_1 and the node n_2.
τ_{l_1,l_2}	The location matrix indicates the latency (in ms) between the location l_1 and the location l_2.
$\tau_{n,l}$	The node location matrix indicates the latency (in ms) between node n and location l.
$\tau_{u,l}$	The user location matrix indicates the latency (in ms) between user u and location l.
$\tau_{u,n}$	The user node matrix indicates the latency (in ms) between user u and node n.
C_{s_i,s_j}	The Communication matrix indicates the minimum amount of bandwidth (in Mbit/s) between the micro-services s_i and s_j for their proper operation. The micro-service s_i is the source of the network flow while s_j is the sink.
$E_{n,l}$	If $E_{n,l} = 1$, the node n is at location l.
$E_{u,l}$	If $E_{u,l} = 1$, the user u is at location l.
α_{s_i,s_j}	The service matrix. If $\alpha_{s_i,s_j} = 1$, the flow bandwidth between the micro-service s_i and the micro-service s_j needs to be guaranteed. If $\alpha_{s_i,s_j} = 0$, the flow bandwidth does not need to be guaranteed.

Table B.2: Input variables related to the wireless dimensioning

Symbol	Description
GW	The set of wireless gateways.
SR	The set of sensors.

continued on next page...

Table B.2: Input variables related to the wireless dimensioning (continued)

Symbol	Description
SL	The set of network slices.
Θ_{gw}	The total association identifiers (AIDs) available on a gateway $gw \,\varepsilon\, GW$.
θ_{sr}	Each sensor $sr \,\varepsilon\, SR$ needs an AID to connect with a gateway.
$\underline{\Phi_{sr,a}}$	The sensor assignment matrix. If $\Phi_{sr,a} = 1$, the sensor sr makes use of the application a.
$D_{gw,sr}$	The Distance matrix indicates the distance (in meters) between the gateway gw and the sensor sr.
$PL_{gw,sr}$	The Path Loss matrix indicates the path loss (in dB) between the gateway gw and the sensor sr.
$A_{sr,gw}$	The Association matrix. If $A_{sr,gw} = 1$, the sensor sr can associate with the gateway gw.
$\underline{A_{a,sl}}$	The Application Slice matrix. If $A_{a,sl} = 1$, the slice sl is responsible for the wireless traffic belonging to app. a.
$\underline{A_{sr,sl}}$	The Slice Association matrix. If $A_{sr,sl} = 1$, the sensor sr needs to be assigned to the slice sl due to its associated application.
$\underline{B_{sl,gw}}$	The Bandwidth matrix indicates the available bandwidth capacity (in Mbit/s) for the slice sl in the gateway gw.
$\underline{\eta_{sr,sl,gw}}$	The Data Rate matrix indicates the available bandwidth (in Mbit/s) for the sensor sr assigned to slice sl in the gateway gw.
$\underline{\tau_{gw,l}}$	The gateway location matrix indicates the latency (in ms) between gateway gw and location l.
$\underline{\tau_{gw,n}}$	The gateway node matrix indicates the latency (in ms) between gateway gw and node n.
$\underline{I_{gw}}$	The IEEE 802.11 ah matrix. If $I_{gw} = 1$, the gateway gw is an IEEE 802.11 ah gateway.
$\underline{L_{gw}}$	The LoRaWAN matrix. If $L_{gw} = 1$, the gateway gw is a LoRaWAN gateway.
$\underline{K_{gw}}$	The gateway access delay matrix represents the propagation time (in ms) from the gateway gw to the Fog-cloud infrastructure.
$\underline{\pi_{a}}$	The number of bits needed to be transmitted in each upload message for application a.
$\underline{\pi_{sr}}$	The number of bits needed to be transmitted by each sensor sr based on the assigned application.
$E_{gw,l}$	If $E_{gw,l} = 1$, the gateway gw is at location l.
$E_{sr,l}$	If $E_{sr,l} = 1$, the sensor sr is at location l.

Table B.3: Decision variables of the MILP model

Symbol	Description
$\underline{G_{a,id}}$	The application acceptance matrix. If $G_{a,id} = 1$, the application a with the SFC identifier id can be allocated. If $G_{a,id} = 0$, the application a with the SFC identifier id cannot be deployed.
$\underline{G_{a,id,s}}$	The micro-service acceptance matrix. If $G_{a,id,s} = 1$, the micro-service s for the application a with the SFC identifier id can be allocated. If $G_{a,id,s} = 0$, the micro-service s for the application a with the SFC identifier id cannot be deployed.
$\underline{G_{u,a,id}}$	The user acceptance matrix. If $G_{u,a,id} = 1$, the user u is associated with application a with the SFC identifier id.
$\underline{\beta_{a,id,s}}$	The replication factor of the micro-service s for the application a with the SFC identifier id.
$\underline{P^{a,id}_{s,\beta_i}(n)}$	The placement matrix. If $P^{a,id}_{s,\beta_i}(n) = 1$, the replica β_i of micro-service s is executed on node n for the application a with the SFC identifier id.
$\underline{P^{a,id}_{s,\beta_i}(n)^{i-1}}$	The placement matrix from the previous iteration.
$\underline{F^{a,id,s_i,\beta_i}_{s_j,\beta_j}(n_1, n_2)}$	The binary flow matrix indicates that the replica β_i from the micro-service s_i and the replica β_j from the micro-service s_j are allocated on node n_1 and n_2, respectively, for the application a with the SFC identifier id.
$\underline{\Lambda_{a,id}}$	The SFC latency matrix indicates the time (in ms) to traverse all possible paths in the service chain for the application a with the SFC identifier id.
$\underline{M^{a,id}_{s,\beta_i}}$	The service migrations matrix. If $M^{a,id}_{s,\beta_i} = 1$, the replica β_i of micro-service s for the application a with the SFC identifier id has been reallocated to another node based on the previous iteration.
U_{gw}	The gateway utilization matrix. $U_{gw} = 1$ indicates that there is at least one sensor associated with gateway gw.
U_n	The node utilization matrix. $U_n = 1$ indicates that there is at least one micro-service replica allocated on node n.
$U_{s,n}$	The micro-service execution matrix. If $U_{s,n} = 1$, a replica of the micro-service s is allocated on node n. If $U_{s,n} = 0$, no replica of the micro-service s is allocated on node n.

continued on next page...

Table B.3: Decision variables of the MILP model (continued)

Symbol	Description
$\underline{U^{u,a,id}_{s,\beta_i}(n)}$	The user service matrix. If $U^{u,a}_{id,s,\beta_i}(n) = 1$, the user u is associated with the replica β_i of micro-service s allocated on node n for the application a with the SFC identifier id.
$U_{sr,gw}$	The sensor association matrix. If $U_{sr,gw} = 1$, the sensor sr is associated with gateway gw.
$\underline{U_{sr,sl}}$	The slice association matrix. If $U_{sr,sl} = 1$, the sensor sr is associated with the slice sl.
$\underline{U_{sr,sl,gw}}$	The sensor execution matrix. If $U_{sr,sl,gw} = 1$, the sensor sr is assigned to slice sl in the gateway gw.
$\underline{\zeta_{gw}}$	The gateway bandwidth factor.
$\underline{T_{sr}}$	The data transfer time matrix indicates the propagation time (in ms) of a single upload message from the sensor sr.
$\underline{\psi_u}$	The user latency matrix indicates the propagation time (in ms) of a request from the user u to reach the last micro-service in the assigned service chain.

The input variables used in the model are shown in Table B.1 and in Table B.2, while decision variables are shown in Table B.3. All variables added to previous work have been underlined. Thirty-one new input variables have been included in the model. Furthermore, thirteen new decision variables have been added to the model, while two others have been slightly modified to address micro-service replication. Regarding cloud formulation, a set of applications A composed of micro-services S need to be allocated on nodes $n \varepsilon N$. Each application a has a given SFC identifier $id \varepsilon ID$. All micro-services have a maximum number of replicas given by β. The replication factor for a particular micro-service $s \varepsilon S$ for the application a with the SFC identifier id is given by $\beta_{a,id,s}$. Thus, the model determines the exact number of replicas for each micro-service depending on the considered objective (e.g. maximizing user requests, reducing user latency). Each micro-service s has a CPU and a memory requirement represented by ω_s (in cpu) and γ_s (in GB) respectively. For instance, a CPU requirement equal to 0.5 cpu (i.e. 500 millicpu) means that the micro-service needs 50% of a core to operate properly. Also, each micro-service s has a minimum bandwidth requirement represented by δ_s (in Mbit/s). A binary placement matrix P is used to represent in which node n, the replica β_i of a micro-service s is allocated. Also, the previous placement matrix $P^{a,id}_{s,\beta_i}(n)^{i-1}$ is added to the model so that decisions can be made in terms of service migrations since the model has information on where micro-service instances have been allocated on the previous iteration. Then, the service migrations matrix M is used to indicate if a particular micro-service replica has been reallocated on another node. If $M^{a,id}_{s,\beta_i} = 1$, the replica β_i of micro-service s for the application a with the SFC identifier id has been reallocated based on the previous placement matrix. Additionally, the service migration factor μ represents the maximum allowed percentage of service reallocations between model iterations.

Regarding wireless formulation, the model supports two LPWAN technologies, which have been represented through linear equations: IEEE 802.11ah and LoRaWAN. The LoRaWAN formulations are based on the work presented in [32]. The binary matrix I_{gw} indicates if the gateway gw is an IEEE 802.11ah gateway, while the binary matrix L_{gw} determines if the gateway gw is a LoRaWAN gateway. Also, the gateway access delay matrix given by K_{gw} represents the access delay from the gateway gw to the Fog-cloud infrastructure. Without loss of generality, if $I_{gw} = 1$ (i.e. IEEE 802.11 ah gateway), the access delay corresponds to 2 ms. Otherwise, if $L_{gw} = 1$ (i.e. LoRaWAN gateway), the access delay is equal to 5 ms. The distance matrix D indicates the distance (in meters) between a gateway gw and a sensor sr. An additional binary association matrix A is used to indicate if a sensor sr can associate with a gateway gw. This association is based on the distance matrix D and on the AIDs available on each gateway. An IEEE 802.11ah [33] gateway cannot have more than 8192 associated stations as stated in the latest standard. However, the association limitation has been set to 50, since an urban macro deployment with extended range has been considered [34]. For LoRaWAN gateways, the association limitation has been set to 100, since the maximum data rate achieved by each sensor decreases considerably when a higher number of sensors is associated [35]. Thus, by setting up a lower limitation, it is assumed that good channel conditions are always achieved and that all sensors can access the deployed applications, if connected with one gateway. Furthermore, for IEEE 802.11 ah, the distance limitation is set to one thousand meters because this is the maximum coverage range in IEEE 802.11ah networks [36] while for LoRaWAN the limit is set to four thousand meters [37]. If $D_{gw,sr}$ is lower than the imposed limit, the sensor sr can associate with gateway gw and then $A_{sr,gw} = 1$, otherwise, $A_{sr,gw} = 0$. Furthermore, the concept of network slicing has been included in the model by adding the set of slices SL. The binary application slice matrix $A_{a,sl}$ indicates if the slice sl is responsible for the wireless traffic belonging to application a while the binary slice association matrix $A_{sr,sl}$ indicates if the sensor sr needs to be associated with slice sl due to its assigned application. The bandwidth matrix $B_{sl,gw}$ contains the available bandwidth (in Mbit/s) for the slice sl in the gateway gw. Then, the data rate matrix $\eta_{sr,sl,gw}$ contains the available bandwidth for the sensor sr assigned to slice sl in the gateway gw. For LoRaWAN, the data rate of each gw depends on the Spreading Factor (SF) adopted by each sensor sr. The SF concerns the ratio between the symbol rate and the chip rate. LoRaWAN spreads each symbol in a rate of 2^{SF} chips per symbol with $SF = \{7, .., 12\}$. Increasing the spreading factor reduces the transmitted data rate at the expense of offering longer range. In the model, SF has been set to 9. Additionally, the data transfer time matrix T_{sr} corresponds to the propagation time (in ms) for a single upload message from the sensor sr to reach the associated gateway. The data transfer time depends on the selected LPWAN technology and on the total number of connected sensors to the particular gateway, which is affected by ζ_{gw}, the gateway bandwidth factor. Essentially, the available bandwidth per sensor may drop depending on the number of connected sensors on each gateway. Further details on how data transfer time is affected by this factor are given in the next section. Then, to fully minimize the latency expected by each user when accessing the given application, another decision variable ψ_u has been added, which contains the propagation time (in ms) of a request from the user u to reach the assigned application, more precisely to access an instance of the last micro-service in the application's service chain with which user u is connected to. Each optimization objective is detailed below. All constraints previously presented in [31] have been considered in this extended model. To avoid repetition, only constraints related to novel variables are described.

B.3.3 Optimization Objectives & Constraints

B.3.3.1 Maximization of User Requests (MAX R)

This objective is related to the maximization of acceptance of user requests. Multiple constraints have been added to reflect the extensions regarding service replication, service chaining and wireless formulations made in the model. Firstly, the objective has been updated to consider the user's assigned application as shown in (B.1).

$$max \sum_{u \,\varepsilon\, U} \sum_{a \,\varepsilon\, A} \sum_{id \,\varepsilon\, ID} \Phi_{u,a} \times G_{u,a,r} \tag{B.1}$$

To limit the association of users to a certain application, a constraint represented by (B.2) has been used to guarantee that the maximum number of users per application is respected.

$$\forall a \,\varepsilon\, A, id \,\varepsilon\, ID : \sum_{u \,\varepsilon\, U} \Phi_{u,a} \times G_{u,a,id} \leq \rho_a \tag{B.2}$$

A constraint has been also included to ensure that each user is associated with only one application as shown in (B.3).

$$\forall u \,\varepsilon\, U : \sum_{a \,\varepsilon\, A} \sum_{id \,\varepsilon\, ID} \Phi_{u,a} \times G_{u,a,id} = 1 \tag{B.3}$$

Service association constraints have also been added to guarantee that users are associated with one particular micro-service instance for all micro-services of their admitted application. Thus, users can only access their assigned application if they are associated with all of their micro-services as shown in (B.4). Also, each user can only be admitted to only one replica of the same micro-service as given by (B.5).

$$\begin{aligned}&\forall u \,\varepsilon\, U, a \,\varepsilon\, A, id \,\varepsilon\, ID :\\ &\sum_{s \,\varepsilon\, S} \sum_{\beta_i \,\varepsilon\, \beta} \sum_{n \,\varepsilon\, N} U^{u,a,id}_{s,\beta_i}(n) = \Phi_{u,a} \times \sum_{s \,\varepsilon\, S} I_{a,s}\end{aligned} \tag{B.4}$$

$$\begin{aligned}&\forall u \,\varepsilon\, U, a \,\varepsilon\, A, id \,\varepsilon\, ID, s \,\varepsilon\, S :\\ &\sum_{\beta_i \,\varepsilon\, \beta} \sum_{n \,\varepsilon\, N} U^{u,a,id}_{s,\beta_i}(n) = \Phi_{u,a} \times I_{a,s}\end{aligned} \tag{B.5}$$

Each micro-service replica is then subject to an association limitation based on ρ_s and the user's association cost λ_u as shown in (B.6).

$$\begin{aligned}&\forall a \,\varepsilon\, A, id \,\varepsilon\, ID, s \,\varepsilon\, S, \beta_i \,\varepsilon\, \beta, n \,\varepsilon\, N :\\ &\sum_{u \,\varepsilon\, U} \lambda_u \times \Phi_{u,a} \times U^{u,a,id}_{s,\beta_i}(n) \leq \rho_s\end{aligned} \tag{B.6}$$

Also, a constraint has been included to assure that users are only connected with micro-service instances allocated in the network as shown in (B.7).

$$\begin{aligned}&\forall u \,\varepsilon\, U, \forall a \,\varepsilon\, A, id \,\varepsilon\, ID, s \,\varepsilon\, S, \beta_i \,\varepsilon\, \beta, n \,\varepsilon\, N :\\ &U^{u,a,id}_{s,\beta_i}(n) \leq P^{a,id}_{s,\beta_i}(n)\end{aligned} \tag{B.7}$$

Then, two constraints have been added to reflect the relations between the acceptance matrices G. A micro-service is considered instantiated if the expected number of micro-service instances is equal to the number of allocated replicas in the network. Also, an application is only accepted if all its micro-services have been admitted in the network (i.e. all instances deployed on the network). These constraints are represented by (B.8) and (B.9), respectively.

$$\begin{aligned}&\forall a \,\varepsilon\, A, id \,\varepsilon\, ID, s \,\varepsilon\, S :\\ &G_{a,id,s} = \begin{cases} 1.0 & \text{if } I_{a,s} \times \beta_{a,id,s} = \sum_{\beta_i, n \,\varepsilon\, \beta, N} P^{a,id}_{s,\beta_i}(n) \\ 0.0 & \text{Otherwise} \end{cases}\end{aligned} \tag{B.8}$$

$$\forall a \,\varepsilon\, A, id \,\varepsilon\, ID :$$
$$G_{a,id} = \begin{cases} 1.0 & \text{if } \sum_{s \,\varepsilon\, S} I_{a,s} = \sum_{s \,\varepsilon\, S} G_{a,id,s} \\ 0.0 & \text{Otherwise} \end{cases} \tag{B.9}$$

A constraint has been added to limit the allocation of one instance of the same micro-service per node. Thus, only one micro-service replica can be deployed on the same node. This constraint is shown in (B.10).

$$\forall a \,\varepsilon\, A, id \,\varepsilon\, ID, s \,\varepsilon\, S, \beta_i \,\varepsilon\, \beta : \sum_{n \,\varepsilon\, N} P^{a,id}_{s,\beta_i}(n) \leq 1.0 \tag{B.10}$$

CPU and memory constraints have also been updated as shown in (B.11) and (B.12), respectively. Similarly, in (B.13), bandwidth limitations have been defined. Bandwidth requirements have been added to the model so that expected bandwidth demands can be met given the limited network link capacity.

$$\forall n \,\varepsilon\, N : \sum_{a \,\varepsilon\, A} \sum_{id \,\varepsilon\, ID} \sum_{s \,\varepsilon\, S} \sum_{\beta_i \,\varepsilon\, \beta} P^{a,id}_{s,\beta_i}(n) \times \omega_s \leq \Omega_n \tag{B.11}$$

$$\forall n \,\varepsilon\, N : \sum_{a \,\varepsilon\, A} \sum_{id \,\varepsilon\, ID} \sum_{s \,\varepsilon\, S} \sum_{\beta_i \,\varepsilon\, \beta} P^{a,id}_{s,\beta_i}(n) \times \gamma_s \leq \Gamma_n \tag{B.12}$$

$$\forall n \,\varepsilon\, N : \sum_{a \,\varepsilon\, A} \sum_{id \,\varepsilon\, ID} \sum_{s \,\varepsilon\, S} \sum_{\beta_i \,\varepsilon\, \beta} P^{a,id}_{s,\beta_i}(n) \times \delta_s \leq \Delta_n \tag{B.13}$$

Secondly, several modifications have been made to the wireless formulations since one additional LPWAN technology has been included in the model. Sensors associate with gateways through an AID, a unique value assigned to a sensor by the gateway during association handshake. A constraint has been added to the model, ensuring that the AID limit on each gateway is respected. Therefore, by using the sensor association matrix $U_{sr,gw}$, the AID limitation can be expressed as shown in (B.14). The total number of AIDs attributed to a gateway must be lower than the total number of available AIDs.

$$\forall gw \,\varepsilon\, GW : \sum_{sr \,\varepsilon\, SR} \theta_{sr} \times U_{sr,gw} \leq \Theta_{gw} \tag{B.14}$$

A constraint is also added to ensure that sensors are connected with at least one gateway to be able to send the collected data. This constraint is represented by (B.15).

$$\forall sr \,\varepsilon\, SR : \sum_{gw \,\varepsilon\, GW} U_{sr,gw} \times A_{sr,gw} = 1 \tag{B.15}$$

Then, the association of sensors is based on the distance matrix $D_{gw,sr}$ and on the path loss matrix $PL_{gw,sr}$. On the one hand, for IEEE 802.11 ah, the path loss is calculated based on the path loss formula for urban macro deployments at a central frequency (f_c) of 900 MHz. This formulation can be expressed as in (B.16), with the distance d in meters. On the other hand, for LoRaWAN, the path loss is calculated based on the path loss formula presented in [37] for the Dortmund use case at a central frequency (f_c) of 868 MHz. This formulation can be expressed as in (B.17), with the distance d in kilometers.

$$PL(dB) = 8 + 37.6 \log_{10}(d) \tag{B.16}$$

$$PL(dB) = 132.25 + 10 \times 2.65 \log_{10}(d) \tag{B.17}$$

Thirdly, constraints have been added on network slicing. The main goal behind virtual slicing is to bring flexibility to the network by splitting the wireless traffic. Resources are reserved for each slice sl on the different gateways. Each slice sl is characterized by a bandwidth matrix $B_{sl,gw}$. The bandwidth matrix $B_{sl,gw}$ depends on the LPWAN technology and on the corresponding maximum bandwidth allowed for each sensor. For IEEE 802.11ah, the value of 256 Kbit/s has been considered, while for LoRaWAN, 50 Kbit/s has been chosen, since this is the theoretical maximum possible bandwidth for each sensor. The bandwidth matrix $B_{sl,gw}$ (in Mbit/s) can be expressed as in (B.18).

$$B_{sl,gw} = \begin{cases} \frac{\Theta_{gw} \times 0.256}{SL} & \text{if } I_{gw} = 1 \\ \frac{\Theta_{gw} \times 0.050}{SL} & \text{if } L_{gw} = 1 \end{cases} \tag{B.18}$$

The data rate matrix $\eta_{sr,sl,gw}$ (in Mbit/s) estimates the available bandwidth capacity for the sensor sr depending on the bandwidth assigned to the slice sl of gateway gw, which is given by (B.19).

$$\eta_{sr,sl,gw} = \begin{cases} 0.256 & \text{if } I_{gw} = 1 \\ SF \times \frac{B_{sl,gw}}{2^{SF}} & \text{if } L_{gw} = 1 \end{cases} \tag{B.19}$$

An additional constraint is added to ensure that sensors access the correspondent slice of their associated application. This constraint is represented by (B.20).

$$\forall sr \,\varepsilon\, SR, sl \,\varepsilon\, SL : U_{sr,sl} = A_{sr,sl} \tag{B.20}$$

To limit the association of sensors to only one slice from a given gateway, a constraint represented by (B.21) has been applied. Two constraints are also added to make sure that the formulation only selects one slice and one gateway for each sensor. These two constraints are shown in (B.22).

$$\forall sr \,\varepsilon\, SR, sl \,\varepsilon\, SL : \sum_{gw \,\varepsilon\, GW} U_{sr,sl,gw} = 1 \tag{B.21}$$

$$U_{sr,sl,gw} = \begin{cases} 1 & \text{if } U_{sr,sl} = 1 \wedge U_{sr,gw} = 1 \\ 0 & \text{if } U_{sr,sl} = 0 \vee U_{sr,gw} = 0 \end{cases} \tag{B.22}$$

To limit the traffic in each slice, a constraint given by (B.23) has been used to guarantee that the total wireless traffic on each slice sl does not exceed the maximum data rate capacity on each gateway gw.

$$\forall sl \,\varepsilon\, SL, gw \,\varepsilon\, GW : \sum_{sr \,\varepsilon\, SR} \eta_{sr,sl,gw} \times U_{sr,sl,gw} \leq B_{sl,gw} \tag{B.23}$$

Fourthly, constraints are added about the data transfer time T_{sr} and the gateway bandwidth factor ζ_{gw}. The data transfer time of a single message can be expressed by using the sensor's data rate $\eta_{sr,sl,gw}$ and the total number of bits to be transmitted π_{sr}. Thus, the data transfer time of each sensor sr is given by (B.24), in which the data transfer time is affected by ζ_{gw}. The gateway bandwidth factor ζ_{gw} is used to make sure that the formulation considers that a higher number of sensors associated to a single gateway will decrease the available bandwidth per sensor and, consequently, increase the data transfer time. The various constraints added to the model are shown in (B.25). These values are based on the work presented in [38]. Their experiments showed that increasing the number of sensors per gateway greatly decreases the maximum attainable data rate of each sensor.

$$\begin{aligned} &\forall sr \,\varepsilon\, SR, sl \,\varepsilon\, SL, gw \,\varepsilon\, GW : \\ &T_{sr} = \left(\frac{\pi_{sr} \times 1000}{\eta_{sr,sl,gw}} \right) \times \zeta_{gw} \quad \text{(in ms)} \end{aligned} \tag{B.24}$$

$$\zeta_{gw} = \begin{cases} 1.0 & \text{if } \sum_{sr,gw \,\varepsilon\, SR,GW} : U_{sr,gw} \leq 3 \\ 1.11 & \text{if } \sum_{sr,gw \,\varepsilon\, SR,GW} : 3 < U_{sr,gw} \leq 5 \\ 1.25 & \text{if } \sum_{sr,gw \,\varepsilon\, SR,GW} : 5 < U_{sr,gw} \leq 8 \\ 1.43 & \text{if } \sum_{sr,gw \,\varepsilon\, SR,GW} : 8 < U_{sr,gw} \leq 12 \\ 1.67 & \text{if } \sum_{sr,gw \,\varepsilon\, SR,GW} : 12 < U_{sr,gw} \leq 15 \\ 2.0 & \text{if } \sum_{sr,gw \,\varepsilon\, SR,GW} : 15 < U_{sr,gw} \leq 18 \\ 2.5 & \text{if } \sum_{sr,gw \,\varepsilon\, SR,GW} : 18 < U_{sr,gw} \leq 26 \\ 3.33 & \text{if } \sum_{sr,gw \,\varepsilon\, SR,GW} : 26 < U_{sr,gw} \leq 33 \\ 5.0 & \text{if } \sum_{sr,gw \,\varepsilon\, SR,GW} : 33 < U_{sr,gw} \leq 40 \\ 10.0 & \text{if } \sum_{sr,gw \,\varepsilon\, SR,GW} : U_{sr,gw} \geq 40 \end{cases} \tag{B.25}$$

Finally, constraints are added about the user latency ψ_u. The user latency corresponds to the propagation time (in ms) of a request from the user u to reach an instance of the last micro-service in the application's service chain with which user u is connected to, as expressed by (B.26). As shown, the user latency depends on which node the last micro-service replica is allocated. If the micro-service instance is deployed on a node far from the user, the user latency will thus be higher. Consequently, the E2E latency is two times the value of ψ_u since it represents the time it takes for a user request to reach the last service in the chain and coming back to the user.

$$\begin{aligned}
&\forall u \,\varepsilon\, U, a \,\varepsilon\, A, id \,\varepsilon\, ID, s \,\varepsilon\, S, \beta_i \,\varepsilon\, \beta, n \,\varepsilon\, N : \\
&\textbf{if } G_{u,a,id} = 1 \quad (u \text{ associated with app. } a \text{ with SFC id. } id) \\
&\textbf{if } \iota_s \times U^{u,a,id}_{s,\beta_i}(n) = 1 \quad (u \text{ connected with last instance}) \\
&\textbf{Then } \psi_u = \tau_{u,n} \quad (\text{in ms})
\end{aligned} \tag{B.26}$$

B.3.3.2 Minimizing Service Migrations - (MIN M)

Although this objective has already been considered in the previous version of the model, the service migration estimation in the model has been reformulated. In a dynamic use case, micro-service replicas may need to be reallocated from one node to another to provide the optimal provisioning solution. However, it may be desirable to find a sub-optimal solution, in which service migrations are kept to a minimum to reduce the delay caused by service reallocations. Since the model is executed iteratively, the placement matrix from the previous iteration P^{i-1} is added to the model, which is compared with the current placement matrix P to reduce the service migrations needed to achieve the next objective. The decision variable $M^{a,id}_{s,\beta_i}(n)$ is used to determine the correspondent service migrations as shown by (B.27).

$$\begin{aligned}
&\forall a \,\varepsilon\, A, id \,\varepsilon\, ID, s \,\varepsilon\, S, \beta_i \,\varepsilon\, \beta, n \,\varepsilon\, N : \\
&M^{a,id}_{s,\beta_i}(n) = \begin{cases} 0 & \text{if} \quad P^{a,id}_{s,\beta_i}(n) = 1 \land P^{a,id}_{s,\beta_i}(n)^{i-1} = 1 \\ & \text{if} \quad P^{a,id}_{s,\beta_i}(n) = 0 \land P^{a,id}_{s,\beta_i}(n)^{i-1} = 0 \\ 1 & \text{Otherwise} \end{cases}
\end{aligned} \tag{B.27}$$

Then, the service migrations matrix $M^{a,id}_{s,\beta_i}$ is calculated as given by (B.28).

$$\begin{aligned}
&\forall a \,\varepsilon\, A, id \,\varepsilon\, ID, s \,\varepsilon\, S, \beta_i \,\varepsilon\, \beta : \\
&M^{a,id}_{s,\beta_i} = \begin{cases} 1 & \text{if} \quad \sum_{n \,\varepsilon\, N} M^{a,id}_{s,\beta_i}(n) \geq 1 \\ 0 & \text{if} \quad \sum_{n \,\varepsilon\, N} M^{a,id}_{s,\beta_i}(n) = 0 \end{cases}
\end{aligned} \tag{B.28}$$

Thus, the minimization of service migrations compared to the previous iteration can be expressed as shown in (B.29).

$$min \sum_{a \,\varepsilon\, A} \sum_{id \,\varepsilon\, ID} \sum_{s \,\varepsilon\, S} \sum_{\beta_i \,\varepsilon\, \beta} M^{a,id}_{s,\beta_i} \tag{B.29}$$

In most cases, this objective will obtain solutions where service migrations are not admitted at all (i.e. 0% service migrations). To allow the model to achieve intermediate solutions based on a predefined limit on the maximum number of allowed service migrations, a constraint has been added to the model based on the service migration factor μ. This constraint is given by (B.30).

$$\begin{aligned} &\forall a \,\varepsilon\, A, id \,\varepsilon\, ID : \\ &\sum_{s \,\varepsilon\, S} \sum_{\beta_i \,\varepsilon\, \beta} M^{a,id}_{s,\beta_i} \times I_{a,s} \leq \mu \times \sum_{s \,\varepsilon\, S} I_{a,s} \times \beta_{a,id,s} \end{aligned} \tag{B.30}$$

B.3.3.3 Minimizing Active Nodes - (MIN N)

This objective concerns the minimization of the number of nodes used in the service allocation, which results in cost and energy savings. This optimization can be expressed as shown in (B.31) by using the node utilization matrix U_n.

$$min \sum_{n \,\varepsilon\, N} U_n \tag{B.31}$$

B.3.3.4 Minimizing Active Gateways - (MIN GW)

This optimization aims to minimize the number of active gateways in the network. This objective ensures that a minimum number of gateways is used to provide connectivity to all the sensors. This results in energy and cost savings, however, it increases the data transfer time for each sensor. A clear trade-off exists between this objective and the minimization of the sensor's data transfer time. Service providers may opt for one of the two strategies, depending on their service characteristics and the network behavior at a given moment. By using the gateway utilization matrix U_{gw}, the minimization can be expressed as shown in (B.32).

$$min \sum_{gw \,\varepsilon\, GW} U_{gw} \tag{B.32}$$

B.3.3.5 Minimizing Network Latency - (MIN NL)

This objective is related to latency reduction in the communication between microservices from the same application corresponding to the proper service chain path.

This is expressed by the Flow Factor $\Upsilon s_i, s_j$ shown in (B.33). Thus, by allocating each micro-service as close to the next micro-service in the service chain as possible, the latency is reduced. The SFC latency matrix $\Lambda_{a,id}$ is determined by using the Flow matrix F as stated in (B.34).

$$\Upsilon s_i, s_j = I_{a,s_i} \times I_{a,s_j} \times \alpha_{s_i,s_j} \tag{B.33}$$

$$\begin{split}\forall a \,\varepsilon\, A, id \,\varepsilon\, ID : \Lambda_{a,id} = \sum_{s_i \,\varepsilon\, S} \sum_{\beta_i \,\varepsilon\, \beta} \sum_{s_j \,\varepsilon\, S} \sum_{\beta_j \,\varepsilon\, \beta} \sum_{n_1 \,\varepsilon\, N} \sum_{n_2 \,\varepsilon\, N} \\ \Upsilon s_i, s_j \times \tau_{n_1,n_2} \times F^{a,id,s_i,\beta_i}_{s_j,\beta_j}(n_1, n_2)\end{split} \tag{B.34}$$

The Flow matrix F is also subjected to various constraints to accurately represent network flows, specifically in terms of flow conservation, which ensures no flow is lost within the network, as shown in (B.35).

$$\begin{split}\forall a \,\varepsilon\, A, id \,\varepsilon\, ID, s_i \,\varepsilon\, S, \beta_i \,\varepsilon\, \beta, s_j \,\varepsilon\, S, \beta_j \,\varepsilon\, \beta : \\ \sum_{n_1 \,\varepsilon\, N} \sum_{n_2 \,\varepsilon\, N} F^{a,id,s_i,\beta_i}_{s_j,\beta_j}(n_1, n_2) = \\ = \begin{cases} 1 & \text{if } P^{a,id}_{s_i,\beta_i}(n_1) = 1 \land P^{a,id}_{s_j,\beta_j}(n_2) = 1 \\ 0 & \text{Otherwise} \end{cases}\end{split} \tag{B.35}$$

Thus, this objective can be expressed as shown in (B.36).

$$min \sum_{a \,\varepsilon\, A} \sum_{id \,\varepsilon\, ID} \Lambda_{a,id} \tag{B.36}$$

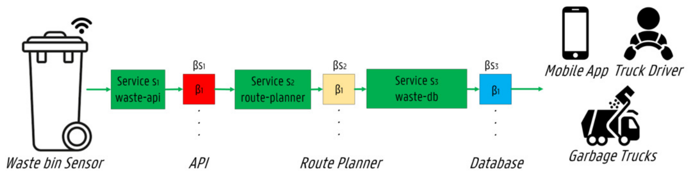
B.3.3.6 Minimizing Data Transfer Time - (MIN T)

In an IoT scenario, sensors need to communicate with gateways to send their collected data to the Fog-cloud infrastructure. This objective concerns the reduction of the data transfer time of a single message by optimizing the sensor's association with the several heterogeneous gateways available in the network. If a given gateway is supporting a high number of sensors, the maximum bandwidth for each sensor drops and thus the data transfer time increases. In the model, the data transfer time T_{sr} is affected by the gateway bandwidth factor ζ_{gw}, which depends on the number of associated sensors with a given gateway. This minimization can be expressed as shown in (B.37).

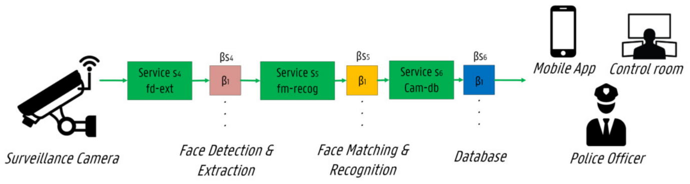
$$min \sum_{sr \,\varepsilon\, SR} T_{sr} \tag{B.37}$$

Figure B.1 The container-based Service Function Chains envisioned for the three evaluated use cases.

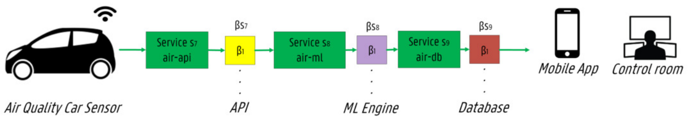
(a) Waste Management Use Case.



