Department of Computer Science
Series of Publications A
Report A-2019-6

Edge Computing Platforms and Protocols

Nitinder Mohan

Doctoral dissertation, to be presented for public examination with the permission of the Faculty of Science of the University of Helsinki, in Auditorium A127, Chemicum building, on the 8th of November, 2019 at 12 o'clock noon.

University of Helsinki
Finland

Supervisor
Jussi Kangasharju, University of Helsinki, Finland

Pre-examiners
Dirk Kutscher, University of Emden/Leer, Germany
Satyajayant Misra, New Mexico State University, USA

Opponent
Abhishek Chandra, University of Minnesota, Twin Cities, USA

Custos
Jussi Kangasharju, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Pietari Kalmin katu 5)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://cs.helsinki.fi/
Telephone: +358 2941 911

Copyright © 2019 Nitinder Mohan
ISSN 1238-8645
ISBN 978-951-51-5560-3 (paperback)
ISBN 978-951-51-5561-0 (PDF)
Helsinki 2019
Unigrafia

Edge Computing Platforms and Protocols

Nitinder Mohan

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
nitinder.mohan@helsinki.fi
https://www.nitindermohan.com/

PhD Thesis, Series of Publications A, Report A-2019-6
Helsinki, May 2019, 87+69 pages
ISSN 1238-8645
ISBN 978-951-51-5560-3 (paperback)
ISBN 978-951-51-5561-0 (PDF)

Abstract

Cloud computing has created a radical shift in expanding the reach of application usage and has emerged as a de-facto method to provide low-cost and highly scalable computing services to its users. Existing cloud infrastructure is a composition of large-scale networks of datacenters spread across the globe. These datacenters are carefully installed in isolated locations and are heavily managed by cloud providers to ensure reliable performance to its users. In recent years, novel applications, such as Internet-of-Things, augmented-reality, autonomous vehicles etc., have proliferated the Internet. Majority of such applications are known to be time-critical and enforce strict computational delay requirements for acceptable performance. Traditional cloud offloading techniques are inefficient for handling such applications due to the incorporation of additional network delay encountered while uploading pre-requisite data to distant datacenters. Furthermore, as computations involving such applications often rely on sensor data from multiple sources, simultaneous data upload to the cloud also results in significant congestion in the network.

Edge computing is a new cloud paradigm which aims to bring existing cloud services and utilities near end users. Also termed edge clouds, the central objective behind this upcoming cloud platform is to reduce the network load on the cloud by utilizing compute resources in the vicinity of users and IoT sensors. Dense geographical deployment of edge clouds in an area not only allows for optimal operation of delay-sensitive applications but also provides support for mobility, context awareness and data aggregation in computations. However, the added functionality of edge clouds comes at the cost of incompatibility with existing cloud infrastructure. For example, while data center servers are closely monitored by the cloud providers to ensure reliability and security, edge servers aim to operate in unmanaged publicly-shared environments. Moreover, several edge cloud approaches aim to incorporate crowdsourced compute resources, such as smartphones, desktops, tablets etc., near the location of end users to support stringent latency demands. The resulting infrastructure is an amalgamation of heterogeneous, resource-constrained and unreliable compute-capable devices that aims to replicate cloud-like performance.

This thesis provides a comprehensive collection of novel protocols and platforms for integrating edge computing in the existing cloud infrastructure. At its foundation lies an all-inclusive edge cloud architecture which allows for unification of several co-existing edge cloud approaches in a single logically classified platform. This thesis further addresses several open problems for three core categories of edge computing: hardware, infrastructure and platform. For hardware, this thesis contributes a deployment framework which enables interested cloud providers to effectively identify optimal locations for deploying edge servers in any geographical region. For infrastructure, the thesis proposes several protocols and techniques for efficient task allocation, data management and network utilization in edge clouds with the end-objective of maximizing the operability of the platform as a whole. Finally, the thesis presents a virtualization-dependent platform for application owners to transparently utilize the underlying distributed infrastructure of edge clouds, in conjunction with other co-existing cloud environments, without much management overhead.

Computing Reviews (2012) Categories and Subject Descriptors:

Computer systems organization → Distributed architectures
Networks → Network services
Networks → Network performance evaluation

General Terms:
Design, Experimentation, Performance, Protocols, Platforms

Additional Key Words and Phrases:
Edge computing, Task deployment, Multipath TCP, Virtualization

I dedicate this thesis to my elder sister, Gagandeep Verma, who lost her life at the young age of 15. You will always be missed.

Acknowledgements

My Ph.D. has been an amazing journey full of highs of jubiliation from paper accepts and lows from experiment failures. It was a journey that challenged me to push the boundaries of my knowledge and capabilities constantly. It was a journey which gave me many memories that I will always cherish but would remain incomplete without acknowledging people who made it possible. I was lucky to work with and make friends with several people who helped me become a researcher, and here I would like to thank them all for their endless love, support and understanding.

My deepest heartfelt gratitude goes to my advisor, Jussi Kangasharju. Back in 2015, he put his blind trust in an unknown student whom he had never met before and took him under his wing. Always calm and collected, Jussi instilled in me his unique research mantra – Think outside the lines! His constant feedback, support and guidance showed me what good research is. He also became my mentor outside the walls of the research lab, a person I can always approach for advice. His understanding, compassion and encouragement towards his students taught me several valuable lessons for my future career in research and academia.

Sincere thanks are due to my co-supervisors, Dr. Fabian Schneider and Prof. Xiaoming Fu, with whom I also had the pleasure of doing secondments. Fabian taught me to always consider the practical implications of research problems and the impact of the work outside the scope of potential publications. From Xiaoming I learned the art of communication and networking, and how there is little difference between pursuing a Ph.D. and an MBA, as both require you to advertise your work and remain visible. I will always cherish and imbibe the values I learned while working in their research groups. I am also grateful to the pre-examiners, Prof. Dirk Kutscher and Dr. Satyajayant Misra, for their thorough work to ensure that the thesis is conveyed in its best form. I also convey my deepest gratitude to Prof. Abhishek Chandra for agreeing to be my opponent.

I would like to thank my colleagues in the CoNe group, who made my time spent at the department worthwhile and something I will always treasure: Aleksandr Zavodovski, Pengyuan Zhou, Otto Waltari, Walter Wong and Ossi Karkulahti. My heartfelt gratitude is towards Ashwin Rao, Julien Mineraud and Suzan Bayhan who gave me wonderful advice for my research and personal problems over countless lunches and coffees. To my close friend and collaborator, Tanya Shreedhar, who made me appreciate the theoretical side of networking research, I owe the deepest gratitude. I express my thanks towards members of the CleanSky ITN group, especially David Koll, Mayutan Arimaithurai, Alessio Silvestro and Besmir Tola, who made me look forward to sharing my progress (and break the monotony of research) in periodically organized summer schools and seminars. I am thankful to Prof. K.K. Ramakrishnan for his invaluable guidance and feedback in the formative years of my Ph.D. which provided much-needed illumination for this journey.

I also want to thank the staff members of the Department of Computer Science at the University of Helsinki where I spent the majority of my time during my PhD work. To Pirjo Moen, who offered her invaluable help from my recruitment interview until my thesis submission for every kind of practicality and problem, I owe my sincere gratitude. I extend my thanks to the staff of NEC Laboratories, Germany and Institute of Computer Science, University of Göttinggen, Germany for being kind and excellent hosts.

Finally, I would like to thank my parents and my brother for their love, understanding and much-needed encouragement throughout my PhD. To my father and role model, Dr. H.R. Verma, who instilled in me the love for science and sense of wonderment of the unknown, I will always be indebted. Without the support of my family, I wouldn't be finishing my PhD.

In Helsinki, Finland, October, 2019

Nitinder Mohan

List of Abbreviations

$5G$	5th Generation
ACK	ACKnowledgement
AP	Access Point
API	Application Programming Interface
AR	Augmented Reality
ARM	Advanced RISC Machines
AWS	Amazon Web Services
$BALIA$	BAlanced Linked Increase Algorithm
$BLEST$	BLocking ESTimation scheduler
$BSID$	Base Station ID
$CAPEX$	CAPital EXpenditure
$CAT4$	CATegory 4
CDN	Content Delivery Networks
CPU	Central Processing Unit
$DAPS$	Delay Aware Packet Scheduler
$DASH$	Dynamic Adaptive Streaming over HTTP
DC	Datacenter
DNS	Domain Name Service
ECF	Earliest Completion First
$eLPCF$	energy-efficient Least Processing Cost First
ES	Edge Server
$ExEC$	Extensible Edge Clouds
$FiWi$	Fiber-Wireless
$FPGA$	Field-Programmable Gate Array
$HTTP$	HyperText Transfer Protocol
$IaaS$	Infrastructure as a Service
$ICON$	Intelligent CONtainers
IEP	Independent Edge Providers
IoT	Internet of Things
IP	Internet Protocol

ISP	Internet Service Provider
kmph	Kilometer per hour
LIA	Linked Increase Algorithm
LPCF	Least Processing Cost First
LTE	Long Term Evolution
MACC	Mobile Ad-hoc Clouds
MEC	Multi-Access Edge Clouds
MC	Mobile Clouds
MPTCP	MultiPath TCP
NC	Network Cost
NIC	Network Interface Controller
OPEX	OPerational EXpenditure
OLIA	Opportunistic Linked Increase Algorithm
PaaS	Platform as a Service
PC	Processing Cost
PTR	PoinTeR
QAP	Quadratic Assignment Problem
QAware	QueueAware
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RPi	Raspberry Pi
RRM	Radio Resource Management
RTT	Round Trip Time
SaaS	Software as a Service
SIM	Subscriber Identity Module
SFC	Service Function Chain
SLA	Service Level Agreement
SRTT	Smoothed Round Trip Time
SRV	SeRVice
TEE	Trusted Execution Environment
TCP	Transport Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
VM	Virtual Machine
VR	Virtual Reality
WiFi	Wireless Fidelity

Contents

Chapter 1

Introduction

Cloud computing has created a radical shift in application services and has expanded the reach of computing, networking and data storage to its users. The cloud service providers such as Google, Microsoft, Amazon etc., deploy a network of large datacenters spread across the globe. Such datacenters are home to large pools of highly managed servers, disks and networking switches which are carefully architected to ensure consistent performance [4]. Cloud providers encapsulate datacenter hardware into virtualized and programmable resources, which can be dynamically provisioned and re-configured on-the-go. This configurability is beneficial to application services whose utilization varies from time to time. For such services, the cloud providers offer their resources with an “on-demand” model which allows application providers only to pay for resources they utilize [14]. The key services offered by the cloud providers include hardware resources (Infrastructure as a Service), operating environments (Platform as a Service) and developer softwares (Software as a Service) [127].

Recently, applications such as Internet-of-Things (IoT), connected vehicles, smart cities etc. have proliferated the network. These applications are often dependent on a large number of sensor devices which record information of the surroundings and upload it to cloud for its processing needs. Recent studies suggest that there will be more than 50 billion devices connected to the Internet by 2020 [35, 87]. Such devices will generate a staggering amount of digital data, also known by the buzzword – big data, the size of which has already been increasing exponentially over the past few years [38]. Computations on big data in a remote datacenter is not only inefficient due to bandwidth constraints but also impacts the performance of time-sensitive and location-aware applications due to (i) network delay for offloading data to the cloud, and (ii) computational dependencies between data generated by nearby sensors. Self-driving cars, augmented reality, video surveillance, automated industry and gaming are examples of a few such applications. For example, queries such as *"what is the distribution of velocities of all vehicles within a mile of an intersection?"* requires data from multiple vehicles and traffic cameras and is time-sensitive for effective traffic control. In worst cases, failures to satisfy by the latency constraints posed by such applications can even result in loss of life and capital [47].

To decouple the network delay from the computation time for processing big data, researchers have proposed to bring compute servers closer to data generators and consumers. Several different cloud models have been proposed by academia and industry alike to solve this problem [11, 32, 45, 59, 99, 116, 122]. *Fog cloud* proposes to augment existing network fabric, i.e. switches, routers etc. with compute capability such that it can extend the support of cloud application logic [11]. The objective of the fog cloud is to perform low-latency computation and aggregation on data while it is en route to the centralized cloud. Another approach is *edge compute cloud*, which aims to utilize crowdsourced compute-capable devices, such as smartphones, desktop etc. [45]. As such devices are available in the same environment as that of data generating sensors, they can support the ultra-low latency requirements of IoT applications. Moreover, there already exist several frameworks that incentivizes the owners of such devices to auction the excessive/not-in-use computational power for monetary benefits [139, 143].

Rather than focusing on differences between several such proposals, we recognize the core idea of this research direction, which is *utilizing compute resources closer to data generators/consumers*, and term it as *edge computing*. We believe that despite having the ability to disrupt the current cloud landscape, future edge computing deployments will co-exist with the traditional centralized cloud and the effectiveness of both cloud platform technologies will allow for transparent cooperation and coordination. In this thesis, we emphasize providing practical solutions to several issues surrounding *adoption and functionality of edge computing in the existing cloud-dominant environment.*

1.1 Problem Statement

Centralized cloud infrastructure is designed to provide a generic deployment platform for application owners while they remain oblivious to management complexities of the underlying hardware. Strides in cloud computing are made possible by standardized protocols and technologies which are specifically designed to mitigate the deployment complexity of application owners [40, 42, 94, 95, 131]. However, the same is not valid for edge clouds primarily due to differences in deployment and server characteristics which does not allow for proper operation of such technologies. Even existing approaches which aim to transform heterogeneous clouds into a homogeneous platform [129] do not work well for integrating edge computing due to the following challenges.

1. **Constrained hardware.** While datacenters are composed of servers with significant compute and storage capacity, the majority of edge approaches aim to utilize hardware-constrained devices which restrict the extent of computation capability that can be exploited by an application. Although existing cloud-based protocols are designed to provision multiple applications onto a single server, protocols for edge clouds would require distribution of a single application on multiple edge servers due to resource constraints.

2. **Constrained environment.** Datacenters depend on high-speed, high-bandwidth, reliable network fabric to support the required application QoS. However, despite extensive management and the isolated operational environment of datacenters, it is commonplace that cloud providers struggle to mitigate network congestion due to the increased traffic load within a datacenter facility [44]. On the other hand, the majority of edge cloud models require servers to operate within public infrastructure, utilize loss-prone wireless networks such as LTE, WiFi etc. and compete with unrelated user traffic for its share of the spectrum.

3. **Availability and Reliability.** While existing cloud protocols are designed to be resilient to operational failures and faults, such schemes depend on the exact knowledge of the auxiliary hardware capacity and existing load on each server in the facility for this functionality [28, 138]. On the other hand, edge cloud aims to incorporate several different types of resources in its platform, ranging from network switches to smartphones, distributed in a geographical area. This exposes several problems for existing cloud-based protocols. (i) The availability of edge servers can be relatively unknown and vary from region to region, e.g. the density of crowdsourced devices can be higher in populated areas like city centers. (ii) The majority of such compute resources are not reliable as they have the capability to leave the system at any time per owner's discretion.

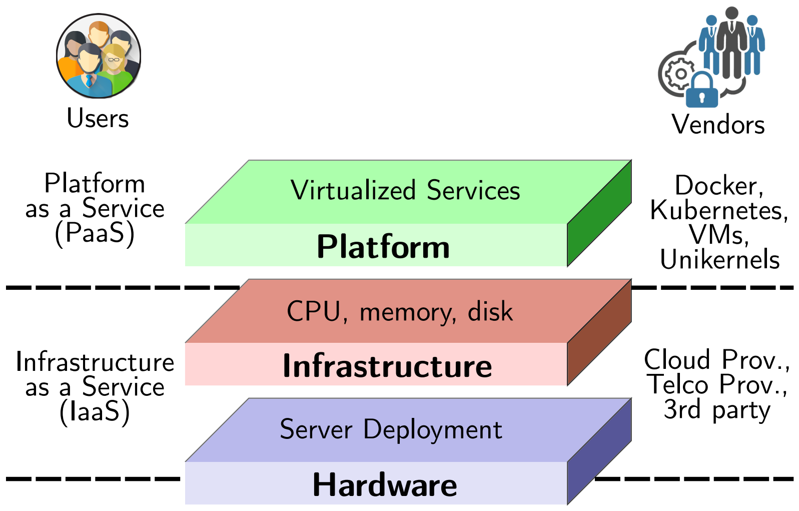

Users
Vendors
Platform as a Service (PaaS)
Virtualized Services
Platform
Docker, Kubernetes, VMs, Unikernels
CPU, memory, disk
Infrastructure
Infrastructure as a Service (IaaS)
Cloud Prov., Telco Prov., 3rd party
Server Deployment
Hardware

Figure 1.1: Edge computing service architecture.

4. **Limited battery capacity.** Several proposed edge models employ servers that operate in mobile environments to support applications of that type. Such devices need to rely on their limited battery capacity to complete assigned computations, which is in stark contrast to server racks in a traditional data center. Limited power availability impacts the extent of server's task execution, which needs to be considered while designing protocols for edge computing.

The challenges mentioned above highlight key issues obstructing the effective operation of edge clouds, and require immediate resolutions. While one approach is to handle each challenge in its entirety, we argue that the issues presented are highly intertwined and require a systematic ground-up redressal. Furthermore, the resulting solutions need to be compatible with traditional cloud computing technologies such that pre-existing cloud-based applications can be ported to edge environment with minimal overhead.

1.2 Edge Computing Service Model

Acknowledging the challenges above, we present **edge computing service model**, illustrated in Figure 1.1, which is a hierarchical organization of different areas of research opportunities for successfully integrating edge and cloud environments. The model is largely derivered from cloud computing service architecture [30]. The primary aim of edge computing is to act as an extension to the current cloud infrastructure; to the point that existing cloud applications can migrate to the edge without *any* modifications. To this purpose, the model focuses on solutions for all but the last layer of the cloud service architecture, i.e. *hardware*, *infrastructure* and *platform*.

Hardware

The foremost research direction which requires exploration concerns physical deployment of edge servers in a region. Although the edge computing needs differ for different applications, the strictest requirements necessitate extensive availability of compute servers close to the data generators (read sensors) and consumers (read users). Some applications such as IoT, surveillance cameras etc. require edge servers as aggregation points somewhere in the network. Other applications such as AR/VR, autonomous vehicles etc. demand direct one-hope connectivity to the compute server for optimal operation. To satisfy the variability in requirements and yet support the majority of applications, the need for extensive installation of servers in a public space increases significantly. This scenario is in stark contrast to the siloed deployment of datacenters, which are protected extensively from human habitat and constantly monitored by cloud providers to ensure service reliability, availability and security. The following research questions address the key challenges of this layer.

RQ1. *Where should the cloud providers install compute servers in the physical world to satisfy the application requirements at the "edge"?*

RQ2. *How can independent entities enroll their compute resources in an existing edge cloud platform?*

RQ1 and RQ2 explore the optimal placement of edge servers in both physical and logical sense. As the edge cloud aims to operate in a semi/unmanaged environment, RQ1 targets cloud providers which scan for optimal deployment locations offering some level of control. Moreover, cloud providers also have the option of integrating compute resources from independent entities if they satisfy the required terms of service. Although the aforementioned allows for the proliferation of servers at the network edge, it imposes several management issues for cloud providers as they need to offer performance reliability over servers outside their administrative control.

Infrastructure

Heterogeneity in availability, configurations, reliability, capability and ownership of servers within the edge cloud platform calls for an aggressive rethinking of traditional cloud solutions. Existing protocols for managing cloud infrastructure depend on complete knowledge of underlying hardware capability and capacity, which needs to be constantly monitored by the provider. While servers in a datacenter are physically arranged in racks and interconnected by fast, high-bandwidth network switches, the same cannot be promised for edge cloud resources. As the edge servers operate in public infrastructure, a similar granularity of control cannot be imposed on them. Furthermore, the variance in edge server capability and capacity far exceeds that of datacenters due to variability in edge deployment. Such restrictions imposed by edge computing calls for an effective redesign of existing cloud solutions as summarized in the following research questions.

RQ3. *How do we utilize availability and variability of edge servers for computing application tasks with different requirements?*

RQ4. *What techniques should be employed for pre-caching computational data within edge servers to improve system promptness?*

RQ5. *Can existing network technologies available at the edge support the requirements imposed by end-applications for optimal performance?*

RQ6. *How do we assure datacenter-like network behavior over edge servers which operate on a public wireless network?*

RQ3 reflects the need to improve application task allocation on available edge servers in the vicinity of data consumers. RQ4 considers the availability and management of pre-requisite data in edge servers for a particular task to achieve minimal processing delay. RQ5 and RQ6 aim at providing reliable networking for edge servers which utilize access technologies over publicly shared mediums. Specifically, RQ5 explores the possibility to simultaneously combine multiple network interfaces at the edge server using Multipath TCP (MPTCP) to simulate high-bandwidth connectivity of datacenter networks. On the other hand, RQ6 suggests improvements to MPTCP design such that it can support data transfer requirements for edge computing.

Platform

A common practice of deploying application services is to encapsulate it as a virtual machine or a container which can then be executed in any cloud environment. This process is known as virtualization and it allows the application service to easily scale with increasing utilization demand. In order to integrate edge computing in existing cloud infrastructure, there is a strong need for solutions which enables standardized virtualization technologies to function properly at the edge. In addition to ease-of-access for application providers, use of virtualization in edge clouds can assist in dynamic resource provisioning based on the density of user requests, migration to locations closest to the source of requests and security. The research questions prevalent in this layer are as follows.