(b) Surveillance Camera Use Case.



(c) Air Quality Monitoring Use Case.



B.3.3.7 Minimizing User latency - (MIN UL)

Previously, two optimization objectives regarding latency have been presented. However, neither objective addresses the expected latency for each user based on their assigned application. To fully address user latency, the model needs to make decisions not only in terms of micro-service allocation but also on the replication factor for each micro-service. If users are spread across the network, it may be beneficial to allocate more micro-service instances to reduce the latency expected by each user. All these factors should be taken into account to fully achieve the minimization of the user latency. In the model, the user latency is between the user and the last service in the chain, since it is expected that users access the last service to obtain the required information. This objective can be expressed as shown in (B.38) based on the previously presented constraints regarding user latency.

$$min \sum_{u \,\varepsilon\, U} \psi_u \tag{B.38}$$

B.4 Use Cases

In this section, three Smart City use cases within the scope of Antwerp's City of Things testbed [39] are introduced. First, a Waste Management use case is presented where sensors are installed in waste bins to collect bins' fill levels to optimize garbage collection through route optimization services. Then, a surveillance camera scenario is detailed where cameras are placed in crowded streets to send continuous video streams to Fog locations where face detection and recognition services are applied in a distributed manner. Finally, an air quality monitoring use case is described where sensors are installed on vehicles to collect air quality data and then alert citizens in case air pollution levels are detected through machine learning (ML) services. In Fig. B.1, the container-based service chains for the three use cases are illustrated and the correspondent deployment requirements are shown in Table B.4.

B.4.1 Waste Management Scenario

Although waste bins are located everywhere (e.g. restaurants, office buildings, retail stores), garbage collection has been an inefficient service for years. Garbage trucks follow a predefined route without knowing if waste bins are currently empty or full. Furthermore, waste bins may get overloaded before the planned collection. This traditionally results in high maintenance and fuel costs. However, IoT can improve the performance of waste collection by gathering bin data [40]. For instance, sensors can be installed into waste bins to monitor which bins are full. By sending the collected data to a Fog-cloud infrastructure, route planning services can be executed to find the optimal route for each truck based on the bins' fill levels. Thus, drivers do not waste time driving to empty bins and broken bins may be repaired quickly. Drivers can access their optimized route through a mobile application that connects to a database service, enabling them to improve their customer service. Therefore, an IoT-based waste management service provides a more efficient waste collection through route optimization and higher driver productivity. The objective of this use case is to enable access to waste bin information.

Considering that for the waste management use case, each upload message is composed of a string of 12 chars (GPS Location - geohash) equal to 12 bytes, a 32-bit integer (timestamp) equal to 4 bytes and 1 floating point 64-bit number (fill level measure) equal to 8 bytes, the total number of payload bytes to be transmitted from the sensor to the Fog-cloud infrastructure is 24 bytes. In the evaluation, a general 13-byte header has been considered in each message as in LoRaWAN technologies. Therefore, each upload message is transmitted with at least 37 bytes, which is equal to 296 bits.

Table B.4: Deployment properties of the three evaluated use cases.

App.	Service	Chain Pos.	CPU (cpu)	RAM (Mi)	Min. Band. (Mbit/s)	Max. Users	User Cost	Device Upload Message Size (in B)
Waste (a_1)	waste-api (s_1)	1	0.25 (ω_{s_1})	0.25 (γ_{s_1})	5.0 (δ_{s_1})	5 (ρ_{s_1})	0.25 (λ_u)	37 ($\pi_a =$ 296 bits)
	route-planner (s_2)	3	0.5 (ω_{s_2})	1.0 (γ_{s_2})	8.0 (δ_{s_2})	8 (ρ_{s_2})		
	waste-db (s_3)	2	0.5 (ω_{s_3})	1.0 (γ_{s_3})	5.0 (δ_{s_3})	5 (ρ_{s_3})		
Camera (a_2)	fd-ext (s_4)	1	0.5 (ω_{s_4})	0.5 (γ_{s_4})	4.0 (δ_{s_4})	4 (ρ_{s_4})	1.0 (λ_u)	1500 ($\pi_a =$ 12000 bits)
	fm-recog (s_5)	2	1.0 (ω_{s_5})	2.0 (γ_{s_5})	8.0 (δ_{s_5})	8 (ρ_{s_5})		
	cam-db (s_6)	3	0.5 (ω_{s_6})	0.5 (γ_{s_6})	5.0 (δ_{s_6})	5 (ρ_{s_6})		
Air (a_3)	air-api (s_7)	1	0.25 (ω_{s_7})	0.25 (γ_{s_7})	4.0 (δ_{s_7})	4 (ρ_{s_7})	0.5 (λ_u)	93 ($\pi_a =$ 744 bits)
	ml-engine (s_8)	2	0.5 (ω_{s_8})	1.0 (γ_{s_8})	8.0 (δ_{s_8})	8 (ρ_{s_8})		
	air-db (s_9)	3	0.5 (ω_{s_9})	0.5 (γ_{s_9})	4.0 (δ_{s_9})	4 (ρ_{s_9})		

B.4.2 Surveillance Camera Scenario

Surveillance services have become highly relevant due to the possibility of identifying individuals or even objects in crowded areas [41]. However, connectivity issues still need to be addressed, including data transfer over limited bandwidth and high latency in sensor-cloud communications. For instance, imagine a low-quality surveillance camera requiring a continuous streaming bandwidth of 256 Kbit/s. Sending the entire data from a single video camera to the cloud translates into approximately 0.08 TB/monthly. To reduce the amount of data transmitted to the cloud, Fog-cloud infrastructures may be adopted by performing data operations locally. Surveillance cameras placed on particular streets or crowded areas send continuous video streams to Fog locations, where face recognition algorithms are performed in a distributed manner. Thus, suspicious individuals can be detected in near real-time. Fog nodes located close to the surveillance cameras receive their video streams and perform a first-level analysis, such as face detection and feature extraction tasks. Then, Fog nodes send the results to the cloud for global analysis operations, such as face matching and recognition operations, due to its high computational requirements. Afterwards, global outcomes can be presented in a central dashboard in a control room. Also, police officers may access the detection results through a mobile application. This approach has been previously presented in [42], to enable anomaly detection services in Fog Computing. An IoT-based surveillance camera service enables an efficient way of recognizing individuals in crowded areas by distributing tasks between Fog and cloud. The objective of this use case is to provide a near real-time face detection system.
Considering that the surveillance camera use case is a video streaming scenario, each upload message is composed of 1500 bytes, equal to 12000 bits. The value of 1500 bytes has been selected due to the Maximum Transmission Unit (MTU) limitation, which defines the largest packet size that can be sent over a network connection.

B.4.3 Air Quality monitoring scenario

Air pollution has become the largest environmental and public health challenge in the world. Air pollution leads to adverse effects on human health and climate change. As an initial proof of concept within Antwerp's City of Things project, air quality sensors have been installed on vehicle rooftops in collaboration with the Belgian postal services, Bpost. For daily mail delivery, Bpost keeps cars continuously driving around the city of Antwerp, thus enabling air quality sensors gathering data with broad city coverage. These sensors collect measures of typical gases (e.g. carbon dioxide (CO_2), Nitrogen Dioxide (NO_2), Particulate Matter indicators (PMIs)), and also climate data, such as temperature and humidity, which are then annotated with GPS locations. By sending the collected data to a Fog-cloud

infrastructure [43], ML services can be executed to find patterns and anomalies in the data. Then, citizens can be alerted in case unhealthy levels of air pollution are detected through a mobile application in near real-time. Thus, an IoT-based air quality monitoring system enables the detection of high concentrations of organic compounds and contributes to improved public health [44].
Considering that for the air quality monitoring use case, each upload message is composed of a string of 12 chars (GPS Location - geohash) equal to 12 bytes, a 32-bit integer (timestamp) equal to 4 bytes, 3 floating-point 64-bit numbers (PMIs) equal to 24 bytes and 5 floating-point 64-bit numbers (Nitrogen, Ozone, Carbon Monoxide, Temperature, Humidity) equal to 40 bytes, the total number of payload bytes to be transmitted from the sensor to the Fog-cloud infrastructure is 80 bytes. Therefore, each upload message is transmitted with at least 93 bytes, which is equal to 744 bits.

B.5 Evaluation Setup

In this section, the network infrastructure used for the MILP model is described. Then, input variables and optimization policies used in the evaluation are shown.

B.5.1 Network Infrastructure

Table B.5: Input variables for the MILP model.

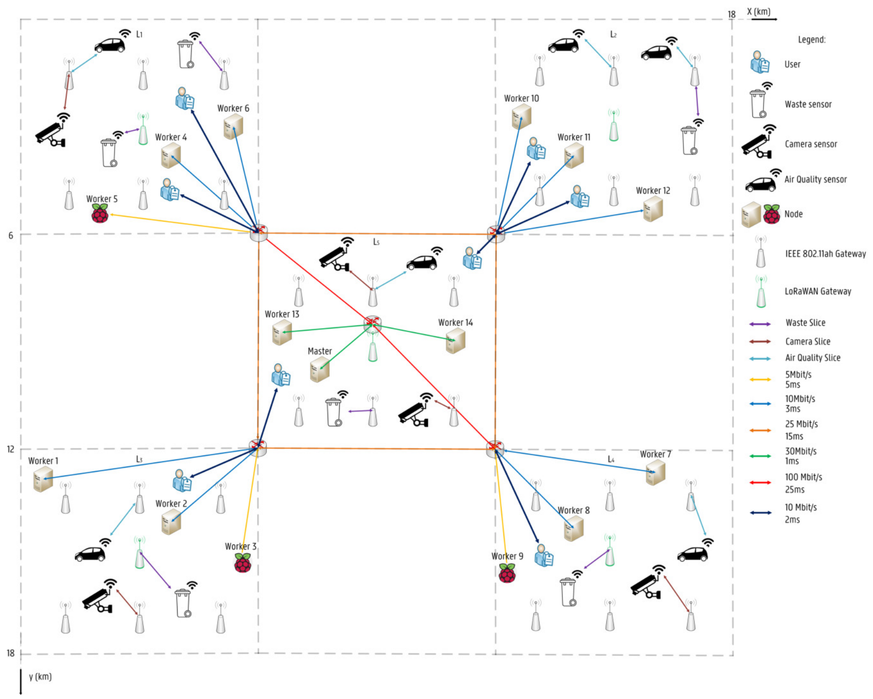
Variables	**Value**
A	3
S	9
L	5
SL	3
GW	35
	$a_1 : s_1 \to s_2 \to s_3$
$I_{a,s}$	$a_2 : s_4 \to s_5 \to s_6$
	$a_3 : s_7 \to s_8 \to s_9$
β	10 (Maximum Replication factor)
L_{gw}	5 (LoRaWAN gateways)
I_{gw}	30 (IEEE 802.11ah gateways)

The Fog-cloud infrastructure illustrated in Fig. B.2 has been represented in the model. Input variables are presented in Table B.5 based on the described network infrastructure. A total area of 324 km^2 has been considered. The Fog-cloud infrastructure is deployed on five locations L, where the micro-service provisioning is

Table B.6: The hardware configurations of each node.

Node	CPU (cpu)	RAM (Mi)	Band. (Mbit/s)
Worker 1 (n_1)	2.0 (Ω_{n_1})	4.0 (Γ_{n_1})	10.0 (Δ_{n_1})
Worker 2 (n_2)	2.0 (Ω_{n_2})	4.0 (Γ_{n_2})	10.0 (Δ_{n_2})
Worker 3 (n_3)	1.0 (Ω_{n_3})	2.0 (Γ_{n_3})	5.0 (Δ_{n_3})
Worker 4 (n_4)	2.0 (Ω_{n_4})	4.0 (Γ_{n_4})	10.0 (Δ_{n_4})
Worker 5 (n_5)	1.0 (Ω_{n_5})	2.0 (Γ_{n_5})	5.0 (Δ_{n_5})
Worker 6 (n_6)	2.0 (Ω_{n_6})	4.0 (Γ_{n_6})	10.0 (Δ_{n_6})
Worker 7 (n_7)	2.0 (Ω_{n_7})	4.0 (Γ_{n_7})	10.0 (Δ_{n_7})
Worker 8 (n_8)	2.0 (Ω_{n_8})	4.0 (Γ_{n_8})	10.0 (Δ_{n_8})
Worker 9 (n_9)	1.0 (Ω_{n_9})	2.0 (Γ_{n_9})	5.0 (Δ_{n_9})
Worker 10 (n_{10})	2.0 ($\Omega_{n_{10}}$)	4.0 ($\Gamma_{n_{10}}$)	10.0 ($\Delta_{n_{10}}$)
Worker 11 (n_{11})	2.0 ($\Omega_{n_{11}}$)	4.0 ($\Gamma_{n_{11}}$)	5.0 ($\Delta_{n_{11}}$)
Worker 12 (n_{12})	2.0 ($\Omega_{n_{12}}$)	4.0 ($\Gamma_{n_{12}}$)	10.0 ($\Delta_{n_{12}}$)
Worker 13 (n_{13})	6.0 ($\Omega_{n_{13}}$)	16.0 ($\Gamma_{n_{13}}$)	30.0 ($\Delta_{n_{13}}$)
Worker 14 (n_{14})	6.0 ($\Omega_{n_{14}}$)	16.0 ($\Gamma_{n_{14}}$)	30.0 ($\Delta_{n_{14}}$)
Master (n_{15})	8.0 ($\Omega_{n_{15}}$)	24.0 ($\Gamma_{n_{15}}$)	30.0 ($\Delta_{n_{15}}$)

Figure B.2 A Fog-cloud infrastructure for the IoT service allocation problem.



possible. Each location manages a set of three nodes. The hardware configurations of each node are shown in Table B.6. Each node, gateway, and sensor have a given location associated. Coordinate positions (x, y) are randomly attributed to each sensor, while for each gateway, coordinate positions are strategically attributed to cover the entire five locations. Based on these coordinates, the distance matrix D is calculated by the euclidean distance formula as shown in (B.39). Then, the path loss matrix PL is calculated based on the path loss formulas previously described by using the calculated distance matrix values. The bandwidth matrix B_{n_1,n_2} is based on the available bandwidth capacity. Also, all latency matrices τ are calculated based on the shown latency values.

$$D(gw, sr) = \sqrt{(x_{gw} - x_{sr})^2 + (y_{gw} - y_{sr})^2} \tag{B.39}$$

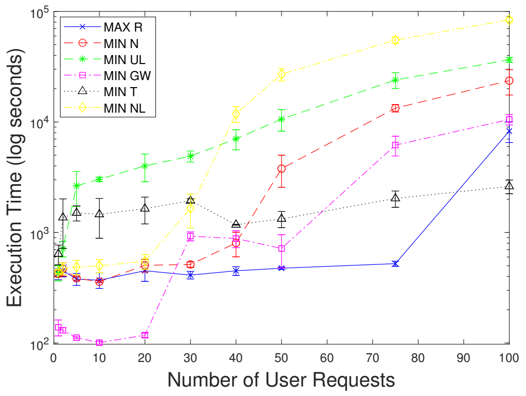
B.5.2 Input variables & Optimization Policies

Table B.7: The evaluated optimization policies.

A - Minimum Latency	B - Energy Efficiency
1 - MAX R	1 - MAX R
2 - MIN UL	2 - MIN N
3 - MIN NL	3 - MIN GW
4 - MIN T	

The described MILP formulation has been implemented in Java using the IBM ILOG CPLEX ILP solver [45]. As previously mentioned, several optimization objectives are executed iteratively, where the optimal solutions of previous objectives are added as additional constraints to the model for the subsequent iterations. Two policies have been evaluated. In each iteration of the model, a different optimization objective has been considered. In Table B.7, the evaluated optimization policies are shown. On the one hand, policy A is responsible for finding micro-service allocations focused on reducing latency. Also, it tries to minimize the sensor's data transfer time by spreading wireless traffic across multiple gateways. On the other hand, policy B is related to energy efficiency, since it minimizes the number of active nodes needed to allocate all the required micro-service instances. Furthermore, policy B maximizes gateway usage by connecting all sensors to the minimum number of gateways possible. It should be highlighted that for the two policies, the number of accepted user requests is maximized first. The model has been executed on the HPC-UGent supercomputing infrastructure [46]. Cluster nodes have been requested to run the experiments (2 x 18-core Intel Xeon Gold 6140 @ 2.3 GHz processor with 32 GB of memory). All policies have been evaluated 30 times and confidence intervals of 95% have been considered in the evaluation.

Figure B.3 The execution time of the different optimization objectives for the waste management use case.



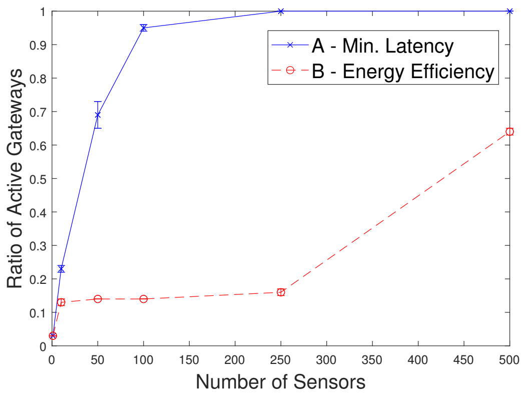
B.6 Results

In this section, the results are detailed. First, the execution time of the different optimization objectives is shown, followed by performance outcomes on the three presented use cases. Finally, a joint use case is shown followed by an analysis concerning the impact of service migrations when provisioning strategies are altered.

B.6.1 Execution Time per Objective

In Fig. B.3, the execution time of the different optimization objectives is shown for the waste management use case. The MILP model has been executed until the optimal solution for each objective has been found. The number of sensors has been set to 100 throughout the experiment. By increasing the number of user requests, the execution time of most objectives increases due to the increased allocation complexity. More micro-service instances need to be allocated since more users are sending requests. The most affected objectives by the increase of user requests are the *MIN UL* and the *MIN NL*. The complexity increases significantly for these two objectives since several users are spread across the network area and the latency needs to be reduced. In fact, the *MIN UL* requires on average 43 minutes to find the optimal solution already for 5 user requests while the *MIN NL* requires on average 8 minutes for the same number of user requests. Additionally, the execution time of the *MIN N* objective significantly increases for user requests higher than 40. Over 40 requests, still minimizing the number of nodes used in the micro-service allocation becomes a difficult task. For example, the *MIN N* requires on average 13 minutes and 63 minutes to find the optimal solution for 40 and 50 user requests, respectively. Also, the execution time of both *MIN T* and *MIN G* objectives slightly increase during the experiment since the number of sensors is

Figure B.4 The gateway usage rate for the waste management use case.



kept constant and, thus, the complexity of both objectives does not increase significantly. The *MAX R* objective slightly increases throughout of the experiment, however, it greatly increases for 100 user requests requiring more than an hour and a half to find the optimal solution, meaning that finding any solution that meets all constraints and maximizes the acceptance of 100 requests is quite time-consuming. Additionally, it should be highlighted that regarding sequential convergence speed, the optimization order that is applied has a strong impact on execution time and on the solution space. For instance, the two strategies evaluated can be combined into a single optimization strategy. As a first objective, *MIN N* is applied followed by the *MIN UL* optimization. As expected, the second optimization is quite limited since the overall value found in the first iteration is added as a constraint. So, the optimization in terms of user latency only occurs if the current allocation scheme can be further optimized without changing the overall value found in the first iteration. Thus, the execution time of the second objective is significantly faster when the first objective is considered since user latency cannot be fully optimized.

In summary, the MILP model can find a solution for most objectives in rather small execution time for user requests lower than 20 (i.e most cases 4 - 10 minutes). However, for the MIN UL, the model takes on average 1 hour and 6 minutes to find a solution for 20 user requests. Both *MIN UL* and *MIN NL* objectives related to latency reduction do not scale well. For example, the *MIN UL* requires on average 4 hours and 18 minutes to find the optimal solution for 50 user requests while the *MIN NL* requires on average 14 hours and 15 minutes for the same number of user requests. The execution time of the other objectives is still acceptable for values lower than 75 user requests.

Figure B.5 The data transfer time per sensor for the waste management use case.

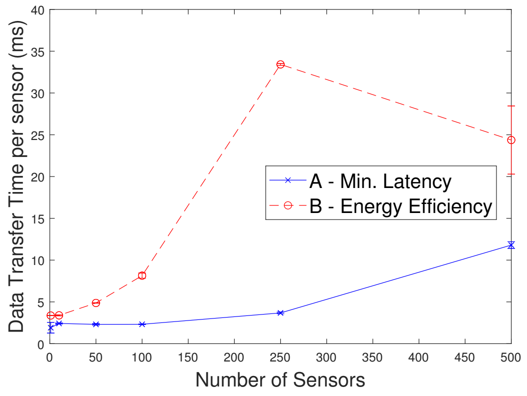


Figure B.6 The node utilization rate for the waste management use case.

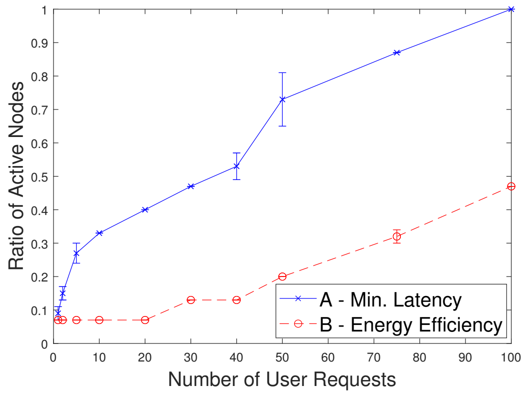


Figure B.7 The number of micro-service instances allocated for the waste management use case.

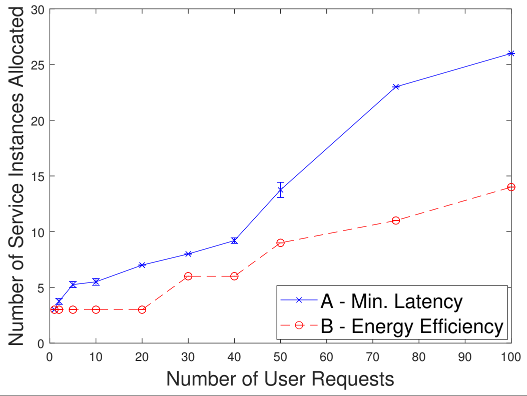
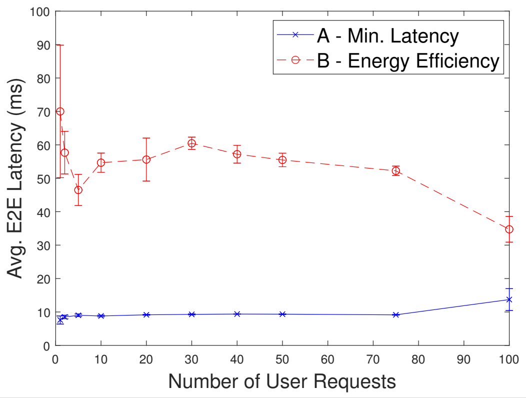


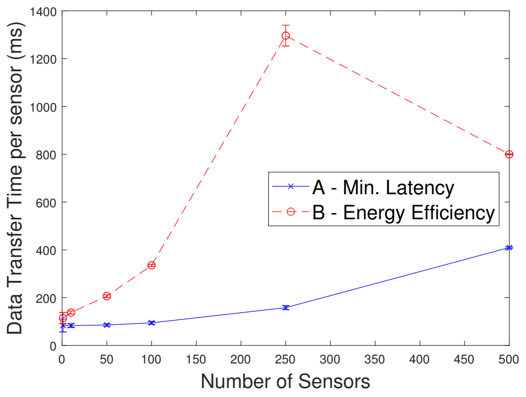
Figure B.8 The expected E2E latency per user for the waste management use case.



B.6.2 Waste Management Use Case

In Fig. B.4, the ratio of active gateways for each policy is shown. On the one hand, high usage rates are shown for policy A since no optimization objective is included regarding gateway efficiency. On the other hand, for policy B, the ratio of active gateways slightly increases with the increase of connected sensors. For example, for policy B, results show that only 16% of the gateways are needed for 250 sensors and that only for 500 sensors, the ratio is higher than 60%. In contrast, policy A requires on average more than 90% of active gateways already for 100 sensors. Thus, policy B shows a higher gateway efficiency. In Fig. B.5, the sensor's data transfer time for each policy is illustrated. As expected, policy A achieved the lowest values of data transfer time per sensor while policy B achieved the highest values since both policies represent opposing strategies. For example, for 250 sensors, each sensor requires on average 33 ms to send a message for policy B while for policy A, each sensor requires only 3.5 ms. This difference occurs because of the gateway usage rate. As previously shown, policy B needs on average only 16% active gateways while policy A requires 90% to minimize the sensor's data transfer time. In Fig. B.6, the ratio of active nodes for each policy is shown. On the one hand, for policy B, the active number of nodes slightly increases with the increase of user requests, since this policy is focused on maximizing energy efficiency in the infrastructure. On the other hand, for policy A, the active nodes greatly increase since this policy is focused on reducing latency. It should be noted that for 40 and 50 user requests, the confidence interval for policy A is higher because user latency can be reduced by activating a different number of nodes, which depends on the user's location. This can be considered a borderline case, where a single user on a particular location could mean that another node needs to be active and thus the higher confidence interval. In Fig. B.7, the exact number of

Figure B.9 The sensor's data transfer time for the surveillance camera use case.



micro-service instances allocated on the network for both policies is depicted. As expected, policy A allocates more micro-service instances on the network than policy B. This results in reduced E2E latency as shown in Fig. B.8. Latency values between 8 and 10 ms are obtained for policy A throughout the experiment even for a high number of user requests while values between 30 and 70 ms are obtained for policy B since the micro-service allocation is not focused on E2E latency but energy efficiency. However, for 100 user requests, policy A is not able to further minimize latency since the infrastructure is already exhausted. Thus, the average E2E latency increases to 13 ms, which is still considerably low when compared to policy B, which obtains latency values of 34.73 ms.

In summary, the Waste Management scenario represents a low bandwidth use case. As shown, differences can be achieved in terms of micro-service allocation depending on the provisioning strategy. Node usage rates are distinct among the two evaluated strategies, which resulted in different results in terms of E2E latency. The gateway usage rate directly affects the data transfer time of each sensor, which could be an important factor. Nevertheless, the sensor's data transfer time variations for the two evaluated strategies (between 33 ms - 3.5 ms in the worst case) can be classified as not particularly relevant for this use case.

B.6.3 Surveillance Camera Use Case

The variations of the ratio of active gateways for both policies are similar to the waste management use case, thus the graph is not included to avoid repetition. Nevertheless, the sensor's data transfer time variations are higher as illustrated in Fig. B.9 since this use case is related to a video streaming scenario (i.e. high bandwidth requirements). As shown, policy A achieved the lowest values of data transfer time per sensor. For instance, for 250 sensors, each sensor requires on

Figure B.10 The node utilization rate for the surveillance camera use case.

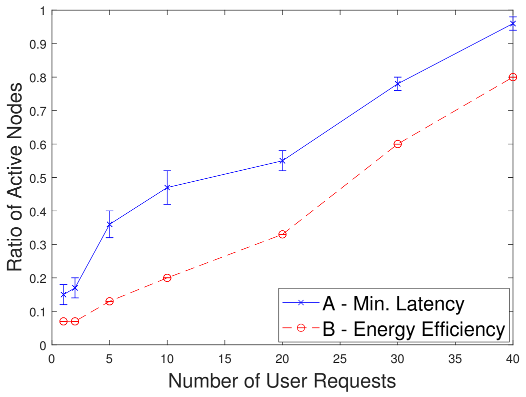


Figure B.11 The number of micro-service instances allocated for the surveillance camera use case.

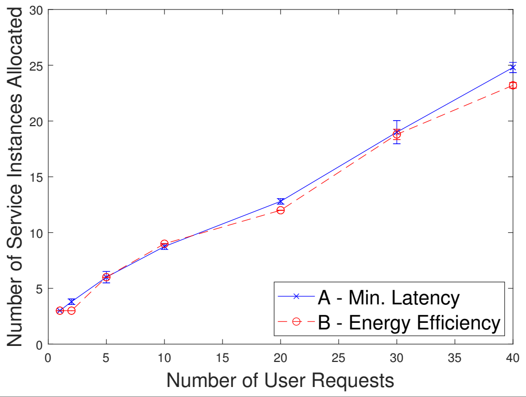


Figure B.12 The expected E2E latency per user for the surveillance camera use case.

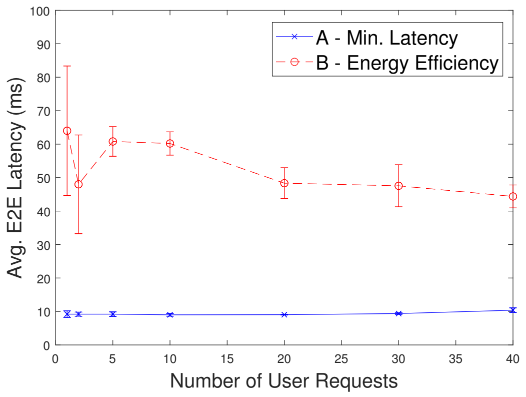
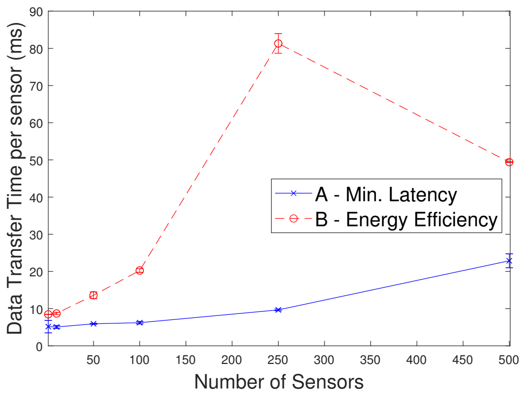


Figure B.13 The sensor's data transfer time for the air quality monitoring use case.



average 158 ms to send a message for policy A while for policy B, each sensor requires at least 1.3 seconds. This significant difference occurs due to the high bandwidth requirements for this use case. In Fig. B.10, the ratio of active nodes for each policy is shown. The ratio of active nodes for both policies slightly increases with the increase of user requests. The difference between policy A and B is lower when compared to the waste management use case, due to the difference in user cost (λ_u), 1.0 and 0.25, respectively for both use cases. Also, the number of micro-service instances allocated on the network demonstrated in Fig. B.11 clearly shows that the fluctuations are attenuated due to the user cost. Nevertheless, the pattern in terms of E2E latency remains similar as shown in Fig. B.12. Latency values between 8 and 10 ms are obtained for policy A while values between 30 and 70 ms are obtained for policy B.

In summary, the Surveillance Camera scenario represents a high bandwidth use case regarding video streaming. As shown, significant differences are achieved in terms of micro-service allocation and the sensor's data transfer time. For instance, notable data transfer time variations (between 158 ms - 1.3 seconds in the worst case) have been shown, which make this use case extremely challenging in the IoT ecosystem.

B.6.4 Air Quality Monitoring Use Case

As expected, the variations of the ratio of active gateways for both policies are similar to the waste management and the surveillance use case, thus the graph is not included. In Fig. B.13, the sensor's data transfer time for each policy is illustrated. For instance, for 250 sensors, each sensor requires on average 9.6 ms to send a message for policy A while for policy B, each sensor requires 81 ms. In Fig. B.14, the ratio of active nodes for each policy is shown. The ratio

Figure B.14 The node utilization rate for the air quality monitoring use case.

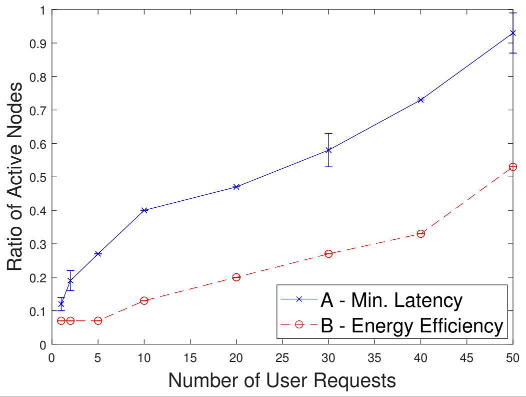


Figure B.15 The number of micro-service instances allocated for the air quality monitoring use case.

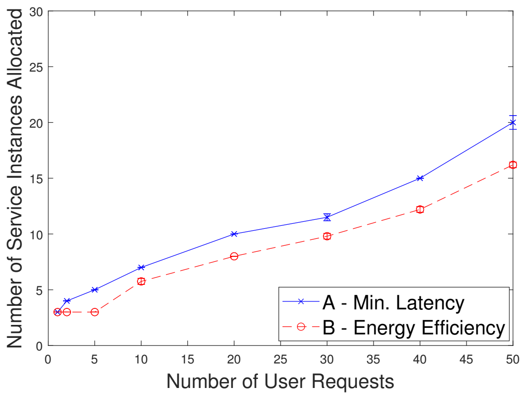


Figure B.16 The expected E2E latency per user for the air quality monitoring use case.

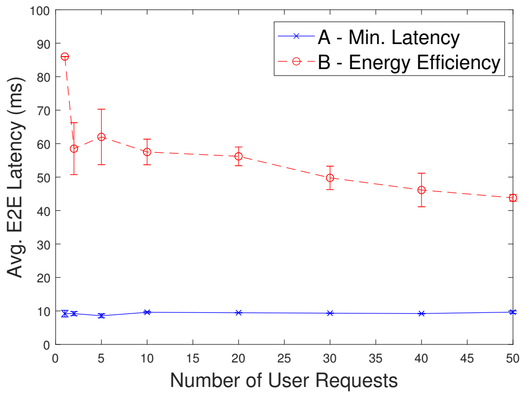
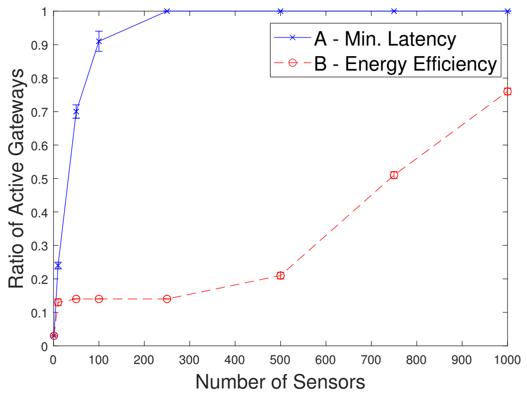


Figure B.17 The gateway usage rate for the joint use case.



of active nodes slightly increases for policy B with the increase of user requests while for policy A, the ratio significantly increases to minimize the user latency. For example, for 10 user requests, 40% of nodes are active for policy A while for policy B only 13.3% of nodes are needed. In Fig. B.15, the number of micro-service instances allocated on the network is shown. Similar to the surveillance scenario, the differences in terms of micro-service allocation are attenuated due to the user cost. Finally, in Fig. B.16, the expected E2E latency is shown. As previously demonstrated for the other two scenarios, the E2E pattern remains the same. Latency values between 8 and 10 ms are obtained for policy A while values between 40 and 80 ms are obtained for policy B.

In summary, the Air Quality Monitoring scenario represents a medium bandwidth use case regarding a near real-time air pollution detection service. As shown, differences are achieved for the two evaluated strategies, specifically in terms of node usage rates and E2E latency. Also, variations in the sensor's data transfer time are obtained depending on the applied strategy. Results show differences between 9.6 ms - 81 ms, which make this use case more challenging than the waste management use case.

B.6.5 Joint Use Case: Optimizing all Use Cases together

The Joint use case corresponds to a real-world scenario where multiple applications need to be deployed in the network. The three previously evaluated use cases need to be allocated in the network at the same time since users need to access all their respective applications. The user's assigned application has been randomized throughout the experiment while increasing user requests. In Fig. B.17, the ratio of active gateways for each policy is shown. Higher usage rates are shown for policy A since no optimization objective is included regarding gateway effi-

Figure B.18 The sensor's data transfer time for the joint use case.

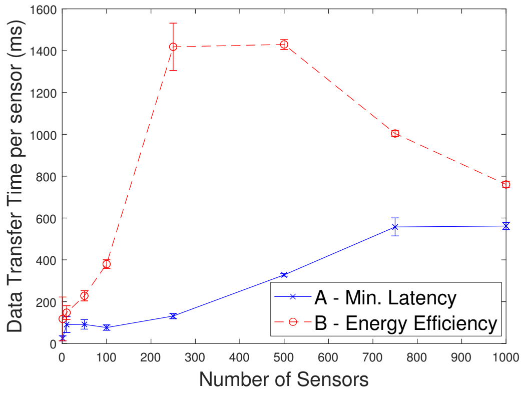


Figure B.19 The node utilization rate for the joint use case.

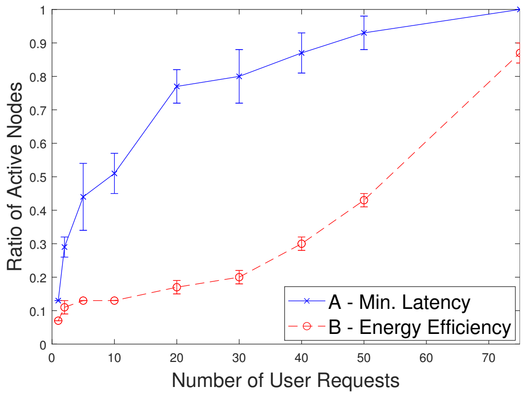


Figure B.20 The number of micro-service instances allocated for the joint use case.

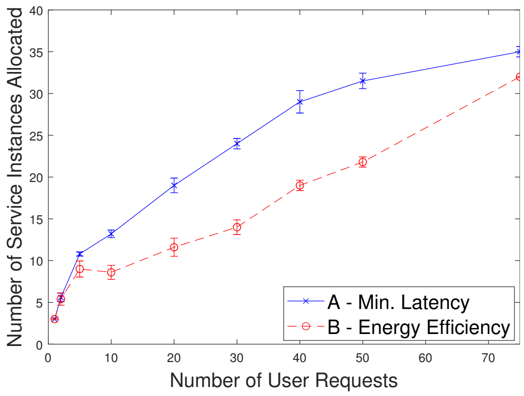
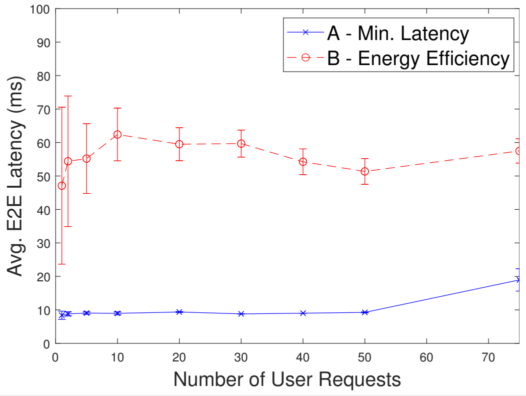
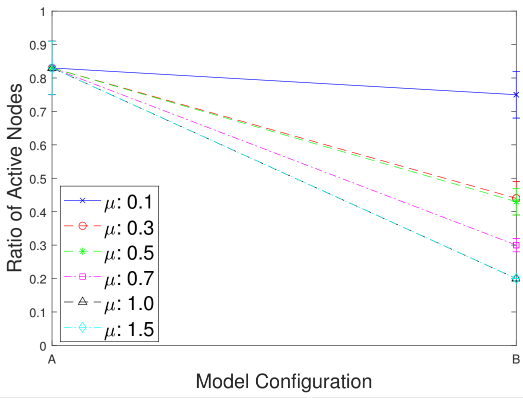


Figure B.21 The expected E2E latency for the joint use case.



ciency. In contrast, for policy B, the ratio of active gateways slightly increases with the increase of connected sensors. Results show that only 51% of the gateways are needed for 750 sensors and that only for 1000 sensors, the ratio is higher than 70%. Policy A requires already more than 65% of active gateways for only 50 sensors. Previous use cases showed similar results in terms of gateway usage rates, but in contrast, for this particular scenario, differences are achieved. This occurs because, in the previous use cases, all sensors are assigned to the single application deployed in the network while in this scenario, three applications are deployed and sensors have different application associations. In Fig. B.18, the data transfer time per sensor for each policy is illustrated. As expected, the differences between the two strategies have been obtained. For example, for 250 sensors, each sensor requires on average 1.4 seconds to send a message for policy B while for policy A, each sensor takes on average only 130 ms. Policy A can reduce the data transfer time of each sensor at the cost of a higher gateway usage rate. Additionally, in Fig. B.19, the ratio of active nodes for each policy is shown. For instance, for 20 user requests, 17% of the nodes are needed to deploy all the required micro-service instances for policy B while for policy A, 77% of nodes are active to fully achieve the optimal solution. For high values of user requests, policy B is not able to reduce the number of active nodes. For example, for 75 user requests, on average 87% of nodes are needed while policy A needs all nodes active in the network to minimize latency. In Fig. B.20, the number of micro-service instances allocated in the network for both policies is shown. The differences between both provisioning strategies significantly increase for values higher than 5 user requests. In fact, for high values of user requests, policy A deploys 10 more instances than policy B. In Fig. B.21, the obtained results in terms of E2E latency are shown. Policy A can minimize the E2E latency since it deploys more micro-service instances, however, for 75 user requests, the infrastructure is exhausted and the latency increases to 19

Figure B.22 The node utilization rate by changing from policy A to B.



ms. Policy B by focusing on optimizing energy efficiency is only able to minimize latency to 57 ms for the same number of user requests.

In summary, the Joint Use Case represents a real-world scenario where multiple applications are deployed in the network and cloud providers want to offer their users the best QoS for all of their services. As shown, large differences are achieved in terms of micro-service allocation depending on the provisioning strategy. Cloud providers must determine which requirements are more important for their services to optimize these at the maximum. Clear trade-offs have been presented. Reducing the sensor's data transfer time implies an increase of active gateways and, thus, higher energy costs. Optimizing the energy efficiency in the infrastructure does not translate into low latency values. In fact, to minimize the latency a higher number of service instances needs to be allocated.

B.6.6 Changing Provisioning Strategies: Impact of Service Migrations

Table B.8: The evaluated policies for the dynamic scenario.

	A → B	B → A
Service Migration Factor (μ)	150% 100% 70% 50% 30% 10%	150% 100% 70% 50% 30% 10%

The last part of the evaluation considered a use case where cloud providers may want to change their provisioning strategy. Thus, an important factor to measure is the number of service migrations needed to find the optimal allocation scheme. For this scenario, the joint use case has been considered with 50 sensors and 30 users.

Figure B.23 The number of micro-service instances allocated by changing from policy A to B.

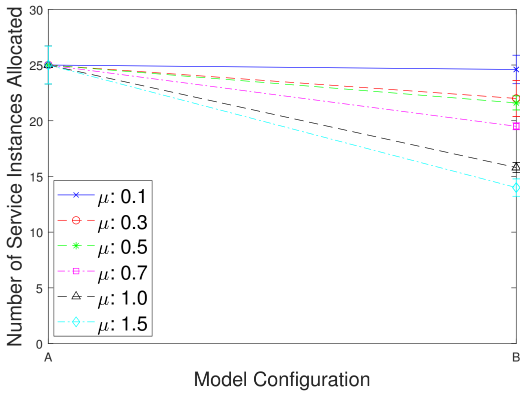


Figure B.24 The expected E2E latency per user by changing from policy A to B.

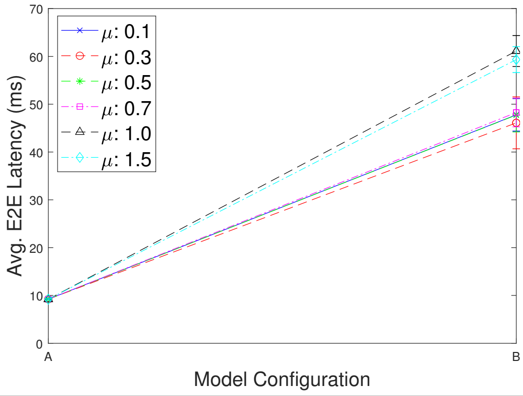


Figure B.25 The node utilization rate by changing from policy B to A.

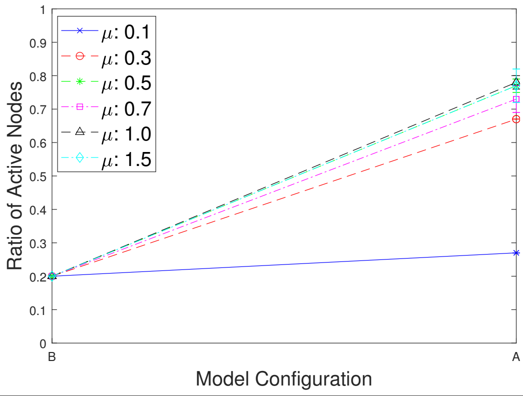


Figure B.26 The number of micro-service instances allocated by changing from policy B to A.

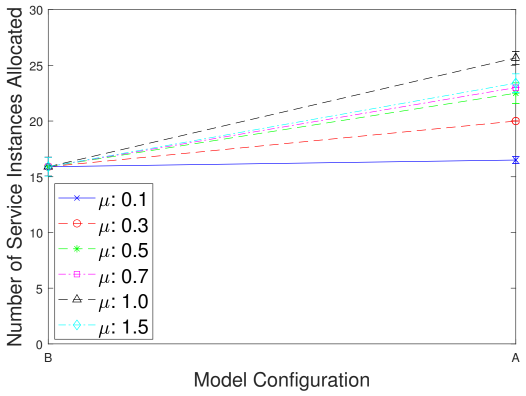
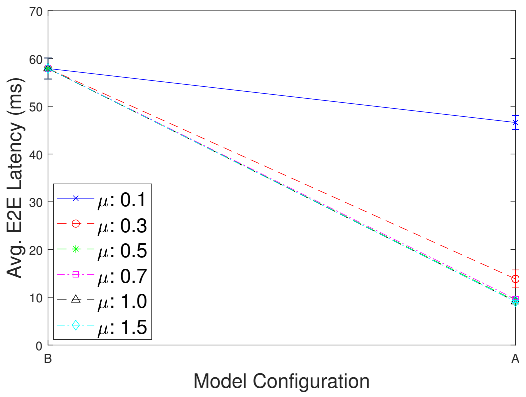


Figure B.27 The expected E2E latency per user by changing from policy B to A.



Sensors and user locations have been changed between strategies by attributing new x and y coordinates. Constraints have been included in the model to make sure that the newly calculated positions are inside the evaluation area. The allocation strategy has been changed and different service migration factors have been evaluated as shown in Table B.8. It should be highlighted that a migration factor of 150% has been evaluated since in the model a previous iteration may only need to deploy 4 micro-service instances in the network, but a further iteration focused on low latency may allocate 6 micro-service instances. Thus, a migration factor higher than 100% is allowed in the MILP formulation. The impact of service migrations must be studied because it can lead to service disruptions and, thus, it may be desirable to find a sub-optimal solution, in which service migrations are kept to a minimum to reduce the delay caused by service reallocations.

In the evaluation, three factors have been evaluated: the ratio of active nodes, the number of micro-service instances allocated and the expected E2E latency for each user. On the one hand, in Fig. B.22, the ratio of nodes by changing from policy A to B is shown while in Fig. B.23 and Fig. B.24, the number of micro-service instances allocated and the expected E2E latency are shown, respectively. As expected, policy A requires a high number of active nodes to minimize latency and when policy B is applied, it will reallocate micro-service instances to only a small percentage of nodes to maximize energy efficiency. For example, for a service migration factor of 0.5, the number of active nodes is reduced to 43%, which corresponds to a total reduction of 50%, but it increases the expected E2E latency per user from 9ms to 47.7 ms. Furthermore, policy B can significantly reduce the number of micro-service instances allocated in the network from 25 to 14 replicas for the highest migration factor ($\mu = 1.5$). On the other hand, in Fig. B.25, the ratio of nodes by changing from policy B to A is shown while in Fig. B.26 and Fig. B.27, the number of micro-service instances allocated and the expected E2E latency are shown, respectively. As expected, policy B requires a small number of nodes active and when policy A is applied, it will reallocate micro-service instances to reduce latency. For example, even for a small migration factor of 0.3, the expected E2E latency can be reduced from 57 ms to 13.8 ms by deploying on average 4 more micro-service replicas, which increases the node utilization rate from 20% to 66.6%. For higher migration factors, policy A can reduce the expected E2E latency per user between 9 and 10 ms at a cost of a high node utilization rate and a high number of deployed micro-service instances. It should also be noted that for a migration factor of 1.5, the results are similar to the joint use case for the same number of user requests, which was expected.

In summary, this scenario corresponds to a real-world use case where service reallocations may be needed when provisioning strategies are changed. The evaluation has been carried out to quantify what the differences are in terms of service allocation by restricting the MILP model with different migration factors. As shown, significant differences can be achieved. Service providers must decide which allocation strategy is suitable for their services and at what time a different allocation strategy should be applied. The evaluation proved that ILP-based solutions can provide the optimal solution, however, at the expense of significant execution time. Depending on the problem complexity, several hours might be needed to find the optimal allocation scheme. Nevertheless, service providers can adopt the model as a reference benchmark for their allocation algorithms and they may even calculate the optimal scheme depending on the current network demand. For instance, if they want to change their allocation policy between the two evaluated provisioning strategies, they can use the model to calculate the optimal scheme and then apply it in their network. It might be worth waiting a few hours and properly allocate all resources for this new policy than just quickly change everything and end up with a sub-optimal scheme. Results showed that to change between the two evaluated provisioning strategies a substantial percentage of service migrations are required to find the optimal allocation solution (higher than 70%). Clear trade-offs have been presented between optimizing energy efficiency and minimizing latency. The model not only allocates service chains, but also deals with sensor's data transfer times and bandwidth requirements which impact the E2E service quality. The model can be executed weekly or even daily depending on the expected demand.

B.7 Conclusions

In this appendix, a MILP formulation for the IoT service provisioning problem is presented, which takes SFC concepts, different LPWAN technologies and multiple optimization objectives into account. In recent years, the need for resource provisioning strategies for Fog Computing is increasing due to the deployment of IoT use cases. Billions of connected devices are expected which will make it impractical for current network architectures to support this massive growth since services will be requested on-demand simultaneously by multiple devices at different locations. Cloud providers will need proper service allocation solutions to minimize infrastructure costs and maximize QoS. The proposed MILP model considers not only cloud requirements but also characteristics stemming from wireless aspects to deal with these demanding requirements. The model considers multiple objectives, such as the acceptance of user requests, user latency reduction or increasing gateway efficiency. To the best of our knowledge, the work goes beyond the current state-of-the-art by providing a complete E2E resource provisioning in Fog-cloud environments while considering both cloud and wireless requirements. Evaluations have been performed to assess the proposed MILP formulation for Smart City use cases. Results show clear trade-offs between the different applied strategies. Cloud providers must decide which allocation strategy is suitable for their services and at what time a different strategy could be applied. Service migrations must also be considered as an important factor in the service allocation. Obtained results show that to change between provisioning strategies a substantial percentage of service migrations are required to find the optimal allocation solution (higher than 70%). The result of the work can serve as a benchmark in research covering IoT provisioning issues in Fog-cloud environments since the model approach is generic, considers several cloud and wireless aspects and can be applied to a wide range of IoT use cases. Future heuristics can be evaluated based on the measured execution times (e.g. latency-aware algorithms vs the *MIN UL* objective). As future work, the MILP model will be validated through real service deployments. Additionally, dynamic user demands will be studied, which will allow the approach to learn from network behavior.

Acknowledgment

This research was performed within the project "Intelligent DEnse And Longe range IoT networks (IDEAL-IoT)" under Grant Agreement
#S004017N, from the fund for Scientific Research-Flanders (FWO-V). The computational resources (Stevin Supercomputer Infrastructure) and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by Ghent University, FWO and the Flemish Government – department EWI.

References

[1] Mung Chiang and Tao Zhang. Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, 2016.

[2] Bhagya Nathali Silva, Murad Khan, and Kijun Han. Towards sustainable smart cities: A review of trends, architectures, components, and open challenges in smart cities. *Sustainable Cities and Society*, 38:697–713, 2018.

[3] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017–2022 White Paper, 2019. URL https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.pdf.

[4] Mithun Mukherjee, Lei Shu, and Di Wang. Survey of fog computing: Fundamental, network applications, and research challenges. *IEEE Communications Surveys & Tutorials*, 20(3):1826–1857, 2018.

[5] Pengfei Hu, Sahraoui Dhelim, Huansheng Ning, and Tie Qiu. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of network and computer applications*, 98:27–42, 2017.

[6] Charith Perera, Yongrui Qin, Julio C Estrella, Stephan Reiff-Marganiec, and Athanasios V Vasilakos. Fog computing for sustainable smart cities: A survey. *ACM Computing Surveys (CSUR)*, 50(3):32, 2017.

[7] Kais Mekki, Eddy Bajic, Frederic Chaxel, and Fernand Meyer. A comparative study of lpwan technologies for large-scale iot deployment. *ICT express*, 5(1):1–7, 2019.

[8] Andrew J Wixted, Peter Kinnaird, Hadi Larijani, Alan Tait, Ali Ahmadinia, and Niall Strachan. Evaluation of lora and lorawan for wireless sensor networks. In *2016 IEEE SENSORS*, pages 1–3. IEEE, 2016.

[9] Juan Carlos Zuniga and Benoit Ponsard. Sigfox system description. *LPWAN@ IETF97, Nov. 14th*, 25, 2016.

[10] Minyoung Park. Ieee 802.11 ah: sub-1-ghz license-exempt operation for the internet of things. *IEEE Communications Magazine*, 53(9):145–151, 2015.

[11] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. In *Present and ulterior software engineering*, pages 195–216. Springer, 2017.

[12] Deval Bhamare, Raj Jain, Mohammed Samaka, and Aiman Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155, 2016.

[13] Yanghao Xie, Zhixiang Liu, Sheng Wang, and Yuxiu Wang. Service function chaining resource allocation: A survey. *arXiv preprint arXiv:1608.00095*, 2016.

[14] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Towards delay-aware container-based service function chaining in fog computing. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2020.

[15] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. Leveraging cloudlets for immersive collaborative applications. *IEEE Pervasive Computing*, 12(4):30–38, 2013.

[16] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[17] Lien Deboosere, Bert Vankeirsbilck, Pieter Simoens, Filip De Turck, Bart Dhoedt, and Piet Demeester. Efficient resource management for virtual desktop cloud computing. *The Journal of Supercomputing*, 62(2):741–767, 2012.

[18] Antonio Brogi and Stefano Forti. Qos-aware deployment of iot applications through the fog. *IEEE Internet of Things Journal*, 4(5):1185–1192, 2017.

[19] Olena Skarlat, Stefan Schulte, Michael Borkowski, and Philipp Leitner. Resource provisioning for iot services in the fog. In *2016 IEEE 9th international conference on service-oriented computing and applications (SOCA)*, pages 32–39. IEEE, 2016.

[20] Olena Skarlat, Matteo Nardelli, Stefan Schulte, and Schahram Dustdar. Towards qos-aware fog service placement. In *2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC)*, pages 89–96. IEEE, 2017.

[21] Swati Agarwal, Shashank Yadav, and Arun Kumar Yadav. An efficient architecture and algorithm for resource provisioning in fog computing. *International Journal of Information Engineering and Electronic Business*, 8(1):48, 2016.

[22] Anila Yasmeen, Nadeem Javaid, Obaid Ur Rehman, Hina Iftikhar, Muhammad Faizan Malik, and Fatima J Muhammad. Efficient resource provisioning for smart buildings utilizing fog and cloud based environment. In *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 811–816. IEEE, 2018.

[23] Jingjing Yao and Nirwan Ansari. Qos-aware fog resource provisioning and mobile device power control in iot networks. *IEEE Transactions on Network and Service Management*, 16(1):167–175, 2018.

[24] Jingjing Yao and Nirwan Ansari. Fog resource provisioning in reliability-aware iot networks. *IEEE Internet of Things Journal*, 2019.

[25] Nouha Kherraf, Hyame Assem Alameddine, Sanaa Sharafeddine, Chadi Assi, and Ali Ghrayeb. Optimized provisioning of edge computing resources with heterogeneous workload in iot networks. *IEEE Transactions on Network and Service Management*, 2019.

[26] Francesco Chiti, Romano Fantacci, Federica Paganelli, and Benedetta Picano. Virtual functions placement with time constraints in fog computing: a matching theory perspective. *IEEE Transactions on Network and Service Management*, 2019.

[27] Hamid Reza Arkian, Abolfazl Diyanat, and Atefe Pourkhalili. Mist: Fog-based data analytics scheme with cost-efficient resource provisioning for iot crowdsensing applications. *Journal of Network and Computer Applications*, 82:152–165, 2017.

[28] Xiaosha Chen, Supeng Leng, Ke Zhang, and Kai Xiong. A machine-learning based time constrained resource allocation scheme for vehicular fog computing. *China Communications*, 16(11):29–41, 2019.

[29] Masoumeh Etemadi, Mostafa Ghobaei-Arani, and Ali Shahidinejad. Resource provisioning for iot services in the fog computing environment: An autonomic approach. *Computer Communications*, 2020.

[30] Ranesh Kumar Naha, Saurabh Garg, Andrew Chan, and Sudheer Kumar Battula. Deadline-based dynamic resource allocation and provisioning algorithms in fog-cloud environment. *Future Generation Computer Systems*, 104:131–141, 2020.

[31] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Resource provisioning for iot application services in smart cities. In *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2017.

[32] Samir Dawaliby, Abbas Bradai, Yannis Pousset, and Roberto Riggio. Dynamic network slicing for lorawan. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 134–142. IEEE, 2018.

[33] Toni Adame, Albert Bel, Boris Bellalta, Jaume Barcelo, and Miquel Oliver. Ieee 802.11 ah: the wifi approach for m2m communications. *IEEE Wireless Communications*, 21(6):144–152, 2014.

[34] Evgeny Khorov, Andrey Lyakhov, Alexander Krotov, and Andrey Guschin. A survey on ieee 802.11 ah: An enabling networking technology for smart cities. *Computer Communications*, 58:53–69, 2015.

[35] Ferran Adelantado, Xavier Vilajosana, Pere Tuset-Peiro, Borja Martinez, Joan Melia-Segui, and Thomas Watteyne. Understanding the limits of lorawan. *IEEE Communications magazine*, 55(9):34–40, 2017.

[36] Stefan Aust and Tetsuya Ito. Sub 1ghz wireless lan propagation path loss models for urban smart grid applications. In *2012 International Conference on Computing, Networking and Communications (ICNC)*, pages 116–120. IEEE, 2012.

[37] Pascal Jörke, Stefan Böcker, Florian Liedmann, and Christian Wietfeld. Urban channel models for smart city iot-networks based on empirical measurements of lora-links at 433 and 868 mhz. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6. IEEE, 2017.

[38] Amina Šljivo, Dwight Kerkhove, Le Tian, Jeroen Famaey, Adrian Munteanu, Ingrid Moerman, Jeroen Hoebeke, and Eli De Poorter. Performance evaluation of ieee 802.11 ah networks with high-throughput bidirectional traffic. *Sensors*, 18(2):325, 2018.

[39] Jose Santos, Thomas Vanhove, Merlijn Sebrechts, Thomas Dupont, Wannes Kerckhove, Bart Braem, Gregory Van Seghbroeck, Tim Wauters, Philip Leroux, Steven Latre, et al. City of things: enabling resource provisioning in smart cities. *IEEE Communications Magazine*, 56(7):177–183, 2018.

[40] Alexey Medvedev, Petr Fedchenkov, Arkady Zaslavsky, Theodoros Anagnostopoulos, and Sergey Khoruzhnikov. Waste management as an iot-enabled service in smart cities. In *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, pages 104–115. Springer, 2015.

[41] Junjue Wang, Brandon Amos, Anupam Das, Padmanabhan Pillai, Norman Sadeh, and Mahadev Satyanarayanan. A scalable and privacy-aware iot service for live video analytics. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 38–49. ACM, 2017.

[42] José Santos, Philip Leroux, Tim Wauters, Bruno Volckaert, and Filip De Turck. Anomaly detection for smart city applications over 5g low power wide area networks. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2018.

[43] Obelisk. a platform for building scalable applications on iot centric timeseries data, 2020. URL https://obelisk.ilabt.imec.be/api/v2/docs/.

[44] Yun Cheng, Xiucheng Li, Zhijun Li, Shouxu Jiang, Yilong Li, Ji Jia, and Xiaofan Jiang. Aircloud: a cloud-based air-quality monitoring system for everyone. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 251–265. ACM, 2014.

[45] IBM ILOG. Ibm cplex ilog optimization studio, 2020. URL https://www.ibm.com/products/ilog-cplex-optimization-studio.

[46] HPC-UGent. High performance computing infrastructure, 2020. URL https://www.ugent.be/hpc/en.

C
Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and Research Directions

This appendix extends the conclusions discussed in Chapter 6 by addressing open challenges and future directions regarding low latency service delivery in next-generation networks. It presents a comprehensive review of ongoing research on low latency service delivery and proposes a taxonomy and specific evaluation criteria to classify research across different domains. Emerging applications (e.g., Virtual Reality (VR), autonomous cars) add even more stringent requirements to the infrastructure, calling for considerable advancements towards cloud-native architectures. The state-of-the-art has been reviewed to identify the most promising trends that will impact the applicability and performance of next-generation applications. Thus, the main contributions of this appendix are the taxonomy for low latency service delivery, the comprehensive review of current literature, the discussion of open challenges and future directions alongside lessons learned, and prospects on emerging use cases.

⋆ ⋆ ⋆

José Santos, Tim Wauters, Bruno Volckaert and Filip De Turck.

Published in IEEE Communications Surveys & Tutorials, 2021.

Abstract The advent of softwarized networks has enabled the deployment of chains of virtual network and service components on computational resources from the cloud up to the edge, creating a continuum of virtual resources. The next generation of low latency applications (e.g. Virtual Reality (VR), autonomous cars) adds even more stringent requirements to the infrastructure, calling for considerable advancements towards cloud-native micro-service-based architectures. This appendix presents a comprehensive survey on ongoing research aiming to effectively support low latency services throughout their execution lifetime in next-generation networks. The current state-of-the-art is critically reviewed to identify the most promising trends that will strongly impact the full applicability and high performance of low latency services. This appendix proposes a taxonomy as well as specific evaluation criteria to classify research across different domains addressing low latency service delivery. Current architectural paradigms such as Multi-access Edge Computing (MEC) and Fog Computing (FC) alongside novel trends on communication networks are discussed. Among these, the integration of Machine Learning (ML) and Artificial intelligence (AI) is introduced as a key research field in current literature towards autonomous network management. A discussion on open challenges and future research directions on low-latency service delivery leads to the conclusion, offering lessons learned and prospects on emerging use cases such as Extended Reality (XR), in which novel trends will play a major role.

C.1 Introduction

The deployment of high-bandwidth and low latency 5G network infrastructures has been driving the digital transformation of network services in Industry 4.0, Smart Cities, Healthcare and connected vehicles. Founded on the principles of Software-Defined Networking (SDN) [1] and Network Function Virtualization (NFV) [2], programmable networks interconnect virtual cloud, fog, and edge resources, which help to bring low latency services to reality. These technological advancements pave the way to support high reliability and low latency services in 5G such as Ultra-Reliable Low-Latency Communication (URLLC) services required for autonomous driving and factory automation use cases. The massive growth of the Internet of Things (IoT) is pushing the boundaries of network architectures by transforming everyday objects into smart connected devices. To overcome the hurdles to arrive at truly end-to-end (E2E) services, which meet the even more stringent requirements (e.g. higher bandwidths, lower latencies) of future applications, next-generation (6G) networks [3] have to provide distributed orches-

tration and management functionalities to integrate a continuum of virtual computing resources with a wide variety of ultra-broadband (radio access and core) and high-precision network links. Bandwidth requirements for Extended Reality (XR) or Holographic Type Communication (HTC) applications will rise well above 1Tbps, while their interactive experiences require sub-millisecond latencies [4]. Pervasive and ambient connectivity (billions of devices per km^2 in the Internet of Everything (IoE) with ultra-low latency, near-proximity communication) and autonomous service delivery (with levels up to 99.99999% reliability, even at high speeds of users or Unmanned Aerial Vehicles (UAVs) [5]) add further stringent requirements, as shown in Table C.1. Supported by pervasive network telemetry and analytics, Artificial Intelligence (AI) capable of dynamically meeting the real-time requirements of cloud-native applications will be brought into play as well. Distributed learning at both the network and application levels will contribute to the development of smart, highly interactive and reliable environments, allowing for a high Quality of Experience (QoE) to end-users. On the business side, increasingly more stakeholders will be involved in the E2E service chain, requiring very dynamic service contracts and relationships beyond mere network connectivity, expanding towards control and management aspects. Extensions to the ETSI NFV MANO model [6] should incorporate these micro-operators, their interfaces, roles and templates to accelerate network slice setup.
To ensure low E2E service latency for all emerging use cases, current network architectures need to drastically change. Several improvements are currently being implemented at the Radio Access Network (RAN) and core alongside novel networking systems incorporating SDN, NFV and caching concepts. Multiple steps in the service execution causing increased delays have to be recognized as potential barriers to low latency service delivery. Distributed (hierarchical) SDN architectures are considered better candidates than current centralized approaches [7]. However, further research is needed to understand how such architectures can incorporate local or global network measurements and analysis to steer the message routing, and how service-level objectives can be enforced in the network. In the network path, fog-cloud infrastructures need to be set up to execute several micro-services of a service chain, enabling flexible deployments [8]. Moreover, the integration of intelligence at the edge will lead to ML-driven networks able to support highly dynamic management updates under varying network circumstances, meeting the requirements of cloud-native applications and services over the continuum of virtual resources. This appendix revisits several aspects of the state-of-the-art on low latency service delivery in next-generation networks. The contributions of the appendix can be summarized as follows:

- Present up-to-date research and novel trends in low latency service delivery by conducting a comprehensive review of the current literature.
- Propose a taxonomy on low latency service delivery by considering different

aspects of next-generation networks that will impact the applicability and performance of low latency services.

- Provide specific evaluation criteria to classify research across different domains based on the presented taxonomy.
- Identify current open challenges and future directions.
- Provide lessons learned and prospects of emerging use cases such as Smart Cities and XR.

Despite the importance of low latency services in next-generation networks, to the best of our knowledge, a comprehensive and detailed survey on novel trends for low latency service delivery is still missing. The close interplay between computing and networking is key to support low latency 6G services in the future. The integration of AI/ML at the edge will also play a major role in enabling autonomous networks. This appendix presents valuable insights for the industry and research community into ongoing research and novel trends pushing towards distributed cloud-native infrastructures capable of supporting low E2E service latency. The next section outlines how our survey differs from the current state-of-the-art and presents the taxonomy on low latency service delivery.

Table C.1: Requirements for emerging use cases [3, 4].

Use Case	Latency	Reliability	Throughput
E-Health	< 1 ms	99.99999%	1 - 100 Mbps
Smart Cities	10ms - 1s	$> 99.999\%$	1 - 100 Mbps
UAV services	1 - 10 ms	99.99999%	1 - 10 Mbps
Industrial IoT	1 - 10 ms	99.99999%	1 - 10 Mbps
Extended Reality	< 1 ms	99.99999%	> 1 Tbps
Self-driving cars	< 1 ms	99.99999%	1 - 10 Mbps

C.2 Survey Methodology

This section starts by introducing the existing surveys and tutorials in the literature related to low latency services. Second, our literature review is explained followed by a taxonomy on low latency service delivery. Then, the evaluation criteria used to classify research across different domains is introduced. Finally, the appendix structure is presented.

C.2.1 Existing Surveys & Tutorials

Several surveys and tutorials on 5G networks are available in the literature. In [9], architectural paradigms and emerging technologies are presented. 5G research projects are also briefly introduced. In [10] and [11], SDN and NFV concepts for 5G networks are addressed. Both surveys highlight how SDN and NFV complement each other and discuss the key role both technologies will play on next-generation networks. In [12], the integration of Multi-access Edge Computing (MEC) in 5G systems is assessed. It highlights the deployment of applications and services at the edge as one of the main benefits of MEC. Service migration and mobility support in MEC are also addressed in [13]. In [14], RAN, core network and caching concepts for 5G are discussed in detail, while in [15] resource management for 5G RAN systems is considered. Recently, in [16], service placement in Fog Computing (FC) is discussed while a comprehensive taxonomy for FC is proposed in [17]. The classification of FC applications based on ML is also presented in [18]. Hardware-accelerated platforms and infrastructures (e.g. Field-Programmable Gate Arrays (FPGAs), microprocessors) are also covered in [19], highlighting relevant studies for the softwarized execution of network services. Furthermore, a few surveys have already been published on 6G networks focused on architectures [20, 21] and wireless access networks [22]. Regarding applications, surveys and tutorials on emerging use cases exist such as Smart Cities [23], Autonomous cars [24], Augmented Reality (AR) [25] and Industrial IoT (IIoT) [26].

Low latency has also been addressed in recent surveys. Previous work [27] discusses latency reduction techniques by comparing their advantages with their overhead in implementation and deployment. The authors focus on communication protocols and how these techniques impact the latency perceived by end-users. Novel architectures and emerging applications fall out of their scope. Design principles and enabling technologies to deploy low-latency wireless communication networks have been analyzed in [28]. The authors reflect on how to meet the stringent requirements of future use cases while discussing the trade-offs between low latency and traditional performance metrics. Other surveys related to low latency exist in the current literature, but their scope is limited [29] or focused on a specific use case [30]. These surveys do not assess each contribution in the presence of specific criteria, as it is performed in this appendix. An exhaustive literature review is also missing, especially considering novel trends on low latency service delivery. In contrast, this appendix comprehensively reviews recent research and novel trends focused on enabling services that require low E2E latency in next-generation networks while proposing a taxonomy on low latency service delivery.

Figure C.1 Overview of the surveyed research.