RQ7. *How can existing cloud virtualization technologies be exploited to optimize the application service deployment in edge clouds?*

RQ8. *How can edge cloud ensure the promised Quality-of-Service despite significant variability in user requests and infrastructure hardware?*

RQ9. *How can independent edge providers generate revenue at par with cloud providers for their service?*

RQ7 focuses on understanding how virtualization technologies can cope in edge clouds where the underlying resources may/may not be available throughout the application operation. RQ8 explores possible solutions, using which virtualization technologies can operate in unreliable environments while ensuring the promised QoS to applications. RQ9 delivers information regarding generating revenue for independent owners of edge resources for the application services hosted on their servers. Together, RQ7 – RQ9 look for platform solutions which allow incorporation of the datacenter cloud and edge cloud infrastructure without the need for pre-signed contractual agreements between the two providers. Answers to these research questions would allow cloud providers to discover edge servers in previously unknown locations near the source of incoming requests and utilize in accordance with service demands.

Figure 1.2 presents an infographic of how the research questions mentioned above are covered in the publications attached to this thesis and form the core components of the edge computing service model.

1.3 Thesis Contributions

This thesis identifies critical focus areas for cloud providers, as edge cloud adoption model, which aims to deploy and utilize the edge computing infrastructure. The thesis explores existing problems in all layers of the adoption model and provides technical solutions for each one of them. The key contributions are as follows:

- We propose Edge-Fog cloud architecture which acts as a blueprint for organizing future deployment of edge computing infrastructure.

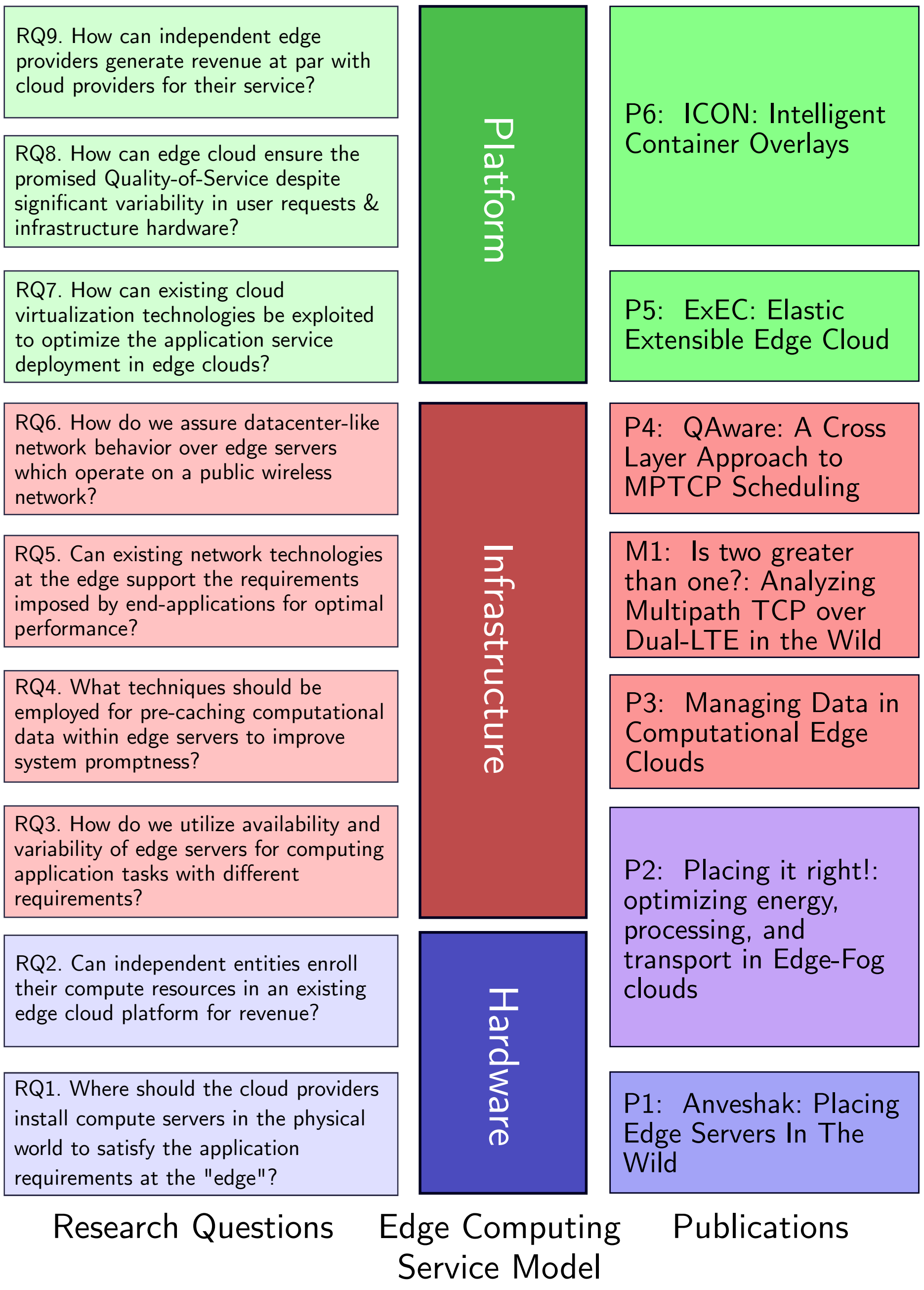

RQ9. How can independent edge providers generate revenue at par with cloud providers for their service?
RQ8. How can edge cloud ensure the promised Quality-of-Service despite significant variability in user requests & infrastructure hardware?
RQ7. How can existing cloud virtualization technologies be exploited to optimize the application service deployment in edge clouds?
RQ6. How do we assure datacenter-like network behavior over edge servers which operate on a public wireless network?
RQ5. Can existing network technologies at the edge support the requirements imposed by end-applications for optimal performance?
RQ4. What techniques should be employed for pre-caching computational data within edge servers to improve system promptness?
RQ3. How do we utilize availability and variability of edge servers for computing application tasks with different requirements?
RQ2. Can independent entities enroll their compute resources in an existing edge cloud platform for revenue?
RQ1. Where should the cloud providers install compute servers in the physical world to satisfy the application requirements at the "edge"?
Platform
Infrastructure
Hardware
P6: ICON: Intelligent Container Overlays
P5: ExEC: Elastic Extensible Edge Cloud
P4: QAware: A Cross Layer Approach to MPTCP Scheduling
M1: Is two greater than one?: Analyzing Multipath TCP over Dual-LTE in the Wild
P3: Managing Data in Computational Edge Clouds
P2: Placing it right!: optimizing energy, processing, and transport in Edge-Fog clouds
P1: Anveshak: Placing Edge Servers In The Wild
Research Questions
Edge Computing Service Model
Publications

Figure 1.2: Research questions and their matching publications along with associated layers of the edge computing service model.

The model builds on the requirements of edge-based applications and segregates edge resources into distinct layers depending on their capability, ownership and functionality. The thesis also proposes a novel deployment framework, Anveshak, which assists edge service providers to efficiently identify prime locations in a geographical area that would benefit from installing an edge server.

- For enabling the reliability and availability of edge infrastructure, this thesis provides protocols which improve processing task allocation, manages caching of compute data and enhances the network capability of wireless interfaces in edge servers. Our infrastructure solutions are based on the requisite needs of applications which plan to function at the edge and present full-fledged support for efficient edge cloud performance.

- This thesis enables effective functioning of virtualized applications in edge computing environments. It includes solutions which enable dynamic discovery of edge providers in previously unknown locations that can be exploited to maintain optimal application QoS. We also provide a novel self-management platform solution which allows containerized services to adapt to environmental and user request changes, and self-migrate to new locations satisfying the targets for optimal service delivery.

The research reported in this thesis encompasses the work published in six original articles and a manuscript under submission. These articles and manuscripts also construct the outline of the thesis and an emphasis is drawn towards the work in which the author has contributed himself.

Publication I: Anveshak: Placing Edge Servers In The Wild. Nitinder Mohan, Aleksandr Zavodovski, Pengyuan Zhou, and Jussi Kangasharju. Published in Proceedings of the ACM Workshop on Mobile Edge Communications (MECOMM '18). pages 7-12. Budapest, Hungary, August 20, 2018.

Contribution: *The publication was led by the author who formulated the problem and methodology and designed the solution algorithm. The author also implemented the solution and procured the required data for evaluation. Aleksandr Zavodovski and Pengyuan Zhou contributed significantly to data analysis and solution implementation. The author, along with others, participated in the writing process.*

Publication II: Placing it right!: optimizing energy, processing, and transport in Edge-Fog clouds. Nitinder Mohan and Jussi Kangasharju. Published in Annals of Telecommunications, Springer Nature, Volume 73, 2018, pages 463-474.

Contribution: *This journal paper is an extension of the work published in [79]. The author was in the lead of planning and studying the literature survey and requirements for the work, designing the closing architecture, analyzing and providing the solution to associated placement problem. Prof. Jussi Kangasharju contributed with sharing of ideas and active discussion throughout problem formulation. The author and Prof. Jussi Kangasharju were involved in the writing process of the publication.*

Publication III: Managing Data in Computational Edge Clouds. Nitinder Mohan, Pengyuan Zhou, Keerthana Govindaraj, and Jussi Kangasharju. Published in Proceedings of the ACM Workshop on Mobile Edge Communications (MECOMM '17). pages 19-24. Los Angeles, CA, USA, August 21, 2017.

Contribution: *The author was in the lead of the problem formulation, algorithm design, evaluation and writing of final publication. Pengyuan Zhou contributed to the implementation and simulation setup. Keerthana Govindaraj provided critical insights regarding related work and scope driven by her personal industrial experience and also contributed to improving the writing and paper structure. Prof. Jussi Kangasharju was involved in the planning, discussion and writing process of the article.*

Publication IV: QAware: A Cross-Layer Approach to MPTCP Scheduling. Tanya Shreedhar, Nitinder Mohan, Sanjit K. Kaul and Jussi Kangasharju. Published in Proceedings of IFIP Networking Conference (IFIP Networking '18), pages 190-198. Zurich, Switzerland, May 14-16, 2018.

Contribution: *The publication was led by Tanya Shreedhar who formulated the problem, delivered main ideas, algorithm, evaluation using simulations and defined the structure of the publication under the supervision of Dr. Sanjit K. Kaul. The author's contributions were focused towards solution design for implementation – which were heavily driven by real-world use cases and constraints. The author was also responsible for implementing the solution for Linux and evaluating over real systems and applications. The author also contributed to writing and suggested significant improvements in the publication.*

Publication V: ExEC: Elastic Extensible Edge Cloud. Aleksandr Zavodovski, Nitinder Mohan, Suzan Bayhan, Walter Wong and Jussi Kangasharju. Published in ACM Workshop on Edge Systems, Analytics and Networking (EdgeSys '19), pages 24-29, March 25, 2019.

Contribution: *The research for this publication was led by Aleksandr Zavodovski who designed the main ideas, methodology and implementation. The author contributed by participating in the problem formulation, design of the solution's workflow/algorithm and provided significant writing improvements to the publication. Dr. Walter Wong was responsible for data collection and cleaning necessary for evaluation and Dr. Suzan Bayhan contributed to the literature survey for the publication. Dr. Jussi Kangasharju provided his insights and supervision at all stages and participated in the writing process.*

Publication VI: ICON: Intelligent Container Overlays. Aleksandr Zavodovski, Nitinder Mohan, Suzan Bayhan, Walter Wong and Jussi Kangasharju. Published in ACM Workshop on Hot Topics in Networks (HotNets '18), pages 463-474, November 15-16, 2018.

Contribution: *The publication was led by Aleksandr Zavodovski who provided the key ideas, formed the problem scope and designed the solution's workflow. The author contributed to the technical design of the solution, including the architecture of the central algorithm, ensuring adaptability to existing protocols and correctness in real-world systems. The author also contributed to the writing process and provided comments for improving the quality of the publication. Dr. Suzan Bayhan designed the optimization algorithm which was critical to the solution and Dr. Walter Wong collected data necessary for evaluation. Dr. Jussi Kangasharju provided his supervision to the design, scope and writing phases of the work.*

Manuscript I: Is two greater than one?: Analyzing Multipath TCP over Dual-LTE in the Wild. Nitinder Mohan, Tanya Shreedhar, Aleksandr Zavodovski, Jussi Kangasharju and Sanjit K. Kaul. Manuscript available at `arxiv.org/abs/1909.02601`.

Contribution: *The manuscript has a strong focus on measurements and data analysis in real systems. The author held primary responsibility for designing the measurement study, setup and methodology. The author also implemented necessary scripts for data collection and carried out in-depth analysis. Tanya Shreedhar played an instrumental role in measurement study design, literature survey, analysis/discussion and writing of the manuscript. Aleksandr Zavodovski contributed by participating in measurement setup, data collection and writing. Prof. Jussi Kangasharju and Dr. Sanjit K. Kaul provided essential insights throughout the study and were involved in the writing process.*

1.4 Thesis Organization

The thesis is organized as follows. Chapter 2 provides the necessary background knowledge to the thesis and discusses the state of the art in cloud technologies with a focus on edge computing approaches and challenges. Rest of the thesis follows the components of edge cloud service model discussed in Section 1.2. Chapter 3 presents *Edge-Fog Cloud* architecture which forms the foundation for organizing edge computing resources in a physical space. Further, we provide a deployment framework catered towards assisting existing cloud providers in identifying optimal locations for installing edge servers in a region. Chapter 4 focuses on effectively utilizing the available edge servers in an area for compute tasks and provide data management that impacts overall task completion time. We also concentrate on enhancing the network capability of edge servers by simultaneously utilizing multiple network interfaces via Multipath TCP. Chapter 5 details platform solutions which enable virtualized applications to utilize the underlying edge cloud infrastructure while maintaining the required QoS. Finally, Chapter 6 concludes the thesis with a summary of solutions and outlooks for future work.

Chapter 2

Rise of Edge Computing

This chapter gives necessary background on edge computing and the associated challenge to redesign traditional datacenter-based cloud infrastructure. We start with a brief understanding of the components and design of traditional datacenters and extensive care undertaken by the provider to ensure consistent optimal performance. Further, we discuss the challenges posed by emerging applications which call for a radical shift in cloud computing infrastructure. The chapter includes a discussion on several edge cloud architectures proposed by researchers in the past and their design choices which make them suitable for specific application operation. We end this chapter with several related state-of-the-art solutions that aim to enable utilizing edge clouds in a real-world environment.

2.1 Overview of Datacenters

While the requirements and features of *datacenter* (DC) appeared as early as the first computer operation in 1960s, the term was standardized by Telecommunications Infrastructure Standard for Datacenters (i.e., ANSI/ TIA-942-2005) in 2005 as "a building which houses multiple servers, communication equipment and other support areas" [8]. Since the standardization of its definition, the extent of DC deployment and business entities involved in providing compute services to users has seen significant growth. The advent of DC-based computing has brought a radical shift in the reliability and performance of application services. Almost all major tech giants, such as Microsoft, Amazon, Google, Dell, HP, Apple, IBM, etc., have set up DCs globally to ensure the premium performance of their software products. While the number of DCs have been growing consistently, with over 7500 datacenters operating worldwide [22], the size of the facilities has also increased to satisfy the growing demand in cloud services. It is estimated that the top 10 largest DC facilities cover an area from 750,000 to 6.3 million square feet and house more than 3000 server racks [108].

The core physical component of a datacenter is a server, which is responsible for processing, analyzing and communicating with servers within (and across) DC facilities. While each server has inbuilt storage to support local computations, the cloud providers also install network-attached storage in the same facility which provides high-speed storage access to all servers in the DC via fiber connections. Multiple servers are organized in stacks, also known as racks, for effective space management. Servers in a single rack are interconnected via high-bandwidth, low latency ethernet cables (up to 100Gbps [67]) to a highly capable top-of-rack (ToR) switch. The ToR switch allows servers within the same rack to communicate with the lowest cost of connectivity.

The cloud provider physically arranges server racks in the facility in logically organized architectures to minimize the network traffic load within the DC. The traditional approach resembles a multi-rooted tree which aggregates the network load as you move towards the root (also known as the core) [66]. The server racks reside at the leaf of the tree and are interconnected via high-speed aggregation switches. The root (also known as core) connects the DC with other DC facilities deployed across the globe via an extensively managed network fiber controlled by the cloud provider. This allows cloud traffic to travel inter-continental distances without any scope of congestion from public traffic via completely separate dedicated network links. Such extensive cost of deployment and management is undertaken to support the strictest reliability and availability requirements imposed by application owners. Several other architectures have been proposed and widely deployed to further reduce the networking and routing latencies for ever-increasing DC loads. A few examples are Fat-tree [5], Diamond [124], DCell [51], BCube [50] etc.

2.2 Emerging Application Requirements

Although traditional clouds present themselves as a repository of highly compute capable servers, the demands of the emerging applications over the internet sway away from features that typical DCs strive to provide. Internet-of-Things (IoT) is one such use-case. IoT devices and sensors have already proliferated the network and their deployment follows an exponential rising trend. More than 75 *billion* of such devices are predicted to be connected over the Internet by 2025, resulting in an average of 7 sensors per person [123]. The trend is assisted by a parallel increase in the market share of IoT-based applications, which is expected to generate 11 *trillion* US dollars worth of revenue by 2025 [125]. However, the majority of IoT (and similar) applications require latency-critical processing which cannot be supported by traditional clouds due to excessive network latency and bandwidth congestion between IoT sensor and DC. We now discuss the requirements of several such applications.

1. **Industry Automation**. The demand to interconnect factories and automate entities on the factory floor without human involvement has given rise to *Industry 4.0* [69]. The application requirements in this area are two-fold. (i) Provide shorter development periods by allowing connected machines to interact and share tasks and (ii) incorporate flexibility in operation such that task recalibration can be performed on-the-fly. Manufacturers have designed sophisticated robots, such as Bosch APAS [109], to work in tandem with human workers on the factory floor. Such machines generate a multitude of data from attached sensors and cameras (exceeding 100,000 GB a day) with processing deadlines faster than human reaction time [73]. The service requirements imposed are strict, and breaches have the potential to affect human safety and derail the entire production process.

2. **Autonomous Vehicles**. Automated driving is becoming a reality due to the last three decades of active efforts from academia and industry alike. The key motivator is not just to reduce road accidents due to human errors (which attributes to 94% of all reported accidents), but also to reduce human resource expenditure in the transport sector due to ineffective fuel usage, parking spot utilization etc. Similar to factory automation, connected vehicles rely on many embedded sensors and cameras which generate data exceeding 100GB per vehicle. Applications in this area require efficient data processing such that the resulting decisions are quick and correct. Failures to comply with the latency deadlines can result in loss of property, capital and (in worst cases) life [47].

3. **Gaming**. The competitive gaming industry has shown rapid growth in recent years and has attracted significant investment from major sporting brands in tournaments and game development [89]. Hardware manufacturers continue to find ways to reduce on-device delays of monitors and peripherals, such that it does not exceed the human perception time ($<$ 10ms). However, the processing limitations of gaming consoles have given rise to cloud gaming, perpetuated by

industries such as Google [48], Microsoft [78] etc. While cloud gaming supports the processing requirements due to high availability of capable GPUs, it adds a significant delay in control due to network latency for accessing the remote DC.

4. **Health**. Effective and responsive healthcare is one of the dominant applications in the IoT domain. In 2015, the IoT healthcare market was valued at 24.2 *billion* US dollars and was projected to grow by a factor of 14 to reach 337 *billion* US dollars by 2025 [111]. Apart from providing remote patient assistance and health monitoring (such as stroke mitigation [13], ECG monitoring [46] etc.), remote surgery performed from far-off regions is one of several growing needs of the current healthcare market. Such scenarios not only require high network bandwidth and uninterrupted access but also necessitates ultra-low processing latencies for optimal control.

Smart cities, traffic control, augmented reality, drone-based delivery etc. are a few more examples of use cases that require latency-critical processing which cannot be supported by traditional DC-based cloud.

2.3 Edge Computing Approaches

Considering the motivations posed by the applications mentioned above, researchers have proposed a number of edge computing architectures. While the proposals differ in resource management and configuration, all of them aim to satisfy the following key requirements: (i) low latency support for delay-sensitive application services, (ii) data aggregation support for combining readings from multiple IoT sensors to reduce bandwidth usage, and (iii) support for context-aware location-dependent services. Table 2.1 provides a summary of edge computing architectures and key features of their design which we now briefly discuss below.

Cloudlets. Cloudlets are a collection of resource capable servers deployed near end users within the managed infrastructure offered by providers such as Nokia, Elisa, Deutsche Telekom etc. Originally described by its authors as “datacenter-in-a-box”, cloudlets assume a strong connection to the cloud and acts as its extension [116]. Cloudlets are designed and managed with the intention to support cloud virtualization technologies/protocols and planned to be deployed by an existing cloud provider. The core advantage of a cloudlet lies in its smaller deployment footprint which allows it to significantly reduce the user access latency to a cloud-hosted service. Armed

Architecture	Year	Key Feature	Operator/Organization
Cloudlets [116]	2009	Extension of cloud as micro datacenters; Managed environment and operation	Cloud Providers
Fog Cloud [11]	2012	Collection of managed servers deployed alongside network devices within cellular infrastructure; Ethernet connectivity for intra-fog and cellular access for users	Cloud & Cellular Providers
Mobile Cloud Computing [32]	2001	Crowdsourced mobile devices and servers deployed at edge of cellular infrastructure e.g. basestations	Cloud, Cellular Providers & Users
Edge Cloud [45]	2015	Crowdsourced user-managed resources (not limited to mobile devices); WiFi and cellular connection	Cellular Providers, Users & Self-organized
Mobile Ad-hoc Cloud [59]	2001	Temporary dynamic network of mobile devices in the vicinity; WiFi and cellular connectivity	Users, Self-organized
Mist Cloud [99]	2015	Computation on IoT devices; inter-connected via WiFi, Bluetooth and ZigBee	Independent Providers & Self-organized

Table 2.1: Edge computing architecture along with their key features and operators; *orgaznized in decreasing distance from users.*

with high processing and network capacity, cloudlets are best fit for applications which require low-latency yet resource-intensive computing, e.g. real-time video surveillance, IoT big data analytics, etc.