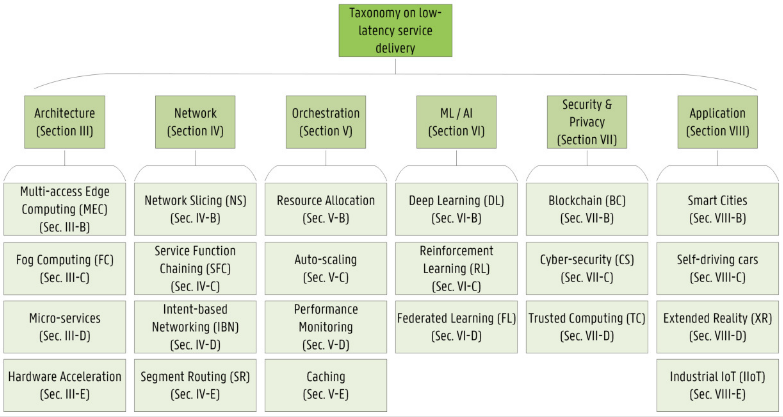
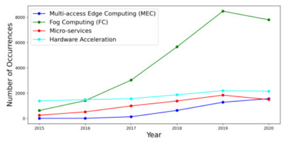
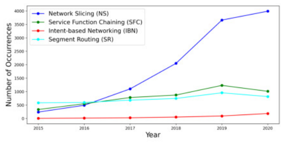

Taxonomy on low-latency service delivery
Architecture (Section III)
Network (Section IV)
Orchestration (Section V)
ML / AI (Section VI)
Security & Privacy (Section VII)
Application (Section VIII)
Multi-access Edge Computing (MEC) (Sec. III-B)
Fog Computing (FC) (Sec. III-C)
Micro-services (Sec. III-D)
Hardware Acceleration (Sec. III-E)
Network Slicing (NS) (Sec. IV-B)
Service Function Chaining (SFC) (Sec. IV-C)
Intent-based Networking (IBN) (Sec. IV-D)
Segment Routing (SR) (Sec. IV-E)
Resource Allocation (Sec. V-B)
Auto-scaling (Sec. V-C)
Performance Monitoring (Sec. V-D)
Caching (Sec. V-E)
Deep Learning (DL) (Sec. VI-B)
Reinforcement Learning (RL) (Sec. VI-C)
Federated Learning (FL) (Sec. VI-D)
Blockchain (BC) (Sec. VII-B)
Cyber-security (CS) (Sec. VII-C)
Trusted Computing (TC) (Sec. VII-D)
Smart Cities (Sec. VIII-B)
Self-driving cars (Sec. VIII-C)
Extended Reality (XR) (Sec. VIII-D)
Industrial IoT (IIoT) (Sec. VIII-E)

Figure C.2 Search occurrence of keywords based on the proposed taxonomy on low latency service delivery in Google Scholar.

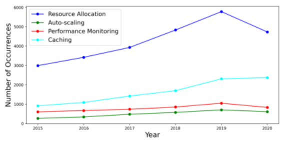
(a) Architecture.



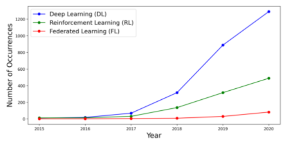
(b) Network.



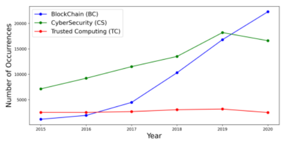
(c) Orchestration.



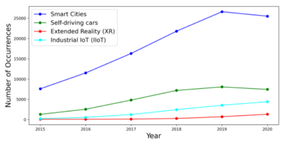
(d) ML/AI.



(e) Security & Privacy



(f) Application.



C.2.2 Literature Review

The literature on low latency service delivery encompasses several domains, thus structuring and classifying the most relevant research is not a trivial task. Fig. C.1 shows the proposed taxonomy. Based on an exhaustive literature search, six main categories have been identified: *Architecture*, *Network*, *Orchestration*, *Security & Privacy*, *ML / AI* and *Application*. These six domains have been identified as crucial for the full applicability and high performance of emerging low latency services in next-generation networks. The review of research contributions on all six domains allows us to identify open issues relevant for several areas and aggregate reviewed work under these main categories. Efforts on radio and wireless access networks have been excluded from our survey since these topics are usually covered in dedicated surveys [31, 32]. Nevertheless, the importance of improving current access networks is acknowledged. Efforts focused only on SDN and NFV paradigms have also been excluded since these topics have been thoroughly addressed in the literature [33].
After performing various reiterative fine-tuned search processes based on multiple keywords, several research domains have been recognized under the six main categories. By dividing the main category into different domains, readers can easily access references addressing a specific issue. Within the first main category, *Architecture*, four research domains have been identified: MEC, FC, Micro-services and Hardware acceleration. These domains have been perceived as important enablers towards fully cloud-native infrastructures in next-generation networks. Research related to Cloudlets [34] has been left out since both MEC and FC are emerging in the last few years and are the main alternatives in this domain. Regarding the second category, *Network*, four research domains have been distinguished: Network Slicing (NS), Service Function Chaining (SFC), Intent-based Networking (IBN) and Segment Routing (SR). Within the third main category, *Orchestration*, four domains have been identified: Resource allocation, Auto-scaling, Performance monitoring and Caching. These areas demonstrate the importance of efficient management practices in the life-cycle of service components. The fourth category, *Security & Privacy*, addresses three research domains: BlockChain (BC), CyberSecurity (CS) and Trusted Computing (TC). Recent security guidelines focused on data protection, trust and authentication/authorization mechanisms have been identified since security aspects are commonly left out in current literature. This appendix tackles this gap in the state-of-the-art by reviewing novel security trends. Three domains have been recognized in the fifth main category, *ML / AI*: Deep Learning (DL), Reinforcement Learning (RL) and Federated Learning (FL). These emerging areas in ML/AI research are being investigated in the context of distributed clouds towards autonomous management. Finally, within the sixth category, *Application*, four research areas have been identified: Smart Cities, Self-driving cars, XR and IIoT. These application domains represent emerging use

cases in next-generation networks, where low latency is crucial for the proper service operation and end-user satisfaction.

Concerning the literature review, keywords based on the presented taxonomy have been searched on two publication databases, Google Scholar and IEEE Xplore. The number of occurrences in Google Scholar for all keywords is shown in Fig. C.2. For example, the search term used to determine the number of occurrences of FC has been *Architecture "Fog Computing"*. Similar keywords have been used for the other research domains in our taxonomy. Some domains were still unknown in 2017 until a significant increase occurred in 2019, especially regarding IBN and DL research. Some topics are still largely unexplored in academia as FL and IBN, while BC and DL had a tremendous increase over the last three years. Based on these graphs, research published between January 2017 and December 2020 has been examined. Table C.2 shows the number of publications reviewed in this appendix per main category and correspondent domain. Publications have been organized per chronological order on each domain subsection to improve readability since several works fall under different criteria. Moreover, works incorporate concepts falling in different categories, thus, the corresponding percentages are derived accordingly. For instance, for Architectural paradigms, eight papers (15.8%) have been analyzed in the context of MEC, six papers for FC (13.3%) while four papers have been reviewed for both Micro-services (5.8%) and Hardware acceleration (4.1%). The inclusion of research is based on the overall quality of the publication (i.e. number of citations, journal indexed in Web of Science). Only peer-reviewed works have been considered and short conference papers (typically between 2-4 pages) have not been included. Based on these principles, 120 publications have been selected to be thoroughly analyzed in the context of this appendix. The next subsection describes in detail the evaluation criteria applied to classify research on low latency service delivery.

C.2.3 Evaluation Criteria

Table C.3: The evaluation criteria applied to classify research on low latency service delivery.

Evaluation criteria	Description
Mobility (C1)	Mobility is an important requirement for next-generation networks where services can be requested on-demand by multiple devices at different locations.

continued on next page...

Table C.3: The evaluation criteria applied to classify research on low latency service delivery (continued)

Evaluation criteria	Description
Scalability (C2)	Millions of devices will be connected to the network. Cloud-native infrastructures need to accommodate applications with different latency requirements while adapting to the current network demand.
Energy Efficiency (C3)	The number of connected devices and the data they collect is growing exponentially. If data is processed centrally by traditional clouds, it would increase power consumption and latency. Cooperation between the edge, fog and the cloud will become even more important in future networks given the increased use of distributed ML techniques and data processing.
Isolation (C4)	Network slicing enables the efficient execution of different services on the same infrastructure with various latency thresholds.
Security & Privacy (C5)	Traditional security solutions are designed to protect enterprise networks and data centers through perimeter-based protections. These practices are no longer adequate for addressing security challenges in emerging use cases. For example, distributed malware monitoring observes low-powered devices to compensate for their limited security.
Resilience (C6)	Resilience must be an inherent property into next-generation networks. If failures occur, networks need to keep offering satisfactory Quality of Service (QoS) no matter what challenges they face. Otherwise, time and money are lost.
Reliability (C7)	Reliability is essential for low latency services in next-generation networks and will be even harder to maintain in distributed cloud-native infrastructures. Devices and end-users can connect from multiple locations with different access technologies, causing failures at the edge, fog, or cloud.

continued on next page...

Table C.3: The evaluation criteria applied to classify research on low latency service delivery (continued)

Evaluation criteria	Description
Heterogeneity (C8)	Devices hold various computing capacities (e.g. CPU, RAM) and will access the medium through different technologies (e.g. 5G, Wi-Fi). Cloud-native infrastructures must ensure such heterogeneous networks run smoothly.
Throughput (C9)	Future networks need to support low latency service delivery as well as high bandwidth data rates. XR is among the most throughput demanding emerging applications.
Federation (C10)	Computing resources will be geographically distributed across the edge, fog and cloud. Several domains can be operated via different service providers. Federation is required to manage and orchestrate services running on different providers that compose a complete application. Low E2E latency must be guaranteed under cloud-native federation.

The criteria to evaluate current research on low latency service delivery for next-generation networks are presented in Table C.3. Although low latency is the main focus, other requirements are also important to ensure low E2E latency. Several requirements have been translated into individual criteria. If the work addresses any of the given criteria in their methodology (e.g. architecture, algorithm), then it meets the individual criterion. Otherwise, the criterion is unmet when this requirement is not considered. The first criterion (C1) is supporting *Mobility*. Devices and end-users will request services on-demand at different locations. Without efficient mobility support, service discovery procedures for mobile devices may need to restart, causing service disruptions and degraded user experience. Mobility is crucial for low latency services to guarantee service continuity when end-users or devices are moving in the network, ensuring smooth handover processes. The second criterion (C2) is *Scalability*. Cloud-native infrastructures need to support millions of connected devices for multiple applications. The network demand at a given time determines the computing resources allocated to each application, meaning that these systems need to be elastic enough to scale up and down resources according to the current demand. The third criterion (C3) relates to *Energy Efficiency*. Latency and energy efficiency are often considered opposing strategies. If service providers offer low E2E latency, that usually translates into higher energy bills since more computing resources are allocated to provide lower latency to all users (limited resource sharing). Users want the best Quality of Service

Table C.2: The number of publications assessed per category and correspondent research domains.

Main Category	Research Domain	Number of Publications
Architecture (39.0%)	Multi-access Edge Computing	8 (15.8%)
	Fog Computing	6 (13.3%)
	Micro-services	4 (5.8%)
	Hardware Acceleration	4 (4.1%)
Network (30.7%)	Network Slicing	4 (8.3%)
	Service Function Chaining	7 (13.3%)
	Intent-based Networking	5 (4.1%)
	Segment Routing	6 (5.0%)
Orchestration (27.3%)	Resource Allocation	8 (13.3%)
	Auto-scaling	4 (3.3%)
	Performance Monitoring	5 (4.1%)
	Caching	6 (6.6%)
ML / AI (19.9%)	Deep Learning	5 (7.5%)
	Reinforcement Learning	6 (8.3%)
	Federated Learning	5 (4.1%)
Security & Privacy (27.4%)	Blockchain	6 (10.0%)
	Cybersecurity	5 (14.1%)
	Trusted Computing	4 (3.3%)
Application (24.8%)	Smart Cities	7 (10.8%)
	Self-driving cars	4 (4.1%)
	Extended Reality	4 (3.3%)
	Industrial IoT	7 (6.6%)

(QoS) for the minimum cost, while service providers want to meet the agreed QoS level by using a small fraction of their infrastructure, reducing their operational costs and maximizing their profit. The massive number of connected devices will generate a huge volume of data that, if processed centrally by traditional clouds, increases power consumption and latency. Efficient collaboration between edge, fog and cloud is crucial towards a more efficient and greener cloud-native infrastructure [35]. The fourth criterion (C4) reflects on *Isolation*. In next-generation networks, resources should be logically separated through slicing, an abstraction allowing resource sharing among several slices while preventing attacks and faults on a slice from affecting any other slices in the network. A network slice can be defined as a set of Network Functions (NFs) and resources (e.g. connectivity, storage) attributed to a specific service. These functions are chained to form a logical isolated E2E network slice responsible for offering the specified QoS level. NS is expected to significantly reduce operational costs and guarantee different latency demands for each supported application.

The fifth criterion (C5) relates to efficient *Security and Privacy* mechanisms. Security mainly deals with the integrity and availability of the system, while privacy concerns data protection and confidentiality. Any security breach can directly impact the privacy of the company or the individual if unauthorized access is granted to private data. In the past, security practices protected enterprises through perimeter-based solutions. Nowadays, data is spread across the network and stored at different levels (i.e. edge, fog, cloud), making traditional security practices inadequate. Security will be key to most emerging use cases since users would only subscribe to services where their privacy and data are protected. Thus, identifying security and privacy issues in distributed clouds will be the main challenge in security research for the next coming years. The sixth criterion (C6) concerns *Resilience*. Network resiliency is the ability to self-heal the network after unexpected failures [36]. Next-generation networks should offer self-healing features to yield sustained QoS levels under critical situations by predicting or identifying network failures. Reactive and proactive remedies can ensure service continuity and availability. The seventh criterion (C7) is *Reliability*. Reliability is often compared to availability, but a service can be available and not run properly. High reliability will be crucial for low latency services since service providers are expected to match the agreed QoS levels to satisfy their users. Reliability is also associated with resilience. Reliability is the goal of service providers, while resiliency contributes to its accomplishment. Services can be reliable since no failures have occurred, but they cannot be considered resilient since such failure-tolerant capabilities have not been tested. The eighth criterion (C8) relates to network *Heterogeneity*. Infrastructure nodes will hold different computing capabilities. For example, edge and fog nodes possess limited capacity compared to cloud nodes. This heterogeneity needs to be considered in resource allocation decisions - where and when to deploy service components is essential to manage the network heterogeneity and ensure services run as expected. The ninth criterion (C9) is *Throughput*. Emerging use cases such as XR require not only low latency but also high bandwidth data rates. Several network functionalities (e.g. data processing, ML operations) are currently being pushed to the edge to minimize latency and maximize throughput. Finally, the tenth criterion (C10) is supporting *Federation*. Cloud-native infrastructures will be operated via different service providers. Applications following the recent micro-service paradigm will be separated into multiple service components that can be deployed over distinct providers in a continuum of virtual resources from the cloud up to the edge. Cooperation over various network domains ensures proper management and orchestration of these applications and delivers low E2E latency.

C.2.4 Appendix Structure

The remainder of the appendix is organized as follows: Section C.3 presents architectural paradigms such as MEC and FC and respective literature. Section C.4 introduces and reviews recent progress in communication networks. Section C.5 reviews novel management and orchestration practices followed by recent advances on ML and AI towards automated management in section C.6. Section C.7 introduces novel security and privacy methods for next-generation networks. Section C.8 reviews the most prominent low latency use cases. Section C.9 focuses on open challenges and future directions while Section C.10 presents the lessons learned and discusses the prospects of emerging use cases such as Smart Cities and XR, in which novel trends will play a key role. Concluding remarks are presented in Section C.11.

C.3 Towards a continuum of virtual resources - Architectural paradigms

C.3.1 Overview

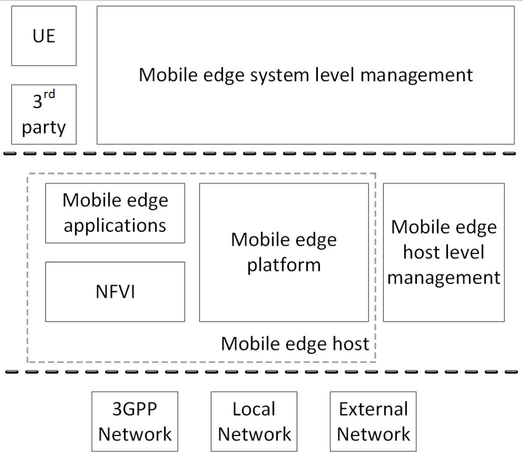
The advent of novel architectural paradigms enabled the deployment of service chains on computational resources from the cloud up to the edge. This brings several benefits such as low latency and mobility support. This section presents the key architectural concepts enabling application deployment in a continuum of virtual resources: MEC, FC, micro-services and hardware acceleration. Then, related research is discussed, in which the works are categorized based on our criteria. Table C.4 summarizes the reviewed works.

Table C.4: Summary of the reviewed works in terms of Architecture.

Research Domain	Authors	Main focus	Year	Evaluation Criteria									
				C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Multi access Edge Computing (MEC) (Sec. C.3.2)	Liu, X., et al. [37]	Service migration	2017	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗
	Ma, L., et al. [38]	Mobility support	2018	✓	✗	✗	✓	✓	✗	✗	✗	✓	✗
	Hsieh, H., et al. [39]	Edge Intelligence	2018	✗	✗	✓	✗	✗	✗	✓	✗	✓	✗
	Liu, J., et al. [40]	Network interoperability	2018	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗
	Yang, S., et al. [41]	Video streaming	2018	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗
	Shah, S., et al. [42]	Mobility Support	2020	✓	✗	✗	✗	✗	✗	✓	✓	✗	✓
	Ranaweera, P., et al. [43]	Security	2020	✗	✓	✗	✗	✓	✗	✗	✓	✗	✗
	Ksentini, A., et al. [44]	Slicing in MEC	2020	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗
Fog Computing (FC) (Sec. C.3.3)	Sookhak, M., et al. [45]	Vehic. networks	2017	✓	✓	✗	✗	✗	✗	✗	✓	✓	✗
	Moreno-V., R., et al. [46]	FC architecture	2017	✓	✓	✗	✗	✓	✗	✓	✓	✓	✓
	Sharma, P. K., et al. [47]	SDN-based FC	2017	✗	✓	✗	✗	✓	✓	✓	✓	✓	✗
	Bruschi, R., et al. [48]	SDN-based FC	2017	✗	✓	✗	✓	✗	✗	✗	✓	✗	✗
	Baccarelli, E., et al. [49]	IoT services	2017	✓	✓	✓	✗	✗	✗	✓	✓	✓	✗
	Santos, J., et al. [50]	Smart Cities	2018	✓	✗	✓	✗	✓	✗	✓	✓	✗	✗
Micro services (Sec. C.3.4)	Brenner, S., et al. [51]	Security	2017	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗
	Guija, D., et al. [52]	Security	2018	✗	✓	✗	✓	✓	✗	✗	✗	✗	✗
	Xu, R., et al. [53]	Blockchain architecture	2019	✗	✓	✗	✗	✓	✗	✗	✓	✗	✗
	Debauche, O., et al. [54]	Edge architecture	2020	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗
Hardware Acceleration (Sec. C.3.5)	Zhang, X., et al. [55]	scalable NFV	2017	✗	✓	✗	✓	✗	✗	✗	✓	✓	✗
	Umuroglu, Y., et al. [56]	Neural networks	2017	✗	✓	✓	✗	✗	✗	✗	✓	✓	✗
	Cai, R., et al. [57]	Neural networks	2018	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗
	Owaida, M., et al. [58]	FPGA-based acceleration	2019	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗

C.3.2 Multi-Access Edge Computing (MEC)

Figure C.3 The mobile edge system reference architecture [50].



MEC [59] is an industry initiative from the European Telecommunication Standards Institute (ETSI). It was launched in 2014 under a different naming: Mobile Edge Computing focused on bringing the current mobile network to the edge by adding Virtual Machine (VM) virtualization. In 2017, ETSI incorporated non-cellular operators' requirements (e.g. MEC hosts deployed in multiple networks owned by different providers, edge applications running collaboratively), thus the name changed to Multi-Access Edge Computing (MEC). The ETSI MEC technical committee is designing a reference architecture [60] for a mobile edge system as shown in Fig. C.3. MEC focuses on evolving the mobile network edges to create a cloud environment close to the RAN that hosts enhanced services provided by the Mobile Network Operator (MNO) or third parties. Mobile Edge (ME) applications run on top of a generic cloud infrastructure within the RAN: the Mobile Edge Host (MEH). The MEH encompasses a Mobile Edge Platform (MEP) responsible for executing ME applications on an NFV Infrastructure (NFVI), which provides computing, storage and network resources for the provisioning and consumption of ME services [60]. The envisioned use cases include IoT, AR, optimized caching and video analytics.
MEC aims to improve the mobile architecture to support low latency services, but a few challenges persist: services may need to be reallocated quickly, thus service migration and mobility support present an enormous challenge. In [37], Liu, X., et al. have proposed a mobility-aware coded probabilistic caching scheme for MEC-enabled small cells. The authors aim to maximize throughput by incorporating user mobility (C1) and distributed storage in their scheme. Results have shown that their scheme outperforms conventional probabilistic methods, attaining higher

throughput (C9) under different degrees of user mobility, content popularity and backhaul capability. In [38], Ma, L., et al. have presented an edge architecture supporting seamless migration (C1) of offloading services while keeping moving users connected to their nearest edge server. Docker container migration has been recognized as an important challenge in the literature. The authors propose to leverage the layered nature of the Docker storage system to reduce file system synchronization overhead. Their framework provides an isolated (C4) running environment for the offloading service via two layers of the system virtualization hierarchy. It also minimizes security risks (C5) posed to offloading services running on the edge servers. The authors state that isolation between different services provides a degree of security. The authors have evaluated the performance of container migration based on several metrics, such as latency, file compression and throughput (C9). Both works are promising, however, no clear strategy or guidelines have yet been defined on how to support mobility in future networks.

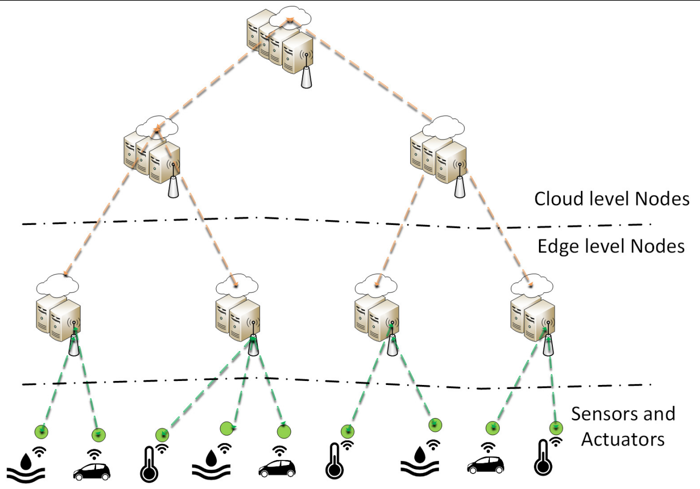
Recent works have also focused on providing intelligence at the edge and in the interoperability between wireless and wired networks or different networking components. In [39], Hsieh, H. et al. have studied virtualized MEC (vMEC) infrastructures to provide intelligence at the edge while reducing latency and increasing the available capacity. Their vMEC infrastructure applies container-based virtualization as an IoT gateway for flow control mechanisms and performance analysis. Results demonstrate that latency can be reduced up to 30%, maintaining high bandwidth for most services. The flow control mechanism has been introduced to reduce the CPU usage of the vMEC platform, reducing energy consumption (C3). Their approach not only reduces latency but also enhances user experience by improving service quality, which relates to reliability (C7). The authors have also focused on adjusting the throughput capacity with their flow control mechanism to avoid network congestion (C9). In [40], Liu, J., et al. have proposed an integrated networking scheme for MEC and fiber-wireless (FiWi) access networks. Their approach focuses on the dynamic orchestration of network, storage, and computing resources to meet diverse application demands, addressing the importance of mobility (C1) management. The lack of mobility data has been solved by collecting user's mobility information, such as locations and time via Access Points, Base Stations (BSs), or even sensors. Furthermore, the connectivity provided by FiWi access networks facilitates the direct communication of edge clouds without the core network, improving reliability (C7). The dynamically controlled routing enables efficient VM migration and service transmission link failure to ensure high reliability and availability of the services. Evaluations on an integrated scheme of edge clouds and multiple heterogeneous (C8) networks (e.g. FiWi) have been performed. Throughput (C9) levels have been verified between the edge and remote cloud servers. Video streaming services in MEC architectures have also been investigated in [41]. Yang, S., et al. have implemented a Proof-of-Concept

(PoC) for a video streaming use case in a MEC-based architecture. It focuses on assessing MEPs to deploy novel applications at the edge, such as intelligent video accelerating services that need low latency and high bandwidth. The authors have developed two ML models for video and radio channel quality prediction to improve the overall QoE of video streaming users. These mechanisms improve service reliability (C7) and meet expected QoS levels. Their approach improves the radio channel quality (and thus the throughput) between mobile devices and the BS (C9). By deploying the video streaming service at the edge, their approach guarantees high throughput for most users. In [42], Shah, S., et al. have proposed the integration of SDN and cloud-native virtualization techniques to facilitate the orchestration and management of MEHs. Their work focuses on E2E mobility support to maintain service continuity when users relocate from one MEH to another (C1). Request/reply messaging patterns have been implemented based on the ZeroMQ protocol for inter-process communication between their SDN application and the target MEP. Protocols such as ZeroMQ and MQ Telemetry Transport (MQTT) have been designed to minimize network bandwidth and ensure reliability (C7). A heterogeneous (C8) Radio Access Technology (RAT) deployment has been considered in the evaluation of their Vehicle-to-Everything (V2X) use case. Multiple mobile operators and different SDN controllers have also been assessed (C10). The authors have also proposed an inter-slice resource sharing and federation model as future work. The authors aim to extend their work to support NS in their framework for mobility management between heterogeneous network slices across edge clouds.

Security in MEC is another open challenge that has been studied in [43]. Ranaweera, P., et al. propose MEC to enable security-as-a-service features. Experiments have determined the security functions scalability deployed in an MEP (C2). Security (C5) has also been discussed in detail. Security services are executed in Docker containers offering application-level isolation. The service deployment follows the SFC concept to dynamically adapt resources and the auto-scaling of security functions to accommodate several traffic profiles. The authors have also considered the inherent heterogeneity (C8) of IoT devices, claiming that the deployment of security services as a third-party solution is needed due to the strict requirements introduced by IoT. Their approach helps to mitigate security concerns and support heterogeneous mobile services through its flexibility. Lastly, Ksentini, A., et al. [44] have proposed the integration of both MEC and NS in a novel scheme compliant with ETSI and 3GPP specifications (the standardization bodies working on MEC and NS, respectively). A novel orchestration architecture has been presented, incorporating the MEC paradigm as a 5G sub-slice. Two different models to support NS in MEC have been proposed. On the one hand, a multi-tenancy model assumes that the MEP is deployed at the edge NFVI and is shared among the multiple slices. On the other hand, an in-slice deployment model considers that the MEP is deployed inside the slice (C4). In both models, the MEP is deployed as a Virtual Network Function (VNF). Security and privacy (C5) concerns have also been addressed. The authors state that each slice should not access the traffic or other information owned by other slices. For example, slices should not access information about the users of another slice, such as their location or channel quality. Multi-tenancy has been addressed, which is related to federation concepts, but from a single provider perspective instead of a multi-provider one. Cited works are adequate alternatives for security enhancement in future MEC infrastructures.

C.3.3 Fog Computing (FC)

Figure C.4 High-level view of a Fog Computing environment [50].



Cisco has introduced the FC paradigm [61] in 2012 as an extension of cloud computing to provide resources on network edges to handle the massive growth of connected devices. Fig. C.4 presents a high-level view of a hierarchical FC architecture. In contrast to a centralized cloud, fog nodes are distributed across the network to act as an intermediate layer between end devices and the cloud. These so named fog nodes or edge locations are essentially small cloud entities that bring processing power, storage, and memory capacity closer to devices and end-users. This enables local operations, crucial for most IoT use cases to reduce the amount of data that needs to traverse the entire network up to the cloud. By deploying services at the network edge, FC can also provide lower latency than traditional clouds. FC and MEC are close concepts [62] differing in the considered interactions (i.e. between edges and cloud): MEC deploys services close to end-users to reduce latency and avoid congestion in the network core, while FC considers bi-directional communications between edges and cloud due to the hierarchical architecture.

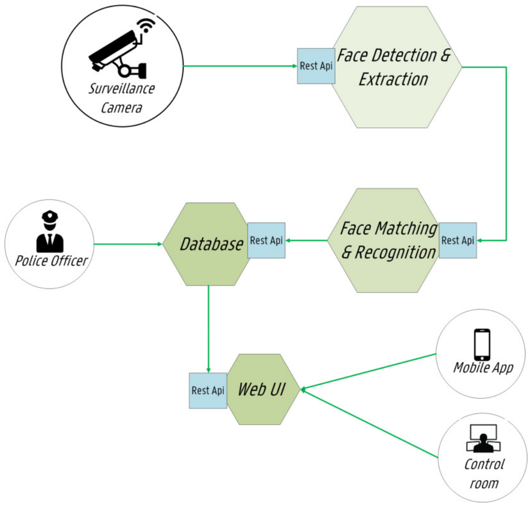
Novel FC architectures have been recently proposed, as in [45]. Sookhak, M., et al. have introduced a novel concept named Fog Vehicular Computing (FVC) to expand the computation and storage capacity of FC architectures. The authors describe a complete cross-layer architecture for FVC and introduce several components alongside a decision-making process for task scheduling. A Shopping Mall FVC use case has been evaluated, demonstrating the effectiveness of the proposed architecture. The feasibility of extending FC towards vehicles has been studied (C1). The evaluation has shown that deploying resources on vehicles could enhance a standard FC architecture. The authors state that their FVC architecture aims to improve the scalability of an FC infrastructure (C2). Different hardware capacities have been considered for fog nodes, which satisfies heterogeneity (C8). Throughput (C9) has also been evaluated in the context of communication costs at the fog layer. Their work compares an FC and their FVC architectures. The authors intend to focus on user security and privacy as future work, especially in terms of secured data access control and data encryption in FVC. In [46], Moreno-V., R., et al. have presented a Hybrid Fog and Cloud (HFC) framework to optimize the automated provisioning of virtual networks to connect geographically distributed fog and cloud sites. The approach adopts an agent-based solution allowing interaction with different cloud providers and fog infrastructures while providing scalability, security and multi-tenancy. The authors state that L2 overlay networks present advantages over L3 overlay networks, including native support for mobility (C1) and migration. Hosts in different sites will share a common overlay addressing scheme, enabling host migrations with minimal reconfiguration. Their HFC framework supports different topologies (e.g. tree or mesh), enabling the addition or removal of a fog or cloud location with minimal configuration (C2). The authors have also stated that it is necessary to guarantee data privacy and integrity in a hybrid environment (where the interconnection of different cloud and fog sites runs over public networks) by implementing the overlay networks over secure communication channels (C5). The authors remark that the reliability (C7) of the overlay network can be increased if HFC agents are deployed in a high-availability cluster that shares a common routing public IP. The authors support their HFC framework stating that tenants can deploy a virtual network over heterogeneous fog infrastructures and clouds (C8). A PoC of the HFC framework has been implemented to assess the throughput (C9) of L2 and L3 overlay networks. The HFC framework has also been designed to support multiple application providers and tenants (C10). Each provider can instantiate its applications and virtual networks to provide services to its end-users. Both works propose suitable FC-based architectures to manage a large number of connected devices and support low latency services.

Several works have also studied the integration of SDN concepts into FC to overcome scalability issues. In [47], Sharma, P. K., et al. have presented a distributed cloud architecture based on SDN and BC for secure and on-demand access to IoT services. The authors adopt SDN and BC to design a highly scalable architecture for IoT (C2). Leveraging BC technology, user privacy can be assured since third-party entities do not access or control user data (C5). Each user manages its own security keys, and each node stores only encrypted fragments of user data. Simulations have evaluated the accuracy of their architecture in detecting and mitigating saturation attacks at the edge of the network. Moreover, the authors state that if nodes fail, the computation should continue on another node to maintain resilience (C6) levels. The authors have also implemented a scheduling algorithm to match users' preferences, with reliability (C7) as a decisive factor. In the evaluation, nodes with different computing capacities have been considered (C8). Evaluations have assessed the delay, response time, throughput (C9), and the ability to detect real-time attacks of their approach. Results have shown that their approach provides higher throughput than traditional cloud infrastructures. The authors propose to focus on energy-efficient communications for edge devices as future work. In [48], Bruschi, R., et al. have discussed an SDN-based slicing scheme for multi-domain fog-cloud services. The approach has been designed with high scalability (C2) in mind, minimizing the number of OpenFlow (OF) rules in the overlay implementation. Their SDN scheme implements NS to provide service isolation (C4). Heterogeneity (C8) has often been addressed (e.g. nodes geographical distribution, different QoS requirements). Solving scalability concerns will definitively help adopting these novel architectural paradigms.

IoT and Smart City services have been thoroughly studied in the FC domain. In [49], Baccarelli, E., et al. have proposed to integrate FC and IoE, describing the main building blocks of a Fog of Everything (FoE) platform. The authors state that fog nodes should be arranged into spatial clusters to serve mobile devices through single-hop links to deal with high mobility patterns (C1). The authors state that the fog layer of the FoE platform should handle resource scalability (i.e. computing, storage, and network) of most big data applications (C2). The authors suggest that inter-device communications should occur through Device-to-Fog (D2F) links in place of Device-to-Device (D2D) links to reduce energy consumption (C3). The performance of the FoE architecture in terms of delay and energy, compared to a typical D2D architecture, has been evaluated. Results have shown that FoE achieves lower delays and higher energy efficiency. The authors also propose to implement Transmission Control Protocol (TCP) NewReno [63] to guarantee reliable E2E connections, even when facing network congestion or link failures (C7). Heterogeneous devices and network topologies have been assessed (C8). The energy assessment of the FoE architecture has also been made based on the average throughput (C9) of all established connections (e.g. F2D). Lastly, in [50], an FC framework for autonomous management and orchestration in 5G-enabled Smart Cities has been proposed. The approach follows the guidelines of ETSI NFV MANO, extending it with software components towards a fully integrated fog node management system. A Peer-to-Peer (P2P) fog protocol has been presented to exchange application service provisioning information between fog nodes. The work also follows guidelines defined by ETSI oneM2M [64] to monitor mobility patterns (C1) and provide proper device management and security (C5) functionalities. FC enables distributed malware monitoring tools to compensate for IoT devices' limited security and detect threats and attacks on time. Energy efficiency (C3) has also been discussed in the context of resource-constrained IoT devices and proper resource allocation. A distributed anomaly detection approach for FC has also been introduced to avoid transmissions of incorrect information and improve network reliability (C7). This anomaly detection scheme has been evaluated based on a heterogeneous (C8) fog-cloud environment. The cited works address important challenges for low latency service delivery by focusing on IoT concerns: mobility support, network reliability and interoperability between heterogeneous devices and environments.

C.3.4 Micro-services

Figure C.5 An example of a Surveillance camera use case based on the micro-service architecture.



Recently, micro-service patterns [65] have gained tremendous attention. Container-based services revolutionized the way developers build their applications. An application is decomposed into a set of loosely coupled services that are developed, deployed and maintained independently. Each service is responsible for a single task and communicates with the other services through lightweight protocols. These services can be developed in different programming languages and even using different technologies. Containers are currently the most promising alternative to the traditional monolithic application paradigm, mostly centralized and code-heavy. Containers are the main alternative to conventional VMs due to their low overhead and high portability. Fig. C.5 presents a high-level view of a Surveillance camera use case envisioned for Smart Cities based on the micro-service paradigm.

Security has been a major concern in micro-service architectures. In [51], Brenner, S., et al. have focused on integrating trusted execution based on Intel Software Guard Extensions (SGX) into micro-services. Their approach increases privacy and data confidentiality to sensitive applications deployed through micro-services (C5). By integrating SGX into the micro-service toolkit, the authors achieve higher levels of security. Results have proved the feasibility of their approach while showing low-performance overheads. Response time and throughput (C9) evaluations demonstrate equivalent throughput levels between secured micro-services and regular ones. In [52], Guija, D., et al. have proposed a 5G platform with integrated authentication and authorization features for micro-services in an NFV environment. Their architecture adopts the NFV-based SONATA Service Platform, which offers continuous integration and management of the entire VNFs life cycle. Keycloak [66], an Identity and Access Management open-source tool, has been proposed to isolate each sub-component in their architecture and ensure higher scalability (C2). The authors note the importance of sub-component isolation (C4) while proposing several authentication and authorization mechanisms (C5). A user management module has been presented to handle identities, permissions, and authorizations, allowing or denying operations to users or internal components. Cited works show that micro-services can efficiently secure and isolate future applications.

Architectural enhancements have been addressed in [53]. Xu, R., et al. have presented a BC-based decentralized architecture named BlendMAS for IoT-based public safety systems. Authentication and access control models have been transcoded into smart contracts deployed on private BC networks. All security mechanisms aim to improve system security and offer higher scalability (C2) and flexibility. The authors state that leveraging BC technologies establishes secure user relationships over computer networks (C5). A PoC of the BlendMAS architecture has been evaluated based on a surveillance use case where micro-services are deployed on distributed edge and fog nodes. A distributed network environment with a large number of heterogeneous (C8) devices (i.e. cameras, laptops, desktops, Raspberry Pis) has been considered. In [54], Debauche, O., et al. have presented an edge-based architecture to deploy AI algorithms and models for IoT. The architecture has been implemented in Kubernetes [67], a well-known container orchestration platform. Their architecture provides several benefits such as lower latency and higher scalability (C2) than traditional cloud infrastructures. It also supports several different types of micro-services, enabling ML features at the edge (C8). The authors propose studying cluster federation as future work. Micro-services promise to relieve the burden of costly service deployments from traditional VMs.

C.3.5 Hardware Acceleration

Hardware acceleration [68] has become a promising research field to mitigate performance degradation and latency introduced by network softwarization. Softwarized NFs have revolutionized network infrastructures by providing higher flexibility, higher portability, and reconfigurability. However, migrating conventional hardware functions towards softwarized VNFs is challenging. Recent works address current barriers to meet the flexibility and scalability demands of modern communication networks. In [55], Zhang, X., et al. have designed a hardware-based approach for NFV. It provides high scalability (C2) and programmability while supporting hardware-level parallelism and reconfiguration. The authors state that it is important to install each software component into a VM to isolate (C4) customized NFs. Their platform consists of heterogeneous (C8) middleboxes adopting both FPGAs and microprocessors to implement NFV operations, dynamically customizing specific network flow needs. Latency and throughput (C9) of both FPGA and VM module implementations have been evaluated.

Recently, hardware-accelerated platforms have focused on Neural Networks (NNs). In [56], Umuroglu, Y., et al. have presented Finn, a framework for building fast and flexible FPGA accelerators. The authors adopt binarized NNs to achieve higher performance in Tera Operations per second (TOPS) on FPGAs. Their evaluation demonstrates the performance and energy efficiency (C3) of binarized NNs for image classification. A heterogeneous (C8) streaming architecture has been designed, in which a custom architecture is built for a given topology rather than scheduling operations on top of a fixed architecture. Separate compute engines are dedicated to each layer, communicating via on-chip data streams. Results have shown benefits in classification throughput (C9), FPGA resource usage and power consumption. In [57], Cai, R., et al. have proposed VIBNN, an FPGA-based accelerator for bayesian NNs. A deep pipelined architecture achieving high scalability (C2) and efficient memory access has been designed. Results have shown that VIBNN can reduce energy consumption (C3) while attaining high throughput (C9) levels. Lastly, Owaida, M., et al. [58] have explored an FPGA-based accelerator to improve the overall performance of data processing pipelines. The authors focus on decision tree ensemble methods, a common approach to score and classify search systems. Results have proved the high scalability (C2) and throughput (C9) of their approach. Hardware impacts the performance of softwarized NFs. All cited works show adequate enhancements to support future VNFs.

C.3.6 Summary

Section 4.5 introduces novel research on architectural paradigms enabling the deployment of service chains on a continuum of virtual resources. ETSI MEC has been designing a reference architecture for future mobile networks, creating a cloud environment close to the RAN while FC is placing resources at the edge to meet the strict requirements of IoT. Micro-services are transforming service deployments from a traditional monolith to loosely-coupled containers, while hardware-based accelerators are pushing the boundaries of hardware platforms to support softwarized NFs. The literature review has shown differences in the evaluation criteria. MEC and FC mainly discuss *Mobility* (C1), while FC, micro-service and hardware acceleration focus on *Scalability* (C2). Mobility support and service migration are open challenges in MEC and FC. Without efficient migration strategies, low latency service delivery cannot be achieved. *Energy efficiency* (C3) and *Isolation* (C4) are unexplored in MEC and FC, while *Security* (C5) is the main focus of micro-service research. Micro-services establish security guarantees while offering flexible service deployments. *Resilience* (C6) is unexplored in all domains, while FC and MEC address *Reliability* (C7). All domains address *Heterogeneity* (C8), while *Throughput* (C9) is more noticeable in FC and hardware acceleration. Hardware-based platforms improve the performance of softwarized NFs, supporting high throughput levels while attaining low latency. *Federation* (C10) concepts are unexplored, though authors acknowledge their importance.

C.4 Recent advances on communication networks for low-latency services

C.4.1 Overview

Novel networking paradigms have opened several possibilities for improving network performance, including higher flexibility and scalability. Recent trends bring software-based automation to current networks: NS, SFC, IBN and SR. Table C.5 presents a summary of the reviewed works.

C.4.2 Network Slicing (NS)

NS [91] implements independent E2E logical networks on top of physical infrastructures. A slice is a virtual network implemented on top of physical nodes, creating the illusion of operating its dedicated physical network. NS allows high flexibility, improved resource allocation and increased service isolation by physically separating network resources. NS has gained significant attention with the advent of 5G. In [69], Campolo, C., et al. have designed 5G network slices for V2X services, addressing mobility (C1) management for V2X slices. The authors

Table C.5: Summary of the reviewed works in terms of Network.

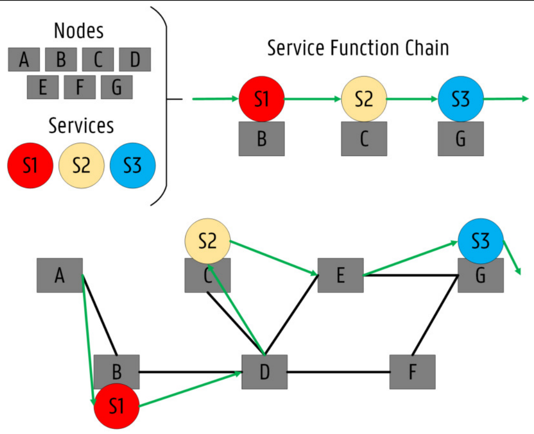
Research Domain	Authors	Main focus	Year	Evaluation Criteria									
				C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Network Slicing (NS) (Sec. C.4.2)	Campolo, C., et al. [69]	Vehicle Networks	2017	✓	✗	✗	✓	✗	✗	✓	✓	✓	✗
	Ksentini, A., et al. [70]	Slicing for RAN	2017	✗	✗	✗	✓	✗	✗	✓	✓	✓	✗
	Taleb, T., et al. [71]	5G use cases	2017	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗
	Popovski, P., et al. [72]	5G use cases	2018	✗	✗	✗	✓	✗	✗	✓	✓	✓	✗
Service Function Chaining (SFC) (Sec. C.4.3)	Qu, L., et al. [73]	Reliability	2017	✗	✓	✗	✗	✗	✗	✓	✗	✓	✗
	Zhang, L., et al. [74]	Network Coding	2017	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗
	Trajkovska, I., et al. [75]	SDN-based SFC	2017	✗	✓	✗	✓	✗	✗	✗	✓	✓	✗
	Jang, I., et al. [76]	SFC placement	2017	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗
	Bhamare, D., et al. [77]	SFC placement	2017	✗	✗	✓	✗	✗	✗	✗	✗	✓	✓
	Hawilo, H., et al. [78]	SFC placement	2019	✗	✓	✗	✗	✗	✗	✓	✓	✗	✗
	Xiang, Z., et al. [79]	Tactile Internet	2019	✗	✓	✗	✓	✓	✗	✗	✓	✓	✗
Intent based Networking (IBN) (Sec. C.4.4)	Cerroni, W., et al. [80]	IBN orchestration	2017	✗	✓	✗	✗	✗	✗	✓	✓	✗	✓
	Arezoumand, S., et al. [81]	SDN-based IBN	2017	✗	✓	✗	✓	✗	✗	✗	✓	✗	✓
	Szyrkowiec, T., et al. [82]	SDN-based IBN	2018	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗
	Abbas, K., et al. [83]	Slicing for IBN	2020	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗
	Wang, Y., et al. [84]	Privacy	2020	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗
Segment Routing (SR) (Sec. C.4.5)	Pang, J., et al. [85]	SDN-based SR	2017	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗
	Giorgetti, A., et al. [86]	Multi-domain SR	2017	✗	✓	✗	✗	✗	✗	✗	✓	✗	✓
	Cianfrani, A., et al. [87]	SR performance	2017	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗
	Desmouceaux, Y., et al. [88]	Load balancing in SR	2018	✗	✓	✗	✗	✗	✓	✓	✗	✓	✗
	Chunduri, U., et al. [89]	SR scalability	2018	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
	Aubry, F., et al. [90]	SR performance	2018	✗	✓	✗	✗	✗	✓	✓	✗	✗	✗

state that mobility prediction models help to optimize caching strategies for vehicles. Isolation (C4) strategies have been discussed, such as intra-slice and inter-slice V2X isolation. The authors remark that life cycle management configuration, adaptation, and monitoring are essential to meet slice isolation constraints and QoS levels. Retransmissions are handled locally by each vehicle in autonomous driving slices to match high-reliability and ultra-low-latency requirements (C7). Heterogeneity (C8) has been satisfied through different requirements (e.g. latency, bandwidth) for different slice types (e.g. URLLC, xMBB). Throughput (C9) has been discussed for distinct slices. The authors state that large Transmission Time Intervals (TTIs) (e.g. 1 ms) should be used for throughput-demanding applications, while short TTIs (e.g. 0.125 ms) can be used for fast retransmissions in teleoperated driving slices. In [70], Ksentini, A., et al. have proposed a framework to enforce NS in RAN. It focuses on separating network traffic towards the appropriate core and uses a two-level scheduler to adapt the resource allocation policy according to the slice's needs. Their architecture shares the usage of logical channels and their mapping to Evolved Packet System (EPS) bearers with legacy LTE. The main difference lies in abstracting physical resource blocks through a slice resource manager responsible for allocating resources for each User Equipment (UE) belonging to its slice (C4). The authors state two important factors for resource allocation in URLLC slices: latency and reliability (C7). To maximize the latter, the authors suggest adapting the modulation and coding scheme used by UEs to improve robustness to channel errors. Thus, robust modulations should be favored over high data rate modulations. Heterogeneity (C8) has also been studied through different slice requirements (e.g. latency, bandwidth, reliability). Throughput (C9) has also been evaluated. Results show that Extreme Mobile Broadband (xMBB) slices achieve the highest throughput while URLLC slices achieve the least levels since this slice focuses on maximizing reliability. The authors propose to study the scalability of the two-level scheduler as future work.

5G use cases have been customized based on NS. In [71], Taleb, T., et al. have personalized mobile networks at different granularity levels (e.g. application, network, group of users). An architecture named PERMIT has been presented, considering user mobility (C1), usage behavioral patterns, and underlying dynamics of the infrastructure for service customization. The authors state that users are aggregated in the same slice when sharing a service or behavior due to scalability (C2) reasons, and to isolate (C4) usage profiles and avoid security (C5) breaches. In [72], Popovski, P., et al. have studied non-orthogonal sharing of RAN resources in uplink communications from a set of enhanced Mobile Broadband (eMBB), massive Machine-Type Communication (mMTC), and URLLC devices to a common BS. NS has been investigated for RAN access (C4). A communication-theoretic model has been presented, considering heterogeneous requirements for the three services (C8). The reliability diversity concept has also been introduced as a design principle that leverages reliability requirements across all services, ensuring the performance of non-orthogonal RAN slicing (C7). Reliability and throughput (C9) levels have been evaluated. 5G has encouraged academia and industry to implement NS for future use cases, allowing service personalization and isolation.

C.4.3 Service Function Chaining (SFC)

Figure C.6 An example of a service chain deployment [92].



SFC [93, 94] has been studied in network management over the last few years. A service chain concerns proper service ordering. Fig. C.6 shows how each user has to traverse the service chain to access a network service. The circles represent different services while the arrows show how traffic is steered in the network. User requests are routed through the service chain following a service graph, which aims to improve resource allocation and application performance. SFC is a flexible and reliable alternative to dynamically reconfigure softwarized services without replacing hardware. SFC has been recently applied to reduce latency in softwarized VNFs. In [73], Qu, L., et al. have proposed a reliability-aware provisioning approach with delay guarantees for NFV-enabled Data Center (DC) networks. A Mixed-Integer Linear Programming (MILP) formulation optimizes VNF placement and traffic routing focused on maximizing reliability and reducing E2E delays. A heuristic has also been introduced to overcome the MILP complexity and consequent high execution time (C2). Several constraints satisfy reliability (C7) guarantees considered by the authors. Throughput (C9) has been evaluated for both methods. Results have shown that their heuristic outperforms existing schemes in average E2E delays and reliability at the expense of additional bandwidth and resource usage. In [74], Zhang, L., et al. have designed and implemented network coding as an NF in VMs for geo-distributed cloud DCs, proposing efficient algorithms for deploying and scaling network coding functions. The authors aim to improve network reliability (C7). Results have shown increased throughput (C9) and higher robustness of multicast sessions. Also, SFC concepts have been adopted in SDN-based approaches as in [75]. Trajkovska, I., et al. have proposed an SDN-based SFC mechanism with performance and scalability (C2) in mind, designed to ensure tenant isolation (C4). A prototype has been evaluated in a real DC, where the impact of heterogeneous (C8) environments has been assessed. Throughput (C9) levels have been evaluated for distinct chain scenarios based on a video traffic use case. Low latency SFC is essential for future applications. Cited works have developed adequate mechanisms to ensure SFC performance and scalability.

Another challenge is SFC allocation and placement. In [76], Jang, I., et al. have studied optimal SFC allocation and flow routing. A multi-objective MILP model has been proposed to maximize the acceptable flow rate and to minimize energy costs for multiple service chains. A polynomial-time algorithm based on linear relaxation has also been presented to approximate the optimal solution provided by the MILP model. The energy cost minimization has been modeled as one of the objectives (C3). Results have shown that their polynomial algorithm obtains near-optimal performance and increases the acceptable flow rate (i.e. throughput (C9)) and service capacity compared to other algorithms. In [77], Bhamare, D., et al. have presented an ILP model for SFC allocation in a multi-cloud scenario. An Affinity-based approach (ABA) has been proposed for large networks. Deployment costs have been considered in the formulation, leading to energy savings (C3). Their heuristic has been compared to greedy algorithms. Results have shown that the ABA algorithm outperforms greedy heuristics in total delays and total resource cost. The evaluation assesses traffic loads (C9) and considers multi-cloud environments (C10). In [78], Hawilo, H., et al. have proposed a MILP model and a heuristic algorithm for VNF placement. The authors study the carrier-grade nature of NFV applications and the minimization of E2E delays in service chains. Their evaluation considers scalability (C2) requirements. The authors state that if delay constraints are violated, the scalability and the traffic offloading capacity of the service chain are affected. The approach enhances the reliability (C7) and the QoS of the service chain by maximizing the number of participating members in a functional group of a VNF instance. Heterogeneity (C8) has also been satisfied through heterogeneous VNF structures and distinct placement requirements (e.g. scalability, E2E delay). SFC has also been studied for latency reduction in future use cases, such as Tactile Internet in [79]. Xiang, Z., et al. have investigated the feasibility of NFV concepts to offer low latency for Tactile Internet. The authors have designed a management framework for distributed SFC in MEC, implementing multiple VNFs in parallel to evaluate its scalability (C2) and flexibility. Their framework considers a virtualized networking overlay on top of the physical infrastructure, where multiple VMs are connected to and isolated from each other (C4). The authors adopt encryption methods to ensure data confidentiality and integrity (C5). Heterogeneity (C8) has been addressed through different networking components (e.g. physical networks, virtual overlays). The trade-off between per-packet latency and throughput (C9) has been evaluated. Their approach reduces packet throughput to achieve shorter per-packet latency, required for typical Tactile Internet applications with a 1 ms round-trip delay budget. Efficient SFC placement and routing is currently the main challenge. Solving these issues will lead to flexible service deployments supporting low latency services throughout their execution.

C.4.4 Intent-based Networking (IBN)

IBN [95] has been recently proposed to communicate intents to the network. Intents are policies written in high-level operational or business objectives that a system should meet. The main idea behind IBN is to communicate to the system how it should behave without detailing how it could achieve the objective. The intent is enforced in the infrastructure by the system, free from human intervention. An Internet Engineering Task Force (IETF) working group has been defining concepts, specifications and functionalities of IBN [96]. IBN aims towards autonomous networks, simple to manage with minimal human intervention powered by ML and AI. IBN concepts have been recently incorporated into management and orchestration practices, including SDN-based platforms. In [80], Cerroni, W., et al. have proposed a reference architecture and an intent-based Northbound Interface (NBI) for E2E service orchestration across multiple domains. The scalability (C2) of the NBI response time has been assessed at the Virtual Infrastructure Manager (VIM) implemented in ONOS [97], an open-source SDN controller. Two QoS features have been evaluated: low latency and high reliability (C7). The approach has been validated in a heterogeneous (C8) multi-domain (C10) testbed (i.e. IoT, OpenFlow, and cloud) based on an IoT use case. In [81], Arezoumand, S., et al. have presented an intent framework named MD-IDN for multi-domain cloud infrastructures. Compilation algorithms have been proposed to achieve high scalability (C2) in multi-domain networks. The authors remark that isolation (C4) is crucial in multi-domain environments, thus, intent frameworks must avoid cross-contamination of intents requested by different tenants. Their algorithms have been assessed over heterogeneous (C8) and multi-domain (C10) networks. Distinct infrastructural nodes have been considered (e.g. VMs, FPGAs, and GPU servers). Results have shown that the MD-IDN framework outperforms current practices that compile intents over a flat network topology. In [82], Szyrkowiec, T., et al. have presented an architecture for automatic intent-based provisioning of security services through a multi-layer SDN Orchestrator. Their orchestrator defines a lightweight NBI, specifying application needs focused on security (C5) configurations. Encryption layer properties have been analyzed regarding latency, throughput (C9), flexibility, and protocol transparency. Cited works show that IBN revolutionizes orchestration practices, enabling highly automated networks.

IBN concepts have also been combined with NS in [83]. Abbas, K., et al. have designed an IBN slicing system to manage core and RAN resources. Users can set intents and their system configures the network accordingly. A DL model for resource management has also been presented. NS has been applied together with IBN to efficiently handle RAN and core resources in 5G networks (C4), considering different slices (i.e. eMBB, IoT, and URLLC) (C8). The uplink and downlink throughput (C9) achieved by the three slices have been assessed. Lastly, privacy concerns have been addressed in [84]. Wang, Y., et al. have presented an intent prediction-based approach named LocJury to preserve location privacy in Internet-of-Vehicles (IoV). Their method estimates the intent of location access and restricts malicious attempts. Mobility (C1) plays a major role in preserving location information in IoV. LocJury applies ML and IBN to learn the motivation behind location accesses and restrain suspected malicious attempts. Results have shown that LocJury preserves vehicles' location privacy (C5).

C.4.5 Segment Routing (SR)

SR [98] leverages the source routing paradigm. A node steers a packet through an ordered list of instructions, named segments. A segment can be composed of any type of instruction (e.g. topological or service-based information), which is then placed as path state information into a packet header at an ingress node. Several segments create unconstrained network paths represented by segment lists. Information flows from packet headers, making nodes stateless and significantly reducing forwarding tables complexity, simplifying traffic engineering and management across network domains. SR is highly responsive to network changes, making networks more agile and flexible. SR has already been explored in SDN and multi-domain scenarios. In [85], Pang, J., et al. have presented a collaboration method of multipath TCP (MPTCP) and SR to address resource consumption in an SDN-based DC Network (DCN). The authors combine MPTCP and SR to reduce forwarding rules required in SDN-based MPTCP solutions. The authors prove that their approach cannot meet the increasing transmission demand in highly scalable scenarios (C2) in a single-controller mode due to the packet header size limitation in SR. In contrast, it can significantly reduce storage, minimizing energy consumption (C3). Simulations have shown that high throughput (C9) is achieved while reducing flow completion time and improving link resource usage in SDN-based DCNs. The authors propose to investigate multi-controller environments as future work. In [86], Giorgetti, A., et al. have focused on two relevant SR use cases:

dynamic traffic recovery and traffic engineering in multi-domain networks. The scalability (C2) of the segment list depth has been assessed. The experiments consider a multi-domain (C10) heterogeneous (C8) testbed exploring an SDN-based implementation of SR.

Performance and scalability are major concerns in SR. In [87], Cianfrani, A., et al. have faced the challenge of transitioning from a pure IP network to a full SR system while optimizing network performance. The authors have proposed an architecture named SR Domain (SRD) to ease the coexistence between IP routers and SR nodes. A MILP formulation has been introduced for the SRD design problem by reducing congestion. Their performance study has shown that the SRD approach can reduce the maximum link usage and increase system stability. Consequently, higher scalability (C2) is achieved. Heterogeneity (C8) has also been studied in hardware routers (IP routers and SR nodes). In [88], Desmouceaux, Y., et al. have presented the concept of 6LB, a load balancer running exclusively within the IP forwarding plane. It applies IPv6 SR to direct data packets from a new flow through a chain of candidate servers. The authors propose a consistent hashing algorithm and an in-band stickiness protocol to reliably distribute 6LB across several instances for scalability purposes (C2). Simulations have assessed the consistent hashing algorithm resiliency (C6). By adopting the power of two choices [99], 6LB significantly increases reliability (C7) compared to single-choice approaches. Packet-forwarding performance assessment shows that higher fairness comes at a negligible cost of CPU overhead. Throughput (C9) levels have been compared to single choice load-balancing approaches. In [89], Chunduri, U., et al. have designed the Preferred Path Routing (PPR) concept to overcome SR limitations. PPR minimizes the data plane overhead (e.g. packet processing) by extending it for IP data planes without replacing existing hardware or even upgrading the data plane. PPR allows dynamic path QoS reservations by providing deterministic queuing latency. Scalability concerns (C2) have been discussed since PPR requires a separate PPR-ID for every possible path. The authors state that they do not expect deployment issues in a practical setting since the total number of preferred paths can be easily supported by network devices. The authors also propose studying fast rerouting and path resiliency schemes as future work. In [90], Aubry, F., et al. have introduced Robustly Disjoint Paths (RDPs): pairs of paths that remain disjoint even after an input set of failures, with no external intervention. The authors have designed efficient algorithms to compute SR-based RDPs. Evaluations on real network topologies have shown that RDPs achieve high scalability (C2) and reliability (C7) in large ISP networks. Fault tolerance (C6) has been assessed through single and multiple link failure experiments. Novel routing paradigms aim to improve the reliability and resilience of current networks. Cited works prove that SR is suitable for addressing multi-domain challenges in future networks.

C.4.6 Summary

Section C.4 presents novel research paradigms on communication networks. NS enables higher levels of flexibility and isolation, while SFC optimizes resource allocation through proper service ordering. IBN communicates intents to the network, thus, enforcing rules without detailing how the system should obtain them. SR revolutionizes the routing paradigm since it leverages the source routing paradigm to move information in the network in packet headers, making nodes stateless. The literature review shows that NS discusses *Mobility* (C1), while SFC focuses on *Scalability* (C2) and *Energy efficiency* (C3). NS also deals with *Isolation* (C4). Both NS and SFC are important for low latency service delivery. On the one hand, NS allows the setup of different QoS levels for distinct applications. On the other hand, SFC brings high degrees of flexibility and reconfiguration, optimizing service placement and traffic flow. IBN addresses *Security* (C5) concerns, while SR studies *Resilience* (C6). SR and IBN enable network automation features that will ensure proper management and orchestration of multi-domain environments. Both concepts enable high automation but also attain high performance and scalability, fully supporting low latency services. *Reliability* (C7) has been explored in NS, SFC and SR, while *Heterogeneity* (C8) has been discussed in NS, SFC and IBN. SFC also focuses on *Throughput* (C9). *Federation* (C10) is still unexplored in these research domains, but authors acknowledge its importance in multi-domain environments.

C.5 Towards efficient orchestration in cloud-native infrastructures

C.5.1 Overview

Cloud-native infrastructures have imposed strict specifications on management functionalities. This section presents trends on orchestration: resource allocation, auto-scaling, performance monitoring and caching. Table C.6 summarizes the reviewed works.

C.5.2 Resource Allocation

Resource allocation, also known as resource provisioning, has been studied for years in the network management domain [123, 124]. It concerns the provisioning of computing, network and storage resources required to instantiate services requested by users and devices over the Internet. Recently, cloud providers and users have been working together towards an efficient allocation of computing resources. Users expect the best QoS at the cheapest rate while cloud providers aim to increase their revenue and respect the agreed QoS level. With the advent of

Table C.6: Summary of the reviewed works in terms of Orchestration.

Research Domain	Authors	Main focus	Year	Evaluation Criteria									
				C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Resource Allocation (Sec. C.5.2)	Liang, L., et al. [100]	D2D comm.	2017	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗
	Arkian, R., et al. [101]	IoT applications	2017	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗
	Santos, J, et al. [102]	IoT applications	2017	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗
	Yao, J., et al. [103]	IoT applications	2018	✓	✗	✓	✗	✗	✗	✗	✓	✗	✗
	Podili, P., et al. [104]	QoS in SDN	2018	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗
	Zhang, H., et al. [105]	Energy eff.	2018	✗	✗	✓	✗	✗	✗	✗	✓	✓	✗
	Zhou, Z., et al. [106]	M2M comm.	2019	✗	✗	✓	✗	✗	✗	✓	✗	✓	✗
	Farhad, A., et al. [107]	Mobility Mgmt.	2020	✓	✗	✓	✗	✗	✗	✓	✓	✗	✗
Auto scaling (Sec. C.5.3)	Aslanp., S., et al. [108]	Web applications	2017	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗
	Rahman, S., et al. [109]	VNFs	2018	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗
	Lee, D., et al. [110]	SFC	2020	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗
	Lin, T., et al. [111]	QoS	2020	✗	✓	✗	✗	✗	✗	✗	✓	✗	✓
Performance Monitoring (Sec. C.5.4)	Moradi, F., et al. [112]	Containers	2017	✗	✓	✗	✓	✗	✗	✗	✗	✓	✗
	Tangari, G., et al. [113]	Decentralized SDN	2017	✗	✓	✗	✗	✗	✗	✗	✓	✓	✗
	Shah, S. Y., et al. [114]	Cloud applications	2017	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗
	Perdices, D., et al. [115]	Net. Performance	2018	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗
	Sanz, I. J., et al. [116]	SFC performance	2018	✗	✗	✗	✓	✓	✗	✗	✗	✓	✗
Caching (Sec. C.5.5)	Chen, M., et al. [117]	5G Mobility-aware	2017	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗
	Tant., K., et al. [118]	Mobility Mgmt.	2017	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗
	Zhang, K., et al. [119]	5G MEC	2018	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗
	Hao, Y., et al. [120]	MEC Energy eff.	2018	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗
	Xiao, L., et al. [121]	Security	2018	✗	✗	✓	✗	✓	✗	✗	✓	✗	✗
	Cheng, F., et al. [122]	Security	2019	✓	✗	✓	✗	✓	✗	✗	✗	✓	✗

IoT and low latency services, resource allocation has become even more important. Delay-sensitive services (e.g. connected vehicles, XR) require latency in the order of milliseconds that centralized infrastructures cannot support, thus, requiring efficient allocation in a continuum of virtual resources. Allocation strategies for vehicular networks and IoT contexts have been studied. In [100], Liang, L., et al. have investigated spectrum sharing and power allocation for D2D vehicular networks. The authors state that fast channel variations caused by high mobility (C1) in a vehicular environment need to be considered in allocation scheme design. Also, different QoS requirements for Vehicle-to-Infrastructure (V2I) and Vehicle-to-Vehicle (V2V) links have been studied. The authors state that high link capacity is desired for V2I connections, while safety-critical information of V2V connections places greater emphasis on link reliability. Their scheme includes reliability (C7) guarantees for D2D users. The heterogeneous (C8) performance of V2I and V2V links for resource allocation purposes has been studied. The maximization of the overall V2I link throughput (C9) has been included as an optimization objective. In [101], Arkian, H. R., et al. have presented a fog-based scheme named MIST for cost-efficient resource allocation of crowdsensing applications in IoT. Firstly, the authors propose a Mixed-Integer Nonlinear Programming (MINLP) model. To tackle its inherent high computational complexity, the MINLP model has then been linearized into a MILP formulation. The authors have studied the joint optimization of data consumer association, task distribution, and VM placement issues towards minimizing the overall cost (C3) while satisfying QoS levels. Heterogeneity (C8) has been satisfied through distinct fog nodes (e.g. routers, access points) and different wireless connectivity (e.g. 4G and Wi-Fi). In [102], Santos, J, et al. have presented an ILP formulation for IoT service placement. The model considers multiple optimization objectives, such as low latency and energy efficiency (C3). Smart City use cases with different placement requirements and a fog-cloud infrastructure with distinct hardware capabilities have been considered (C8). In [103], Yao, J., et al. have addressed the joint optimization of resource allocation and power control in FC to minimize the overall system cost while satisfying QoS requirements. The authors have formulated the problem as an MINLP model and then presented an approximation algorithm to solve it. The authors discuss mobility (C1) management in IoT when maintaining QoS levels. To manage device mobility in fog-aided networks, the authors state that its transmission power can be adapted and that handovers between different IoT gateways and VM migrations should be performed to meet QoS requirements. Simulations have evaluated power control and system cost (C3), based on multiple applications with different QoS requirements (C8).

Resource allocation has also been investigated in SDN and access control in mobile networks. Podili, P., et al. [104] have introduced a resource provisioning approach for virtual networks in SDN focused on E2E delay and bandwidth. Results have shown improvements in availability, scalability (C2), and cost-effectiveness (C3). The authors propose a destination label forwarding mechanism to reduce the number of flow rules in SDN switches: a unique set of labels is assigned to each virtual network, thus, ensuring traffic isolation (C4). In [105], Zhang, H., et al. have studied the problem of energy-efficient user scheduling and power optimization in Non-Orthogonal Multiple Access (NOMA) networks. The trade-off between data rate performance and energy consumption (C3) has been assessed for wireless downlink communications in heterogeneous (C8) NOMA networks. Results have demonstrated improved energy consumption and user throughput (C9). In [106], Zhou, Z., et al. have proposed a two-stage access control and resource allocation algorithm for Machine-to-machine (M2M) communications in industrial automation. Firstly, a contract-based mechanism motivates delay-tolerant devices to postpone their access voluntarily. Then, a long-term cross-layer online resource allocation model jointly optimizes rate control, power allocation, and channel selection without prior knowledge of channel states. Results have shown improvements in sensing rate, queue stability, backlog fluctuation and energy efficiency (C3). The reliability of M2M communications has also been studied in device battery life as a long-term average power consumption constraint in their model (C7). Their scheme has been compared with the snapshot-based throughput optimal algorithm (as baseline), which maximizes physical-layer throughput (C9) without considering long-term constraints and sensing rate control. Lastly, mobility management has been studied in [107]. Farhad, A., et al. have presented a mobility-aware allocation scheme for IoT devices (C1) that enhances the Packet Success Ratio (PSR) by reducing the impact of interference, retransmissions, and packet loss compared with the LoRaWAN-based Adaptive Data Rate (ADR). Energy consumption (C3), PSR and reliability (C7) have been evaluated. Their scheme considers traffic heterogeneity (i.e. static and mobile end devices) based on the gateway sensitivity during the initial deployment phase (C8). Simulations demonstrate the feasibility of their scheme for IoT mobile applications needing high PSR and reliability without high energy consumption. Cited works tackle resource allocation starting from different points of view: several use cases and multiple requirements.

C.5.3 Auto-scaling

Distributed clouds have revolutionized resource management. On the one hand, if services need more resources (i.e. under-provisioning), then they should be added on-demand so that services keep operating. On the other hand, resources should be released when they are not fully used (i.e. over-provisioning). Service over-provisioning wastes resources and increases costs, while under-provisioning schemes degrade performance and violate Service Level Agreements (SLAs). Thus, automatic mechanisms should scale up and down resources according to the network demand without human intervention. This is named Auto-scaling [125], where resources are dynamically added or removed to meet QoS requirements. Designing efficient auto-scaling systems is not a trivial task due to limited hardware resources, dynamic workloads, diverse service requirements and complex infrastructures.

Auto-scaling for web applications has been studied in [108]. Aslanpour, M. S., et al. have proposed a cost-aware auto-scaling mechanism (C2). Their approach focuses on executing scale-down commands via the selection of surplus VMs, which are then quarantined for the rest of their billing period to maximize cost efficiency (C3). The auto-scaling mechanism throughout has been assessed (C9). Results have shown reduced costs with improved response time and decreased SLA violations. In [109], Rahman, S., et al. have proposed an ML-based approach for VNF auto-scaling focused on dynamic traffic changes. The authors have presented an ML classifier that learns proactive measures from both past scaling decisions and temporal traffic behavioral patterns (C2). Results have demonstrated that the ML classifier improves the overall QoS and reduces costs (C3). SFC concepts and QoS metrics have been recently investigated in terms of auto-scaling. Lee, D., et al. [110] have proposed an auto-scaling method using RL for scale-in/out of multi-tier VNF instances (C2). The authors have defined the observation space based on SFC compositions while Service Level Objectives (SLOs) have been applied to design the reward function. Throughput (C9) and response time have been considered as SLOs. Results have shown an optimal number of VNF instances while minimizing SLO violation. In [111], Lin, T., et al. have studied the inclusion of network metrics into auto-scaling and scheduling mechanisms of cloud management systems. The authors have proposed an architecture that monitors the QoS (i.e. latency and bandwidth) of their clients. If QoS levels are violated, the auto-scaling system is activated, and new service instances are strategically deployed to meet QoS levels (C2). The authors have implemented a multi-tier and multi-tenant testbed, incorporating heterogeneous (C8) devices (e.g. sensors, edge devices, DC locations) that span across geographically spread regions (C10). Efficient auto-scaling provides higher autonomy as shown by the adoption of ML-based methods.

C.5.4 Performance Monitoring

Performance monitoring relates to the application's behavioral analysis. Monitoring or tracking applications allows service providers to fine-tune the application runtime performance while reducing allocation costs. This subsection surveys trends in performance monitoring research, discussing existing methods and tools that help to control the application execution and measure the impact of different infrastructures on their performance.