Fog Clouds. Fog cloud is a virtualized platform for managed compute resources that are colocated with devices deployed within the access network, e.g. routers, switches, access points etc. [11]. Unlike cloudlets, fog clouds are managed and deployed primarily by cellular providers operating in the region (in partnership with cloud providers) and aim to integrate wireless technologies for user/sensor access e.g. WiFi, LTE, 5G etc. The principal objective of fog is to provide in-network computation on data as it moves towards centralized clouds for extensive processing. As the hardware for fog requires less physical space, it can be deployed much closer to the users and IoT sensors without significant management overhead. Their deployment design make fog clouds best suited towards smart city and real-time video surveillance applications.

Mobile Cloud Computing. Also referred to as Mobile Clouds (MC) in the research community [32], Mobile Cloud Computing represents a broader platform where specialized compute servers deployed at the edge of the cellular network, e.g. basestations, work in conjunction with a centralized cloud to support applications on mobile devices. Unlike fog, mobile clouds essentially rely on cellular access technologies such as LTE, 5G, for user access. Rather than extending the cloud to the edge, the primary objective of mobile clouds is to provide a one-hop computation offload facility for mobile subscribers. This allows underlying mobile devices to support application services that require computation capacity which far exceeds what mobile devices can offer on-board. That being said, the mobile clouds aim to support cloud virtualization technologies which allow them to integrate with centralized clouds if necessary.

Edge Clouds. Edge clouds is a consolidation of human-operated, voluntary crowdsourced resources with varying levels of compute capacity [45]. Also referred to as Mobile Edge Clouds (MEC) [114], edge clouds extend mobile cloud computing to non-mobile resources. A few examples of edge resources are desktop PCs, tablets and nano-datacenters such as cloudlets set up by independent providers. As edge clouds incorporate a wide variety of compute capable resources and can also ensure low latencies due to their physical proximity to end users and sensors, the model is often considered to be the representative architecture for edge computing. Unlike fog, edge resources are limited by their network capability as they primarily rely on public shared wireless connectivity for interaction with users and cloud.

Mobile Ad-hoc Clouds (MACC). Mobile ad-hoc cloud represents a temporary collection of mobile devices in the vicinity, owned and operated by end users, which form a dynamic network topology to share compute tasks [59]. Composed of limited resource capacity of mobile hardware, MACC *does not* offer any virtualization capability which would allow it to work in conjunction with a centralized cloud. The key advantage of MACC lies in its proximity to users and sensors (often within a single wireless hop) which allows it to support the ultra-low latencies required for applications such as group live video streaming, unmanned drone control etc. However, MACC fails to offer reliability and availability guarantees to application services due to its inherent "ad-hoc" nature.

Mist Clouds. Mist clouds propose a platform where the computation is dispersed to the "extreme edge" and is handled by IoT sensors themselves [99]. The sensor, incorporated as mist resource, uses its limited compute and storage capacity to perform contextual computations locally. Prime examples of mist resources are smart watches, smart fridge, smart speakers etc. Although like MACC, mist forms local network interconnections with other resources in the vicinity via wireless interfaces such as ZigBee, Bluetooth etc., the resulting computations on the platform are not performed ad-hoc, as mist also has consistent connectivity to a centralized cloud for sophisticated computations.

2.4 Edge Computing Services

The sheer number of edge architectures which have been proposed by researchers in the recent past allows for comprehensive support towards all *edge-dependent* applications. However, the management, operation and function of these architectures differ significantly from each other. This has resulted in a unique problem in edge computing research where there is a siloed development of protocols and platforms for specific edge cloud architectures. Such protocols make several assumptions for their operation, which are valid only for their target deployment architecture and fail to function when ported on other variants. The difference in opinions regarding the "correct" reference architecture is not just limited to research but exist even in standardization bodies, such as OpenFog [62] and OpenEdge [24]. This motivates us to envision a future where widely different edge cloud flavors co-exist in the same environment. In this section, we briefly discuss several state-of-the-art edge computing protocols and their limitations. The solutions discussed lie in the scope of the challenges tackled in this thesis.

2.4.1 Task Deployment

Efficiently utilizing the processing capabilities of compute resources (of any edge computing architecture) takes priority over any other protocol requirement. As edge clouds were designed to be an extension of traditional clouds, several researchers assume that edge resources will support execution of virtualized applications such as virtual machines, containers etc. Significant work has been conducted to solve the VM placement problem in the context of edge computing aiming to minimize the amount of data traffic sent to the central cloud for processing [70, 135, 145]. However, such approaches differ significantly due to their secondary objectives and methodology. For example, Ahvar et al. [2] minimize the network cost of connectivity between servers hosting the VM in their computations while Silva et al. [121] model VM deployment as a bin-packing problem and propose greedy heuristics for a near-optimal solution. While such approaches are successful in architectures like cloudlets and fog, their applicability is limited in MACC and mist clouds due to their lack of support for virtualization technologies.

2.4.2 Storage

Utilizing resources at the network edge for storage is central to the deployment of Content Delivery Networks (CDN) [3]. Extensive research has been conducted in the past for perfecting the design of CDN models which aim to disseminate content to end users via distributed servers and edge cache hierarchies [98,146]. The exploitation of in-network caching to enable efficient content distribution also serves as a motivation behind information-centric networking (ICN) research [144]. Yuan et al. [136] study cache placement in MEC to satisfy the requirements of autonomous vehicles in the vicinity. Researchers in [75] provide a probabilistic model for edge caches to improve the delivery of 360° video streams.

However, the primary focus of such solutions is to support effective content retrieval, which departs significantly from *computational data* storage. The task allocation models, discussed in the previous section, assume that the required data for computation is already available in the local caches of assigned edge resources. However, this assumption does not hold water for edge computing architectures incorporating resources with limited cache capacity (read MACC, mist etc.). Such resources would need to fetch the required data from a distant cloud upon every task allocation, thereby delaying the task execution. To the best of our knowledge, we are unaware of any existing work which considers effective caching of prerequisite computational data in edge clouds.

2.4.3 Application Placement

Unlike the task deployment problem, which focuses on processing a specific job on a group of edge servers, edge applications require parallel computation of multiple "services" which can be distributed independently but operate dependently. An example of this can be a streaming application such as YouTube, which may be composed of multiple services such as parental control, video optimizer and a firewall. Existing datacenter-based placement solutions [132] are not influenced by the mobility of end users and changes in subscriber locations due to their distance from the edge. However, deploying applications on edge clouds require dynamic placement approaches and a server discovery mechanism to utilize resources of different edge architectures coexisting in the same region. This would allow application services to enjoy the least possible access latency along with flexible migration to satisfy highly dynamic user request crowds.

The relevant works concerning discovery of edge servers in a network are as follows. Bhardwaj et al. [9] propose an edge discovery protocol as a backend service that hosts a directory of available devices. Varghese et al. [126] present the concept of an EaaS (Edge-as-a-Service) platform, offering a discovery protocol. The downside of both approaches is that they rely on a central directory where the information regarding each server in the network must be stored. For incorporating edge architectures that are based on the ad-hoc availability of devices [59, 99], an open discovery solution is required which is not tied to the pre-known registry of servers in the network. On the other hand, despite several proposals which envision a high availability of edge servers of varying capacities and capabilities, there is a significant shortage of platform solutions that expose the availability and configuration of such servers to application owners for usage. Platform approaches such as [97, 133] are focused on supporting a niche set of resources that operate in a well-managed network environment.

Chapter 3

Hardware Solutions for Edge

The previous chapter discusses the multitude of edge computing models which have been proposed in recent years to handle *specific* task requirements for *specific* applications. In the majority of research proposals we observe a common trend, i.e. utilizing crowdsourced compute resources (smartphones, tablets, smart speakers etc.) available in a geographical region. It has been estimated that billions of such devices will be utilized in the near future as edge servers [18]. This is further substantiated by the increasing processing capability of such devices due to increasing density of transistors in an integrated circuit as predicted by Gordon Moore [92]. Past research has shown that not only is a collection of such devices able to imitate server-like performance, it also results in a significant reduction in energy usage due to efficient power management algorithms employed by mobile CPUs [93].

The variety of resource configurations and capabilities incorporated by various edge cloud approaches presents several hardware challenges that require immediate resolution. *Firstly*, without any standardized model to guide edge cloud deployment, we can envision a future where multiple edge computing architectures operated by different entities coexist in a fragmented sense in a geographical region. Furthermore, not only will there not exist any coordination and cooperation between such co-existing deployments, they will also compete with each other for the larger market share. *Secondly*, dense availability of crowdsourced compute resources in an area will impact the revenue of cloud providers that have deployed their edge servers in the same location; thereby lessening their motivation for deploying managed servers. Still in its nascent stages, both challenges have the potential to affect the real-world adoption of edge computing.

This chapter provides the following solutions for edge cloud deployment.

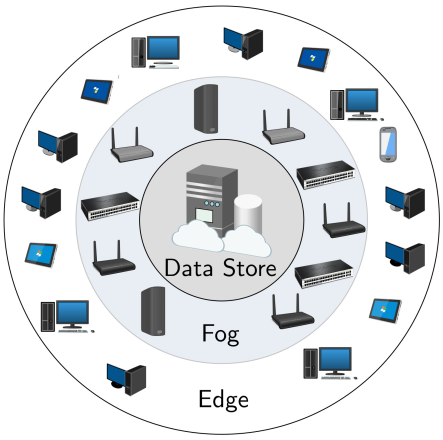


Figure 3.1: Layered Abstraction of *Edge-Fog Cloud*.

I. We present *Edge-Fog Cloud* which is an all-inclusive, node-oriented, edge cloud model. The model does not adhere to design restrictions imposed by specific application scenarios and presents an opportunity for unifying multiple edge cloud approaches. Edge-Fog Cloud was initially published in [79] which was later extended as a journal publication (attached as Publication II along with this thesis [80]).

II. For cloud providers aiming to deploy managed edge servers in any geographical region, we develop *Anveshak*. Anveshak is a deployment framework that discovers the optimal locations for installing edge servers. The framework prioritizes regions with lower installation costs and maximum investment returns to providers. Anveshak predicts future density of crowdsourced edge servers throughout the region along with the utilization of the installed server in its decisions. The framework was published in Publication I attached to this thesis [83].

3.1 Edge-Fog Cloud

Edge-Fog Cloud is a loosely-coupled edge cloud architecture which due to its unique edge-centric design, supports the majority of application types with varying QoS requirements. Figure 3.1 shows the resource-oriented view of *Edge-Fog Cloud*.

While in reality, edge clouds are a consolidation of heterogeneous resources with different storage, compute and network capacities distributed in a physical region, *Edge-Fog Cloud* logically segregates the resources in *three* abstract layers, *Edge*, *Fog* and *Data Store*. The categorization of compute resources into different layers of *Edge-Fog Cloud* is based on their hardware capacity, ownership and network latency from the end user. It must be noted that our segregation criteria *does not* include the exact deployment environment or location of the edge server. This allows us to easily integrate different edge architectures (discussed in Section 2.3 of Chapter 2) managed by different entities with different goals (independent providers, telco operators, cloud providers etc.) as a single logical platform.

Edge. The *Edge*[1] layer is composed of loosely-coupled compute resources with one-to-two hop network latency from the end users. Because of their close proximity to the user, *Edge* servers rely on direct wireless connectivity (e.g. short-range WiFi, LTE, 5G, Bluetooth etc.) for communication and coordination with other *Edge* servers and users alike. These devices are often of a smaller size with limited computational and storage capacity. Typical examples of *Edge* servers are smartphones, tablets, desktops, smart speakers etc. The majority of *Edge* devices participate as crowdsourced servers in *Edge-Fog Cloud* and are owned and operated by independent third-party entities. Despite their limited capability, *Edge* resources are highly dynamic in terms of availability, routing etc., and can collectively address several strict application requirements. The ad-hoc nature of this layer makes it the perfect candidate for handling contextual user requests and support services catering to disaster relief, vehicular and robotic systems etc.

Fog. The *Fog* layer is composed of high-capacity compute servers co-located with network-capable nodes near end users. Routers, network switches, WiFi access points and cellular base stations are prime examples of *Fog* resources. Unlike *Edge*, the devices are specifically designed, manufactured, operated and deployed by cloud vendors. *Fog* servers also support existing cloud virtualization technologies and protocols out-of-the-box. The central objective of the *Fog* layer is to provide in-network computation as the data traverses to the centralized cloud for storage. Furthermore, the *Fog* resources are tightly-coupled and utilize a large bandwidth and highly reliable dedicated network links. However, compared to *Edge* servers, their deployment is widespread in a geographical region.

Data Store. At the core of *Edge-Fog Cloud* architecture lies the *Data Store*. *Data Store* closely resembles a centralized cloud infrastructure with

[1]For the rest of the thesis, we denote "*Edge*" as an abstract layer of *Edge-Fog Cloud* and "edge" as a generic term for edge computing.

the exception that it only acts as a data-storage repository and does not provide any computation to user requests. Our primary reason to impose this restriction is to maximize the utilization of *Edge* and *Fog* resources[2]. The primary aim of *Data Store* in *Edge-Fog Cloud* is to ensure secure, reliable and global access to data by *Edge* and *Fog* nodes which are otherwise distributed, failure-prone and equipped with limited cache size. Therefore, the *Data Store* is responsible for maintaining the end user's Quality-of-Experience (QoE) by assuring availability and consistency of application data in case of *Edge* and *Fog* resource failures.

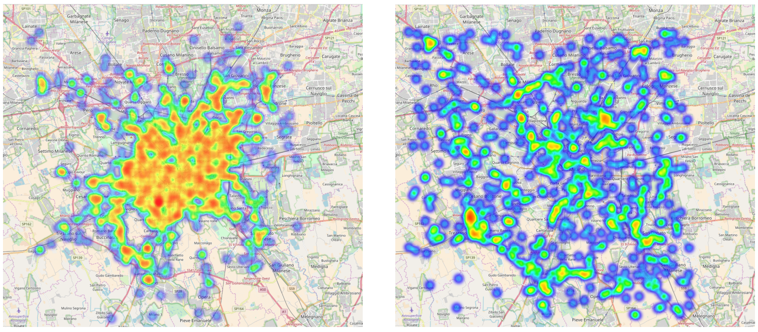
As discussed in Chapter 2, different edge-based applications impose different requirements on the underlying edge cloud architecture. The unique design of *Edge-Fog Cloud* allows it to behave as a unifying edge model for the majority of previously proposed approaches. For example, while the proposals like edge clouds [45], mist [99], MACC [59], MEC [34], MC [114] etc. bear a close similarity to the *Edge* layer, cloudlets [116] and fog clouds [11] can be integrated as the *Fog* in *Edge-Fog Cloud*. This allows *Edge-Fog Cloud* to also inherit the benefits and design objectives of such architectures. Furthermore, by removing its dependency on the centralized cloud for computation, *Edge-Fog Cloud* also provides a decentralized platform which maximizes the utilization of available servers near sensors and the end users to support the application QoS.

3.2 Physical Deployment of Edge and Fog Servers

While the modular design of *Edge-Fog Cloud* allows for an edge cloud variant composed entirely of *Edge*/*Fog* servers, the co-existence of both layers is imminent and necessary to support the requirements of several emerging edge-dependent applications. As the *Edge* is primarily composed of crowdsourced servers, the availability and reliability of such resources cannot be ensured throughout the application run-time. Cloud providers can rectify this by installing managed *Fog* servers in the same area, which would ensure that the application SLA requirements are met throughout its operation. Therefore, the presence of servers from both layers in a physical region is necessary to ensure a consistent and reliable compute backbone to application providers. However, installing a *Fog* server in a location which already has a high density of crowdsourced *Edge* devices in the vicinity can result in a significant under-utilization of either resource type.

Furthermore, the ISP-backed cloud provider must bear a combination of *CAPEX* (purchase and deployment) and *OPEX* (operation, maintenance

[2]Deployment constraints imposed by centralized cloud on *Edge-Fog Cloud* is in [80].



(a) WiFi access points (b) Telekom Italia basestations

Figure 3.2: Heatmap WiFi access points and Telecom Italia's cellular base stations over Milan, Italy.

and security) for every *Fog* server installation. It is only natural that cloud providers seek to maximize their profits by discovering specific installation locations in the region, which consistently observe high user requests such that the deployed server can remain almost always fully utilized. However, previous research has shown that user request distribution in any area is not static but temporally and behaviorally influenced [77]. As a majority of *Edge* servers participate as crowdsourced entities, shifts in user densities in an area also impact the availability of such resources, which hold a priority over *Fog* in *Edge-Fog Cloud* due to their ultra-low latency to the end user.

To illustrate the problem further, we extrapolate the extent of *Edge* and *Fog* resources in Milan, Italy. Figure 3.2 shows the heatmap of WiFi access points (APs) and Telekom Italia base station densities in the city[3]. While WiFi access points behaviorally resemble the properties of *Edge*, cellular-provider backed base stations are prime locations for deploying managed *Fog* servers. As can be observed in the figure, the density of access points is concentrated around the city center (center of the map) which boasts of large user crowds as residential, commercial and tourist spots are clustered around that region. As we move away from the center, the availability of access points decreases. On the other hand, the deployment of Telekom Italia base stations in Milan is almost evenly distributed throughout the region. This is by design as the primary goal of a cellular provider is to ensure consistent user connectivity throughout the operational zone such as to maintain the required user QoE.

Such deployment density trends makes *Fog* server placement problem unique to existing server deployment solutions. Naive installation of servers

[3]Data illustrated in the figure is extracted from open source datasets [63, 128].

in higher population density areas will result in sub-optimal utilization as the majority of requests will be satisfied by densely available user-managed *Edge* devices in the vicinity. In such cases, a compute request will only be offloaded to the *Fog* server if it exceeds the capacity of the *Edge*. On the other hand, deploying *Fog* servers at all base stations in the region is equally naive as the majority of servers will remain under-utilized due to the absence of local user requests. Furthermore, existing server placement solutions [101, 102, 112] are not directly applicable in this scenario as the end-goal for such solutions is to discover deployment locations which maximize the user-to-server bandwidth and latency requirements. On the other hand, potential solutions for the *Fog* placement problem must focus on optimizing server utilization and service availability.

Based on these observations, we design Anveshak [83] which is a server deployment framework for ISP-backed cloud providers. Anveshak identifies optimal base station locations in a metropolitan area for installing a *Fog* server. Unlike existing placement approaches, which only consider user population density, Anveshak predicts the availability of *Edge* resources and identifies areas where the *Fog* server will handle user requests maximally. Anveshak's operation is based on the assumption that the interested cloud provider partners with an existing cellular provider operating in the region for deploying *Fog* servers. The motivation for this assumption is two-fold. *Firstly*, it allows the cloud provider to co-locate servers with a base station which is already managed by the ISP and the resulting maintenance and operational overheads can be shared between the two parties. *Secondly*, the ISP can provide the cellular provider access to a historical user request database which may include, but is not limited to, Call Detail Records (CDR), SMS connections, internet usage, etc.

Figure 3.3 shows Anveshak's workflow. We design Anveshak as a three-phase modular system where each stage is responsible for handling a specific task. The modular design allows cloud providers to swap and add stages to the protocol tailoring to the additional complexities of the region. In ***Phase I***, Anveshak identifies high communication zones of the area as they will likely result in higher server utilization. The zone identification process is as follows. Anveshak starts with logically dividing the geographical region into equally spaced grids where the size of the grid is inversely proportional to the base station density and the number of *Fog* servers the cloud provider aims to install. The system then utilizes the historical data of cellular link access by subscribers of the cellular provider (normalized over a longer time to remove any temporal outliers) originating from each grid. The request densities are clustered depending on their *closeness*. The resulting clusters

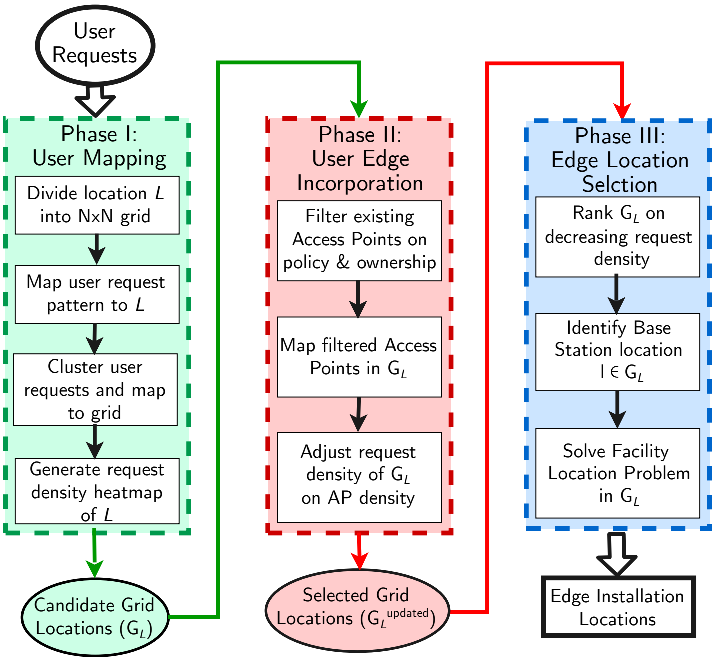

User Requests
Phase I: User Mapping
Divide location L into NxN grid
Map user request pattern to L
Cluster user requests and map to grid
Generate request density heatmap of L
Candidate Grid Locations (G_L)
Phase II: User Edge Incorporation
Filter existing Access Points on policy & ownership
Map filtered Access Points in G_L
Adjust request density of G_L on AP density
Selected Grid Locations ($G_L^{updated}$)
Phase III: Edge Location Selction
Rank G_L on decreasing request density
Identify Base Station location $l \in G_L$
Solve Facility Location Problem in G_L
Edge Installation Locations

Figure 3.3: Workflow of Anveshak.

closely identify regions where subscriber request density is on the higher end. As the clusters can be overlapping, varyingly dense and of different shapes, Anveshak normalizes them as a flat heatmap of grid locations. At the end of the first phase, Anveshak presents the list of grid areas, ranked in decreasing user request density, to the cloud provider.