Containers, VMs and SDN applications have been a major topic in performance monitoring. In [112], Moradi, F., et al. have proposed an automated system for monitoring network performance of container-based applications named ConMon. It identifies newly instantiated containers and passively observes their traffic. Based on these observations, it configures and executes monitoring functions inside adjacent containers. Consequently, monitoring is isolated from the application and does not require instrumenting the image of the container or running additional processes inside it (C4). The feasibility and scalability (C2) of ConMon have been validated based on container performance and system resources. The authors have also assessed the impact of passive monitoring on the application's throughput (C9). In [113], Tangari, G., et al. have presented a decentralized approach for resource monitoring in SDN. Their proposal supports a wide range of measurement tasks and requirements regarding monitoring rates and information granularity levels. The authors have compared their decentralized system to a centralized approach based on a realistic use case, where a distributed management application coordinates a content distribution service in an ISP network. The trade-off between application reactivity/accuracy and monitoring scalability/overhead (C2) has been studied. The evaluation has considered requirements stemming from heterogeneous (C8) management applications. The average flow throughput (C9) has also been investigated in their monitoring scheme. In [114], Shah, S. Y., et al. have studied runtime dependencies among micro-services to detect anomalous behavior and meet SLAs. The authors have proposed a novel method based on Long-Short Term Memory (LSTM) recurrent NNs to find those dependencies in heterogeneous (C8) environments (i.e. public/private clouds). The authors focus on finding the strongest performance predictors, discovering temporal dependencies, and improving the accuracy of forecasting for a given performance metric. Results have proved its feasibility compared to existing literature methods, such as Granger causality and classical statistical time series models. Throughput (C9) levels have been assessed. In [115], Perdices, D., et al. have presented dPRISMA (distributed Passive Retrieval of Information, and Statistical Multipoint Analysis), a passive monitoring system generating statistical models for network measurements and raising alarms in case of anomalous behaviors. dPRISMA implements a distributed data gathering strategy, a promising approach to improve the scalability (C2) of monitoring systems. dPRISMA has been validated by correlating measurements retrieved from heterogeneous (C8) data sources (e.g. virtual environments, real-world data sets). SFC performance has also been analyzed in [116]. Sanz, I. J., et al. have developed a framework for the performance evaluation of SFC named SFCPerf. Their framework provides flexibility for experimenting with different NFs and virtualization infrastructures. Isolation (C4) has been addressed in the context of SFC since micro-service isolation has been applied through packet encapsulation. The authors adopt the Network Service Header (NSH) concept to perform packet forwarding in the service chain and isolate micro-services. To demonstrate the feasibility of SFCPerf, the authors consider a service chain constituted by virtual security (C5) functions. A service chain composed of an Intrusion Detection System (IDS) and a reactive firewall has been assessed concerning latency, throughput (C9) and the number of replied HTTP requests. Cited works show why monitoring tools are crucial for the orchestration life cycle. Without proper monitoring, performance decreases since adequate procedures are not employed.

C.5.5 Caching

With the advent of modern wireless technologies and higher demand for multimedia services, new challenges have emerged for the support of multimedia content in next-generation networks. Caching at the edge is a promising approach to alleviate the burden on backhaul infrastructures, reduce data transmissions and minimize startup latency for multimedia delivery. The aim is to control data traffic and keep popular content at the edge close to end-users. Caching research has addressed mobility management in 5G contexts. Chen, M., et al. [117] have studied caching placement on small cell BSs and mobile devices by leveraging user mobility (C1), aiming to maximize the cache hit ratio. The authors have focused on delivering content while reducing energy consumption (C3). Results have shown that their approach improves cache hit ratio and energy efficiency compared to existing strategies. In [118], Tantayakul, K., et al. have studied a caching policy focused on mobility (C1) support in SDN networks. Two policies have been proposed: an on-off policy and an adaptive mechanism. Both methods have been analyzed for packet loss, channel occupancy, transmission time, throughput (C9) and bandwidth fairness. Both policies improve SDN mobility regarding packet loss. The authors state that the adaptive policy outperforms the on-off policy for latency-sensitive applications. The on-off policy provides long transmission times, though attaining enhanced channel occupancy, thus suitable for file transfer or media applications which download all contents before display. In [119], Zhang, K., et al. have proposed a cooperative edge caching architecture for 5G MEC. Their architecture focuses on mobility-aware hierarchical caching, where smart vehicles act as collaborative caching agents for sharing content operations with BSs (C1). Heterogeneous (C8) nodes (e.g. vehicles, mobile devices, BSs) have been considered as potential cache-enabled devices. In [120], Hao, Y., et al. have introduced the concept of task caching for MEC, referring to the caching of completed task applications and their related data in edge-cloud infrastructures. The authors have studied the joint optimization of task caching and offloading through a MILP formulation. The authors have also proposed a heuristic algorithm that reduces energy costs compared to other schemes based on their simulations (C3). Heterogeneity (C8) has been satisfied through different task demands, several data sizes, and distinct service requirements. Cited works are promising approaches to support mobility in edge caching schemes.

Security has also been recently discussed. In [121], Xiao, L., et al. have studied attack models in MEC by focusing on both mobile offloading and caching procedures. The authors propose RL-based security solutions for safe offloading to edge nodes against jamming attacks. The aim is to improve the offloading quality, such as the Signal-to-Noise-plus-Interference Ratio (SINR) and Bit Error Rate (BER) of the signals received by edge nodes against jamming and interference to reduce energy consumption (C3). Their scheme reduces energy consumption and delay in mobile offloading while increasing the SINR of the signals received by edge nodes compared to benchmark schemes. The authors have also presented lightweight authentication and secure caching to preserve data privacy (C5). The authors state that security and data privacy are bottlenecks of MEC due to its heterogeneity (C8) in terms of devices (e.g. mobile, edge) and networks (e.g. physical, virtual). In [122], Cheng, F., et al. have proposed a novel caching scheme securing UAV-relayed wireless networks via jointly optimizing the UAV trajectory and time-scheduling. The authors suggest using UAVs as relay nodes for long-distance communications since they can be deployed on-demand because of their high mobility (C1) and agility. The authors state that energy consumption (C3), access delay, and throughput (C9) are essential factors for proper UAV operation. The authors propose exploiting energy harvesting techniques, such as solar energy, to provide sufficient energy supply to UAVs for delay-tolerant applications. The aim is to maximize throughput in multi-UAV networks while reducing their energy consumption. Simulations have proved the feasibility and efficiency of their scheme, showing the improved security of UAV relaying assisted networks (C5). Improving security in caching schemes will lead to its further acceptance in future networks.

C.5.6 Summary

Section C.5 presents trends on orchestration and management for next-generation networks. Resource allocation enables proper resource provisioning, while auto-scaling mechanisms ensure deployed services handle the current demand. Performance monitoring methods study application behavior, while caching research focuses on improving content delivery and reducing data transmissions, especially for fog/edge infrastructures. The literature review shows that resource allocation and caching address *Mobility* (C1) support, while Auto-scaling focuses on *Scalability* (C2). Reducing costs and managing multiple heterogeneous devices are open challenges of provisioning practices. Efficient provisioning is essential to support low latency service delivery. *Energy efficiency* (C3) is more noticeable in resource allocation and caching, while performance monitoring studies *Isolation* (C4). *Se-*

curity (C5) and data privacy is a major concern in caching schemes. *Resilience* (C6) is unexplored in all domains, while resource allocation addresses *Reliability* (C7). *Heterogeneity* (C8) is studied in all domains, while auto-scaling focuses on *Throughput* (C9) experiments. Auto-scaling systems capable of dealing with dynamic demands is a major future research topic. *Federation* (C10) concepts are still unexplored. All domains will play their role in the life-cycle management of containerized applications in future networks.

C.6 Integrating Machine Learning (ML) and Artificial Intelligence (AI): towards autonomous networks

C.6.1 Overview

Over the past years, ML has become an interesting research field in the networking domain. ML methods have been adapted to traditional network problems. Due to the integration of ML / AI in network management, self-driving networks may emerge as a potential solution for inappropriate human intervention. This section reviews trends in ML/AI: DL, RL and FL. Table C.7 summarizes the reviewed research.

C.6.2 Deep Learning (DL)

DL [142] is a subset of ML employing artificial NNs to learn complex problems from large amounts of data. DL methods learn without human supervision and from both structured and unlabeled data. Recently, DL has been applied to several domains, such as object and speech recognition, language translation, and computer vision. DL has also been recently adapted to traffic load and congestion prediction as in [126]. Tang, F., et al. have presented a DL-based algorithm for future traffic load and congestion prediction in an SDN-IoT network. Another DL algorithm has been proposed for channel assignment to avoid congestion in SDN-IoT. The authors consider a heterogeneous (C8) SDN-IoT network, where several devices can hardly cooperate, making it difficult to predict the traffic load sent by all devices. Simulations have demonstrated that their proposal outperforms conventional channel assignment algorithms regarding delay and throughput (C9). DL has also been applied to SFC placement and routing. In [127], Pei, J., et al. have formulated the VNF selection and chaining problem as a Binary Integer Programming (BIP) model to minimize E2E delay. The authors have also presented a DL algorithm for the SFC routing problem. The evaluation has shown that DL obtains routing paths for SFC requests while achieving higher scalability (C2)

Table C.7: Summary of the reviewed works in terms of ML / AI.

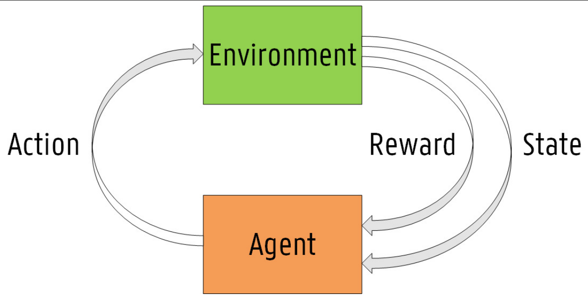
Research Domain	Authors	Main focus	Year	Evaluation Criteria									
				C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Deep Learning (DL) (Sec. C.6.2)	Tang, F., et al. [126]	Traffic load prediction	2018	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗
	Pei, J., et al. [127]	SFC allocation	2018	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
	Jiang, F., et al. [128]	Resource allocation for MEC	2019	✓	✗	✓	✗	✗	✗	✗	✓	✗	✗
	Siasi, N., et al. [129]	SFC allocation	2020	✗	✓	✓	✗	✗	✗	✗	✓	✗	✗
	Ali, Z., et al. [130]	Resource allocation for MEC	2020	✓	✓	✓	✗	✗	✗	✗	✓	✗	✗
Reinforcement Learning (RL) (Sec. C.6.3)	Chen, L., et al. [131]	Auto-scaling	2018	✗	✓	✗	✗	✗	✗	✗	✓	✓	✗
	Xu, Z., et al. [132]	Experience-driven networking	2018	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
	Li, J., et al. [133]	Resource allocation for MEC	2018	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
	Li, R., et al. [134]	Slicing allocation	2018	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗
	Ye, H., et al. [135]	V2V comm.	2019	✓	✗	✗	✗	✗	✗	✓	✓	✗	✗
	Faraci, G., et al. [136]	5G Slicing	2020	✓	✗	✓	✓	✗	✗	✗	✗	✗	✗
Federated Learning (FL) (Sec. C.6.4)	Tran, H., et al. [137]	Wireless Networks	2019	✗	✗	✓	✗	✓	✗	✗	✓	✗	✗
	Chen, M., et al. [138]	Wireless Networks	2020	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
	Qu, Y., et al. [139]	Privacy in FC	2020	✗	✓	✗	✗	✓	✗	✗	✗	✓	✗
	Zhou, C., et al. [140]	Privacy in FC	2020	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
	Kang, J., et al. [141]	Mobile Networks	2020	✗	✗	✗	✗	✓	✗	✓	✗	✗	✗

compared to existing approaches. In [128], Jiang, F., et al. have proposed DL-based algorithms for resource scheduling in hybrid MEC networks. The authors have presented a large-scale path-loss fuzzy c-means algorithm to predict the optimal positions of Ground Vehicles (GVs) and UAVs that help with offloading tasks (C1). Their goal is to minimize the energy consumption (C3) of UEs by jointly optimizing the positions of GVs and UAVs, user association, and resource allocation. Simulations have shown that their framework achieves similar performance to heuristic methods while reducing CPU execution time for heterogeneous (C8) MEC networks. In [129], Siasi, N., et al. have presented a DL algorithm for SFC allocation in FC. Their scheme predicts the popularity of VNFs, mapping the most popular ones to High-Capacity Fog (HCF) nodes while linking unpopular NFs to Low-Capacity Fog (LCF) nodes. The authors remark that their strategy provides significant advantages since cached VNFs are available on nodes close to terminals. Caching alleviates congestion and saves resources by enhancing network scalability (C2) and capacity without increasing network cost. The authors have evaluated several performance metrics, including usage rate, network saturation, energy consumption (C3) and cost, based on a heterogeneous (C8) architecture composed of HCF and LCF nodes with different resource capacities. Lastly, Ali, Z., et al. [130] have presented a resource allocation algorithm for MEC named Power Migration Expand (PowMigExpand). Their algorithm assigns user requests to optimal servers and allocates optimal amounts of resources to UEs. Their approach migrates UE requests to new servers when needed due to user mobility (C1). A DL algorithm for user request allocation has also been proposed, considering the varying load of incoming requests. The authors have demonstrated the scalability (C2) of the PowMigExpand algorithm through simulations evaluating service rate, utility, and energy consumption (C3). Increased performance for a different number of ME servers and varying traffic has been obtained. The heterogeneity (C8) of user requests has been considered. Cited works highlight the multiple applications of DL algorithms, presenting them as an alternative to solve several challenges in the network management domain.

C.6.3 Reinforcement Learning (RL)

In recent years, RL has become an important area in ML research [144]. Figure C.7 represents a typical scenario in RL. RL has been applied to sequential decision-making. An agent learns to make better decisions from interacting with an environment, which represents the problem to solve. In the beginning, the agent knows nothing about the problem at hand and learns by performing actions in an environment. For each action taken, the agent receives a reward and a new observation that describes the new state of the environment. Depending on the goal and how the agent is performing the given task, the reward can be positive or negative.

Figure C.7 The representation schema of most RL scenarios [143].



The agent learns to be successful by repeated interaction with the environment, by determining the inherent synergies between states, actions, and subsequent rewards. Ultimately, RL algorithms maximize the total reward an agent collects by experiencing multiple problem rounds. RL has been recently applied to Autoscaling in [131]. Chen, L., et al. have addressed traffic optimization in a DC by applying Deep RL (DRL) algorithms. The authors have proposed a two-level RL system named AuTO to solve scalability (C2) problems in DCs. Their experiments have assessed homogeneous and heterogeneous (C8) traffic and measured flow throughput (C9) levels.

RL research has also focused on resource allocation and traffic control. In [132], Xu, Z., et al. have investigated the application of DL for model-free control in communication networks, focusing on traffic engineering. Simulations have shown that DL significantly reduces E2E delay and offers similar throughput (C9) to baseline methods. In [133], Li, J., et al. have proposed a DRL approach for computation offloading and resource allocation for wireless MEC. The authors have formulated the total cost of delay and energy consumption (C3) for all UEs as the optimization objective. Results have shown a significant reduction in the total cost compared to other baselines. In [134], Li, R., et al. have studied resource management for NS based on DRL. Simulations have demonstrated the advantages of applying RL to radio resource slicing and priority-based slicing (C4). Heterogeneity (C8) has been addressed in distinct slices (e.g. video, URLLC). In [135], Ye, H., et al. have developed a decentralized resource allocation mechanism for V2V communications based on DRL. The high mobility (C1) of vehicles has been addressed in their scheme. A vehicle or a V2V link can decide which is the optimal sub-band and power level of transmission without requiring global information based on the decentralized nature of their approach. Latency and reliability (C7) requirements for V2V links have been discussed. Heterogeneous (C8) data (e.g. link interference, channel information) has been included in the observation space of the RL environment. The authors state that useful information can be extracted from heterogeneous data, simplifying optimal policy learning. Lastly, Faraci, G., et al. [136] have proposed the extension of 5G network slices with MEC UAVs. A system controller has been designed to handle job offloading to UAVs based on DRL. The authors aim to minimize power consumption (C3) and job loss while considering the mobility of UAVs (C1). An E2E logical network on top of physical infrastructures enables service isolation based on NS (C4). RL is still in the early stages but its applicability has already been proved, improving orchestration practices for low latency services.

C.6.4 Federated Learning (FL)

FL [145] or collaborative learning is a recent ML technique that trains a model across multiple decentralized edge devices or servers holding local data samples without exchanging them. The edge devices download the current model from the central server and update the model based on their local data. Then, these trained local models are sent back to the central server, where they are aggregated (i.e. averaging weights) into a single global model that is sent back to all devices. The main benefit of FL is supporting decentralized learning since training data is kept locally without being transferred to central locations. Nevertheless, challenges persist in FL regarding efficient communications since edge devices need to share small portions of their training execution with the central server without transferring the complete data set. FL has been recently applied to wireless networks. In [137], Tran, N. H., et al. have formulated FL over wireless networks as an optimization problem. The authors have studied how the computation and communication latency of UEs affect their energy consumption (C3) and both learning time and accuracy. The authors remark on the benefits of FL concerning data privacy since local data is not shared (C5). Numerical results have quantified the impact of UE heterogeneity (C8) on the system cost. In [138], Chen, M., et al. have also applied FL over wireless networks. In their scheme, wireless users train their local model using their data samples and transmit the trained model to a BS. The authors have studied the joint problem of wireless resource allocation and user selection for running FL algorithms. Their formulation aims to minimize the training loss while meeting delay and energy consumption (C3) requirements.

Privacy concerns have also been recently addressed. In [139]. Qu, Y., et al. have proposed a BC-enabled FL scheme for privacy preservation in FC. FL enables decentralized learning while BC allows end devices to exchange model updates based on a consensus mechanism without any centralized authority. Their FL approach achieves learning convergence when data sizes are moderate, proving its scalability (C2). Their scheme has been assessed regarding privacy protection, efficiency, and resistance to poisoning attacks (C5). In the evaluation, the authors consider a throughput (C9) model based on the Shannon capacity with a certain loss. In [140], Zhou, C., et al. have proposed a privacy-preserving FL scheme for

FC. In their approach, fog nodes collect data from IoT devices to perform learning tasks, improving training efficiency and accuracy. Differential privacy mechanisms and security aggregation methods have been considered to protect device data and protect devices from collusion attacks (C5). Mobile networks have also been discussed in [141]. Kang, J., et al. have presented the concept of reputation as a metric to discover trustworthy nodes in FL. The authors have leveraged consortium BC for achieving efficient node management without repudiation and tampering (C5). Results have shown improved reliability (C7) of FL tasks over mobile networks. Based on reviewed research, FL has become an adequate solution for user privacy preservation.

C.6.5 Summary

Section C.6 presents trends in ML research being adopted in network management. DL has been applied to networking problems based on large data sets, while RL has studied how to perform tasks by interacting with an environment without being given any information beforehand. In contrast, FL enables decentralized learning where data remains local, and each device runs its copy of the model. The literature review shows that RL and DL focus on *Mobility* (C1) and *Scalability* (C2). *Energy efficiency* (C3) is addressed in all domains, while *Isolation* (C4) is discussed in RL. FL focuses on *Privacy* (C5) preservation. Keeping data private is a major challenge in future networks. Users will share their information (e.g. mobility pattern, application preferences) with service providers to receive higher QoE. Ensuring privacy and security while improving network performance is a future research topic. *Resilience* (C6) is unexplored in all domains, while RL and FL address *Reliability* (C7). *Heterogeneity* (C8) is studied in all domains, while *Throughput* (C9) is more noticeable in RL. *Federation* (C10) is unexplored. Combining these ML trends can lead to fully automated networks with minimum human intervention. Self-configuration and self-repairing features will strongly impact the performance of low latency services.

C.7 Security and privacy mechanisms for cloud-native infrastructures

C.7.1 Overview

Security and Privacy are crucial enablers of emerging use cases in future networks. Without proper authentication/authorization mechanisms and data privacy solutions, service providers will not benefit from improved technologies since users will only subscribe to services that protect their privacy and data. This section discusses trends in security research: BC, CS and TC. Table C.8 summarizes the

Table C.8: Summary of the reviewed works in terms of Security and Privacy.

Research Domain	Authors	Main focus	Year	Evaluation Criteria									
				C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
BlockChain (BC) (Sec. C.7.2)	Dorri, A., et al. [146]	IoT	2017	✗	✓	✓	✗	✓	✗	✗	✗	✗	✗
	Zamani, M., et al. [147]	BC resiliency	2018	✗	✓	✗	✗	✓	✓	✗	✗	✓	✗
	Alvarenga, I. D., et al. [148]	Management of VNFs	2018	✗	✗	✗	✗	✓	✓	✗	✗	✗	✓
	Dinh, T. T. A., et al. [149]	BC framework	2018	✗	✓	✗	✗	✓	✓	✗	✗	✓	✗
	Zheng, W., et al. [150]	BC platform	2019	✗	✓	✗	✗	✓	✗	✓	✗	✗	✗
	Qiu, Z., et al. [151]	IoT	2020	✗	✓	✓	✗	✓	✗	✓	✗	✓	✗
Cyber Security (CS) (Sec. C.7.3)	Varga, P., et al. [152]	SDN	2017	✗	✓	✗	✗	✓	✗	✗	✗	✓	✗
	Diro, A. A., et al. [153]	Fog Computing	2017	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗
	Sohal, A. S., et al. [154]	Fog Computing	2018	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
	Dzeparoska, K., et al. [155]	Architecture	2018	✗	✓	✗	✗	✓	✗	✗	✓	✓	✗
	Joshi, K. D., et al. [156]	Privacy in SDN	2019	✗	✓	✗	✗	✓	✗	✗	✗	✗	✓
Trusted Computing (TC) (Sec. C.7.4)	Aga, S., et al. [157]	Smart memory	2017	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗
	Gjerdrum, A. T., et al. [158]	TC performance	2017	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗
	Coughlin, M., et al. [159]	TC for secured NFV	2017	✗	✗	✗	✓	✓	✗	✗	✗	✓	✗
	Yin, H., et al. [160]	Decentralized TC	2018	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗

reviewed works.

C.7.2 BlockChain (BC)

BC [161] is a decentralized digital ledger of transactions distributed across a network of nodes based on P2P topologies. A list of transactions is named a block, which is then linked with other blocks through cryptographic hashes. Nodes keep a time-stamped series of immutable data records without a central authority. By design, BC is resilient to data modification since once a block is recorded, data modifications cannot happen without altering all subsequent blocks. BC is a potential enabler of efficient and secure sharing of information in next-generation networks due to its decentralized nature. BC has already been studied for IoT contexts in [146]. Dorri, A., et al. have proposed a lightweight BC-based architecture for IoT, eliminating overheads of classic BC methods. The authors adopt a hierarchical structure employing a centralized immutable ledger to reduce overhead, increase network scalability (C2), and optimize resource consumption (C3). Evaluations have demonstrated the robustness of their architecture against several attacks (C5).

BC resilience has also been studied. In [147], Zamani, M., et al. have presented RapidChain, a sharding-based public BC protocol, and assessed its scalability (C2) and security (C5). RapidChain is resilient (C6) to Byzantine faults from up to a 1/3 fraction of its participants. RapidChain employs an optimal intra-committee consensus algorithm achieving high throughput (C9) levels via block pipelining, a novel gossiping protocol for large blocks. In [148], Alvarenga, I. D., et al. have proposed a BC architecture for secure management, configuration, and migration of VNFs. Their approach ensures the integrity and consistency of VNF information while keeping the anonymity of tenants and configuration information (C5). The authors adopt a consensus protocol that validates every transaction before storing it in a block to provide resiliency against faulty systems and collusion attacks (C6). The authors consider a federated consensus model where participants are known, and their asymmetric key pair is certified by a third-party, previously agreed upon by the federation members (C10). In [149], Dinh, T. T. A., et al. have presented BLOCKBENCH, a benchmarking framework for understanding the performance of private BCs against data processing workloads. The authors have evaluated three major BC systems (i.e. Ethereum, Parity, and Hyperledger Fabric) regarding scalability (C2), fault tolerance and security (C5), throughput (C9), and latency. The authors conclude that Ethereum and Parity are more resilient (C6) to node failures but more vulnerable to security attacks that fork the BC. Cited works show that BC architectures can enable adequate resilience for next-generation networks.

BC architectures also acknowledge the importance of reliability. In [150], Zheng, W., et al. have developed a BC-as-a-service platform named NutBaaS, in which BC services (e.g. smart contracts, system monitoring) are enabled over cloud infrastructures. To improve NutBaaS flexibility and scalability (C2), the authors have adopted Kubernetes to automatically deploy container-based services. The authors state that NutBaaS helps developers to detect security vulnerabilities by providing smart contract services, thus avoiding economic losses (C5). The authors have also discussed the reliability (C7) of their NutBaaS platform. In [151], Qiu, Z., et al. have proposed a Dual Vote Confirmation based Consensus (DVCC) mechanism for the integration of BC and IoT. Firstly, the authors have adopted a role division approach to handle limited resources in IoT devices. Then, a dual confirmation algorithm has been designed to improve the security and fairness of the consensus. A voting method has also been presented to enhance delay performance (C2) and avoid energy waste (C3). Results have shown that DVCC guarantees the security (C5) and fairness of the consensus while achieving notable reliability (C7) and throughput (C9). Both works propose adequate BC-based architectures solving current reliability issues while providing efficient security measures.

C.7.3 CyberSecurity (CS)

CS [162] consists of protecting computer systems, networks and data from malicious attacks or threats. Organizations and enterprises apply security methods to protect against unauthorized access to their infrastructures. Hardware-accelerated security practices have been studied in [152]. Varga, P., et al. [152] have introduced an automated method to speed up the reaction lag in SDN-based DCs via FPGA-based processing units. The authors aim to enhance the security (C5) of DCNs and large clouds with a new FPGA-accelerated NFV. The authors evaluate their approach in a DCN with a new security application that detects and mitigates real-world Distributed Denial of Service (DDoS) attacks, with lags from 430 μs up to 3 ms - several orders of magnitude faster than traditional approaches (C2). The authors state that their FPGA-based closed loop achieves higher throughput (C9) levels and lower latency than conventional methods. Security aspects in Fog Computing have also been addressed recently. In [153]. Diro, A. A., et al. have proposed an FC-based publish-subscribe lightweight protocol for IoT and assessed its scalability (C2) and security (C5). Their scheme provides higher scalability and less overhead than traditional methods while guaranteeing similar levels of security. In [154], Sohal, A. S., et al. have proposed a CS framework to identify malicious edge devices in FC. The authors adopt a two-stage hidden Markov model to categorize edge devices while a Virtual Honeypot Device (VHD) stores information of all identified malicious devices to assist the system, securing it from future attacks (C5).

Security for SDN has been studied in [155]. Dzeparoska, K., et al. have proposed a hierarchical, logically centralized architecture, expressing security policies through Autonomous Systems (AS) as intents. The authors state that SD Internet Exchange Points (SDXs) provide flexible and programmable control over wide-area network traffic delivery. SDX collaboration has addressed scalability (C2) challenges since security (C5) intents are compiled and installed at the available SDX closest to the malicious source, effectively protecting against DDoS attacks. A heterogeneous (C8) topology (i.e. SDXs, SDN controllers) has assessed the throughput (C9) of their approach. In [156], Joshi, K. D., et al. have presented PRIME-Q, a privacy-aware E2E QoS framework in Multi-domain SDN. A privacy (C5) index has been defined to quantify the privacy level of E2E QoS coordination. Simulations have assessed the scalability (C2) and operational overhead of PRIME-Q in large multi-domain networks (C10). These works are appropriate answers to attacks or threats in the coming years.

C.7.4 Trusted Computing (TC)

TC [163] refers to technologies and proposals for solving security problems through enhancing hardware or modifying software to secure cloud computing and virtualized systems. In recent years, research has focused on improving hardware performance. In [157], Aga, S., et al. have proposed InvisiMem, a memory approach expanding the trust base to include the logic layer in the smart memory to cryptographic primitives. It allows the secure host processor to send encrypted addresses over the untrusted memory bus. InvisiMem significantly improves memory space, energy (C3), and overhead. The authors also state that InvisiMem properly supports enclave isolation (C4). The authors conclude that InvisiMem ensures similar security (C5) to Oblivious RAM (ORAM)-based solutions. In [158], Gjerdrum, A. T., et al. have studied the performance characteristics of SGX technology to understand how it could enforce privacy policies in cloud-hosted Software-as-a-Service (SaaS) architectures (C5). The authors also state that developers should pre-provision enclaves in a disposable pool of resources to prevent reuse between isolation (C4) domains if before-the-fact usage of enclaves is accurately predicted. In [159], Coughlin, M., et al. have studied the application of TC to circumvent limitations on encrypted data. Encrypted data introduces an overhead while providing a limited set of operations since encrypted schemes are not completely homomorphic. The authors have studied whether SGX technologies can perform secure packet processing since SGX provides proper software isolation (C4) and attestation (C5). A real NFV environment has been evaluated, including throughput (C9) experiments. In [160], Yin, H., et al. have presented HyperNet, a decentralized TC and networking paradigm to address control loss over data. The decentralized trusted connection is based on BC smart contracts enabling secure digital object management and an identifier-driven routing mechanism (C5). The authors remark that HyperNet handles data sovereignty and helps to build a Universal Data object Identifier (UDI)-driven network capable of indexing and routing data. Cited works show that increasing hardware trust is the first step towards efficient security in future networks.

C.7.5 Summary

Section C.7 reviews novel security mechanisms enabling data privacy and secure transmissions in cloud-native systems. BC enables a decentralized trust system, while CS implements security practices to defend cloud systems from potential threats. TC focuses on trust and software isolation through hardware-based protection. The literature review shows that *Mobility* (C1) is unaddressed, while BC and CS address *Scalability* (C2) issues arising when adopting novel security mechanisms. *Energy efficiency* (C3) is more noticeable in BC, while TC focuses on *Isolation* (C4). As expected, these domains mainly discuss *Security* and *Privacy* (C5). BC also addresses *Resilience* (C6) and *Reliability* (C7), while *Heterogeneity* (C8) is studied in CS. BC and CS also address *Throughput* (C9) and *Federation* (C10). Securing future networks requires all three domains. A decentralized architecture fully supports low latency services, as long as efficient security against attacks and high trust when executing NFs in hardware is guaranteed.

C.8 Emerging Applications for next-generation networks

C.8.1 Overview

The emerging cloud-native infrastructures and novel technologies lead to new use cases requiring even more stringent requirements (e.g. higher reliability, lower latency). This section presents research focused on four emerging use cases: Smart Cities, Self-driving cars, XR, and IIoT. Table C.9 summarizes the reviewed works.

C.8.2 Smart Cities

A Smart City [186] applies technology to improve the performance of its urban services by transforming simple objects into smart connected devices. Sensors are spread around the city to collect data and then insights are extracted from these data to improve urban operations and services. Several domains of urban life are affected by these types of applications, such as waste management, environmental monitoring, urban mobility, and healthcare. Research has been studying suitable architectures for IoT services. In [164], Montori, F., et al. have introduced an

Table C.9: Summary of the reviewed works in terms of Application.

Research Domain	Authors	Main focus	Year	Evaluation Criteria									
				C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Smart Cities (Sec. C.8.2)	Montori, F., et al. [164]	Architecture	2017	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗
	Cheng, B., et al. [165]	Fog Computing	2017	✗	✓	✓	✗	✗	✗	✓	✓	✓	✓
	Ejaz, W., et al. [166]	Energy Mgmt.	2017	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗
	Santos, J., et al. [167]	Resource alloc.	2018	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗
	Liu, Y., et al. [168]	Energy Mgmt.	2019	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
	Aloqaily, M., et al. [169]	Security	2019	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗
	Santos, J., et al. [170]	Resource alloc.	2020	✓	✓	✓	✓	✗	✗	✗	✓	✗	✗
Self driving cars (Sec. C.8.3)	Maqueda, I., et al. [171]	Steering Prediction	2018	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
	Chen, S., et al. [172]	Testing platform	2019	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
	Chernik., A., et al. [173]	Security	2019	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
	Ndik., A., et al. [174]	Caching for MEC	2020	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗
Extended Reality (XR) (Sec. C.8.4)	Liu, Y., et al. [175]	MEC-assisted VR	2018	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
	Doumanoglou, A., et al. [176]	QoE	2018	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗
	Hooft, J., et al. [177]	Adaptive VR	2019	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
	Hooft, J., et al. [178]	Point cloud comp.	2019	✗	✗	✗	✗	✗	✓	✗	✗	✓	✗
Industrial IoT (IIoT) (Sec. C.8.5)	Zhou, L., et al. [179]	Data processing	2017	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗
	Cheng, J., et al. [180]	5G smart mfg.	2018	✗	✗	✓	✗	✓	✗	✓	✓	✗	✗
	Luvisotto, M., et al. [181]	indoor IIoT	2018	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗
	Hiller, J., et al. [182]	Low Latency	2018	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗
	Yang, H., et al. [183]	Low Latency	2019	✓	✗	✓	✗	✗	✗	✓	✓	✓	✗
	Niya, S. R., et al. [184]	BC architecture	2020	✗	✗	✓	✗	✓	✗	✓	✓	✓	✗
	Kumar, T., et al. [185]	BC architecture	2020	✗	✓	✓	✗	✓	✗	✓	✓	✗	✗

architecture to handle data sources in IoT. The authors have also proposed crowd-sensing management functionalities for environmental data, addressing reliability (C7). Their approach focuses on the integration of heterogeneous (C8) data sources. In [165], Cheng, B., et al. have designed and implemented an FC framework for Smart Cities named FogFlow. Message propagation latency, scalability (C2) and energy consumption (C3) have been assessed based on an anomaly detection use case. The authors state that FogFlow is a reliable (C7) option to handle multiple IoT brokers in a Smart City environment. The heterogeneity (C8) of IoT has been considered through different device profiles (e.g. temperature sensor, alarm) and distinct contexts (e.g. data processing, service management). Throughput (C9) has also been evaluated. The authors also propose a federated broker to exchange context information with other federated brokers in a multi-domain environment (C10). In [166], Ejaz, W., et al. have discussed efficient energy management in Smart Cities. The authors have addressed energy harvesting in a Smart City to extend the lifetime of low-powered devices (C3). The heterogeneous (C8) nature of IoT has also been considered. In [167], the City of Things (CoT) framework has been presented for data collection and analysis and automated resource provisioning in Smart Cities. The framework has been evaluated on an air quality monitoring use case by deploying air quality sensors in cars. CoT is a flexible and scalable (C2) approach for the Smart City ecosystem. Heterogeneity (C8) has been discussed through different devices and distinct functionalities (e.g. ML algorithms, data operations). In [168], Liu, Y., et al. have designed an IoT-based energy management system with DRL. An efficient energy scheduling method has been presented to manage the uncertainty of energy supply and demand in a Smart City (C3). All cited architectures are appropriate to deal with massive numbers of connected devices and the stringent requirements of IoT.

Security mechanisms have also been studied in [169]. Aloqaily, M., et al. have introduced a cloud framework for connected vehicles focused on intrusion detection mechanisms against security attacks while meeting user QoS levels. The authors have discussed the future role of connected cars in the electric power grid (C3), by suggesting the stabilization of the power grid through wirelessly charging batteries. Simulations have shown that their approach significantly mitigates real attacks (C5). Lastly, in [170], a MILP formulation for IoT service allocation has been proposed, which considers SFC concepts, different Low Power Wide Area Network (LPWAN) technologies, and multiple optimization objectives. Mobility (C1) has been addressed since users move around in the network, and the MILP model considers the impact of service migrations. The scalability (C2) of the MILP formulation has been assessed for several optimization objectives, including energy efficiency (C3) and E2E latency. The concept of NS has also been included in the model by adding different slices to multiple applications (C4). Heterogeneity (C8) has been satisfied through different service requirements (e.g. CPU, RAM) and distinct LPWAN technologies (i.e. IEEE 802.11ah and LoRaWAN).

C.8.3 Self-driving cars

Self-driving or autonomous vehicles [187] are cars or trucks in which little to no human intervention is needed. These vehicles sense the environment and drive safely through software-based control decisions. Self-driving cars are getting significant attention in several research domains. In [171], Maqueda, A. I., et al. have applied DL for steering prediction in self-driving cars. The authors aim to investigate whether DL algorithms provide robust steering prediction based on data samples from event cameras. Their approach produces reliable (C7) steering angle predictions in challenging situations (e.g. poor lighting, fast motion). In [172], Chen, S., et al. have proposed an integrated simulation system for self-driving vehicles. The approach consists of four main components: the vehicle kinematic model simulation, the multi-sensor simulation, the environment simulation, and the Electronic Control Unit (ECU) hardware. Several experiments have validated their platform, showing its performance and reliability (C7).

Security in self-driving cars has also been recently studied. In [173]. Chernikova, A., et al. have proved that evasion attacks are a threat to steering angle predictions in autonomous vehicles (C5). The authors have stated that these attacks threaten the safe applicability of DL in autonomous driving. In [174], Ndikumana, A., et al. have proposed infotainment caching in self-driving cars, in which caching decisions are made based on passenger features obtained through DL. DL predicts cached contents in autonomous cars or MEC servers attached to roadside units. The authors state that lower delays for downloading operations are achieved if autonomous cars select available MEC servers en route since self-driving cars are delay-sensitive due to their high mobility (C1). In their evaluation, heterogeneous contents (i.e. movies with different demands and user recommendations) have been considered and throughput (C9) levels have been assessed.

C.8.4 Extended Reality (XR)

XR [188] refers to all real and virtual environments generated via computers or even wearables that blend virtual and physical worlds to create fully immersive experiences. In AR, objects are overlaid into the real world by enhancing the user experience through AR glasses. XR is one of the emerging use cases for future networks that will completely revolutionize multimedia service delivery. MEC-assisted VR has been studied in [175]. Liu, Y., et al. have proposed a Panoramic VR Video (PVRV) streaming system designed for millimeter-wave (mmWave) mobile networks in combination with MEC. The authors remark that adopting MEC servers can improve wireless bandwidth utilization and UE energy efficiency (C3). Simulations have shown that their PVRV system improves energy efficiency and the quality of received viewport over conventional methods. QoE of VR streaming services has also been addressed in [176]. Doumanoglou, A., et al. have studied the QoE of real-time 3D media content streamed to VR headsets for entertainment purposes. The aim is to embed real users within virtual environments of interactive games to provide a fully immersive experience. The evaluation has considered multiple users under varying network conditions to assess the overall QoE concerning a range of visual quality and latency parameters. The trade-off between latency introduced by a reliable transport protocol (i.e. TCP) versus frame loss rate has been assessed (C7). In their experiments, lag issues occurred when using TCP, though the authors claim that a buffering mechanism potentially mitigates these issues. Throughput (C9) assumptions for TCP and User Datagram Protocol (UDP) have been included in their scheme.

VR adaptive streaming has also been investigated. In [177], three research challenges in immersive streaming have been tackled: viewport prediction, tile-based rate adaptation and application layer optimizations. A content-agnostic viewport prediction scheme based on spherical walks alongside a novel rate adaptation heuristic for tile-based video has been proposed. The advantages of using HTTP/2 server push have been studied since it significantly improves viewport prediction, video quality, and throughput (C9) compared to existing methods. In [178], HTTP adaptive streaming of VR content through Point Cloud Compression (PCC) has been studied. Several rate adaptation heuristics have been presented to decide the most appropriate quality representation of each VR object. The trade-off between accurate prediction and resilience (C6) to playout freezes has been discussed. Increasing the buffer size results in lower interactivity, prediction accuracy, and video quality. However, a larger buffer results in higher resilience to playout freezes, a crucial factor for over-the-top video streaming. Throughput (C9) traces have been considered in the evaluation. Cited works aim to provide efficient mechanisms towards maximizing users QoE.

C.8.5 Industrial IoT (IIoT)

IIoT [189] refers to the extension of IoT towards the industrial sector. Sensors interconnect with manufacturing processes and robots easing data collection to improve productivity and efficiency of industrial processes. Energy consumption has been an important subject in IIoT. In [179], Zhou, L., et al. have studied the performance of different computing methods in IIoT by analyzing the relationship between data processing and energy consumption (C3). Heterogeneity (C8) has been discussed in the consumption estimations of data transmissions from multiple sources. In [180], Cheng, J., et al. have proposed a 5G-based IIoT architecture while describing different manufacturing functionalities for three use cases:

eMBB, mMTC and URLLC. The authors discuss the importance of low-cost and low-energy consumption in 5G (C3). Data privacy and security (C5) aspects have been considered alongside reliability (C7) and heterogeneity (C8). The authors remark that real-time monitoring in IIoT requires a packet loss rate lower than 1×10^{-12}. Current 4G technologies cannot meet these requirements, while 5G wireless communications are promising for URLLC services. In [181], Luvisotto, M., et al. have assessed the performance of LoRaWAN for typical IIoT use cases, such as indoor industrial monitoring. The authors discuss how particular parameters can be adapted to increase the performance of their industrial scenario. Simulations have shown that LoRaWAN is a viable alternative for these applications since it ensures low energy consumption (C3) and high reliability (C7).

Low latency communications have also been recently addressed. In [182], Hiller, J., et al. have studied secure low latency communications in IIoT. The authors propose antedated encryption and fast data authentication with templates, securing low-powered devices (C3). The authors state that implementing security (C5) measures on constrained IoT devices adds further latency overhead. Security processes execution time increases the perceived latency. Evaluations have quantified the overhead of their approach focused on latency and energy consumption, showing that IIoT latency requirements are met through antedated encryption and fast data authentication. In [183], Yang, H., et al. have proposed a heterogeneous Radio Frequency (RF) / Visible Light Communication (VLC) architecture to satisfy different QoS requirements for URLLC devices. A resource management approach has also been formulated as a Markov Decision Process (MDP), followed by an RL algorithm that learns the optimal policy. The authors consider user mobility (C1) in the RL policy strategy. The heterogeneous (C8) RF/VLC architecture has been evaluated concerning energy consumption per device (C3), reliability (C7) and throughput (C9). Results have shown that their approach enables energy-efficient communications, satisfies URLLC requirements (i.e. high reliability, low latency), and ensures high data rates at different scenarios in IIoT. Low latency in IIoT is essential since high delays can have major repercussions (e.g. damaged machines, injured employees). Both works show promising results to guarantee low latency communications in industrial scenarios.

BC architectures have been proposed for IIoT. In [184], Niya, S. R., et al. have proposed a BC-agnostic architecture named BIIT for the IIoT. BIIT aims to reduce computational overhead and enhance energy efficiency (C3). The authors intend to offer higher levels of trust (C5), transparency, and data reliability (C7) by leveraging BC. Heterogeneity (C8) has been discussed regarding data collection. The authors have also studied performance issues in their architecture in combination with LoRa, focusing on throughput (C9) concerns. The authors maximize throughput while simultaneously providing data integrity through cryptographic signatures. The authors suggest performing large-scale experiments with BIIT as future work, validating the scalability of the system in LoRa and cellular networks. In [185], Kumar, T., et al. have proposed BlockEdge, a framework combining edge computing with BC to address the stringent requirements introduced by IIoT. The feasibility of BlockEdge has been assessed regarding scalability (C2), latency, power consumption (C3) and network usage compared to non-BC approaches. Results prove that BlockEdge provides decentralized trust and security (C5) management in IIoT without compromising system performance and resource efficiency. In contrast, the system reliability (C7) is improved. The Heterogeneity (C8) of IIoT environments has also been addressed. Cited works show that BC supports reliable communications in IIoT.

C.8.6 Summary

Section C.8 reviews emerging applications for future networks. Self-driving cars address *Mobility* (C1) concerns, while Smart Cities and IIoT focus on *Scalable* (C2) and *Energy-efficient* (C3) platforms, acknowledging the importance of low power consumption in these environments. Few works address *Isolation* (C4), while *Security* (C5) is a major concern in self-driving cars and IIoT. *Resilience* (C6) is mostly unexplored, while *Reliability* (C7) concerns are addressed in self-driving cars and IIoT. Smart Cities and IIoT address their *Heterogeneous* (C8) nature due to IoT, while XR discusses *Throughput* (C9) since these applications require high bandwidth data rates, as well as low latency as part of QoE expectations. *Federation* (C10) is mostly unaddressed. All these applications are pushing towards paradigm shifts in architecture, communication, orchestration and security. Addressing current hurdles will help next-generation networks to fully support these applications and deliver low E2E latency. The next section discusses open challenges and future directions.

C.9 Open Challenges & Future Directions

Table C.10: Summary of revised works on low latency service delivery.

Evaluation Criteria	Relevant reviewed works	Number of Publications
Mobility (C1)	[37], [38], [40], [42], [45], [46], [49], [50], [69], [71], [84], [100], [103], [107], [117], [118], [119], [122], [128], [130], [135], [136], [170], [174], [183]	25 (20.8%)

continued on next page...

Table C.10: Summary of revised works on low latency service delivery (continued)

Evaluation Criteria	Relevant reviewed works	Number of Publications
Scalability (C2)	[43], [45], [46], [49], [48], [47], [52], [53], [54], [55], [56], [57], [58], [71], [73], [75], [78], [79], [80], [81], [85], [86], [87], [88], [89], [90], [104], [108], [109], [110], [111], [112], [113], [115], [127], [129], [130], [131], [139], [146], [147], [149], [150], [151], [152], [153], [155], [156], [165], [167], [170], [185]	52 (43.3%)
Energy Efficiency (C3)	[39], [49], [50], [56], [57], [76], [77], [85], [101], [102], [103], [104], [105], [106], [107], [108], [109], [117], [120], [121],[122], [128], [129], [130], [133], [136], [137], [138], [146], [151], [157], [165], [166], [168], [169], [170], [175], [179], [180], [181], [182], [183], [184],[185]	44 (36.6%)
Isolation (C4)	[38], [44], [48], [52], [55], [69], [70], [71], [72], [75], [79], [81], [83], [104], [112], [116], [134], [136], [157], [158], [159], [170]	22 (18.3%)
Security & Privacy (C5)	[38], [43], [44], [46], [47], [50], [51], [52], [53], [71], [79], [82], [84], [116], [121], [122], [137], [139], [140], [141], [146], [147], [148], [149], [150], [151], [152], [153], [154],[155], [156], [157], [158], [159], [160], [169], [173], [180], [182], [184], [185]	41 (34.2%)
Resilience (C6)	[47], [88], [90], [147], [148], [149], [178]	7 (5.8%)
Reliability (C7)	[39], [40], [41], [42], [46], [47], [49], [50], [69], [70], [72], [73], [74], [78], [80], [88], [90], [100], [106], [107], [135], [141], [150], [151], [164], [165], [171], [172], [176],[180], [181], [183], [184], [185]	34 (28.3%)

continued on next page...

Table C.10: Summary of revised works on low latency service delivery (continued)

Evaluation Criteria	Relevant reviewed works	Number of Publications
Heterogeneity (C8)	[40], [42], [43], [45], [46], [47], [48], [49], [50], [53], [54], [55], [56], [69], [70], [72], [75], [78], [79], [80], [81], [83], [86], [87], [100], [101], [102], [103], [105], [107], [111], [113], [114], [115], [119],[120], [121], [126], [128], [129], [130], [131], [134], [135], [137], [155], [164], [165], [166], [167], [170], [174], [179], [180], [183], [184], [185]	57 (47.5%)
Throughput (C9)	[37], [38], [39], [40], [41], [45], [46], [47],[49], [51], [55], [56], [57], [58], [70], [69], [72], [73], [74], [76], [77], [75], [79], [82], [83], [85], [88], [100], [106], [105], [108], [110], [112], [113], [114], [116], [118], [122], [126], [131], [132], [139], [147], [149], [151], [152], [155], [159], [165], [174], [176], [177], [178], [183], [184]	55 (45.8%)
Federation (C10)	[42], [46], [77], [80], [81], [86], [111], [148], [156], [165]	10 (8.3%)

To tackle open challenges in low latency service delivery, several works addressing different domains need to be combined to meet all criteria shown in previous summary Tables (i.e. IV to IX). This section discusses challenges not yet fully addressed by literature and highlights future directions. Table C.10 summarizes the reviewed works aggregated per evaluation criteria. By combining research from different domains, readers can easily consult works addressing a particular criterion of their interest.

Mobility (C1) has been addressed by several works. Nevertheless, no generic approach has been proposed to ensure service operation when end-users or devices are moving. MEC (e.g. [37], [38]) and FC (e.g. [45],[46]) address mobility focused on mobile devices and IoT, respectively. A viable alternative is merging both domains and provide an integrated solution for the mobility support of different devices (e.g. end-users, IoT sensors and vehicles). Integrating the mobile and the cloud domain is a research direction that will certainly help address future mobility requirements. NS [69], [71] and IBN [84] have also been proposed to handle user mobility and usage patterns. Mobility and access functionalities can be deployed and managed independently. Future research will focus on NS and IBN to offer mobility alongside high degrees of flexibility and customization. Resource allocation and caching have also addressed mobility requirements. However, most works focus only on specific use cases, such as D2D networks [100] or MEC [119]. ML has also been studied for mobility management in V2V communications [135] or MEC (e.g. [128], [130]). These methods aim to autonomously manage user mobility, but no clear approach has been yet applied in practice. Emerging use cases such as autonomous vehicles, IIoT, and in particular, Smart City services would definitively benefit from enhanced mobility support in future cloud-native infrastructures.

Scalability (C2) has been tackled in the revised literature, starting from different points of view. Novel architectures (e.g. [43], [48]) have been designed to focus on application scalability or in supporting a high number of connected devices. Hardware-accelerated platforms have also been studied for performance improvement regarding NFV reconfiguration [55], memory access [57] and data processing pipelines [58]. Their purpose is to enhance the performance and scalability of hardware-based platforms running softwarized NFs. Several works have addressed scalability issues in SFC (e.g. [73],[75]), IBN [80],[81] and SR (e.g. [87], [88]). All these networking paradigms aim to provide higher levels of flexibility and reconfigurability to current networks, but their impact on network scalability needs to be acknowledged. Future research will focus on implementing novel mechanisms in practical testbeds and assess their scalability and overhead concerning execution time and resource usage. Auto-scaling (e.g. [108],[109]) and performance monitoring systems (e.g. [112],[113]) are usually designed with scalability in mind. Also, the scalability of ML algorithms is not neglected. A few works (e.g. [129], [131]) study how scalable ML techniques are compared to existing methods. In turn, the implementation of novel security practices focuses on increasing network scalability as BC architectures (e.g. [150], [151]) and on how faster these methods can handle security breaches than traditional approaches (e.g. [153],[155]). Scalability requirements have also been more noticeable in Smart Cities (e.g. [165], [167]) and IIoT [185]. A fully scalable system is needed to fully provide low latency service delivery. All domains will play their part in delivering an architecture capable of handling high-demand patterns with efficient management capabilities that enable scalable deployments of future applications.

Energy Efficiency (C3) has been addressed in several domains. Trends on architectural paradigms such as MEC, FC and Hardware acceleration have covered energy efficiency by reducing power consumption in the infrastructure [39], network links [49] and the high performance of NNs [56], [57]. Energy-efficient hardware is a future research direction. Energy savings have also been addressed in SFC placement [76],[77] and SR performance [85]. Efficient service placement and traffic routing are also future research directions, in which service providers will focus on reducing their deployment costs while providing high QoS. The development of container-based service chains and VNFs will focus even more on minimizing resources. Works on resource allocation (e.g. [104],[105]) and caching (e.g. [121],[122]) have also studied energy consumption to deliver more efficient and greener architectures. Distributed and energy-efficient architectures are another future direction. ML has also been applied to reduce energy consumption [128], [136]. All connected devices will generate a massive volume of data that, if not properly handled, leads to slow decision-making and increased power consumption. Distributed ML operations offer an adequate solution to handle massive amounts of data. In turn, BC architectures aim to enhance performance and avoid energy waste [146], [151]. Another direction is to further optimize the performance of low-powered devices. Regarding applications, current literature focuses mostly on Smart Cities (e.g. [169], [170]) and IIoT ([181], [182]) scenarios.

Isolation (C4) and NS concepts have also been getting significant attention in the last few years. However, standard implementations or detailed interfaces are still missing. Most mobile operators have already implemented PoCs to provide service isolation inside their network, but no interfaces or management models have yet been standardized. The proper administration of slice-based VNFs is still being defined by ETSI NFV MANO, including operations for the life-cycle control of slice instances. A few works address the architectural challenges of implementing isolated running environments [38] or adding NS features to their schemes [44], [48]. Also, NS has been studied mostly for 5G (e.g. [69], [71]), but a mature design and concrete implementation guidelines are missing. Isolation has also been investigated in SFC [75],[79] and monitoring [112],[116] contexts, where increased performance has gained notable attention. Isolation offers high levels of security and TC research has been adopting isolation features to enhance current hardware platforms (e.g. [158],[159]). Emerging use cases demand QoS-specific network slices on a per-application basis, in which providers rely on fully cloud-native infrastructures to provide service isolation and meet QoS levels. Also, NS in a federation model is still unexplored. Federated slices raise further issues since multi-domain resource discovery and slice brokering are needed. How to effectively protect and secure network slices in a federation is a future direction.

Security and Privacy (C5) concerns are still commonly left out in most works unless security is the main focus as in BC, CS, or TC research domains. Other research fields are also tackling security issues, though no generic approach has been given. Works on architecture mostly address security risks in service offloading [38], unauthorized access [44] and secure communications [46]. Also, privacy is even more important in future networks due to the massive gathering of user data. Taleb, T. et al. [71] have addressed privacy concerns while customizing services based on NS. Their work is an adequate solution for user privacy preservation in future networks. Protect infrastructures in a continuum of virtual resources is the main research direction. BC (e.g. [149], [150], [151]) has positioned itself as the main enabler of security and privacy while providing resiliency and reliability guarantees. Combining BC concepts with existing approaches within architecture, network, and orchestration is another future direction. Lastly, research on emerging applications has studied how to enable security features without compromising performance, such as in self-driving cars [173] and IIoT (e.g. [180], [182]). Without efficient security mechanisms, users do not benefit from these use cases since safety is their main priority.

Resilience (C6) has been viewed as one of the main features of future architectures, but only a few works address resilience concerns. Sharma, P. K., et al. [47] have presented a distributed cloud architecture providing service guarantees when failures occur in the infrastructure. Desmouceaux, Y., et al. [88] have proposed an SR-based load balancer and its resiliency has been evaluated. Aubry, F., et al. [90] have introduced the concept of RDP, including a fault tolerance assessment. Works leveraging BC (e.g. [147], [148]) have also focused on resilience aspects. SR and BC are promising domains to efficiently provide resilience features in a cloud-native infrastructure, ensuring service continuity under critical situations. The integration of SR and BC is a future research direction. Resilience has also been studied for XR services towards avoiding playout freezes in VR adaptive streaming [178]. Emerging applications require extremely low latency but also high resilience guarantees against attacks and unexpected failures.

Reliability (C7) has been viewed as an important enabler of the next evolution of high-precision services. Research on novel architectures (e.g. [40],[47]) has studied mechanisms to improve service reliability. However, only simulations or small-scale scenarios have been evaluated and implementations in practical environments are missing. The reliability assessment of these approaches in large-scale scenarios is a future direction. Literature has also addressed reliability for URLLC services based on NS (e.g. [70],[72]). Also, works have investigated how to provide reliability guarantees for SFC deployments (e.g. [74], [78]) and improve network reliability focused on SR [88],[90]. SFC and SR are both future directions to improve the reliability of next-generation networks. Resource allocation research has also studied reliability improvements for D2D [100] and M2M communications [106]. In addition, autonomous cars [171], [172] and UAVs are pushing the limits of infrastructures to offer at least seven nines of reliability while precision demanding services, such as XR [176] and IIoT (e.g. [180],[181]) are stretching current infrastructures to the edge to provide high levels of reliability and large throughput data rates.

Heterogeneity (C8) has been addressed by several works since the heterogeneous nature of future networks is widely accepted. Architectures discuss heterogeneity based on several aspects, such as integrated network schemes [40], IoT support [43] and different computing capacities [47]. These works aim to support a large number of devices while improving network performance. Hardware-based platforms [55], [56] have also addressed heterogeneity concerns. A future direction is studying the implications of the adoption of these novel architectural paradigms. NS (e.g. [70],[72]), SFC (e.g. [75], [78]) and IBN (e.g. [81],[83]) have also addressed the interoperability between different domains and services. Novel orchestration practices (e.g. [102], [103],[114]) have also considered the heterogeneous nature of services and environments. Caching research (e.g. [119],[120]) has also addressed heterogeneity concerning distinct device (e.g. vehicles, mobile devices, BSs). DL (e.g. [126],[128]) and RL (e.g. [131],[134]) also consider different networks and architectures. In terms of applications, Smart Cities (e.g. [166],[167]) and IIoT (e.g. [180], [183]) distinct themselves due to their highly complex environment encompassing several stringent requirements. The analysis of heterogeneity implications in these scenarios is a future research direction.

Throughput (C9) is essential for the new range of use cases that evolve beyond 5G. MEC (e.g. [38], [37]) and FC (e.g. [47], [49]) have tackled architectural challenges to attain higher throughput under different scenarios. Hardware-accelerated platforms aim to solve performance degradation while maintaining high throughput levels (e.g. [57], [58]). NS also considers throughput as an application-specific requirement to constitute different slices [69]. SFC research (e.g. [75], [79]) has also addressed throughput under distinct chain scenarios. The impact of throughput performance should be acknowledged in future research. Resource allocation (e.g. [106],[105]) and monitoring operations (e.g. [113],[114]) have become promising areas to maximize throughput. Efficient allocation attains higher throughput levels while proper motoring identifies problems faster. ML is also providing automated allocation and monitoring operations to enhance system performance including throughput (e.g. [126], [132]). Security research (e.g. [149], [151]) has also studied how to maintain notable throughput performance while guaranteeing system security. Regarding applications, throughput requirements are more prominent in XR since it requires real-time operations in a fully immersive environment and data cannot be compressed. Throughput requirements are expected to surpass 1 Tbps due to its data-hungry nature. Current literature (e.g. [177],[178]) has proposed efficient adaptive streaming mechanisms for VR content to improve throughput performance.

Federation (C10) is still unaddressed in most works. A potential research direction is the design of federated models for cloud-native infrastructures focused on the interoperability between several providers. MEC and FC have addressed architectural challenges based on multiple providers [42], [46]. Efficient SFC placement in multi-cloud environments has also been studied in [77]. IBN [80], [81] and SR [86] have also considered multi-domain environments to solve federation challenges. The design of efficient networking schemes for federated environments is a future research direction. Auto-scaling and scheduling mechanisms for cloud systems across geographically spread regions have also been studied in [111]. Developing efficient orchestration mechanisms for deploying and maintaining application components across federated clouds is another research direction. Security research has also addressed federation concepts in BC architectures [148] or multi-domain networks [156]. Existing works acknowledge the need for federation and already mention the intention to address multi-domain environments or federation concepts as future work. These extensions will lead to future federation research.

C.10 Lessons Learned and Prospects

C.10.1 Lessons Learned

Several lessons have been derived from the literature review relevant to low latency service delivery. *Architecture* has shown that MEC [37-44] and FC [45-50] are essential for the evolution of the mobile network to create a cloud-native environment where services can be deployed in a continuum of virtual resources. The integration of MEC and FiWi access networks has been studied in [40], while FC has been recently extended through connected vehicles [45]. Both works bring resources even further to the edge. Micro-services [51-54] have also revolutionized service deployments and conventional resource-hungry monoliths have been transformed into small loosely coupled micro-services. Micro-service research mostly focuses on isolation, security, and scalability aspects. Hardware acceleration [55-58] is a promising domain to mitigate performance degradation and latency introduced by network softwarization. Both [56] and [57] are adequate approaches for performance improvement of NNs. Migrating conventional hardware functions towards softwarized VNFs is needed to fully achieve low E2E latency. Hardware-accelerated NFs further help supporting low latency service delivery.

Communication Networks have proved that novel networking concepts provide higher levels of flexibility and scalability towards low latency service delivery. NS [69-72] addresses isolation through the creation of E2E logical slices for each group of users accessing the same service. NS has even been extended in [69] by integrating vehicles in the infrastructure. SFC [73-79] has been explored to provide a complete E2E service chain. SFC allocation has already been addressed for MEC [128] or FC [129] architectures. Recently, IBN [80-84] has been conceptualized to communicate intents to the network. Enforcing rules without detailing the system how it should perform is the main purpose of IBN research. It is still in the early stages, but IBN is an adequate answer to improve network reliability and scalability. An E2E service orchestration approach across multiple domains has been presented in [80]. The work discusses low latency challenges but also reflects on reliability and scalability issues. SR [85-90] deals with scalability concerns but also tackles resilience and reliability. Resilience has been addressed in

[88]. Results have shown that IBN improves the scalability and throughput performance at a negligible cost of CPU overhead. These four networking concepts are viable alternatives to enable the support of low latency service delivery in future networks. The combination of these concepts is key to obtain a more integrated solution.

Orchestration practices for distributed cloud infrastructures are also gaining significant attention. Resource allocation [100-107] addresses proper service deployment, while auto-scaling [108-111] mechanisms guarantee that deployed services are sufficient to handle current network demand. Efficient provisioning strategies have been studied for vehicular networks and IoT contexts. Both [101] and [102] are adequate strategies for massive numbers of connected devices while optimizing resources in fog-cloud infrastructures. These provisioning strategies need to be complemented with auto-scaling features, such as [110] and [111], where appropriate actions are made based on the current demand. Efficient allocation and auto-scaling are only possible if proper monitoring [112-116] tools are employed. Both [113] and [115] are viable alternatives to monitor distributed systems and raise alerts in case of anomalous behaviors. Caching [117-122] also improves content delivery and reduces data transmissions if adopted at the fog or edge. Mobility and security benefit from novel caching strategies as shown in [117] and [121], respectively. All these domains will play their role in next-generation networks. However, no standardization has yet been accepted in these domains. Current works continue investigating proper solutions for efficient and cost-aware orchestration in future networks.

ML and AI have positioned themselves as crucial enablers of autonomous networks. DL [126-130] has already been applied to traffic congestion [126] and SFC placement and routing [127], [128]. DL has achieved high performance compared to existing methods. Recently, RL [131-136] has been applied to resource allocation and traffic control [132], [133] and auto-scaling [131], [110]. RL has shown its potential applicability in these domains due to its performance and scalability. Existing methods (e.g. ILP formulations) cannot deal with dynamic demands and their implementation in practice is difficult due to their high-resolution time. Developing RL systems capable of reallocating services in the infrastructure by reacting to sudden network demands is a major research topic in network management. Also, FL [137-141] provides decentralized learning features for wireless networks [137], [138] alongside adequate privacy guarantees [139], [140]. All these domains will help achieve higher levels of independence in next-generation networks. The combination of these trends can lead to fully automated networks with minimum human intervention, providing self-configuration and self-repairing features that will strongly impact the performance of low latency services.

Security and Privacy are gaining even more importance in future networks. Traditional practices are no longer adequate since data and services are spread around in the network and stored at different levels (i.e. edge, fog, cloud). BC [146-151] provides a fully decentralized architecture capable of providing higher levels of reliability and resiliency. BC architectures as [150] and [151] have shown promising performance while mitigating security vulnerabilities. BC is one of the most promising technologies in the coming years. Novel CS [152-156] practices have also been introduced to protect infrastructures [150], mitigate attacks [152] and guarantee user privacy [156]. These works are appropriate answers to attacks or threats in the coming years. Also, TC [157-160] has been studying hardware enhancements focused on security issues. TC addresses trust while increasing hardware performance. All domains will play their role in securing distributed cloud systems enabling the support of low latency services in next-generation networks. *Emerging Applications* are pushing towards more efficient and reliable infrastructures. Smart cities [164-170] focus mostly on stringent requirements coming from IoT. Both [165] and [167] are adequate alternatives to handle large volumes of data in a connected city. Self-driving cars [171-174] have addressed security and reliability concerns. DL has been adopted in [171] for steering prediction in autonomous cars while evasion attacks have been studied in [173]. Reliability and security are essential for the support of self-driving cars. Both works have shown promising results. XR [175-178] focuses on providing fully immersive experiences through AR glasses. XR deployments will revolutionize multimedia service delivery in future networks. MEC-based systems have already been proposed for VR content delivery [175] while a few works have studied the optimization of VR streaming systems [177], [178]. The adoption of novel architectural concepts in combination with novel streaming mechanisms is the major research direction. IIoT [179-185] has studied the adoption of IoT in the industrial sector. Low latency communications [182] and security [184] are major concerns. Reliability is also important, especially for URLLC services [180]. New ambitious and challenging use cases will emerge in the coming years leading networking innovation and the creation of new business models in next-generation networks. Prospects of emerging use cases are discussed next.

C.10.2 Prospects of emerging use cases

All research domains introduced in this appendix will play a major role in enabling emerging use cases. *Architectural* paradigms such as MEC and FC will reduce latency in the communication between devices and services. Smart City and IIoT services will benefit the most from these two concepts since services are deployed at the edge and fog providing lower latency and enabling local operations. Also, micro-services allow flexible low-cost deployments as opposed to rigid costly VM allocations. Hardware-based platforms will also support efficient softwarized VNFs aiming to mitigate performance degradation.

Communication Networks will also play their role. Emerging use cases have diverse service requirements. XR requires throughput levels above 1Tbps, while their interactive experiences need sub-millisecond latency. In contrast, autonomous cars and UAVs demand seven levels of reliability without necessarily requiring higher throughput. NS allows setting up specific network slices for each of the envisioned use cases. Also, SFC allows developers to create service chains of containerized services for each of their applications. Container-based service chains for Smart City use cases have already been proposed in [170]. In turn, IBN and SR will completely revolutionize current networks. IBN enables network management through intents while SR provides scalable and flexible routing mechanisms, simplifying traffic engineering. Both technologies will help achieve the referenced sub-millisecond latency.
Orchestration practices will adapt. A plethora of allocation and auto-scaling mechanisms have been developed to address the stringent requirements of emerging applications. A trade-off between requirements is crucial for proper service allocation and scaling. XR requires low latency and high throughput, while IIoT needs high reliability and high resiliency guarantees. Cost-efficient allocations have been addressed in [101] while low latency provisioning has been studied in [102]. Anomaly detection and monitoring operations will be executed close to end-users, allowing faster responses. These methods will maintain agreed QoS levels and perform operational adjustments when needed in near real-time. Also, edge caching schemes will overcome high mobility patterns, especially necessary for self-driving cars. Cars could then access services while moving as shown in [174].
ML and AI will automate several tasks currently being solved via human intervention. Resource allocation [128, 130], NS [136] and privacy preservation [140] are among the most envisioned operations being resolved by these algorithms. Also, FL will contribute to an enriched collaborative learning experience where devices train a common model. This is particularly relevant for Smart City and IIoT scenarios, where a common model can increase performance and reliability.
Security and Privacy mechanisms will mitigate attacks and avoid service disruptions. Security is crucial for self-driving cars [173] and IIoT [184]. A failure in these scenarios may have costly repercussions (e.g. damaged machines, car accidents). TC will also keep hardware secured. Isolation guarantees will ensure secure packet processing [159]. The close interplay between all domains is key to fully support low latency 6G services in the future.

C.11 Conclusion

This appendix surveys the literature on ongoing research aiming to support low latency services in next-generation networks. A taxonomy on low latency service delivery has been proposed alongside a specific set of criteria to classify research across different domains. Open challenges and future directions have been discussed, while lessons learned have been derived from our literature review. Also, prospects have been provided with a focus on the role that novel trends will play in emerging use cases such as XR.

Acknowledgment

This research was performed within the project "Intelligent DEnse And Longe range IoT networks (IDEAL-IoT)" under Grant Agreement #S004017N, from the fund for Scientific Research-Flanders (FWO-V). The authors thank the FlexNGIA consortium members.

References

[1] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[2] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications surveys & tutorials*, 18(1):236–262, 2015.

[3] White paper on 6g networking, 2020. URL https://www.6gchannel.com/portfolio-posts/6g-white-paper-networking/.

[4] Marco Giordani, Michele Polese, Marco Mezzavilla, Sundeep Rangan, and Michele Zorzi. Toward 6g networks: Use cases and technologies. *IEEE Communications Magazine*, 58(3):55–61, 2020.

[5] Nils Tijtgat, Wiebe Van Ranst, Toon Goedeme, Bruno Volckaert, and Filip De Turck. Embedded real-time object detection for a uav warning system. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 2110–2118, 2017.

[6] Mehmet Ersue. Etsi nfv management and orchestration-an overview. *Presentation at the IETF*, 88, 2013.

[7] Hemanth Kumar Ravuri, Maria Torres Vega, Tim Wauters, Bin Da, Alexander Clemm, and Filip De Turck. An experimental evaluation of flow setup latency in distributed software defined networks. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 432–437. IEEE, 2019.

[8] Shihabur Rahman Chowdhury, Mohammad A Salahuddin, Noura Limam, and Raouf Boutaba. Re-architecting nfv ecosystem with microservices: State of the art and research challenges. *IEEE Network*, 33(3):168–176, 2019.

[9] Akhil Gupta and Rakesh Kumar Jha. A survey of 5g network: Architecture and emerging technologies. *IEEE access*, 3:1206–1232, 2015.

[10] Faqir Zarrar Yousaf, Michael Bredel, Sibylle Schaller, and Fabian Schneider. Nfv and sdn—key technology enablers for 5g networks. *IEEE Journal on Selected Areas in Communications*, 35(11):2468–2478, 2017.

[11] Bego Blanco, Jose Oscar Fajardo, Ioannis Giannoulakis, Emmanouil Kafetzakis, Shuping Peng, Jordi Pérez-Romero, Irena Trajkovska, Pouria S Khodashenas, Leonardo Goratti, Michele Paolino, et al. Technology pillars in the architecture of future 5g mobile networks: Nfv, mec and sdn. *Computer Standards & Interfaces*, 54:216–228, 2017.

[12] Sami Kekki, Walter Featherstone, Yonggang Fang, Pekka Kuure, Alice Li, Anurag Ranjan, Debashish Purkayastha, Feng Jiangping, Danny Frydman, Gianluca Verin, et al. Mec in 5g networks. *ETSI white paper*, 28:1–28, 2018.

[13] Shangguang Wang, Jinliang Xu, Ning Zhang, and Yujiong Liu. A survey on service migration in mobile edge computing. *IEEE Access*, 6:23511–23528, 2018.

[14] Imtiaz Parvez, Ali Rahmati, Ismail Guvenc, Arif I Sarwat, and Huaiyu Dai. A survey on low latency towards 5g: Ran, core network and caching solutions. *IEEE Communications Surveys & Tutorials*, 20(4):3098–3130, 2018.

[15] Thomas O Olwal, Karim Djouani, and Anish M Kurien. A survey of resource management toward 5g radio access networks. *IEEE Communications Surveys & Tutorials*, 18(3):1656–1686, 2016.

[16] Farah Aït Salaht, Frédéric Desprez, and Adrien Lebre. An overview of service placement problem in fog and edge computing. *ACM Computing Surveys (CSUR)*, 53(3):1–35, 2020.

[17] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. Application management in fog computing environments: A taxonomy, review and future directions. *ACM Computing Surveys (CSUR)*, 53(4):1–35, 2020.

[18] Judy C Guevara, Ricardo da S Torres, and Nelson LS da Fonseca. On the classification of fog computing applications: A machine learning perspective. *Journal of Network and Computer Applications*, 159:102596, 2020.