In ***Phase II*** shown in Figure 3.3, Anveshak estimates future availability of *Edge* servers in every grid area by extrapolating the density of currently operational WiFi APs in the region. For this purpose, the cloud provider can exploit several openly available datasets of WiFi APs operating throughout the globe [128]. Anveshak filters the WiFi APs operational in the target region and clusters them based on their density. The system then negatively adjusts grid areas which have a high AP density. The updated heatmap is a collection of grid locations which have high user request density *and* low availability of edge devices to satiate those requests. Anveshak presents this list to the cloud provider in its final phase. Based on the provider's deployment budget and required service coverage area in the region, the service provider can select *top-k* grid locations for deploying the *Fog* server. In case there exists only a single serving base station in the selected grid, the cloud provider may choose to co-locate the server at that site. However, if there are more than one cell sites operational in that grid, the system must provide a priority level to each location depending on its reachability extent for satisfying maximum user requests originating in the vicinity. ***Phase III*** of Anveshak is designed to solve this problem. After identifying the exact locations of the cellular provider's base stations in the grid, Anveshak calculates the probability of one-hop latency from each site to the majority of user requests. The model utilizes a coordinate-based network latency approximation technique [90] and calculates the maximum tolerated network distance (R^{max}) and network cost ($n_{(S,u)}$) between users U in grid G_i to server S installed at base station in the grid $L^{G_i} = \{l_1, \ldots, l_x\}$. Formally, the variables are defined as follows:

$$R^{max}_{(u_l,S_l)} = max[u - S_l] \quad \forall u \in U \tag{3.1}$$

$$n_{(S,u)} = \alpha * R_{(S,u)} \tag{3.2}$$

Variable α in Equation 3.2 denotes the maximum server access cost for the user in the grid. Based on the network utility of each base station, Anveshak identifies the optimal server location for an arbitrary user u in the grid by solving the following equation.

$$S_u = min \sum_{l \in L^{G_i}} \{S_l | S_l \in S, n_{(S_l,u)} < n_{max}\} x_l \tag{3.3}$$

Equation 3.3 closely resembles a Facility Location Problem [21] which is known to be *NP-hard*. Anveshak approximates the solution to the problem by limiting the grid size in *Phase I* such that even the worst-case iterative solution completes within reasonable time. Further details regarding the implementation of the problem and subsequent results can be found in Publication II attached to this thesis [83].

To summarize, in this chapter, we presented a unifying edge cloud architecture *Edge-Fog Cloud* which maximizes the computational capability at the network edge by incorporating resource types of varying hardware capacity and ownership but ensures data resilience through a centralized data store. We also presented Anveshak, which is a deployment framework designed for cloud providers that aim to install *Fog* servers amidst crowdsourced *Edge* resources. Anveshak predicts the future density of user request patterns and preexisting *Edge* servers in the area to holistically select the base station locations best for augmenting *Fog* servers. While the solutions presented in this chapter were designed to unite multiple cloud entities to ensure a consistent global edge compute covering, they can be easily adapted to specific requirements of cloud providers competing in the same region.

Chapter 4

Infrastructure services for Edge

Although *Edge-Fog Cloud* aims to be an extension to traditional cloud computing, the underlying architecture of the two models is in stark contrast to each other. As discussed in Chapter 2, the difference in the two cloud models is not just limited to the organization of its compute resources but extends to the properties and behavior of its infrastructure as well. For example, while centralized clouds operate in well-managed and controlled environments, *Edge-Fog Cloud* offer no such characteristics. Moreover, the majority of salient features of DC servers, such as reliability, availability, significant compute capacity, large local data storage, high bandwidth and minimal network congestion, are not inherently supported by *Edge-Fog Cloud*. This, along with additional complexity imposed by *Edge* and *Fog* resources, such as wireless shared access medium, battery capacity, mobility, etc., begs for a complete redesign of existing cloud-based solutions such that the hardware of these devices can be utilized effectively.

In this chapter, we focus on providing solutions for the three pillars of cloud infrastructure; *compute*, *storage* and *networking.*

I. **Compute.** An effective computation strategy for *Edge-Fog Cloud* must consider the following resource constraints. (i) *Edge* devices are severely compute constrained and multiple such servers may need to work in a distributed fashion to support a single application task, (ii) computation power of several *Edge* resources may be limited by their available battery capacity, and (iii) executing tasks on *Edge-Fog Cloud* incurs an associated processing, networking and energy cost. Considering the restrictions above, we design *Least Processing Cost First (LPCF)*, and *energy-efficient Least Processing Cost First (eLPCF)*, which are task allocation mechanisms for *Edge-Fog Cloud*, discussed in Section 4.1. While LPCF minimizes the associated processing and network cost for executing application tasks on *Edge-Fog Cloud*, eLPCF additionally minimizes resource energy used to support the computation.

II. **Storage.** While *Edge-Fog Cloud* utilizes centralized *Data Store* to ensure data permanence and availability, the *Edge* and *Fog* servers must rely on their local disk/cache capacity for storing prerequisite data for computation. This data is usually fetched from sensors/*Data Store* and must be available in local caches of all involved servers prior computation start time. Moreover, as *Edge* resources have limited cache capacity and pre-cached data is rarely re-used in applications with varying workloads, new (often non-overlapping) data must be fetched for every subsequent task allocation. This significantly impacts the performance of *Edge-Fog Cloud* as the network delay for acquiring the required data far exceeds the latency gains of computing at the edge. For this purpose, we design an *edge caching mechanism* which maximizes the probability of re-using locally cached data or retrieve it from caches of nearby servers. Discussed in Section 4.2, our mechanism also ensures coherency of shared data distributed across the platform.

III. **Networking.** Unlike DC networks, which boast of reliable, high bandwidth, isolated ethernet connectivity, *Edge-Fog Cloud* primarily relies on public wireless access networks, such as LTE, WiFi etc. To imitate the characteristics of DC networking at the edge, we explore the possibility of parallelly utilizing multiple LTE connections at an edge server via Multipath TCP (MPTCP) in Section 4.3. MPTCP is a standardized extension to TCP which allows applications to transparently use multiple network interfaces by creating a TCP flow over each path. However, our results in Section 4.3.2 indicate that due to its internal design policies, MPTCP is unable to maintain its performance under mobility due to excessive on-path queueing delays. As mobility is one of the critical features that edge computing aims to support, we design *QAware* in Section 4.3.3 which is a custom scheduler for MPTCP. QAware proactively avoids the network path with large packet queues which results in a significant increase in MPTCP performance over constrained wireless networks.

4.1 Deploying Tasks on the Edge

The *Edge-Fog Cloud* architecture discussed in Chapter 3 poses a double-edged sword for edge computing infrastructure solutions. On one hand, the cloud model presents an amalgamation of devices of ranging compute and network capacity for handling a multitude of edge application requirements. On the other hand, the server capacity is often limiting to the point that a resource is unable to single-handedly complete user-assigned tasks and requires constant coordination with other nearby servers in the platform. Existing cloud-based task deployment frameworks such as MapReduce [28], Spark [138] are not capable of effectively utilizing edge servers due to two reasons. Firstly, the primary objective of such datacenter-based frameworks is to maximize the utilization of the processing capacity offered by the underlying servers, and network latency or congestion between the servers is not considered in their operation. Secondly, while datacenters provide consistent power and cooling facilities for optimal server operation, *Edge* servers such as mobiles, tablets, laptops etc. operate in public environment with limited battery capacity. However, past research has shown that with effective task allocation, computations on such devices can significantly reduce the overall energy used without sacrificing computation time when compared to a datacenter server [93,110]. This is due to highly-efficient power management algorithms built into the mobile CPU, which constantly controls the processor's clock speed based on the available battery capacity. Despite their energy-efficient behavior, additional care is required to incorporate these devices as edge resources as battery depletion in the midst of computation can result in an inconsistent application state.

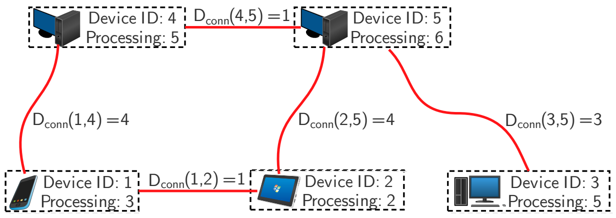
In this chapter, we present two deployment algorithms, *Least Processing Cost First* (LPCF) and *energy-efficient LPCF* (eLPCF). The algorithms are part of Publication II attached to this thesis. Before we describe the inner workings of our solutions, we first discuss the core objectives of a task deployment strategy for edge clouds.

4.1.1 Naive Task Deployment Strategy

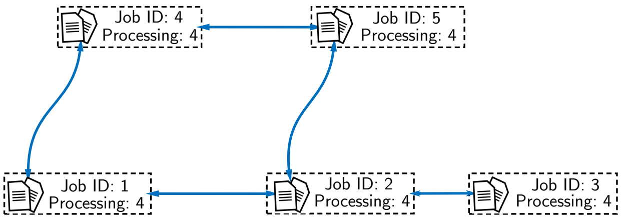
Figure 4.1a shows a snapshot of *Edge-Fog Cloud* with five compute resources while Figure 4.1b shows an application job distributed into five distinct sub-tasks. We represent an *Edge-Fog Cloud* resource by a unique ID D_i and maximum available processing power $D_{proc}(i)$. The red interconnections in the cloud snapshot denote direct network connectivity between servers, which has an associated cost denoted by $D_{conn}(i,j)$[1]. Similarly, each job[2] in Figure 4.1b is represented with a job ID J_i with required processing cost for completion denoted by $J_{size}(i)$. Furthermore, there can

[1] D_{conn} can signify multiple network parameters such as bandwidth, latency, packet loss etc.

[2] We use the term *sub-task* and *job* interchangeably.



(a) Edge-Fog cloud resource graph



(b) User assigned task graph

Figure 4.1: Input constraints for task deployment on *Edge-Fog Cloud*.

exist two-way dependencies between jobs that operate on shared data or need to be processed sequentially. Blue links in the task graph denote such dependencies.

The primary objective of a task deployment algorithm is to optimize the job-to-device placement such that both costs for processing (PC) and the network interaction (NC) for the computation is minimized. These two optimizations are formalized as follows.

$$PC_{min} = \sum_{i,j \in A} \left(\frac{J_{size}(i)}{D_{proc}(j)} \right) x_{ij} \tag{4.1}$$

$$NC_{min} = \sum_{i,j \in A} J_{conn}(i,j) * D_{conn}(f(i), f(j)) \tag{4.2}$$

where A is the set of all arcs in the graph and x_{ij} is a binary decision variable.

Equations 4.1 and 4.2 show the minimization function for processing jobs on *Edge* resources and network connectivity between dependent tasks respectively. An ideal task deployment algorithm aims to discover optimal placement with the least PC_{min} and NC_{min}. While there already exist algorithms in the literature which can guarantee optimization of PC_{min} in $O(n^3)$ bound [64], the same is not true for optimizing NC_{min}. Equation 4.2 closely resembles an NP-hard problem, Quadratic Assignment Problem

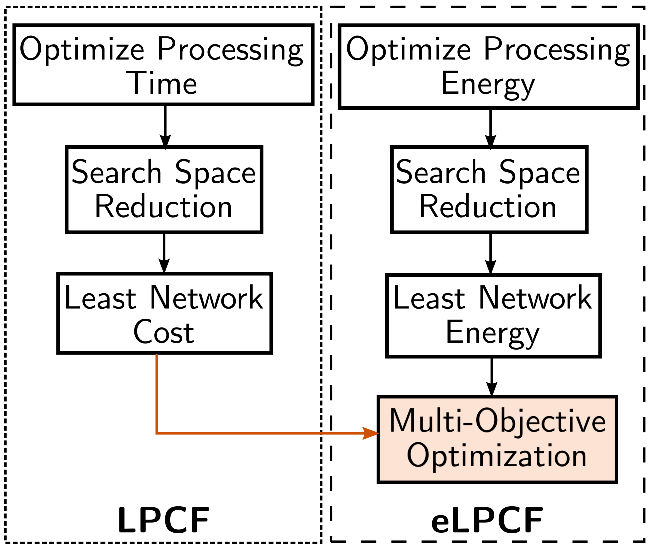


Figure 4.2: LPCF and eLPCF algorithm workflow.

(QAP), which can only be approximated by applying strict constraints. Without constraints, the optimization can take a significantly long time. For example, computing the deployment for 30 nodes can take well over a week on a grid of 2500 machines [137]. Existing approximation approaches, e.g., Branch-and-Bound, only work for small problem sizes and is not applicable in *Edge-Fog Cloud* where hundreds of edge servers can be candidates for a single task deployment.

4.1.2 Optimized Task Deployment for *Edge*

Unlike existing task deployment strategies, our aim was to develop algorithms that can provide semi-optimal job placements *quickly*! Instead of applying generic constraints for approximating QAP, we exploit an inherent property of an edge cloud environment to reduce our problem search space significantly, i.e. *both Edge and Fog layers are composed of similar devices with matching attributes* such as CPU capacity, architecture, etc.. Considering this observation, we develop Least Processing Cost First (LPCF), which guarantees a job assignment within polynomial time while optimizing both processing and network costs. We extend our principal approach to devise an energy optimizing variant of LPCF, energy-efficient LPCF (eLPCF). Figure 4.2 shows the anatomy of both algorithms' workflow.

Least Processing Cost First (LPCF)

Instead of optimizing for both processing and network costs, LPCF starts with solving the optimal task deployment with the least associated processing cost shown in Equation 4.1. Once the algorithm has a valid job-to-device placement, it builds a search space from the solution by interchanging jobs assigned to resources with equal processing costs and resources with jobs requiring similar processing. At the end of this step, LPCF generates a set of deployments, all with the least processing cost derived from the first optimization step. Further, LPCF iteratively[3] calculates the network cost of each deployment in the search space and picks the variant with the least value. The final deployment calculated by LPCF has the least processing cost and an *almost* optimal network cost. Our results in Publication II show that the strategy employed by LPCF reduces the problem search space by several hundred orders of magnitude. This allows LPCF to complete its execution within polynomial time while the network cost of the resulting deployment always stays within a 10% range of the optimal solution.

Energy-efficient Least Processing Cost First (eLPCF)

eLPCF is an energy-efficient variant of LPCF that searches for deployment which would result in least energy spent (E_{total}) due to processing (E_{proc}) and network usage (E_{netw}).

$$E_{total} = \sum_{i \in \mathcal{N}} [E_{proc}(i) + E_{netw}(i)] \tag{4.3}$$

$$E_{proc}(i) = PC(i) * e_p \qquad \forall i \in \mathcal{N} \tag{4.4}$$

$$E_{netw}(i) = NC(i) * e_n \qquad \forall i \in \mathcal{N} \tag{4.5}$$

Here, e_n and e_p denote networking and processing cost-to-energy metrics for D_i respectively. Both values depend on D_i's hardware configuration.

The first two steps of eLPCF workflow closely resemble that of LPCF as it exploits the homogeneity in server hardware and job characteristics to build a reduced search space. In step 3, eLPCF computes a deployment solution with the least energy for processing and almost optimal energy for networking in polynomial time. It is to be noted that eLPCF E_{total} optimization can be influenced by the resource's e_n and e_p which have no correlation with PC and NC. As a result, the task deployment can have

[3]Intuitively, a branch-and-bound variant of LPCF can reduce the number of pointless calculations, but for the sake of simplicity we opt for an iterative approach.

Size	5	10	20	50	100	150
NC-Perm	0.068s	23m 20s	> 1h	> 1h	> 1h	NA
NC-QAP	0.026s	36.273s	18m 38s	> 1h	> 1h	NA
LPCF	0.0005s	0.002s	0.073s	51.686s	12m24s	> 1h
eLPCF	0.0011s	0.0116s	0.126s	1m14s	24m43s	> 1h

Table 4.1: Computation time comparison between deployment algorithms.

significantly high completion time as e_n and e_p are largely dependent on the CPU algorithms and NIC peripherals of the device. So as to not be completely swayed by the makeup of the device hardware, eLPCF combines its search space with that calculated by LPCF. This ensures that the resulting deployment also considers reduction of both processing and network cost metrics. In its combined search space, eLPCF utilizes a multi-objective function to select deployment with optimal E_{total}, PC and NC metrics. We also provide weights to each deployment cost parameter, which can be tweaked on-the-go by cloud providers according to their liking [80].

We compare both algorithms' performance with the permutation and the QAP variant of the network cost optimization function (Equation 4.2). Table 4.1 compares the completion time for proposed algorithms to naive and QAP-based network optimizing solutions for increasing number of participating *Edge-Fog Cloud* resources[4]. The results show that LPCF's strategy of reducing the problem search space shows a significant gain in performance over its competitors. Where NC-based solvers struggle to compute any solution for topology sizes exceeding nodes, both LPCF and eLPCF discover a task deployment within minutes. Surprisingly, for increasing problem sizes, eLPCF performs twice as bad as LPCF. We attribute this additional computational delay of eLPCF to the *multi-objective optimization* step of the algorithm that waits for outputs from two parallel optimization branches (see Figure 4.2). Detailed comparison between both task deployment algorithms and the impact of *Edge* resource characteristics on deployment is further discussed in Publication II attached to this thesis.

4.2 Storing and Retrieving Data on *Edge*

Edge-Fog Cloud relies on a centralized *Data Store* to provide data permanence and availability. *Data Store* offers vast storage capacity and ease-of-access for underlying *Edge* and *Fog* servers but with higher access delay. However, the *Edge-Fog Cloud* dependence on a centralized data store is

[4]Maximum task computation time is set to one hour

critical for two reasons. Firstly, the local cache of *Edge* and *Fog* resources is often of limited capacity due to which the devices are unable to retain relevant data for extended time periods. Secondly, as task operations in *Edge-Fog Cloud* involve executions over multiple servers, all of which operate on their local copy of relevant data, periodic synchronization with a centralized data repository ensures that the computational data in the resource's cache always remains consistent and coherent.

On the other hand, centralized *Data Store* also acts as a bottleneck for scaling *Edge-Fog Cloud* performance. A typical data exchange in *Edge-Fog Cloud* is as follows. Task deployment algorithms, such as LPCF, select a set of *Edge* and *Fog* resources to fulfill a user-assigned task which requires prerequisite data for computation. This data can either be information from end-sensors or results of previously computed queries. Consider the example of a fully-automated automobile factory which houses multiple machines (read resources) that actively collaborate and coordinate with each other to complete a specific task. A machine with a drilling tool must be able to change its settings for the next workpiece (e.g. chassis to engine) which requires data updates from sensors attached to the belt. However, it is also possible that the machine needs to switch to a different tool altogether, such as screwdriver, which requires coordination and specifications from other machines involved in the project across the factory. Assuming the sensor and the resources offload their data to a *Data Store*, the machines need to retrieve the required specifications into their local caches before initiating the task operation [84]. In this case, each assigned resource sends a retrieval request to the *Data Store*, the phase also known as *fetching*.

As the waiting period for acquiring data is directly proportional to the number of participating resources and latency between the resource and *Data Store*, fetching data at the start of every computation can induce significant delays in task completion. Furthermore, in a system with varying workloads, cache reusability can be quite low as subsequent computations may require data different from local cache content. Considering the complications above, pre-caching required data at the edge seems to be the obvious solution. However, this is not straightforward as majority of existing approaches [106, 107] were designed for Content Delivery Networks (CDN), which have very different requirements when compared to computation-first *Edge-Fog Cloud*. Specifically, unlike CDN's, the compute data in *Edge-Fog Cloud* has shorter temporal relevance and receives frequent updates.

We now present an efficient edge caching mechanism, published in Publication III, which leverages the already cached content in *Edge-Fog Cloud* resources to predict and prefetch required data for upcoming computations.

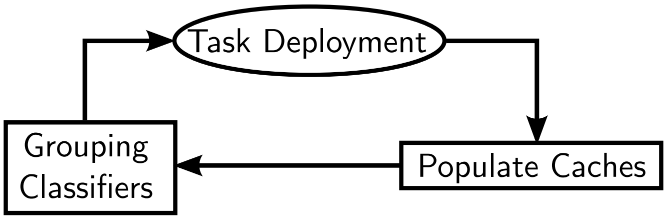


Figure 4.3: Edge-Fog cloud cache grouping algorithm.

4.2.1 Grouping Cached Content of Edge Resources

Our approach for improving data caching at the edge involves a three-step *iterative* cache-grouping mechanism, which is built on the principles of the task deployment algorithms discussed in Section 3.2. The core idea of our approach is shown in Figure 4.3 and explained as follows. In its first iteration, the task deployment algorithm assigns an application task on a set of $\{RC\}_i$ resources (phase *Task Deployment*). At time $t = 0$, the participating resources are "fresh" with empty local caches[5] and request for the required data from *Data Store* (phase *Populate Cache*). In its next phase, the algorithm exploits the insight that *resources assigned to execute a single task will cache the same/similar prerequisite content.* This allows the algorithm to classify each computation within a broader workload W and data required for that particular workload as D_i^k. All resources $\{RC\}_i$ housing data for W_i can be grouped in a logical cluster, termed as a cache group (denoted as $\{CG\}_i$). As *Edge-Fog Cloud* handles several computations on distinct tasks *parallelly*; at arbitrary time $t = n$ there are CG_k groups existing in *Edge-Fog Cloud* for W_k workloads[6]. The classification of computation into workloads can be based on several metrics, which includes (but is not limited to) location, time, sensor type etc. A comprehensive list of different classifiers for workloads is in [85].

In its subsequent iterations, the cache grouping algorithm coordinates with the task deployment algorithm to prioritize the further deployment of tasks belonging to a workload on the member resources of its respective cache group. Along with increasing the probability of re-utilizing pre-cached data of the assigned resource, the algorithm also improves the chances for retrieving the required data from neighboring cache group mem-

[5]Even if resources have pre-existing data in their cache, the algorithm assumes that either there will be available space to house data for the current computation or required space would be freed through a cache replacement.

[6]It must be noted that membership to a cache group is *strictly* logical and a resource can belong to more than one group.

ber resources instead of fetching it from *Data Store*. In case the present group members of that workload are less than ideal for computing the task, the algorithm is free to choose any other resource that ends up becoming a part of the cache group at the end of the computation. As the algorithm cyclically progresses, the number of distinct groups in the system stabilizes and sharing within a group far exceeds retrieval requests from *Data Store*.

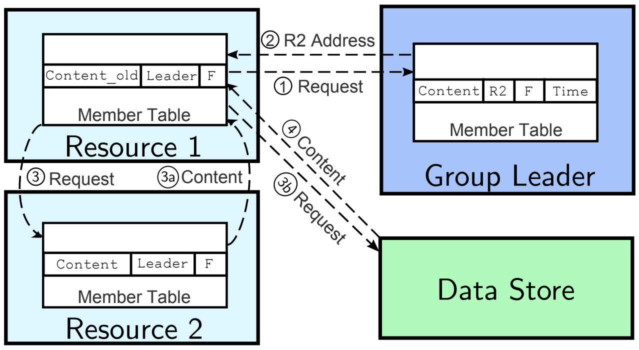
We evaluate our algorithm on the Icarus simulator [113] for large network topologies. We artificially generate requests for $96 * 10^4$ content items belonging to 32 different workloads. Our algorithm significantly outperforms non-grouping-based caching algorithms and achieves almost twice the cache hit ratios resulting in 45% reduction in retrieval requests. Further details of our evaluation are present in Publication III to this thesis.

4.2.2 Maintaining Edge Cache Coherence

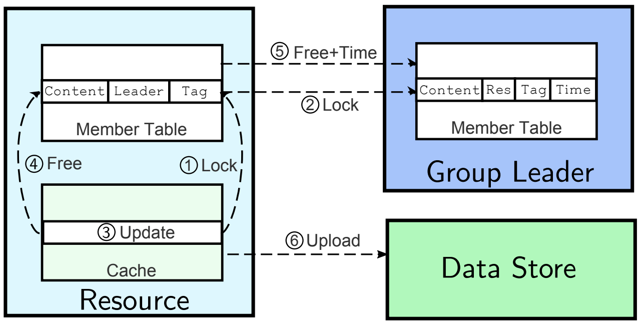
Although the cache grouping algorithm compels servers to share required content at the edge, it opens up another issue for *Edge-Fog Cloud* operation, i.e. *How do you ensure the correctness of shared data?* Isolated operations on a local copy of the shared content by multiple edge servers may result in an incoherent data state. A resource *must* always operate on the most relevant version of the required data, failure of which can lead to inconsistent computations that may require multiple roll-backs and re-executions. This scenario (as well as its consequences) bears a close resemblance to the classical distributed shared memory problem [100].

Intuitively, the problem can be mitigated by employing one of the following approaches. *One*, at the completion of every computation the involved resources send update messages to all other servers in the cache group/system updating them of their local state. *Second*, instead of sharing data at the edge, the assigned servers periodically synchronize their local cache contents with the *Data Store* which is then responsible for ensuring consistency. We argue that both approaches are equally impractical in a system operating at the scale of *Edge-Fog Cloud*. The first approach would induce unnecessary network traffic in the system that would increase with the increasing number of participating entities. On the other hand, the second approach mars the sole objective of the cache grouping algorithm i.e. maximizing sharing at the edge.

To resolve this issue, we design a resource communication model for *Edge-Fog Cloud* for *updating* and *retrieving* the prerequisite data. The overview of communication in both scenarios is shown in Figure 4.4. The model is inspired by the directory cache coherence algorithm for networked processing systems [1] and ensures *causal coherence* on shared data.



(a) Data retrieval within cache group



(b) Data update within a cache group

Figure 4.4: Communication model for ensuring coherency.

Our communication model is strictly *pull-based*, which reduces unnecessary message exchange in the system significantly[7]. Periodically, the model elects a *leader* resource in each cache group, which is responsible for serializing the synchronization of cached content for the member resources. Any resource can nominate itself for the leader election provided it has a consistent connectivity to the rest of the group. The leader only acts as a central database for enhancing communication between resources in the group and is not involved in any data transfers. As shown in Figure 4.4, each resource involved in the communication (leader, group member and *Data Store*) uses specifically-designed tabular data structures that serve as bookmarks for the communication. While the majority of the data structure fields are self-explanatory, the most notable field for the communication model is "Tag". Tag denotes the state of locally cached data. If the data is currently in-use by the resource, the Tag entry will be *locked* otherwise it will be *free*.

[7]While resources can fail, we assume that messages are not lost in transmission.

Although we assume that resources regularly upload their computed data to *Data Store* (preferably at the end of each computation), the design of our communication model refrains from synchronizing content via centralized *Data Store* but uses it only for failure recovery purposes.

Retrieving Content

The model restricts resources to only share information of the last updated content with their respective group leader. Before initiating a new computation, every resource in the group requests their group leader for the latest copy of prerequisite data, irrespective of whether a locally cached copy is available or not. The group leader replies with the address of the resource which last reported on updating that particular content. At this point, the requesting server directly queries the last-updater resource to share its cached copy of the content such that the content remains consistent in the group. In case the node is no longer alive or has replaced its cache contents, the retrieval request is escalated to the *Data Store*.

Updating Content

To avoid stale writes on data, the model follows a step-wise checkpoint approach for update requests. After retrieving the required data following the protocol described above, the updating resource locks the cached content by marking its Tag field as *locked* before initiating computation. At the same time, the resource also requests the leader to update its own Tag for that particular content in its directory. If the group leader receives any retrieval requests for that content from any other resource while the data is under lock, the leader simply denies that request. After successfully executing the task, the updating resource "un-tags" the content in its local directory and notifies the group leader of the update. The leader responds by freeing the content's Tag in its own directory along with storing the time at which it received the notification[8].

In conclusion, we designed a cache grouping strategy which works in conjunction with task deployment for *Edge-Fog Cloud*. The algorithm maximizes the data sharing between the resources, thereby removing *Data Store* as a bottleneck for efficient computation. The strategy employs a communication model which provides causal data coherence for cached content at the edge while enabling parallel updates.

[8]The timestamp field resolves any race condition due to message delays.

4.3 Optimizing Networking at the *Edge*

Support for ultra-low network latencies due to proximity to end-users and sensors alike heralds edge computing as a “silver bullet” for emerging latency-sensitive applications. However, unlike traditional datacenters which house an interconnection of large bandwidth ethernet pipes in isolated and managed locations, edge servers plan to operate in public infrastructure and overcome unrelated network traffic. Moreover, a variety of applications, such as augmented reality, smart homes, smart vehicles etc., incorporate mobility in their functionality and require access to nearby edge servers via wireless network interfaces. Catering to such increasing demands, manufacturers plan to incorporate a myriad of wireless connectivity standards in edge servers, e.g. LTE, WiFi, 5G etc. [23]. The performance of such technologies is plagued by intermittent delays, congestion and failures during active data transmissions as the research to support a wide variety of applications and environments is still ongoing [56]. Even in isolated datacenters, attempts to design a fully wireless architecture for inter-server communication has been proven to be sub-optimal due to careful placement restrictions of directional antennas, interference and MAC layer contention [26, 118, 147]. As a result, the adoptability of wireless technologies in edge clouds remains under question as the application providers must also embrace its inherent flaws to maintain the required application QoS.

Multipath TCP (MPTCP) [10] is an extension to TCP that allows devices with multiple network interfaces, such as smartphones equipped with WiFi and LTE, to form multiple parallel TCP connections. The transport protocol allows multi-homed devices to exploit available bandwidth over multiple interfaces and offers increased robustness and resilience to packet failures. Due to several benefits of MPTCP, researchers have proposed to utilize the protocol for a variety of applications and configurations such as datacenters [103], augmented reality [54], railway networks [71], video streaming [53], opportunistic networks [105] etc. The MPTCP kernel is available as open-source [19] and is actively supported and utilized by industries, such as Apple [6], in their products.

Due to several inherent benefits of MPTCP as mentioned above, the protocol presents itself as a prime candidate to mitigate prominent connectivity issues in edge clouds. MPTCP’s functionality can be well-exploited in edge clouds as majority of edge resources will be equipped with multiple network interfaces. For example crowdsourced resources such as smartphones support simultaneous dual-LTE and WiFi connectivity. Manufacturers plan to equip specifically designed *Edge*/*Fog* servers with even more connectivity options [23].

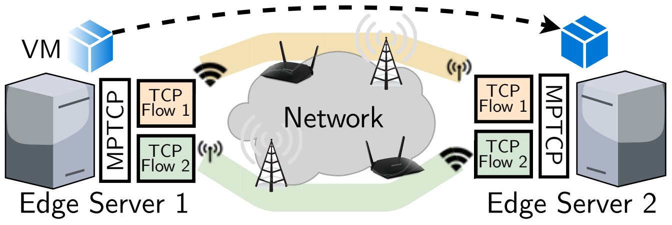


Figure 4.5: Edge Servers utilizing MPTCP over two interfaces.

Figure 4.5 illustrates an edge-to-edge VM migration scenario using MPTCP in such a utopia. As shown in the figure, both edge servers are equipped with a combination of WiFi and LTE NIC. The end-to-end connection between the servers comprises of two parallel TCP flows over each NIC pair[9]. Individually, either WiFi or LTE would have been unable to support the transfer rate required for fast VM migration. However, with MPTCP both connection types can be utilized in parallel, which allows edge servers to replicate cloud-like network performance. While a few previous works have proposed to utilize the protocol for this purpose [16, 81], comprehensive understanding of MPTCP behavior over public wireless networks is required before drawing any conclusions regarding its applicability in *Edge-Fog Cloud*.

In this section, we analyze, via extensive measurements, whether MPTCP over wireless connections can support the edge-based application demands. Mainly, we focus on understanding the behavior of MPTCP over multiple LTE connections for different mobility types, as required by use-cases such as autonomous vehicles, augmented reality etc. Our results show that all is not well with MPTCP as its performance degrades with increasing mobility [82]. Through a thorough understanding of the root cause for this decline, we propose a scheduling solution for MPTCP, which overcomes this shortcoming and allows edge servers to perform optimally in constrained wireless environments [119].

4.3.1 Internal Workings of MPTCP

We start with understanding the internals of MPTCP. Unlike regular TCP, MPTCP builds connections over *hosts* and not *interfaces*. Figure 4.6 shows the network stack of an MPTCP-capable sender and receiver. The kernel implementation of the protocol adds a unifying abstraction/control layer atop TCP. The user application remains unaware of the presence of MPTCP

[9]The default path-manger of MPTCP creates M$\times$N TCP flows, where M and N are the number of NICs at both ends of the connection.

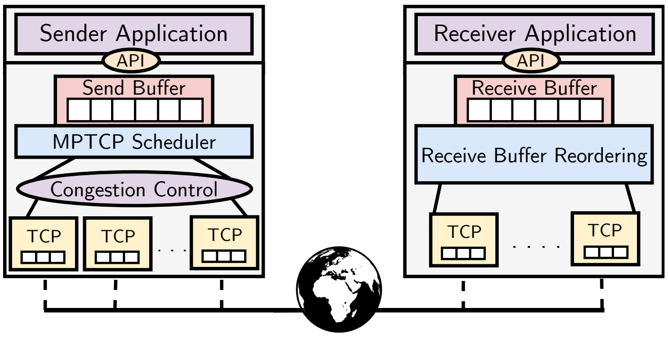


Figure 4.6: Multipath TCP end-to-end architecture.

and interacts with the system using the regular socket API. Instead of establishing an end-to-end TCP path, the server now maintains an MPTCP connection which is responsible for managing multiple TCP “subflows”. MPTCP is a *sender-oriented* protocol as all of its control blocks, such as connection management, scheduling and rate control, are handled by the sender and the receiver is only responsible for accepting incoming packets.

The sender-side of the MPTCP protocol is composed of three modular control blocks: *path manager* (not shown), *scheduler* and *congestion control*. The *path manager* is responsible for establishing and managing subflows between the hosts over all available NICs. The workflow for this is as follows. The path manager sets the MP_CAPABLE TCP option in its SYN packet and sends it to the receiver signaling its MPTCP compliance. If the receiver replies with a positive acknowledgment, a single end-to-end TCP flow over one of its interfaces is established which is known as the “master” subflow. Following this, the path manager adds additional subflows over remaining NICs in the same session using ADD_ADDR TCP option. The default “full-mesh” path manager does not require the sender and the receiver to have the same number of interfaces and establishes M×N subflows, where M and N are the number of distinct NICs at sender and receiver respectively.

The application data arrives in the send buffer from which it is scheduled to one of the underlying subflows by the *scheduler*. Each TCP subflow is monitored by its independent congestion control algorithm which controls the local packet send rate. While deciding the optimal path for a packet, the scheduler only considers subflows that have available space in their congestion window. The default scheduler, `minSRTT` injects application packets into the “fastest” subflow, i.e. TCP connection with the least smoothed round-trip time (SRTT) to the receiver [96]. Overall packet input rate is governed by MPTCP *coupled congestion control* which balances packet congestion over all underlying TCP subflows. While multiple solutions for MPTCP congestion control have been proposed, such as LIA [104], OLIA [65], BALIA [60] etc., a detailed explanation of the internal differences between the variants is out-of-scope of this thesis. As data packets may not arrive in the same order as they were transmitted from the sender due to losses and variable transmission delays, MPTCP employs a *receive buffer* at the receiver to store all out-of-order arriving packets. The receiver-side MPTCP waits for any missing in-sequence packets before re-ordering and delivering them "in-order" to the application.

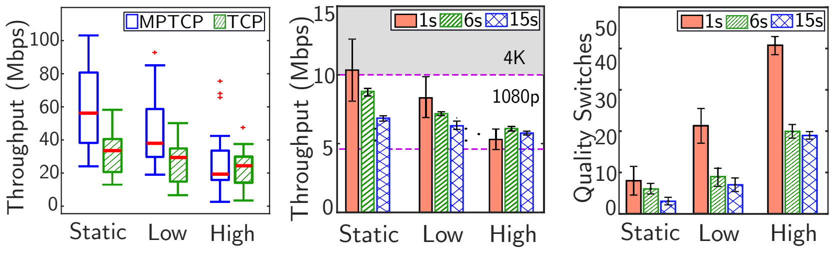
The current implementation of MPTCP enables researchers to incorporate their custom scheduler and congestion control schemes, provided they fulfill two fundamental *design objectives. First,* MPTCP over multiple paths should always achieve more throughput than TCP over the single better performing path. *Second,* the protocol must allow for interchangeability and resilience over all underlying TCP paths.

4.3.2 Poor MPTCP Performance in Mobility

As the typical requirement for a majority of edge-based applications is the support for mobility, we analyze MPTCP behavior for increasing movement speeds through extensive real-world measurements. We opt for analyzing MPTCP over dual-LTE connections due to the vast coverage areas, large combined bandwidth and reliable packet delivery promised out-of-the-box by LTE. Furthermore, considering the internal stake of cellular providers in setting up *Fog* infrastructure in a region (as discussed in Chapter 3), we envision cellular network to be the prime enabler of edge computing in the future. The measurements and resulting analysis is part of Manuscript I.

Our *Edge* server is a Raspberry Pi 2 (RPi) equipped with two Telewell CAT4 LTE USB modems. The RPi runs Raspbian OS over the latest MPTCP v0.94 [20] and is powered by an external battery to enable mobility. We equip USB modems with LTE connections from two major cellular providers operating in Finland, *Sonera* and *DNA* which offer extensive network coverage in the test region. On the other end of the connection is a cloud server hosted in AWS-central Europe. The server is equipped with 32 GB RAM, 1 Gbps Ethernet and 16-core 2.4GHz CPU. We ensure that both cellular connections at *Edge* have non-intersecting paths to the cloud by periodically running `traceroute` on both paths.

Throughout our experiments, we collect packet traces via `tcpdump`, signal strength (dBm) on each modem and associated basestation ID (BSID). We asked three volunteers to carry the RPis along their daily commute,



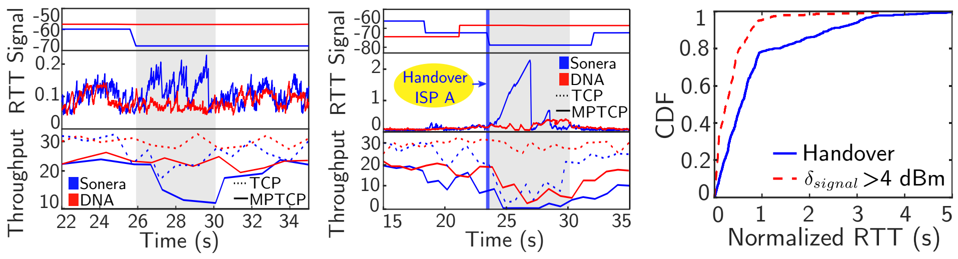
(a) Throughput (b) Video Bitrate (c) Video QoE

Figure 4.7: Effect of mobility on MPTCP performance.

which largely comprised walking, driving and using public transport such as trams, buses etc., from *September 2018* to *February 2019*. This allowed us to cover the full spectrum of target speeds for the bulk of edge-based applications. As the RPi moved in the physical space, both LTE modems experienced signal strength changes and handovers between basestations.

Our measurements reveal that MPTCP shows promise in scenarios where both the sender and receiver are static while the network remains consistent. In such cases, MPTCP achieves almost 1.7× throughput compared to a single TCP over either LTE connection. However, MPTCP performance degrades significantly with the increasing mobility of the RPi, as shown in Figure 4.7a. In slow-speed ($<$ 10kmph) scenarios, we observe a 37% decrease in overall throughput over two LTE links using MPTCP while parallel tests using TCP over single LTE revealed a reduction of 10%. The degradation is intensified in high-speed mobility cases ($<$ 100kmph), which happens to be the prominent network configuration smart vehicular applications. At such speeds, MPTCP throughput can observe up to 65% drops resulting in even lower performance than the single TCP connection. The result is highly intriguing and directly contradicts the design goal of MPTCP, i.e. *always perform better or at-par TCP*.

MPTCP's performance degeneration is not just limited to file downloads but also impacts streaming services over edge clouds, such as gaming, video delivery etc.. Figure 4.7b shows the average bitrate of three different segment-sized DASH streams over MPTCP. We observe up to 49% drop in video bitrate for constant streams (1-second segments) when the RPi is moving at high speeds. In such a case, the receiver struggled to maintain a consistent 1080p stream quality over a combination of two LTE connections when even a single LTE theoretically supports the bandwidth required for that quality. We attribute the primary reason for this degradation to MPTCP's inability to support data transfers at high speeds, as the receiver



(a) Signal strength drop (b) Handover (c) Increased RTT

Figure 4.8: Effect of network events on MPTCP performance.

switched video quality more than *four* times in one minute of playback (as shown in Figure 4.7c). Closer analysis via data traces reveals that MPTCP fails to take advantage of bandwidth gains using two LTE connections in high mobility scenarios and the resulting path utilization over both becomes excessively skewed. This decline in performance is primarily due to: *(i)* impact of the last-mile link changes with increasing mobility, and *(ii)* sub-optimal design and functioning of the default `minSRTT` scheduler.