[19] Prateek Shantharama, Akhilesh S Thyagaturu, and Martin Reisslein. Hardware-accelerated platforms and infrastructures for network functions: A survey of enabling technologies and research studies. *IEEE Access*, 8: 132021–132085, 2020.

[20] Tongyi Huang, Wu Yang, Jun Wu, Jin Ma, Xiaofei Zhang, and Daoyin Zhang. A survey on green 6g network: Architecture and technologies. *IEEE Access*, 7:175758–175768, 2019.

[21] Mohamed Faten Zhani and Hesham ElBakoury. Flexngia: A flexible internet architecture for the next-generation tactile internet. *Journal of Network and Systems Management*, pages 1–45, 2020.

[22] Zhengquan Zhang, Yue Xiao, Zheng Ma, Ming Xiao, Zhiguo Ding, Xianfu Lei, George K Karagiannidis, and Pingzhi Fan. 6g wireless networks: Vision, requirements, architecture, and key technologies. *IEEE Vehicular Technology Magazine*, 14(3):28–41, 2019.

[23] Ghazaleh Javadzadeh and Amir Masoud Rahmani. Fog computing applications in smart cities: A systematic survey. *Wireless Networks*, 26(2): 1433–1457, 2020.

[24] Rasheed Hussain and Sherali Zeadally. Autonomous cars: Research results, issues, and future challenges. *IEEE Communications Surveys & Tutorials*, 21(2):1275–1313, 2018.

[25] Dimitris Chatzopoulos, Carlos Bermejo, Zhanpeng Huang, and Pan Hui. Mobile augmented reality survey: From where we are to where we go. *Ieee Access*, 5:6917–6950, 2017.

[26] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, 10(4):2233–2243, 2014.

[27] Bob Briscoe, Anna Brunstrom, Andreas Petlund, David Hayes, David Ros, Jyh Tsang, Stein Gjessing, Gorry Fairhurst, Carsten Griwodz, and Michael Welzl. Reducing internet latency: A survey of techniques and their merits. *IEEE Communications Surveys & Tutorials*, 18(3):2149–2196, 2014.

[28] Xiaolin Jiang, Hossein Shokri-Ghadikolaei, Gabor Fodor, Eytan Modiano, Zhibo Pang, Michele Zorzi, and Carlo Fischione. Low-latency networking: Where latency lurks and how to tame it. *Proceedings of the IEEE*, 107(2): 280–306, 2018.

[29] Yuan-Yao Shih, Wei-Ho Chung, Ai-Chun Pang, Te-Chuan Chiu, and Hung-Yu Wei. Enabling low-latency applications in fog-radio access networks. *IEEE network*, 31(1):52–58, 2016.

[30] Joachim Sachs, Lars AA Andersson, José Araújo, Calin Curescu, Johan Lundsjö, Göran Rune, Eckehard Steinbach, and Gustav Wikström. Adaptive 5g low-latency communication for tactile internet services. *Proceedings of the IEEE*, 107(2):325–349, 2018.

[31] Patrick Marsch, Icaro Da Silva, Omer Bulakci, Milos Tesanovic, Salah Eddine El Ayoubi, Thomas Rosowski, Alexandros Kaloxylos, and Mauro Boldi. 5g radio access network architecture: Design guidelines and key considerations. *IEEE Communications Magazine*, 54(11):24–32, 2016.

[32] Mohammad Asif Habibi, Meysam Nasimi, Bin Han, and Hans D Schotten. A comprehensive survey of ran architectures toward 5g mobile communication system. *IEEE Access*, 7:70371–70421, 2019.

[33] Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hines. 5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges. *Computer Networks*, 167:106984, 2020.

[34] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. Cloudlets: Bringing the cloud to the mobile user. In *Proceedings of the third ACM workshop on Mobile cloud computing and services*, pages 29–36, 2012.

[35] JMH Elmirghani, T Klein, K Hinton, L Nonde, AQ Lawey, TEH El-Gorashi, MOI Musa, and X Dong. Greentouch greenmeter core network energy-efficiency improvement measures and optimization. *IEEE/OSA Journal of Optical Communications and Networking*, 10(2):A250–A269, 2018.

[36] John C Whitson and Jose Emmanuel Ramirez-Marquez. Resiliency as a component importance measure in network reliability. *Reliability Engineering & System Safety*, 94(10):1685–1693, 2009.

[37] Xinwei Liu, Jiaxin Zhang, Xing Zhang, and Wenbo Wang. Mobility-aware coded probabilistic caching scheme for mec-enabled small cell networks. *IEEE Access*, 5:17824–17833, 2017.

[38] Lele Ma, Shanhe Yi, Nancy Carter, and Qun Li. Efficient live migration of edge services leveraging container layered storage. *IEEE Transactions on Mobile Computing*, 18(9):2020–2033, 2018.

[39] Han-Chuan Hsieh, Jiann-Liang Chen, and Abderrahim Benslimane. 5g virtualized multi-access edge computing platform for iot applications. *Journal of Network and Computer Applications*, 115:94–102, 2018.

[40] Jing Liu, Guochu Shou, Yaqiong Liu, Yihong Hu, and Zhigang Guo. Performance evaluation of integrated multi-access edge computing and fiber-wireless access networks. *IEEE Access*, 6:30269–30279, 2018.

[41] Shun-Ren Yang, Yu-Ju Tseng, Chen-Chia Huang, and Wo-Chien Lin. Multi-access edge computing enhanced video streaming: Proof-of-concept implementation and prediction/qoe models. *IEEE Transactions on Vehicular Technology*, 68(2):1888–1902, 2018.

[42] Syed Danial Ali Shah, Mark A Gregory, Shuo Li, and Ramon Dos Reis Fontes. Sdn enhanced multi-access edge computing (mec) for e2e mobility and qos management. *IEEE Access*, 8:77459–77469, 2020.

[43] Pasika Ranaweera, Vashish N Imrith, Madhusanka Liyanag, and Anca Delia Jurcut. Security as a service platform leveraging multi-access edge computing infrastructure provisions. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2020.

[44] Adlen Ksentini and Pantelis A Frangoudis. Toward slicing-enabled multi-access edge computing in 5g. *IEEE Network*, 34(2):99–105, 2020.

[45] Mehdi Sookhak, F Richard Yu, Ying He, Hamid Talebian, Nader Sohrabi Safa, Nan Zhao, Muhammad Khurram Khan, and Neeraj Kumar. Fog vehicular computing: Augmentation of fog computing using vehicular cloud computing. *IEEE Vehicular Technology Magazine*, 12(3):55–64, 2017.

[46] Rafael Moreno-Vozmediano, Ruben S Montero, Eduardo Huedo, and Ignacio M Llorente. Cross-site virtual network in cloud and fog computing. *IEEE Cloud Computing*, 4(2):46–53, 2017.

[47] Pradip Kumar Sharma, Mu-Yen Chen, and Jong Hyuk Park. A software defined fog node based distributed blockchain cloud architecture for iot. *Ieee Access*, 6:115–124, 2017.

[48] Roberto Bruschi, Franco Davoli, Paolo Lago, and Jane Frances Pajo. A scalable sdn slicing scheme for multi-domain fog/cloud services. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–6. IEEE, 2017.

[49] Enzo Baccarelli, Paola G Vinueza Naranjo, Michele Scarpiniti, Mohammad Shojafar, and Jemal H Abawajy. Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study. *IEEE access*, 5:9882–9910, 2017.

[50] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Fog computing: Enabling the management and orchestration of smart city applications in 5g networks. *Entropy*, 20(1):4, 2018.

[51] Stefan Brenner, Tobias Hundt, Giovanni Mazzeo, and Rüdiger Kapitza. Secure cloud micro services using intel sgx. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 177–191. Springer, 2017.

[52] Daniel Guija and Muhammad Shuaib Siddiqui. Identity and access control for micro-services based 5g nfv platforms. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pages 1–10, 2018.

[53] Ronghua Xu, Seyed Yahya Nikouei, Yu Chen, Erik Blasch, and Alexander Aved. Blendmas: A blockchain-enabled decentralized microservices architecture for smart public safety. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 564–571. IEEE, 2019.

[54] Olivier Debauche, Saïd Mahmoudi, Sidi Ahmed Mahmoudi, Pierre Manneback, and Frédéric Lebeau. A new edge architecture for ai-iot services deployment. *Procedia Computer Science*, 175:10–19, 2020.

[55] Xuzhi Zhang, Xiaozhe Shao, George Provelengios, Naveen Kumar Dumpala, Lixin Gao, and Russell Tessier. Scalable network function virtualization for heterogeneous middleboxes. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 219–226. IEEE, 2017.

[56] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74, 2017.

[57] Ruizhe Cai, Ao Ren, Ning Liu, Caiwen Ding, Luhao Wang, Xuehai Qian, Massoud Pedram, and Yanzhi Wang. Vibnn: Hardware acceleration of bayesian neural networks. *ACM SIGPLAN Notices*, 53(2):476–488, 2018.

[58] Muhsen Owaida, Gustavo Alonso, Laura Fogliarini, Anthony Hock-Koon, and Pierre-Etienne Melet. Lowering the latency of data processing pipelines through fpga based hardware acceleration. *Proceedings of the VLDB Endowment*, 13(1):71–85, 2019.

[59] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[60] Etsi technical specification, mobile edge computing (mec): Framework and reference architecture., 2016. URL http://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/\gs_MEC003v010101p.pdf.

[61] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H Glitho, Monique J Morrow, and Paul A Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 20(1):416–464, 2017.

[62] Paolo Bellavista, Javier Berrocal, Antonio Corradi, Sajal K Das, Luca Foschini, and Alessandro Zanni. A survey on fog computing for the internet of things. *Pervasive and mobile computing*, 52:71–99, 2019.

[63] Nadim Parvez, Anirban Mahanti, and Carey Williamson. An analytic throughput model for tcp newreno. *IEEE/ACM Transactions on Networking*, 18(2):448–461, 2009.

[64] Etsi technical specification, onem2m functional architecture, onem2m ts-0001 version 2.10.0 release 2., 2016. URL http://www.etsi.org/deliver/etsi_ts/118100_118199/118101/\02.10.00_60/ts_118101v021000p.pdf.

[65] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, and Mike Amundsen. *Microservice architecture: aligning principles, practices, and culture*. " O'Reilly Media, Inc.", 2016.

[66] Marcus A Christie, Anuj Bhandar, Supun Nakandala, Suresh Marru, Eroma Abeysinghe, Sudhakar Pamidighantam, and Marlon E Pierce. Managing authentication and authorization in distributed science gateway middleware. *Future Generation Computer Systems*, 111:780–785, 2020.

[67] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. *Queue*, 14(1):70–93, 2016.

[68] Leonardo Linguaglossa, Stanislav Lange, Salvatore Pontarelli, Gábor Rétvári, Dario Rossi, Thomas Zinner, Roberto Bifulco, Michael Jarschel, and Giuseppe Bianchi. Survey of performance acceleration techniques for network function virtualization. *Proceedings of the IEEE*, 107(4):746–764, 2019.

[69] Claudia Campolo, Antonella Molinaro, Antonio Iera, and Francesco Menichella. 5g network slicing for vehicle-to-everything services. *IEEE Wireless Communications*, 24(6):38–45, 2017.

[70] Adlen Ksentini and Navid Nikaein. Toward enforcing network slicing on ran: Flexibility and resources abstraction. *IEEE Communications Magazine*, 55(6):102–108, 2017.

[71] Tarik Taleb, Badr Mada, Marius-Iulian Corici, Akihiro Nakao, and Hannu Flinck. Permit: Network slicing for personalized 5g mobile telecommunications. *IEEE Communications Magazine*, 55(5):88–93, 2017.

[72] Petar Popovski, Kasper Fløe Trillingsgaard, Osvaldo Simeone, and Giuseppe Durisi. 5g wireless network slicing for embb, urllc, and mmtc: A communication-theoretic view. *Ieee Access*, 6:55765–55779, 2018.

[73] Long Qu, Chadi Assi, Khaled Shaban, and Maurice J Khabbaz. A reliability-aware network service chain provisioning with delay guarantees in nfv-enabled enterprise datacenter networks. *IEEE Transactions on Network and Service Management*, 14(3):554–568, 2017.

[74] Linquan Zhang, Shangqi Lai, Chuan Wu, Zongpeng Li, and Chuanxiong Guo. Virtualized network coding functions on the internet. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 129–139. IEEE, 2017.

[75] Irena Trajkovska, Michail-Alexandros Kourtis, Christos Sakkas, Denis Baudinot, João Silva, Piyush Harsh, George Xylouris, Thomas Michael Bohnert, and Harilaos Koumaras. Sdn-based service function chaining mechanism and service prototype implementation in nfv scenario. *Computer Standards & Interfaces*, 54:247–265, 2017.

[76] Insun Jang, Dongeun Suh, Sangheon Pack, and György Dán. Joint optimization of service function placement and flow distribution for service function chaining. *IEEE Journal on Selected Areas in Communications*, 35 (11):2532–2541, 2017.

[77] Deval Bhamare, Mohammed Samaka, Aiman Erbad, Raj Jain, Lav Gupta, and H Anthony Chan. Optimal virtual network function placement in multi-cloud service function chaining architecture. *Computer Communications*, 102:1–16, 2017.

[78] Hassan Hawilo, Manar Jammal, and Abdallah Shami. Network function virtualization-aware orchestrator for service function chaining placement in the cloud. *IEEE Journal on Selected Areas in Communications*, 37(3):643–655, 2019.

[79] Zuo Xiang, Frank Gabriel, Elena Urbano, Giang T Nguyen, Martin Reisslein, and Frank HP Fitzek. Reducing latency in virtual machines: Enabling tactile internet for human-machine co-working. *IEEE Journal on Selected Areas in Communications*, 37(5):1098–1116, 2019.

[80] Walter Cerroni, Chiara Buratti, Simone Cerboni, Gianluca Davoli, Chiara Contoli, Francesco Foresta, Franco Callegati, and Roberto Verdone. Intent-based management and orchestration of heterogeneous openflow/iot sdn domains. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–9. IEEE, 2017.

[81] Saeed Arezoumand, Kristina Dzeparoska, Hadi Bannazadeh, and Alberto Leon-Garcia. Md-idn: Multi-domain intent-driven networking in software-defined infrastructures. In *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–7. IEEE, 2017.

[82] Thomas Szyrkowiec, Michele Santuari, Mohit Chamania, Domenico Siracusa, Achim Autenrieth, Victor Lopez, Joo Cho, and Wolfgang Kellerer. Automatic intent-based secure service creation through a multilayer sdn network orchestration. *Journal of Optical Communications and Networking*, 10(4):289–297, 2018.

[83] Khizar Abbas, Muhammad Afaq, Talha Ahmed Khan, Adeel Rafiq, and Wang-Cheol Song. Slicing the core network and radio access network domains through intent-based networking for 5g networks. *Electronics*, 9(10): 1710, 2020.

[84] Yuhang Wang, Zhihong Tian, Yanbin Sun, Xiaojiang Du, and Nadra Guizani. Locjury: An ibn-based location privacy preserving scheme for iocv. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[85] Junjie Pang, Gaochao Xu, and Xiaodong Fu. Sdn-based data center networking with collaboration of multipath tcp and segment routing. *IEEE Access*, 5:9764–9773, 2017.

[86] Alessio Giorgetti, Andrea Sgambelluri, Francesco Paolucci, F Cugini, and P Castoldi. Segment routing for effective recovery and multi-domain traffic engineering. *Journal of Optical Communications and Networking*, 9(2): A223–A232, 2017.

[87] Antonio Cianfrani, Marco Listanti, and Marco Polverini. Incremental deployment of segment routing into an isp network: A traffic engineering perspective. *IEEE/ACM Transactions on Networking*, 25(5):3146–3160, 2017.

[88] Yoann Desmouceaux, Pierre Pfister, Jérôme Tollet, Mark Townsley, and Thomas Clausen. 6lb: Scalable and application-aware load balancing with segment routing. *IEEE/ACM Transactions on Networking*, 26(2):819–834, 2018.

[89] Uma Chunduri, Alexander Clemm, and Richard Li. Preferred path routing-a next-generation routing framework beyond segment routing. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2018.

[90] François Aubry, Stefano Vissicchio, Olivier Bonaventure, and Yves Deville. Robustly disjoint paths with segment routing. In *Proceedings of the 14th international conference on emerging networking experiments and technologies*, pages 204–216, 2018.

[91] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis, Adlen Ksentini, and Hannu Flinck. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys & Tutorials*, 20(3):2429–2453, 2018.

[92] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Towards delay-aware container-based service function chaining in fog computing. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2020.

[93] Hendrik Moens and Filip De Turck. Vnf-p: A model for efficient placement of virtualized network functions. In *10th International Conference on Network and Service Management (CNSM) and Workshop*, pages 418–423. IEEE, 2014.

[94] Deval Bhamare, Raj Jain, Mohammed Samaka, and Aiman Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155, 2016.

[95] Engin Zeydan and Yekta Turk. Recent advances in intent-based networking: A survey. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5. IEEE, 2020.

[96] Alexander Clemm, Laurent Ciavaglia, Lisandro Granville, and Jeff Tantsura. Intent-based networking-concepts and definitions, 2020. URL https://tools.ietf.org/pdf/draft-irtf-nmrg-ibn-concepts-definitions-02.pdf.

[97] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov,

William Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6, 2014.

[98] Zahraa N Abdullah, Imtiaz Ahmad, and Iftekhar Hussain. Segment routing in software defined networks: A survey. *IEEE Communications Surveys & Tutorials*, 21(1):464–486, 2018.

[99] Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10): 1094–1104, 2001.

[100] Le Liang, Geoffrey Ye Li, and Wei Xu. Resource allocation for d2d-enabled vehicular communications. *IEEE Transactions on Communications*, 65(7): 3186–3197, 2017.

[101] Hamid Reza Arkian, Abolfazl Diyanat, and Atefe Pourkhalili. Mist: Fog-based data analytics scheme with cost-efficient resource provisioning for iot crowdsensing applications. *Journal of Network and Computer Applications*, 82:152–165, 2017.

[102] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Resource provisioning for iot application services in smart cities. In *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2017.

[103] Jingjing Yao and Nirwan Ansari. Qos-aware fog resource provisioning and mobile device power control in iot networks. *IEEE Transactions on Network and Service Management*, 16(1):167–175, 2018.

[104] Prashanth Podili and Kotaro Kataoka. Effective resource provisioning for qos-aware virtual networks in sdn. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2018.

[105] Haijun Zhang, Fang Fang, Julian Cheng, Keping Long, Wei Wang, and Victor CM Leung. Energy-efficient resource allocation in noma heterogeneous networks. *IEEE Wireless Communications*, 25(2):48–53, 2018.

[106] Zhenyu Zhou, Yufei Guo, Yanhua He, Xiongwen Zhao, and Wael M Bazzi. Access control and resource allocation for m2m communications in industrial automation. *IEEE Transactions on Industrial Informatics*, 15(5):3093–3103, 2019.

[107] Arshad Farhad, Dae-Ho Kim, Beom-Hun Kim, Alaelddin Fuad Yousif Mohammed, and Jae-Young Pyun. Mobility-aware resource assignment to iot

applications in long-range wide area networks. *IEEE Access*, 8:186111–186124, 2020.

[108] Mohammad Sadegh Aslanpour, Mostafa Ghobaei-Arani, and Adel Nadjaran Toosi. Auto-scaling web applications in clouds: A cost-aware approach. *Journal of Network and Computer Applications*, 95:26–41, 2017.

[109] Sabidur Rahman, Tanjila Ahmed, Minh Huynh, Massimo Tornatore, and Biswanath Mukherjee. Auto-scaling vnfs using machine learning to improve qos and reduce cost. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[110] Doyoung Lee, Jae-Hyoung Yoo, and James Won-Ki Hong. Deep q-networks based auto-scaling for service function chaining. In *2020 16th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2020.

[111] Thomas Lin and Alberto Leon-Garcia. Towards a client-centric qos auto-scaling system. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2020.

[112] Farnaz Moradi, Christofer Flinta, Andreas Johnsson, and Catalin Meirosu. Conmon: An automated container based network performance monitoring system. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 54–62. IEEE, 2017.

[113] Gioacchino Tangari, Daphne Tuncer, Marinos Charalambides, and George Pavlou. Decentralized monitoring for large-scale software-defined networks. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 289–297. IEEE, 2017.

[114] Syed Yousaf Shah, Zengwen Yuan, Songwu Lu, and Petros Zerfos. Dependency analysis of cloud applications for performance monitoring using recurrent neural networks. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1534–1543. IEEE, 2017.

[115] Daniel Perdices, David Muelas, Luis de Pedro, and Jorge E López de Vergara. Network performance monitoring with flexible models of multi-point passive measurements. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2018.

[116] Igor Jochem Sanz, Diogo Menezes Ferrazani Mattos, and Otto Carlos Muniz Bandeira Duarte. Sfcperf: An automatic performance evaluation framework for service function chaining. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2018.

[117] Min Chen, Yixue Hao, Long Hu, Kaibin Huang, and Vincent KN Lau. Green and mobility-aware caching in 5g networks. *IEEE Transactions on Wireless Communications*, 16(12):8347–8361, 2017.

[118] Kuljaree Tantayakul, Riadh Dhaou, and Beatrice Paillassa. Mobility management with caching policy over sdn architecture. In *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–7. IEEE, 2017.

[119] Ke Zhang, Supeng Leng, Yejun He, Sabita Maharjan, and Yan Zhang. Cooperative content caching in 5g networks with mobile edge computing. *IEEE Wireless Communications*, 25(3):80–87, 2018.

[120] Yixue Hao, Min Chen, Long Hu, M Shamim Hossain, and Ahmed Ghoneim. Energy efficient task caching and offloading for mobile edge computing. *IEEE Access*, 6:11365–11373, 2018.

[121] Liang Xiao, Xiaoyue Wan, Canhuang Dai, Xiaojiang Du, Xiang Chen, and Mohsen Guizani. Security in mobile edge caching with reinforcement learning. *IEEE Wireless Communications*, 25(3):116–122, 2018.

[122] Fen Cheng, Guan Gui, Nan Zhao, Yunfei Chen, Jie Tang, and Hikmet Sari. Uav-relaying-assisted secure transmission with caching. *IEEE Transactions on Communications*, 67(5):3140–3153, 2019.

[123] Lien Deboosere, Bert Vankeirsbilck, Pieter Simoens, Filip De Turck, Bart Dhoedt, and Piet Demeester. Efficient resource management for virtual desktop cloud computing. *The Journal of Supercomputing*, 62(2):741–767, 2012.

[124] Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in nfv: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.

[125] Chenhao Qu, Rodrigo N Calheiros, and Rajkumar Buyya. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys (CSUR)*, 51(4):1–33, 2018.

[126] Fengxiao Tang, Zubair Md Fadlullah, Bomin Mao, and Nei Kato. An intelligent traffic load prediction-based adaptive channel assignment algorithm in sdn-iot: A deep learning approach. *IEEE Internet of Things Journal*, 5 (6):5141–5154, 2018.

[127] Jianing Pei, Peilin Hong, and Defang Li. Virtual network function selection and chaining based on deep learning in sdn and nfv-enabled networks. In

2018 IEEE International Conference on Communications Workshops (ICC Workshops), pages 1–6. IEEE, 2018.

[128] Feibo Jiang, Kezhi Wang, Li Dong, Cunhua Pan, Wei Xu, and Kun Yang. Deep learning based joint resource scheduling algorithms for hybrid mec networks. *IEEE Internet of Things Journal*, 2019.

[129] Nazli Siasi, Mohammed Jasim, Adel Aldalbahi, and Nasir Ghani. Deep learning for service function chain provisioning in fog computing. *IEEE Access*, 8:167665–167683, 2020.

[130] Zaiwar Ali, Sadia Khaf, Ziaul Haq Abbas, Ghulam Abbas, Fazal Muhammad, and Sunghwan Kim. A deep learning approach for mobility-aware and energy-efficient resource allocation in mec. *IEEE Access*, 8:179530–179546, 2020.

[131] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 191–205, 2018.

[132] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. Experience-driven networking: A deep reinforcement learning based approach. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1871–1879. IEEE, 2018.

[133] Ji Li, Hui Gao, Tiejun Lv, and Yueming Lu. Deep reinforcement learning based computation offloading and resource allocation for mec. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2018.

[134] Rongpeng Li, Zhifeng Zhao, Qi Sun, I Chih-Lin, Chenyang Yang, Xianfu Chen, Minjian Zhao, and Honggang Zhang. Deep reinforcement learning for resource management in network slicing. *IEEE Access*, 6:74429–74441, 2018.

[135] Hao Ye, Geoffrey Ye Li, and Biing-Hwang Fred Juang. Deep reinforcement learning based resource allocation for v2v communications. *IEEE Transactions on Vehicular Technology*, 68(4):3163–3173, 2019.

[136] Giuseppe Faraci, Christian Grasso, and Giovanni Schembra. Design of a 5g network slice extension with mec uavs managed with reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 38(10):2356–2371, 2020.

[137] Nguyen H Tran, Wei Bao, Albert Zomaya, Nguyen Minh NH, and Choong Seon Hong. Federated learning over wireless networks: Optimization model design and analysis. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1387–1395. IEEE, 2019.

[138] Mingzhe Chen, Zhaohui Yang, Walid Saad, Changchuan Yin, H Vincent Poor, and Shuguang Cui. A joint learning and communications framework for federated learning over wireless networks. *IEEE Transactions on Wireless Communications*, 2020.

[139] Youyang Qu, Longxiang Gao, Tom H Luan, Yong Xiang, Shui Yu, Bai Li, and Gavin Zheng. Decentralized privacy using blockchain-enabled federated learning in fog computing. *IEEE Internet of Things Journal*, 2020.

[140] Chunyi Zhou, Anmin Fu, Shui Yu, Wei Yang, Huaqun Wang, and Yuqing Zhang. Privacy-preserving federated learning in fog computing. *IEEE Internet of Things Journal*, 2020.

[141] Jiawen Kang, Zehui Xiong, Dusit Niyato, Yuze Zou, Yang Zhang, and Mohsen Guizani. Reliable federated learning for mobile networks. *IEEE Wireless Communications*, 27(2):72–80, 2020.

[142] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[143] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Reinforcement learning for service function chain allocation in fog computing. *Book Chapter in revision, Submitted to Communications Network and Service Management In the Era of Artificial Intelligence and Machine Learning, IEEE Press*, 2020.

[144] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[145] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[146] Ali Dorri, Salil S Kanhere, and Raja Jurdak. Towards an optimized blockchain for iot. In *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 173–178. IEEE, 2017.

[147] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 931–948, 2018.

[148] Igor D Alvarenga, Gabriel AF Rebello, and Otto Carlos MB Duarte. Securing configuration management and migration of virtual network functions using blockchain. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2018.

[149] Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Ji Wang. Untangling blockchain: A data processing view of blockchain systems. *IEEE Transactions on Knowledge and Data Engineering*, 30(7): 1366–1385, 2018.

[150] Weilin Zheng, Zibin Zheng, Xiangping Chen, Kemian Dai, Peishan Li, and Renfei Chen. Nutbaas: A blockchain-as-a-service platform. *Ieee Access*, 7: 134422–134433, 2019.

[151] Zheng Qiu, Jianjun Hao, Yijun Guo, and Yi Zhang. Dual vote confirmation based consensus design for blockchain integrated iot. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–7. IEEE, 2020.

[152] Pál Varga, Georgios Kathareios, Ákos Máté, Rolf Clauberg, Andreea Anghel, Péter Orosz, Balázs Nagy, Tamás Tóthfalusi, László Kovács, and Mitch Gusat. Real-time security services for sdn-based datacenters. In *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2017.

[153] Abebe Abeshu Diro, Naveen Chilamkurti, and Neeraj Kumar. Lightweight cybersecurity schemes using elliptic curve cryptography in publish-subscribe fog computing. *Mobile Networks and Applications*, 22(5):848–858, 2017.

[154] Amandeep Singh Sohal, Rajinder Sandhu, Sandeep K Sood, and Victor Chang. A cybersecurity framework to identify malicious edge device in fog computing and cloud-of-things environments. *Computers & Security*, 74:340–354, 2018.

[155] Kristina Dzeparoska, Hadi Bannazadeh, and Alberto Leon-Garcia. Sdx-based security collaboration: Extending the security reach beyond network domains. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 63–71. IEEE, 2018.

[156] Kalpana D Joshi and Kotaro Kataoka. Prime-q: Privacy aware end-to-end qos framework in multi-domain sdn. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 169–177. IEEE, 2019.

[157] Shaizeen Aga and Satish Narayanasamy. Invisimem: Smart memory for trusted computing. In *International Symposium on Computer Architecture*, volume 10, 2017.

[158] Anders T Gjerdrum, Robert Pettersen, Håvard D Johansen, and Dag Johansen. Performance of trusted computing in cloud infrastructures with intel sgx. In *CLOSER*, pages 668–675, 2017.

[159] Michael Coughlin, Eric Keller, and Eric Wustrow. Trusted click: Overcoming security issues of nfv in the cloud. In *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 31–36, 2017.

[160] Hao Yin, Dongchao Guo, Kai Wang, Zexun Jiang, Yongqiang Lyu, and Ju Xing. Hyperconnected network: A decentralized trusted computing and networking paradigm. *IEEE Network*, 32(1):112–117, 2018.

[161] Michael Nofer, Peter Gomber, Oliver Hinz, and Dirk Schiereck. Blockchain. *Business & Information Systems Engineering*, 59(3):183–187, 2017.

[162] Rossouw Von Solms and Johan Van Niekerk. From information security to cyber security. *computers & security*, 38:97–102, 2013.

[163] Sean W Smith. *Trusted computing platforms: design and applications*. Springer-Verlag, 2005.

[164] Federico Montori, Luca Bedogni, and Luciano Bononi. A collaborative internet of things architecture for smart cities and environmental monitoring. *IEEE Internet of Things Journal*, 5(2):592–605, 2017.

[165] Bin Cheng, Gürkan Solmaz, Flavio Cirillo, Ernö Kovacs, Kazuyuki Terasawa, and Atsushi Kitazawa. Fogflow: Easy programming of iot services over cloud and edges for smart cities. *IEEE Internet of Things Journal*, 5 (2):696–707, 2017.

[166] Waleed Ejaz, Muhammad Naeem, Adnan Shahid, Alagan Anpalagan, and Minho Jo. Efficient energy management for the internet of things in smart cities. *IEEE Communications Magazine*, 55(1):84–91, 2017.

[167] Jose Santos, Thomas Vanhove, Merlijn Sebrechts, Thomas Dupont, Wannes Kerckhove, Bart Braem, Gregory Van Seghbroeck, Tim Wauters, Philip Leroux, Steven Latre, et al. City of things: Enabling resource provisioning in smart cities. *IEEE Communications Magazine*, 56(7):177–183, 2018.

[168] Yi Liu, Chao Yang, Li Jiang, Shengli Xie, and Yan Zhang. Intelligent edge computing for iot-based energy management in smart cities. *IEEE Network*, 33(2):111–117, 2019.

[169] Moayad Aloqaily, Safa Otoum, Ismaeel Al Ridhawi, and Yaser Jararweh. An intrusion detection system for connected vehicles in smart cities. *Ad Hoc Networks*, 90:101842, 2019.

[170] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Towards end-to-end resource provisioning in fog computing over low power wide area networks. *Journal of Network and Computer Applications*, page 102915, 2020.

[171] Ana I Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso García, and Davide Scaramuzza. Event-based vision meets deep learning on steering prediction for self-driving cars. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5419–5427, 2018.

[172] Shitao Chen, Yu Chen, Songyi Zhang, and Nanning Zheng. A novel integrated simulation and testing platform for self-driving cars with hardware in the loop. *IEEE Transactions on Intelligent Vehicles*, 4(3):425–436, 2019.

[173] Alesia Chernikova, Alina Oprea, Cristina Nita-Rotaru, and BaekGyu Kim. Are self-driving cars secure? evasion attacks against deep neural networks for steering angle prediction. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 132–137. IEEE, 2019.

[174] Anselme Ndikumana, Nguyen H Tran, Ki Tae Kim, Choong Seon Hong, et al. Deep learning based caching for self-driving cars in multi-access edge computing. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[175] Yanwei Liu, Jinxia Liu, Antonios Argyriou, and Song Ci. Mec-assisted panoramic vr video streaming over millimeter wave mobile networks. *IEEE Transactions on Multimedia*, 21(5):1302–1316, 2018.

[176] Alexandros Doumanoglou, David Griffin, Javier Serrano, Nikolaos Zioulis, Truong Khoa Phan, David Jiménez, Dimitrios Zarpalas, Federico Alvarez, Miguel Rio, and Petros Daras. Quality of experience for 3-d immersive media streaming. *IEEE Transactions on Broadcasting*, 64(2):379–391, 2018.

[177] Jeroen van der Hooft, Maria Torres Vega, Stefano Petrangeli, Tim Wauters, and Filip De Turck. Optimizing adaptive tile-based virtual reality video streaming. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 381–387. IEEE, 2019.

[178] Jeroen van der Hooft, Tim Wauters, Filip De Turck, Christian Timmerer, and Hermann Hellwagner. Towards 6dof http adaptive streaming through point cloud compression. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2405–2413, 2019.

[179] Liang Zhou, Dan Wu, Jianxin Chen, and Zhenjiang Dong. When computation hugs intelligence: Content-aware data processing for industrial iot. *IEEE Internet of Things Journal*, 5(3):1657–1666, 2017.

[180] Jiangfeng Cheng, Weihai Chen, Fei Tao, and Chun-Liang Lin. Industrial iot in 5g environment towards smart manufacturing. *Journal of Industrial Information Integration*, 10:10–19, 2018.

[181] Michele Luvisotto, Federico Tramarin, Lorenzo Vangelista, and Stefano Vitturi. On the use of lorawan for indoor industrial iot applications. *Wireless Communications and Mobile Computing*, 2018, 2018.

[182] Jens Hiller, Martin Henze, Martin Serror, Eric Wagner, Jan Niklas Richter, and Klaus Wehrle. Secure low latency communication for constrained industrial iot scenarios. In *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, pages 614–622. IEEE, 2018.

[183] Helin Yang, Arokiaswami Alphones, Wen-De Zhong, Chen Chen, and Xianzhong Xie. Learning-based energy-efficient resource management by heterogeneous rf/vlc for ultra-reliable low-latency industrial iot networks. *IEEE Transactions on Industrial Informatics*, 16(8):5565–5576, 2019.

[184] Sina Rafati Niya, Eryk Schiller, Ile Cepilov, and Burkhard Stiller. Biit: Standardization of blockchain-based i 2 ot systems in the i4 era. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2020.

[185] Tanesh Kumar, Erkki Harjula, Muneeb Ejaz, Ahsan Manzoor, Pawani Porambage, Ijaz Ahmad, Madhusanka Liyanage, An Braeken, and Mika Ylianttila. Blockedge: Blockchain-edge framework for industrial iot networks. *IEEE Access*, 8:154166–154185, 2020.

[186] Tai-hoon Kim, Carlos Ramos, and Sabah Mohammed. Smart city and iot, 2017.

[187] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius Brito Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago Meireles Paixão, Filipe Mutz, et al. Self-driving cars: A survey. *Expert Systems with Applications*, page 113816, 2020.

[188] Åsa Fast-Berglund, Liang Gong, and Dan Li. Testing and validating extended reality (xr) technologies in manufacturing. *Procedia Manufacturing*, 25:31–38, 2018.

[189] Charith Perera, Chi Harold Liu, Srimal Jayawardena, and Min Chen. A survey on internet of things from industrial market perspective. *IEEE Access*, 2:1660–1679, 2014.

Smart cities empowered by cloud and fog technologies.