Impact of last-mile link changes

A user experiences both changes in signal strength (δ_{signal}) and handovers as it moves closer/farther from the basestation. Figure 4.8 shows how MPTCP behaves as either connection experiences handovers and drops in signal strength (collectively denoted as *network events* in this thesis). We first focus our attention on Figure 4.8a and 4.8b which show a snippet of MPTCP throughput, RTT behavior as one of the connection experiences a signal drop and handover (vertical line) respectively. The figures also show the simultaneous throughput of individual TCP flows over each LTE connection. It is evident from the results that the occurrence of a network event induces a spike in RTT of that connection. In the case of signal drops the resulting RTT spikes can grow almost *twice* its average value, which in turn results in 22% decline in MPTCP throughput. On the other hand, for handovers, the RTT spikes are exceedingly intense compared to that observed in signal drops, reaching as large as *15 times* its average value (see Figure 4.8c).

Our analysis reveals the primary reason for the elevation in RTT in response to a network event is due to *bufferbloat* [37]. In Figure 4.8a, LTE Radio Resource Management (RRM) at the basestation switches to a lower link rate to accommodate for increased distance from the user (denoted by the decrease in signal strength). This leads to increased packet buffering

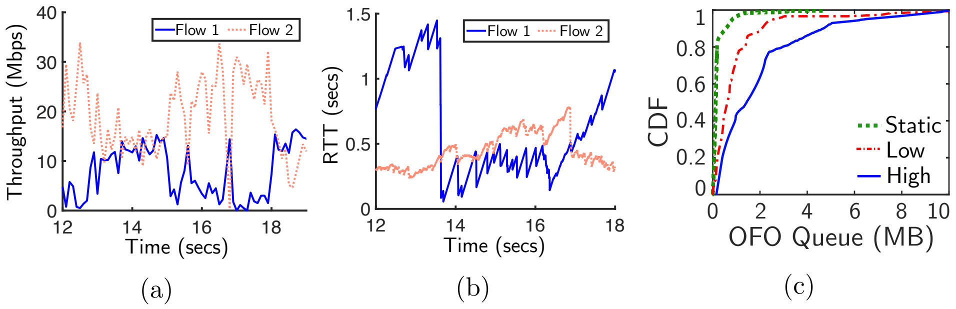


Figure 4.9: Behavior of `minSRTT` scheduler for varying RTT.

at either end of the last-mile link, i.e. user equipment and basestation, as the transport protocol remains unaware of the reduced link rate. The resulting packets on the path encounter additional queueing delay which causes spikes in RTT. In the case of handovers, the RTT spike is primarily due to temporary link outages as LTE follows a hard handover strategy (based on "break-before-make") [31].

At high speeds, both LTE connections can experience extremely frequent and often overlapping network events. The shaded region in Figure 4.8 reveals that the recovery time for the RTT of the affected subflow from a network event can last as long as *40 seconds* and impacts a significant share of subsequent packets sent on that path.

Analyzing the workings of MPTCP

An interesting observation from our measurements was the difference in behavior of MPTCP and TCP in response to network events. Figure 4.8 shows that signal drops and handovers affect MPTCP throughput much more than TCP. The results are quite intriguing as intuitively MPTCP is simply a combination of two parallel TCP connections. Further analysis revealed the root cause for this behavior to be the `minSRTT` scheduler and *out-of-order buffer* at the receiver.

Figure 4.9 shows the throughput and RTTs of both MPTCP subflows. Consider closely the RTT of Flow 1. The RTT captures in a lagged manner the impact of scheduling decisions on the subflow. High RTT values (12-14s) correspond to the time when the subflow was assigned packets by the scheduler while there was a significantly high queuing delay on the path. However, the scheduler remained impervious to the increased buffering as the RTT remained lower compared to the other path. The sharp dip around 14s captures the transition from when the flow was stopped being assigned packets by the scheduler due to high RTT. These packets encounter a lower queue occupancy which causes the dip. The scheduler notices the lower RTT and switches back to Flow 1, which results in increasing RTT from 16-18s. Apart from constantly playing catch-up with better performing subflow in case of frequent network events, `minSRTT` also causes significantly high out-of-order delivery at the receiver.

We can observe in Figure 4.8 that while a signal drop only impacts the performance of the affected subflow, handovers result in throughput drop on both LTE connections. On closer inspection, we find that the *receive buffer stalling* impacts MPTCP performance. With increasing mobility, both LTE links encounter frequent network events (often) simultaneously which results in elevated RTTs of both connections. The scheduling decisions by `minSRTT` does not account for additional intermittent delays on the subflow and the resulting packet arrives later than expected at the receiver. This increases the number of out-of-order deliveries at the receiver, as evident from Figure 4.9c. As perceived from the figure, MPTCP receive buffer size increases with increasing mobility speeds. Larger the buffer occupancy, longer a packet waits in the kernel to be delivered "in-order" to the application. If the network remains unstable for an increased duration, the buffer reaches its maximum capacity and the receiver can no longer accommodate the arriving packets. This remains the case until there is available space in the receive buffer. As a result, MPTCP experiences an overall drop in throughput over handover in Figure 4.8b.

4.3.3 Alleviating bufferbloat in MPTCP

The previous section shows that variable delays on access paths is the primary reason for degradation in MPTCP performance at high mobility. As `minSRTT` does not account for such delays in its scheduling decisions, large queue build ups can often happen at the last-mile, which in turn results in increased out-of-order deliveries at the receiver [120]. It becomes obvious that monitoring the presence of queue build-ups at either end of the last-mile link (user or access point) would help alleviate the problem. However, this is easier said than done. A majority of access points at the last-mile are heavily controlled and managed by their providers, and do not share information other than what is necessary for their functionality. Cellular network providers are well-known for carefully configuring their network according to promised QoE, and a majority of WiFi access points utilize proprietary firmware. Therefore, we focus our attention on the user equipment as it aims to be an integral component of edge clouds. In particular, we concentrate on providing solutions for assisting high-rate upload traffic, which is necessary for data sharing within an edge platform. In such a case, as discussed in the previous section, the edge server can encounter high occupancy in the local NIC queue. The scenario we tackle is not limited to cellular connections but spans all constrained wireless networks where a data packet may be required to wait for the channel to become available.

Before we describe our solution, we first dissect the functionality of `minSRTT` to identify the reason behind increased queueing delays. The default scheduler sends packets on the TCP connection with lowest SRTT to the receiver. The reliance on the SRTT metric for scheduling decisions is the cause of the problem. We explain this as follows. *Firstly*, per-packet instantaneous RTT itself is a delayed metric for gauging delays on the path as it is updated once an acknowledgment of the packet arrives at the sender. If the data packet encounters intermediate delays during delivery, such as in LTE, sender-side TCP (by default) smoothens it out with its pre-existing knowledge of RTT for that path. The instantaneous RTT is given much smaller weight (0.2 in practice). This exercise is carried out as TCP considers intermediate elevations in RTT as outliers and not representative of the behavior of that network. In this case, `minSRTT` continues to insert packets on its assumption of the better performing connection until the excessive delay due to queueing starts to reflect in the SRTT. We argue that due to this delayed feedback mechanism, MPTCP loses out on many opportunities of scheduling packets to subflow with lower queue delay.

Considering the motivations above, we design a custom scheduler for MPTCP, *QAware* which is published in Publication IV. QAware aims to maximize end-to-end throughput over all MPTCP subflows by better estimating segmented delays on the path. The core idea is explained as follows. Consider a simplified queue-theoretic abstraction of an end-to-end MPTCP connection, illustrated in Figure 4.10. The shaded region signifies the internals of sender device equipped with two TCP flows to the receiver. Packets scheduled over either TCP connection (denoted as service facility in the figure) encounters the following delays: (i) buffering at the underlying NIC queue with a service rate μ, (ii) service delay which includes access network and intermittent nodes on the path. If incoming packet rate from the application exceeds the service rate of NIC queue, a scheduled packet may find other packets waiting in the service facility. Bear in mind, we consider a scenario where other non-MPTCP protocols are also actively using the NIC alongside MPTCP e.g. UDP. In this case, an MPCTP packet waits for all other packets to finish service before it enters the server of the facility.

Algorithm 1 captures the workflow of QAware. Consider K service facilities indexed $1, \ldots, K$. Let facility k have a service rate of μ_k. Let

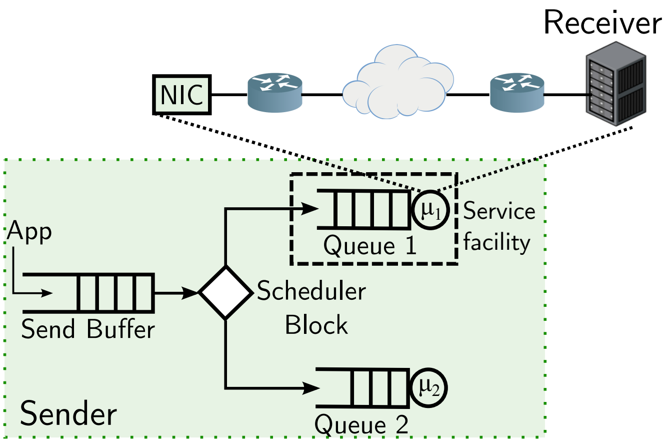


Figure 4.10: Queueing abstraction of an *end-to-end* MPTCP connection with two subflows.

Algorithm 1 QAware Algorithm

```
1: Inputs:
      Available Subflows SF$\in \{1, \ldots, n\}$
2: Initialize at packet arrival $P_k$:
      minService $\leftarrow$ 0xFFFFFFFF
      selectedSubflow $\leftarrow$ NONE
3: //The function below will return best subflow for packet $P_k$
4: for each subflow $\in$SF do
5:     $n_k \leftarrow$ queueSize(subflow)
6:     if $n_k \neq 0$ then
7:         $\Delta t \leftarrow$ sampling time
8:         $\Delta$packets $\leftarrow$ packets dequeued in $\Delta t$
9:         $W_k \leftarrow [1/(\frac{\Delta\text{packets}}{\Delta t})]n_k$
10:    else
11:        $W_k \leftarrow 0$
12:    $\widehat{W} \leftarrow \alpha\widehat{W} + (1-\alpha)W_k$
13:    $\hat{S}_k = [\text{RTT} - \widehat{W}]$
14:    $TS_k = (n_k + 1)\hat{S}_k$
15:    if $TS_k <$ minService then
16:        minService $\leftarrow TS_k$
17:        selectedSubflow $\leftarrow$ subflow
```

$n_k(t)$ be the number of packets waiting for service in facility k at time t. The expected service time $(1/\mu_k)$ of a packet in that facility can be estimated as follows. The RTT includes the time packet waits in the device driver queue of its assigned subflow before it starts service and the time it spends in service. Let W_i be the time the packet i waits in the queue which can be estimated by the sender by constantly monitoring the NIC queue. The time X_i that the packet spends in service begins when the packet enters the NIC for transmission and ends when a TCP ACK for the packet is received, i.e. $X_i = \mathrm{RTT}_i - W_i$. Let $\hat{S}_k$ be the current estimate of the average service time of facility k. On receipt of a TCP ACK for packet i, QAware updates

$$\hat{S}_k = \alpha\hat{S}_k + (1-\alpha)X_i, \tag{4.6}$$

where $0 < \alpha < 1$ applies weights to the last estimate of the average and the current service time (the default being 0.8). Therefore, QAware schedules to the TCP subflow k that satisfies least $\hat{S}_k$ for the packet (line 14 in Algorithm 1).

We implement QAware as a modular scheduler for MPTCP kernel version 0.93 and our implementation is open-source [91]. We evaluate and compare QAware performance for a variety of application workloads and network configurations via both network simulator-3 (ns-3) and real-world testing. Along with comparing our results with that from the default scheduling policy, i.e. `minSRTT`, we also compare the performance gains over other state-of-the-art schedulers recently proposed by researchers, such as Delay Aware Packet Scheduler (DAPS) [68], Blocking Estimation based Scheduler (BLEST) [36] and Earliest Completion First (ECF) [72]. The key results are as follows:

i) QAware achieves 40% increase in upload throughput compared to state-of-the-art when both subflows offer the same bandwidth. In case the paths have unequal bandwidth availability, the gains can be as high as 50%.

ii) For paths experiencing variable packet losses, QAware can detect and exploit both subflows efficiently and achieve 32% throughput improvement over `minSRTT`.

iii) QAware is also applicable in traditional CDN applications over the edge as it is found to improve the performance of video streaming services using DASH [57] by 10%.

Further details of QAware's implementation and evaluation can be found in Publication IV attached to this thesis.

Chapter 5

Platform solutions for Edge

The previous chapters of this thesis focused on providing working solutions for an open edge infrastructure which does not restrict itself to management and control of a specific cloud provider. The scale and service required for adopting edge computing for satiating the requirements of emerging applications begs for the participation of independent entities. The *Edge-Fog Cloud*, discussed in Chapter 3, can be considered as an amalgamation of such managing entities that offer their compute-capable resources. Several different managing entities that participate in the *Edge-Fog Cloud* are independent users (e.g. smartphone owners), telecom providers and small-to-mid cloud providers. For the rest of the thesis, we denote such entities as *Independent Edge Providers* (IEP). It must be noted that an IEP is not just limited to managing a single resource but can host and maintain multiple servers in its authority (e.g. cellular provider hosting *Fog* servers).

The solutions discussed in Chapter 4 were designed to enhance the functionality of servers managed by a single IEP. To elevate the edge cloud as a viable platform for hosting application services, transparent utilization of edge servers across multiple operational IEPs is required. However, resource cooperation across authoritative boundaries presents a unique research challenge. While *Edge-Fog Cloud* envisions a future where multiple IEPs are aware of each other's existence which allows them to collaborate; in reality, an edge provider operates in a siloed environment and remains agnostic of other IEPs operating on the platform. In absence of standardized technologies which allow seamless interaction of servers belonging to different IEPs, the *Edge-Fog Cloud* platform will remain unable to support the need of application developers to run their services in a flexible manner. Consider the following scenario. An application owner deploys its application, composed of both back-end and front-end services, on *Edge-Fog Cloud* facility in the northern USA. As the popularity of the application grows, user crowds from the other half of the earth, e.g., southeast Asia, starts subscribing to the application. Situations such as this are commonplace as the Internet becomes cheaper and more accessible to users across the globe [58]. Let's assume that the cloud provider only offers edge facilities in the USA but not in Southeast Asia. Attempts of administrative personnel to find local IEPs operational in that region are slow and inefficient [142]. Such endeavors are further affected by delays in budget negotiation and ensuring a homogeneous software stack for supporting the application. Since resulting QoS becomes low, the application provider stands at significant risk of losing this new audience.

While the example above extrapolates the problems of future edge deployments, we argue that the scenario is quite prominent even in the current cloud computing landscape. Multiple cloud providers (analogous to IEPs) such as Amazon, Google, Microsoft etc. allow application owners access to their infrastructure via specially designed protocols and technologies which bear little-to-no resemblance to those offered by another provider. This results in a minimal collaboration opportunity between the infrastructure of different providers and restricts the application developers to use technologies of a single cloud platform. The solutions we present in this chapter address several such challenges described above.

- We design an open framework for edge clouds, **Elastic Extensible Cloud** (ExEC), which unifies the infrastructure of multiple IEPs in the *Edge-Fog Cloud*. ExEC enables the cloud provider hosting the application to detect emergent request flows for that service and discover availability of IEPs near the request origin to allow for better serviceability. The platform also facilitates application owners to transparently negotiate with IEPs such that both parties can comply with the required terms of service. ExEC is published in Publication V attached to this thesis [141].

- To utilize the resources across multiple IEPs discovered by ExEC, we develop **Intelligent Container (ICON)** which leverages existing container technologies [25, 33, 42, 43, 86] for deploying application services. However, ICON's objectives differ from such technologies in a significant way. While existing containerized solutions require constant monitoring for service migration and replication, ICON "intelligently" discovers and autonomously moves virtualized services to edge servers on the platform. This self-operation is derived from high-level control thresholds set by the application owner and significantly reduces the management overhead compared to traditional methods.

> The salient feature of ICON is to enable the migration of services across multiple IEPs, making it desirable for application owners who want to utilize resources from multiple cloud providers without any configuration hassles. ICON is published in Publication VI attached with to thesis [140].

We design our platform solutions over open-source, well-maintained standardized protocols such that their applicability and usability in real environments can be maximized.

5.1 Unifying Multiple Edge Providers

The primary aim of ExEC is to enable all IEPs, such as ISPs managing Mobile Clouds (MC), crowdsourced edge resources and cloud providers, to operate on a homogeneous *Edge-Fog Cloud* platform despite varied configuration, ownership and management. For this purpose, we rest our solution on the assumption that interested IEPs are willing to support all necessary technologies for ExEC operation. We believe that the assumption is well-placed as it allows interested IEPs to operate in a more substantial infrastructure umbrella, which in turn increases their extent of revenue.

Figure 5.1 illustrates the operation of ExEC. ExEC employs an edge orchestrator within existing *Edge-Fog Cloud* environment, which allows ExEC to build its own view of the network. Overall operation is composed of two stages, i) discovery of on-path IEPs, and ii) negotiation and contractual agreements between involved entities. For onloading services onto the selected edge servers, ExEC transfers the control to ICON.

5.1.1 Discovery of edge providers in the network

Figure 5.2 shows the discovery mechanism of ExEC. In its first phase, ExEC orchestrator monitors the location and density of incoming user requests for a particular application. The objective of the orchestrator in this phase is to discover best-suited IEPs (in terms of end-to-end latency from the user) for offloading the service. If the orchestrator observes requests from a previously unseen subnet, it initiates a network tomography procedure. The procedure is performed by running a `traceroute` [76] to the user's IP. This allows the orchestrator to record the latencies of the user to on-path routers along with their IP addresses (denoted by path **"Traceroute"**).

ExEC assumes that the IEPs register themselves in their local DNS domain and their resources as a DNS service (SRV) record [61] with a consistent service name (e.g. `edge`). The assumption is essential to the

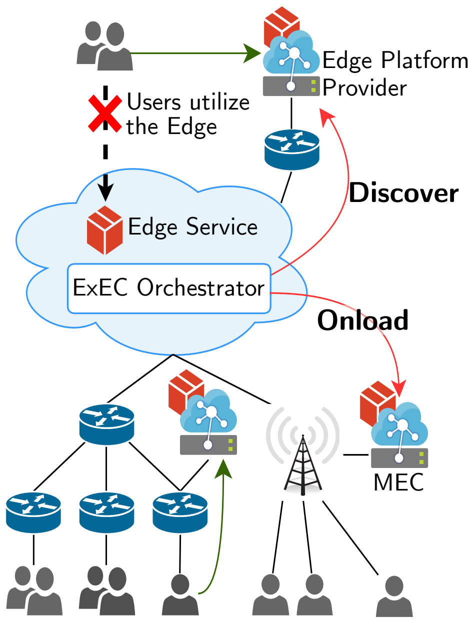


Figure 5.1: Overview of ExEC operation.

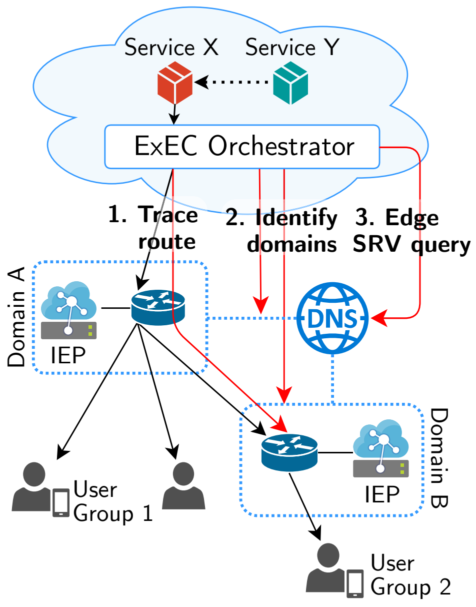


Figure 5.2: Discovery of IEP in ExEC.

effective functioning of ExEC discovery mechanism as it allows IEPs to become discoverable on the network. Considering that IEPs satisfy this requirement, ExEC identifies DNS domains from the IP addresses collected in the first step through DNS PTR records or by augmenting `whois` information. This step is denoted by the path **"Identify domains"** in Figure 5.2 which shows that ExEC discovers two domains on path, *Domain A* and *Domain B*. ExEC further queries the DNS for resources for `edge` service in the discovered zones (path **"Edge SRV query"**) for which a possible reply can be as follows.

```
_edge._tcp.domainA.com.  86400 IN SRV 10 30 5060
                                   mecServer.domainA.com.
mecServer.domainA.com.   86400 IN A 192.168.121.30
```

The SRV record for DNS *Domain A* shows the presence of a TCP server which is accessible through IP address 192.168.121.30 and port 5060. ExEC orchestrator uses this information to build a topology, with intrinsic detail (IP address, user latency) of all edge servers in discovered domains, which lie on the path to end-users. Based on this, ExEC can filter a set of edge servers (irrespective of their managing IEP) based on the minimum latency requirement imposed by the application owner.

5.1.2 Negotiation and agreements with edge providers

Once ExEC collects the information of potential IEPs for deploying the service, it initiates the negotiation procedure by entering a contractual agreement with the IEP. The communication for agreement is divided into two broad steps.

Initial Negotiation. The orchestrator contacts the management service of the target IEP with a `hello` message. The message contains several service-level preferences for optimal application operation e.g. possible time of operation, configuration requirements etc. If the IEP agrees with proposed likings, it acknowledges the orchestrator with available time slots for deployment and maximum hardware capability which can be exploited by the service. If ExEC finds IEP constraints in line with the application QoS, it initiates the process to form a contractual agreement.

Agreement. The agreement between IEP and orchestrator binds both parties for the duration of application service. In order to form an agreement between the two parties, the de-facto method is to utilize a third trusted entity. This role is usually fulfilled by a third-party broker [27, 55] which can be trusted by all participating entities. However, in the case of edge computing, the sheer scale and frequency of incorporating IEPs calls for a process which can be distributed and conclude agreements on-the-fly. Therefore, ExEC utilizes smart contracts to handle payments and concordance [12, 39, 41]. Specifically, smart contracts are blockchain-based self-executing contracts wherein the participants achieving consensus verify the correctness of the execution. Unlike centralized broker systems, smart contract presents itself as a peer-to-peer solution that can achieve agreement on an Internet scale.

The IEP sends the orchestrator-required details for formulating a contractual agreement, e.g. price for service, chosen time slot, ownership verification etc.. Upon receiving the details, the orchestrator calls the smart contract method for initiating an agreement with the IEP. At the end of the contractual agreement process, ExEC orchestrator green-lists the IEP for service placement with a positive acknowledgment message to the IEP.

As the contractual agreement also involves a transfer of funds between parties, there is a possibility of fraud in the system. One such example is as follows. A malicious IEP can advertise fake configurations to the orchestrator at agreement time but does not fulfill its promises in the Service Level Agreement (SLA) at runtime. To avoid such deceptions, future versions of ExEC can incorporate research solutions which allow smart contracts to hold off payment to the seller until the buyer verifies and acknowledges service satisfaction [130].

5.2 Service Placement on IEPs

Once ExEC partners with IEPs to host application services, the next obvious question that needs to be answered is, *where and how do we deploy the application service amongst all discovered edge servers?* The question has been well-explored in the context of Service Function Chaining (SFC) wherein the application services are virtualized using container technologies e.g. Docker [33], Kubernetes [86], Rocket [25] etc. and deployed in virtual infrastructure within a single cloud platform [95]. In such environments, the primary objective of placing application services is to maintain a "service chain", i.e. multiple services follow a hierarchical ordering depending on goals of the application. For example, Netflix may distribute its component services like parental control, traffic encryption and acceleration, in that particular order, for optimal functionality. Even in controlled datacenter environments, formation, migration and replication of such complex, end-to-end services require fine-grained control, which usually involves solving an optimization problem over global view of the infrastructure [15]. The solution to service placement requires optimization over several constraints such as budget, delay to end-users, delays between services, application user QoS and SLA requirements etc. [117]. Therefore, the complexity of the problem can become quite large and may incur a significant management overhead for the cloud provider.

5.2.1 Limitations for placing services on *Edge*

Figure 5.3 illustrates an end-to-end service chain comprising of two distinct services over the *Edge-Fog Cloud* infrastructure composed of two IEPs. Both IEPs operate in different DNS domains, wherein IEP1 manages three edge servers while IEP2 hosts two. The figure also shows that IEP1 is closest to the clients of the application and, therefore, can offer better QoS than IEP2. Let us assume that IEP1 demands a higher price for service deployment than IEP2. The chain terminates at the *Fog* platform, which is composed of two servers. While there exist several possible service placement combinations, we show two probable examples, SFC1 and SFC2, in the figure. SFC1 deploys both services within the IEP1 platform and can, therefore, enjoy low latency to the client along with the homogeneous management by the same provider. However, the benefits accompany the higher costs charged by IEP1. SFC2, on the other hand, distributes the services over both providers so that the front-facing service enjoys lower latency on IEP1 while the other service operates in IEP2 infrastructure. While SFC2 allows the application owner to limit its placement expenditure, it suffers

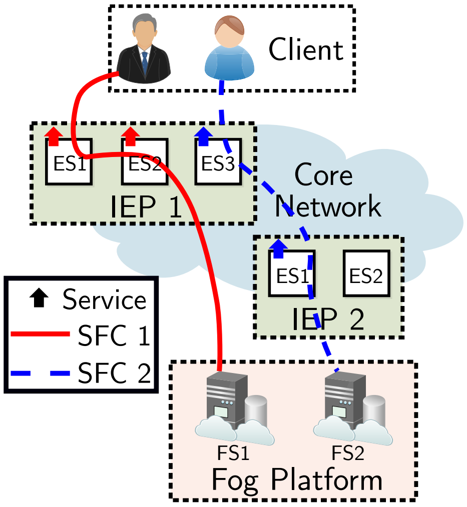


Figure 5.3: Example of application service placement on *Edge-Fog Cloud*.

from inflexibility to adapt to variable user requests due to management overheads for incorporating two different providers.

Considering the example above, we summarize several constraints for deploying services in *Edge-Fog Cloud*.

1. The *Edge-Fog Cloud* infrastructure is highly susceptible to changes – with IEPs constantly joining and leaving the platform. While ExEC discovers IEPs that lie on the path to emerging client requests, existing techniques require a "global" view of the infrastructure for optimal service placement.

2. Unlike datacenters, crowdsourced *Edge* resources can offer significant heterogeneity in their hardware capability, e.g. in processing, network distribution, deployment costs etc. Incorporating this in existing placement problems adds further complexity to deployment decisions. This can result in significant placement delays and, therefore, impact client QoS.

3. As discussed in the example, the coordination overhead between different IEPs participating in *Edge-Fog Cloud* can cause significant delays in service migration, replication and termination.

The solutions presented in this chapter implicitly assume that all edge servers participating in the platform must be able to support container

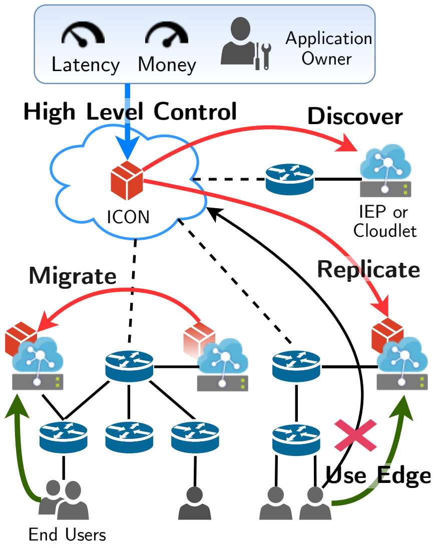


Figure 5.4: Overview of ICON lifecycle.

virtualization technologies. However, this may not be true for server types discussed in approaches such as [99]. In such cases, we propose the following alternative. The independent edge provider utilizes a *resource orchestrator*, which is a server capable of supporting virtualized application services. While the orchestrator acts as the face for deploying applications for that platform, it utilizes task deployment algorithms (e.g. LPCF proposed in Chapter 3) to distribute application tasks on underlying edge resources.

5.2.2 Intelligent Container Overlays (ICON)

Considering the constraints above, there is a need for a flexible service management scheme which offers resilience and adaptation over infrastructure variability of *Edge-Fog Cloud*. To this purpose, we design Intelligent Containers (ICON) [140], the overview of which is illustrated in Figure 5.4.

As the name suggests, ICONs are *self-managing virtualized services* (such as containers, virtual machines, unikernels etc.) which can "intelligently" migrate/replicate/terminate between different IEPs participating in the *Edge-Fog Cloud*. ICONs remove the management overhead for application owners as it opts for a completely decentralized decision and control scheme. ICONs form a tree-shaped logical control overlay, the root of which is the origin application service. The tree grows organically downwards as ICONs deploy replicas on other servers in *Edge-Fog Cloud*, each replica becoming a child of its parent. Such a hierarchical control structure allow ICONs to employ fine-grained management over services on the platform, one in which the communication flow and control is limited to single hop. To elucidate, the child ICON reports its local information (request per second, cost incurred) only to its immediate parent, which aggregates it with its own information and sends it further upstream. Every ICON can independently take one of the following decisions: (i) spawn a child to another server, (ii) migrate to a new location or (iii) terminate itself. This allows each ICON to determine its own state without being overwhelmed by the responsibility of managing its children. The self-managing overlay structure of ICON frees the application owner from the vice of "micromanagement" and allows the application services to react faster to environmental changes.

We now briefly describe the lifecycle of ICON, a high-level description of which is shown in Figure 5.4. The life of an ICON can be divided into three phases: *Initialization*, *Active* and *Termination*.

Initialization. The application owner deploys each service in the chain as a virtualized container managed by ICON. The owner allocates a fixed budget for ICON operation along with the SLA requirements, which impacts the extent of replication the service can achieve in its lifetime.

Active. Each ICON monitors the frequency, location and pattern of incoming user requests and remains deployed at the current location until the request density remains consistent. If an ICON detects a change in the request pattern, it requests ExEC to discover IEPs in subnets close to new request location. ExEC replies with a renewed list of potential IEPs which have agreed to host the service[1] along with the configurations of the edge servers in their management. ICON calculates *utility* ($\mathcal{U}_j$) of each edge server as follows.

$$\mathcal{U}_j = (1-w)b_j + w\sum_{\forall i \in S} \lambda_i l_{i,j} \tag{5.1}$$

Equation 5.1 shows the utility function for edge server at location j which serves λ_i requests per second from a subnet i. b_j is the budget cost of running ICON at location j, $l_{i,j}$ is the latency experienced by users (or ICON for service earlier in the chain) from subnet i if served from network location j. w is the weight parameter ($0 \leq w \leq 1$) which allows the application owner to tune the priority of cost or latency objective. Depending on the available budget, the utility of the current edge server and utility of the target edge server, ICON can perform one of the following operations.

[1]We assume that all participating edge servers support container technologies.

i) **Replication**: If both current and target edge servers have high utility, ICON may choose to deploy a replica of itself at the target edge server. The resulting container is an independent ICON which takes a percentage of the budget from origin ICON (depending on the ratio of utilities of both edge servers) for its own future decisions.

ii) **Migration**: If the utility of the current server is far less than the target location, the ICON migrates itself to the new location and retains its allocated budget. Any user requests arriving at the previous ICON location is redirected to the new server by a *redirection stub*.

For migration and replication, ICONs utilize several well-explored techniques which cover all applicable corner cases such as application state, client redirection etc. [17,52,74,88,134].

Termination. If the utility of ICON dips below the threshold specified by the application owner and ICON is unable to find any new location which requires service, it decides to terminate itself. Although this allows the application owner to refocus the budget requirements of ICONs in locations which require more service, it also opens up two obvious issues in ICON operation. (i) *how do we maintain the tree-like management overlay if the parent ICON decides to terminate?*, and (ii) *what happens to any future requests which were initially to be handled by the now-extinct ICON?*

We tackle the first question by reshuffling the control order of underlying branches of the affected ICON. The terminating container assigns the control of its descendants to its parent, who is then informed of the change in hierarchy. The original budget of the terminating ICON is redistributed amongst the children depending on their (and their sub-branches) utility (calculated via Equation 5.1). For the second issue, ICONs utilize a redirection stub, similar to the one used in the migration phase. As the name suggests, the stub redirects any incoming client requests for terminated ICON to its previous parent.

In case an ICON exceeds its budget requirement but still requires further service, it contacts its parent to allocate more budget for an effective application operation. If the parent is unable to satisfy the demand, it forwards the request up the tree until it reaches the application owner who can assign more money to be redistributed ICONs. Alternatively, if the owner decides not to satisfy the budget requirement, there may be overall re-allocation of budget across all branches, which may result in migration and termination of near obsolete ICONs.

The novel design of ICON lifecycle allows the containers to grow towards the locations which observe larger request densities and prune obsolete branches for minimizing overall deployment expenditure. Although the control operation of ICON is self-reliant, it is also susceptible to several operational and security flaws. One open challenge is deploying ICONs in an untrusted/unmanaged environment. While the IEP servers are protected by virtualization, ICONs are vulnerable to malicious IEPs examining their code and application data. One solution is to allow application owners to express high security requirement in their configuration which curtails ICON deployment to edge servers which support Trusted Execution Environments (TEE) [115]. With TEEs, the application is executed in a protected memory region of the server which even non-accessible for users with root privileges. Although the TEE technology is still in its nascent stages of developement and comes with several drawbacks [7], it shows promise as it is a prominent research area which is supported by leading processor manufacturers such as Intel [49], ARM [29] etc.

To summarize, in this chapter we presented two novel platform solutions for *Edge-Fog Cloud*, namely ExEC and ICON, which opens an opportunity for third-party edge providers and makes it possible for applications to unfold dynamically towards the edge without any administrative overhead. Both solutions utilize existing standardized technologies for their operation and allow for easier integration of edge clouds in the current cloud-dominant environment. Not only are the solutions presented based on already existing protocols and technology, thus making them easy to incorporate, they are also designed as modular blocks and can easily be integrated in emerging virtualization technologies.

Chapter 6

Conclusion

This thesis proposed several solutions necessary for the adoption of edge computing in the current cloud-dominant environment, which are summarized in this chapter. We start by revisiting the research questions posed at the beginning of the thesis in Section 1.1 of Chapter 1. We further discuss shortcomings of the presented solutions and avenues for future work.

6.1 Research Questions Revisited

RQ1. **Where should the cloud providers install compute servers in the physical world to satisfy the application requirements at the "edge"?**

To emulate datacenter-like performance, edge servers installed by a cloud provider require constant monitoring and maintenance. We find that existing basestation locations in an area offers a good fit for future edge deployments as it is already heavily managed by cellular providers. However, provisioning all cellular towers with an edge server is not worthwhile for cloud providers. While deployment of cell towers is usually evenly distributed in the area for consistent connectivity, locations of server deployment must be driven by the need to maximize the density of request resolution and server utilization. To this purpose, we present Anveshak in Section 3.2 of Chapter 3 which prioritizes basestation locations for deploying servers by predicting the future density of user requests arising at that location along with a number of competing crowdsourced resources in the vicinity.

RQ2. **How can independent entities enroll their compute resources in an existing edge cloud platform?**

Non-conventional compute resources such as WiFi AP, smart speakers, smartphones, desktops etc. can fulfill the compute requirements of edge applications while maintaining a low latency to the end-user. This makes their presence in edge clouds crucial for supporting several latency-critical applications. In Section 3.1 of Chapter 3, we introduce *Edge-Fog Cloud* architecture which merges the functionality of independent resources with a back-end of managed edge servers operating in a region. Our solution unifies the benefits of several previous edge cloud approaches and allows multiple cloud providers and heterogeneous edge servers to operate on a single platform.

RQ3. **How to utilize availability and variability of edge servers for computing application tasks with different requirements?**

In Section 4.1 of Chapter 4, we propose two task allocation frameworks designed for clouds, *Least Processing Cost First* (LPCF) and *energy-efficient LPCF* (eLPCF). Both frameworks distribute application jobs on edge servers by actively considering the variability in application tasks and edge server characteristics. Every server may be equipped with different hardware which results in several differences in processing, network and energy utilization while computing the same task. Similarly, an application can be distributed into several tasks which may require different levels of processing and coordination between servers. We design our algorithms to discover the best subset of resources which can minimize both task completion time and the energy used by the servers throughout task processing.

RQ4. **What techniques should be employed for pre-caching computational data within edge servers to improve system promptness?**

We find that fetching the required data at the start of every computation can result in significant processing delays, due to network latency between the edge server and the *Data Store*. However, pre-caching data at the edge is a challenge due to shorter temporal relevance of the data, limited cache size and undetermined number of servers for each arising task. In Section 4.2 of Chapter 4, this thesis provides an edge caching mechanism which leverages the characteristics of pre-existing data in a server's cache to predict and store the prerequisite data for coming computations. Along with maximizing the probability of fetching the required data from neighboring servers instead of the centralized cloud, our technique also provides causal coherency to in-computation cached data at the edge.

RQ5. **Can existing network technologies available at the edge support the requirements imposed by end-applications for optimal performance?**

We answer this question semi-negatively. We find that edge-dependent applications can have significantly varying QoE requirements. For example, even servers with a limited capacity placed somewhere in the network can support the demands of applications such as IoT, smart city and smart delivery. On the other hand, AR/VR, automated vehicles, etc. require constant connectivity with edge servers for optimal operation. Section 4.3 of Chapter 4 shows that almost all edge-dependent applications demand mobility support from edge clouds. Our measurements show that while current wireless technologies, such as WiFi, LTE etc., can theoretically satiate this need, they are often unable to support the strict application QoS due to significant degradation in link performance at high speeds. Analysis of our results finds the reason to be excessive last-mile link changes.

RQ6. **How do we assure datacenter-like network behavior over edge servers which operate on a public wireless network?**

To provide larger bandwidth and reliability over wireless networks, this thesis explores the possibility of exploiting multiple network interfaces of an edge server simultaneously using Multipath TCP (MPTCP). However, our studies show that mobility impacts MPTCP performance more severely than regular TCP, primarily due to excessive packet buffering on one of the paths. To this purpose, we develop QAware, which is a cross-layer scheduler for MPTCP. QAware actively estimates link delays due to on-path packet buffering and proactively counters it by switching to the better performing connection. Our evaluation in real environments shows that QAware achieves more than 50% increase in data rates compared to the state-of-the-art.

RQ7. **How can existing cloud virtualization technologies be exploited to optimize the application service deployment in edge clouds?**

Adoption of existing cloud protocols and technologies in edge computing is necessary for integrating both cloud architectures. However, we show that due to the existence of multiple cloud providers at the edge (both independent and managed), a direct port of current virtualization solutions is not possible. One of the problems we address in this thesis is the discovery of available edge server locations in the network with their latency to end-users. In Section 5.1 of Chapter 5, we provide a platform solution, ExEC, which enables this interaction between cloud and edge platforms. We design ExEC using existing protocols and technologies, and aim to define practices for integrating edge providers on a single unifying platform on-the-go.

RQ8. **How can edge cloud ensure the promised Quality-of-Service despite the significant variability in user requests and infrastructure hardware?**

Although ExEC enables discovery of edge servers close to the users, migration of containerized services to such servers requires the assistance of the application owner. As the user requests at the edge can be fluctuating and localized, existing container protocols cannot cope with such variations. In Section 5.2, this thesis provides ICONs which are self-managing container entities which can make independent decisions to migrate and replicate to new servers without the guidance of the application owner. Moreover, ICONs operation is not limited within a single edge cloud platform as it is intended to cross administrative borders of edge providers and benefit from decentralized control overlay.

RQ9. **How can independent edge providers generate revenue at par with cloud providers for their service?**

This thesis takes the viewpoint that the availability of independent edge servers is key to guaranteeing the efficient performance of edge computing. The technical solutions we propose in this thesis enables a symbiotic relationship between the existing cloud and independent edge providers. However, we also acknowledge that without a monetary reward for service, the benefits of hosting personal resources at the edge are minimized. To this end, we design an agreement module for ICONs which enables cloud providers to settle transactions with independent providers on the fly. The framework utilizes smart contract technology to ensure a secure, transparent, and fault-tolerant book-keeping.

6.2 Future Work

This thesis has presented several techniques which allow for a full-fledged adoption of edge computing in the current infrastructure. While our solutions were designed to complete their required operation, some of our approaches can use more granularity. The infrastructure protocols provided in this thesis require tweaking depending on the exact requirements of the edge cloud provider (prioritizing bandwidth over processing in task allocation, supporting data traffic for specific traffic type etc.). Specifically, the task deployment algorithms assume a direct 1:1 mapping between application jobs and the number of servers involved in the computation. This assumption limits the extent of edge cloud utilization as monolithic jobs only require a single "fat" server for computation. Future versions of the algorithm should take design cues from existing cluster computing frameworks, such as Spark [138] which enable an unequal server-to-job mapping.

Similarly, our platform solution should allow existing application owners to develop required softwares for the edge. For this, we are aware of the shortcomings in ICON design. Firstly, the control interaction of ICON is simplified and requires a complex overhaul, of which multiple possibilities exist. One design choice is to group services by type and have different optimization goals for each group, e.g. creating latency-critical and backend service groups of ICONs. Another possibility is grouping based on responsibility, such as services catered towards crowds in the USA and those in Europe. Furthermore, the one-hop communication model in ICONs may not support complex interactions required for services with large dependencies. Future work involves allowing an application owner to configure the workings of the communication model as per their application requirement.

As edge computing is an up-and-coming concept, it requires significant attention from the research community to resolve several pressing issues obstructing its real-world adoption. Edge servers equipped with specifically designed hardware and communication technologies can impart huge improvements in application performance. Utilizing 5G, fiber-wireless (FiWi) and FPGA-based chipsets as networking and processing components respectively are few such directions worth exploring. The issue of security and trust in edge clouds is also of paramount importance. While this thesis allows integration of crowdsourced resources, we acknowledge the presence of security holes due to malicious edge servers attacking the system from within. To subvert this situation, we proposedly utilized hardware enclaves (i.e. TEE) in our platform solutions to protect the applications from attack. However, TEE's are known to instill significant performance overheads on applications due to their restrictive operation [7]. Furthermore, it does not subvert infrastructure attacks on other edge servers in the system such as sharing wrong results, denial-of-service, network overload etc. Further research is required for ensuring privacy and security of application providers and edge resource vendors for an effective edge cloud infrastructure.

References

[1] Anant Agarwal, Richard Simoni, John Hennessy, and Mark Horowitz. An evaluation of directory schemes for cache coherence. In *ACM SIGARCH Computer Architecture News*, volume 16, pages 280–298. IEEE Computer Society Press, 1988.

[2] 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.

[3] Akamai. Prefetching. ”`https://developer.akamai.com/learn/Optimization/Pre-fetching.html"`, 2014.

[4] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM ’08, pages 63–74, New York, NY, USA, 2008. ACM.

[5] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74. ACM, 2008.

[6] Apple Inc. Use Multipath TCP to create backup connections for iOS. ”`https://support.apple.com/en-us/HT201373,2017"`, 2017.

[7] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O’keeffe, Mark L Stillwell, et al. SCONE: Secure linux containers with intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 689–703, 2016.

[8] Rihards Balodis and Inara Opmane. History of data centre development. In *Reflections on the History of Computing*, pages 180–203. Springer, 2012.

[9] Ketan Bhardwaj et al. Fast, scalable and secure onloading of edge functions using airbox. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, SEC '16, 2016.

[10] Olivier Bonaventure, Mark Handley, and Costin Raiciu. An overview of Multipath TCP. *; login:*, 2012.

[11] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[12] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 2014.

[13] Yu Cao, Songqing Chen, Peng Hou, and Donald Brown. Fast: A fog computing assisted distributed analytics system to monitor fall for stroke mitigation. In *2015 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pages 2–11. IEEE, 2015.

[14] Jason Carolan, Steve Gaede, James Baty, Glenn Brunette, Art Licht, Jim Remmell, Lew Tucker, and Joel Weise. Introduction to cloud computing architecture. *White Paper, 1st edn. Sun Micro Systems Inc*, 2009.

[15] Alberto Ceselli, Marco Premoli, and Stefano Secci. Mobile edge cloud network design optimization. *IEEE/ACM Transactions on Networking (TON)*, 25(3):1818–1831, 2017.

[16] Lucas Chaufournier, Prateek Sharma, Franck Le, Erich Nahum, Prashant Shenoy, and Don Towsley. Fast transparent virtual machine migration in distributed edge clouds. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 10. ACM, 2017.

[17] Yang Chen. Checkpoint and restore of micro-service in docker containers. In *2015 3rd International Conference on Mechatronics and Industrial Informatics (ICMII 2015)*. Atlantis Press, 2015.

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

[19] Christoph Paasch et al. Multipath TCP Linux. "www.multipath-tcp.org/", 2019.

[20] Christoph Paasch et al. Multipath TCP Linux version 0.94. "http://multipath-tcp.org/pmwiki.php?n=Main.Release94", 2019.

[21] Richard Church and Charles R Velle. The maximal covering location problem. *Papers in regional science*, 32(1):101–118, 1974.

[22] Ciena. 12 mind-blowing data center facts you need to know. "https://www.ciena.com/insights/articles/Twelve-Mind-blowing-Data-Center-Facts-You-Need-to-Know.html", 2018.

[23] CISCO. Edge server data spec sheet. "https://www.cisco.com/c/en/us/products/collateral/routers/829-industrial-router/datasheet-c78-734981.pdf".

[24] Consortium. OpenEdge Computing Initiative. "https://www.openedgecomputing.org/", 2019.

[25] CoreOS. Rocket containers. "https://coreos.com/rkt/", 2019.

[26] Yong Cui, Hongyi Wang, Xiuzhen Cheng, and Biao Chen. Wireless data center networking. *IEEE Wireless Communications*, 18(6):46–53, 2011.

[27] Antonio Cuomo, Giuseppe Di Modica, Salvatore Distefano, Antonio Puliafito, Massimiliano Rak, Orazio Tomarchio, Salvatore Venticinque, and Umberto Villano. An SLA-based broker for cloud infrastructures. *Journal of grid computing*, 11(1):1–25, 2013.

[28] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[29] ARM Developers. ARM TrustZone. "https://developer.arm.com/ip-products/security-ip/trustzone", 2019.

[30] T. Dillon, C. Wu, and E. Chang. Cloud computing: Issues and challenges. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 27–33, April 2010.

[31] Konstantinos Dimou, Min Wang, Yu Yang, Muhammmad Kazmi, Anna Larmo, Jonas Pettersson, Walter Muller, and Ylva Timner. Handover within 3GPP LTE: Design principles and performance. In

2009 IEEE 70th Vehicular Technology Conference Fall, pages 1–5. IEEE, 2009.

[32] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[33] Docker. Docker containers. ”`https://www.docker.com/`, 2019.

[34] Utsav Drolia, Rolando Martins, Jiaqi Tan, Ankit Chheda, Monil Sanghavi, Rajeev Gandhi, and Priya Narasimhan. The case for mobile edge-clouds. In *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*, pages 209–215. IEEE, 2013.

[35] Dave Evans. The Internet of Things: how the next evolution of the internet is changing everything (april 2011). *White Paper by Cisco Internet Business Solutions Group (IBSG)*, 2012.

[36] S. Ferlin, . Alay, O. Mehani, and R. Boreli. BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, 2016.

[37] S. Ferlin-Oliveira, T. Dreibholz, and . Alay. Tackling the challenge of bufferbloat in multi-path transport over heterogeneous wireless networks. In *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)*, 2014.

[38] Forbes. How much data do we create every day? the mind-blowing stats everyone should read. ”`https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/"`, 2018.

[39] The Ethereum Foundation. Ethereum smart contract. ”`https://www.ethereum.org/`, 2019.

[40] The Linux Foundation. Kernel virtual machine. ”`http://www.linux-kvm.org`, 2018.

[41] The Linux Foundation. Hyperledger smart contract. ”`https://www.hyperledger.org/`, 2019.

[42] The Linux Foundation. LXC linux containers. ”`http://lxc.sourceforge.net`, 2019.

[43] The Linux Foundation. LXD linux containers. ”`https://linuxcontainers.org/lxd/`, 2019.

[44] Xiaofeng Gao, Linghe Kong, Weichen Li, Wanchao Liang, Yuxiang Chen, and Guihai Chen. Traffic load balancing schemes for devolved controllers in mega data centers. *IEEE Transactions on Parallel and Distributed Systems*, 28(2):572–585, 2016.

[45] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review*, 45(5):37–42, 2015.

[46] Tuan Nguyen Gia, Mingzhe Jiang, Amir-Mohammad Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen. Fog computing in healthcare internet of things: A case study on ecg feature extraction. In *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pages 356–363. IEEE, 2015.

[47] Gizmodo. Tesla autopliot crash. ”`https://gizmodo.com/tesla-autopilot-malfunction-caused-crash-that-killed-ap-1834453661/`, 2019.

[48] Google. Stadia: Cloud gaming. ”`https://stadia.dev/"`, 2019.

[49] James Greene. Intel trusted execution technology: White paper. ”`http://www.intel.com/txt`, 2012.

[50] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 39(4):63–74, 2009.

[51] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 75–86. ACM, 2008.

[52] Kiryong Ha, Yoshihisa Abe, Thomas Eiszler, Zhuo Chen, Wenlu Hu, Brandon Amos, Rohit Upadhyaya, Padmanabhan Pillai, and Mahadev Satyanarayanan. You can teach elephants to dance: agile VM handoff for edge computing. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 12. ACM, 2017.

[53] Bo Han, Feng Qian, Lusheng Ji, and Vijay Gopalakrishnan. MP-DASH: Adaptive video streaming over preference-aware multipath. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, pages 129–143, New York, NY, USA, 2016. ACM.

[54] B. Hayes, Y. Chang, and G. Riley. Controlled unfair adaptive 360 vr video delivery over an MPTCP/QUIC architecture. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2018.

[55] Eric J Horvitz, Harold L Cochrane, Rene A Vega, and Angel S Calvo. Cloud computing resource broker, 2012. US Patent 8,244,559.

[56] Junxian Huang, Feng Qian, Yihua Guo, Yuanyuan Zhou, Qiang Xu, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. An in-depth study of LTE: Effect of network protocol and application behavior on performance. *SIGCOMM Comput. Commun. Rev.*, 43(4):363–374, August 2013.

[57] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM Conference on SIGCOMM.*

[58] Influencer Marketing Hub. The incredible rise of tiktok. "`https://influencermarketinghub.com/the-incredible-rise-of-tiktok-tiktok-growth-visualization/`, 2019.

[59] J-P Hubaux, Thomas Gross, J-Y Le Boudec, and Martin Vetterli. Toward self-organized mobile ad hoc networks: the terminodes project. *IEEE Communications Magazine*, 39(1):118–124, 2001.

[60] Jaehyun Hwang, Steven H. Low, Anwar Walid, and Qiuyu Peng. Balanced Linked Adaptation Congestion Control Algorithm for MPTCP. 2016.

[61] Internet Engineering Task Force (IETF). DNS SRV Record - RFC 2782. "https://tools.ietf.org/rfc/rfc2782.txt, 2000.

[62] Industrial Internet Consortium. OpenFog Initiative. "https://www.iiconsortium.org/index.htm", 2019.

[63] Telecom Italia. Open big data challenge. "https://dandelion.eu/datamine/open-big-data/, 2014.

[64] Roy Jonker and Ton Volgenant. Improving the hungarian assignment algorithm. *Operations Research Letters*, 5(4):171–175, 1986.

[65] Ramin Khalili, Nicolas Gast, Miroslav Popovic, and Jean-Yves Le Boudec. Opportunistic Linked-Increases Congestion Control Algorithm for MPTCP. Internet-Draft draft-khalili-mptcp-congestion-control-05, Internet Engineering Task Force, July 2014.

[66] A Khiyaita, H El Bakkali, M Zbakh, and Dafir El Kettani. Load balancing cloud computing: state of art. In *2012 National Days of Network Security and Systems*, pages 106–109. IEEE, 2012.

[67] Datacenter Knowledge. The year of 100GbE in data center networks. "https://www.datacenterknowledge.com/networks/year-100gbe-data-center-networks", 2018.

[68] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli. Daps: Intelligent delay-aware packet scheduling for multipath transport. In *IEEE International Conference on Communications (ICC)*, 2014.

[69] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & information systems engineering*, 6(4):239–242, 2014.

[70] Kangkang Li and Jarek Nabrzyski. Traffic-aware virtual machine placement in cloudlet mesh with adaptive bandwidth. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 49–56. IEEE, 2017.

[71] Li Li, Ke Xu, Tong Li, Kai Zheng, Chunyi Peng, Dan Wang, Xiangxiang Wang, Meng Shen, and Rashid Mijumbi. A measurement study on multi-path tcp with multiple cellular carriers on high speed rails. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 161–175, New York, NY, USA, 2018. ACM.

[72] Yeon-sup Lim, Erich M. Nahum, Don Towsley, and Richard J. Gibbens. ECF: An MPTCP Path Scheduler to Manage Heterogeneous Paths. In *Proceedings of the 13th International Conference of CoNEXT*, 2017.

[73] Madhusanka Liyanage, Pawani Porambage, and Aaron Yi Ding. Five driving forces of multi-access edge computing. *arXiv preprint arXiv:1810.00827*, 2018.

[74] Lele Ma, Shanhe Yi, and Qun Li. Efficient service handoff across edge servers via docker container migration. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 11. ACM, 2017.

[75] Anahita Mahzari, Afshin Taghavi Nasrabadi, Aliehsan Samiei, and Ravi Prakash. Fov-aware edge caching for adaptive 360 video streaming. In *2018 ACM Multimedia Conference on Multimedia Conference*, pages 173–181. ACM, 2018.

[76] Linux man page. Traceroute. ”`https://linux.die.net/man/8/traceroute`, 2019.

[77] Rod Meikle and Joseph Camp. A global measurement study of context-based propagation and user mobility. In *4th ACM International Workshop on Hot Topics in Planet-scale Measurement*, HotPlanet ’12, 2012.

[78] Microsoft. Xbox Project xCloud. ”`https://www.techradar.com/news/project-xcloud-everything-we-know-about-microsofts-cloud-streaming-service"`, 2019.

[79] Nitinder Mohan and Jussi Kangasharju. Edge-fog cloud: A distributed cloud for internet of things computations. In *2016 Cloudification of the Internet of Things (CIoT)*, pages 1–6, Nov 2016.

[80] Nitinder Mohan and Jussi Kangasharju. Placing it right!: optimizing energy, processing, and transport in edge-fog clouds. *Annals of Telecommunications*, 73(7):463–474, Aug 2018.

[81] Nitinder Mohan, Tanya Shreedhar, Aleksandr Zavodavoski, Otto Waltari, Jussi Kangasharju, and Sanjit K. Kaul. Redesigning MPTCP for edge clouds. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom ’18, pages 675–677, New York, NY, USA, 2018. ACM.

[82] Nitinder Mohan, Tanya Shreedhar, Aleksandr Zavodovski, Jussi Kangasharju, and Sanjit K Kaul. Is two greater than one?: Analyzing multipath TCP over dual-LTE in the wild. *arXiv preprint arXiv:1909.02601*, 2019.

[83] Nitinder Mohan, Aleksandr Zavodovski, Pengyuan Zhou, and Jussi Kangasharju. Anveshak: Placing edge servers in the wild. In *Proceedings of the 2018 Workshop on Mobile Edge Communications*, MECOMM'18, pages 7–12, New York, NY, USA, 2018. ACM.

[84] Nitinder Mohan, Pengyuan Zhou, Keerthana Govindaraj, and Jussi Kangasharju. Grouping computational data in resource caches of edge-fog cloud. In *Proceedings of the 4th Workshop on CrossCloud Infrastructures & Platforms*, page 8. ACM, 2017.

[85] Nitinder Mohan, Pengyuan Zhou, Keerthana Govindaraj, and Jussi Kangasharju. Managing data in computational edge clouds. In *Proceedings of the Workshop on Mobile Edge Communications*, MECOMM '17, pages 19–24, New York, NY, USA, 2017. ACM.

[86] Cloud Native. Kubernetes containers. ”`https://kubernetes.io/`, 2019.

[87] NCTA. Broadband by the numbers. ”`https://www.ncta.com/broadband-by-the-numbers/`, 2018.

[88] Michael Nelson, Beng-Hong Lim, Greg Hutchins, et al. Fast transparent migration for virtual machines. In *USENIX Annual technical conference, general track*, pages 391–394, 2005.

[89] New Zoo. Statistics of Gaming Industry. ”`https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/"`, 2019.

[90] TS Eugene Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 170–179. IEEE, 2002.

[91] Nitinder Mohan – GitHub. MPTCP QAware kernel implementation. ”`https://github.com/nitindermohan/mptcp-QueueAware"`.

[92] NorthEastern. Analytics of smartphone capability. ”`https://www.northeastern.edu/levelblog/2016/04/21/smartphones-supercomputer-in-your-pocket/`, 2019.

[93] University of Helsinki. Ubispark project. "`https://ubispark.cs.helsinki.fi/`, 2017.

[94] OpenNebula. Opennebula technology. "`http://opennebula.org/`, 2018.

[95] OpenStack. Openstack technology. "`http://www.openstack.org/`, 2018.

[96] Christoph Paasch, Simone Ferlin, Ozgu Alay, and Olivier Bonaventure. Experimental Evaluation of Multipath TCP Schedulers. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, CSWS '14, pages 27–32, New York, NY, USA, 2014. ACM.

[97] Claus Pahl, Sven Helmer, Lorenzo Miori, Julian Sanin, and Brian Lee. A container-based edge cloud paas architecture based on raspberry pi clusters. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pages 117–124. IEEE, 2016.

[98] Guillaume Pierre and Maarten Van Steen. Globule: a collaborative content delivery network. *IEEE Communications Magazine*, 44(8):127–133, 2006.

[99] Jürgo S Preden, Kalle Tammemäe, Axel Jantsch, Mairo Leier, Andri Riid, and Emine Calis. The benefits of self-awareness and attention in fog and mist computing. *Computer*, 48(7):37–45, 2015.

[100] Jelica Protic, Milo Tomasevic, and Veljko Milutinovic. Distributed shared memory: Concepts and systems. *IEEE Parallel & Distributed Technology: Systems & Applications*, 4(2):63–71, 1996.

[101] Lili Qiu, Venkata N Padmanabhan, and Geoffrey M Voelker. On the placement of web server replicas. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 3, pages 1587–1596. IEEE, 2001.

[102] Pavlin Radoslavov, Ramesh Govindan, and Deborah Estrin. Topology-informed internet replica placement. *Computer Communications*, 25(4):384–392, 2002.

[103] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving Datacenter

Performance and Robustness with Multipath TCP. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 266–277, New York, NY, USA, 2011. ACM.

[104] Costin Raiciu, Mark J. Handley, and Damon Wischik. Coupled Congestion Control for Multipath Transport Protocols. RFC 6356, October 2011.

[105] Costin Raiciu, Dragos Niculescu, Marcelo Bagnulo, and Mark James Handley. Opportunistic mobility with multipath tcp. In *Proceedings of the Sixth International Workshop on MobiArch*, MobiArch '11, 2011.

[106] Lakshmish Ramaswamy, Ling Liu, and Arun Iyengar. Cache clouds: Cooperative caching of dynamic documents in edge networks. In *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 229–238. IEEE, 2005.

[107] Lakshmish Ramaswamy, Ling Liu, and Arun Iyengar. Scalable delivery of dynamic content using a cooperative edge cache grid. *IEEE Trans. Knowl. Data Eng.*, 19(5):614–630, 2007.

[108] Computer Business Review. Top 10 biggest data centres from around the world. "`http://www.cbronline.com/news/data-centre/infrastructure/top-10-biggest-data-centres-from-around-the-world-4545356`", 2015.

[109] Robert Bosch GmbH. Intelligent systems for the flexible factory. "`https://www.bosch-apas.com/en/home/`", 2019.

[110] Rodrigo Roman, Javier Lopez, and Masahiro Mambo. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78:680–698, 2018.

[111] Safe at last. 80 Mind-Blowing IoT Statistics. "`https://safeatlast.co/blog/iot-statistics/`", 2019.

[112] Jagruti Sahoo, Mohammad A Salahuddin, Roch Glitho, Halima Elbiaze, and Wessam Ajib. A survey on replica server placement algorithms for content delivery networks. *IEEE Communications Surveys & Tutorials*, 19(2):1002–1026, 2017.

[113] Lorenzo Saino, Ioannis Psaras, and George Pavlou. Icarus: a caching simulator for information centric networking (icn). In *SimuTools*, volume 7, pages 66–75. ICST, 2014.

[114] Zohreh Sanaei, Saeid Abolfazli, Abdullah Gani, and Rajkumar Buyya. Heterogeneity in mobile cloud computing: taxonomy and open challenges. *IEEE Communications Surveys & Tutorials*, 16(1):369–392, 2013.

[115] Nuno Santos, Krishna P Gummadi, and Rodrigo Rodrigues. Towards trusted cloud computing. *HotCloud*, 9(9):3, 2009.

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

[117] Marco Savi, Massimo Tornatore, and Giacomo Verticale. Impact of processing costs on service chain placement in network functions virtualization. In *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pages 191–197. IEEE, 2015.

[118] Ji-Yong Shin, Emin Gün Sirer, Hakim Weatherspoon, and Darko Kirovski. On the feasibility of completely wirelesss datacenters. *IEEE/ACM Transactions on Networking*, 21(5):1666–1679, 2013.

[119] Tanya Shreedhar, Nitinder Mohan, Sanjit Kaul, Jussi Kangasharju, et al. QAware: A Cross-Layer Approach to MPTCP Scheduling. *IFIP Networking 2018, May 14-16, 2018, Zürich, Switzerland, Proceedings*, 2018.

[120] Tanya Shreedhar, Nitinder Mohan, Sanjit K Kaul, and Jussi Kangasharju. More than the sum of its parts: Exploiting cross-layer and joint-flow information in MPTCP. *arXiv preprint arXiv:1711.07565*, 2017.

[121] Pedro Silva, Christian Perez, and Frédéric Desprez. Efficient heuristics for placing large-scale distributed applications on multiple clouds. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 483–492. IEEE, 2016.

[122] Alessio Silvestro, Nitinder Mohan, Jussi Kangasharju, Fabian Schneider, and Xiaoming Fu. MUTE: Multi-tier edge networks. In *Proceedings of the 5th Workshop on CrossCloud Infrastructures & Platforms*, pages 1–7. ACM, 2018.

[123] Statista. Internet of Things - number of connected devices worldwide 2015-2025. "`https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/`", 2019.

[124] Yantao Sun, Jing Chen, Q Lu, and Weiwei Fang. Diamond: An improved fat-tree architecture for large-scale data centers. *Journal of Communications*, 9(1):91–98, 2014.

[125] TechJury. Internet of Things Statistics 2019. "`https://techjury.net/stats-about/internet-of-things-statistics/`", 2019.

[126] Blesson Varghese et al. Edge-as-a-service: Towards distributed cloud architectures. *arXiv preprint arXiv:1710.10090*, 2017.

[127] David Villegas, Norman Bobroff, Ivan Rodero, Javier Delgado, Yanbin Liu, Aditya Devarakonda, Liana Fong, S Masoud Sadjadi, and Manish Parashar. Cloud federation in a layered service model. *Journal of Computer and System Sciences*, 78(5):1330–1344, 2012.

[128] WiGle. Open-source wireless base station dataset. "`https://wigle.net/`, 2019.

[129] Dan Williams, Hani Jamjoom, and Hakim Weatherspoon. The xenblanket: virtualize once, run everywhere. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 113–126. ACM, 2012.

[130] Kwame-Lante Wright, Martin Espinoza, Uday Chadha, and Bhaskar Krishnamachari. Smartedge: A smart contract for edge computing, 2018.

[131] Xen. Xen hypervisor. "`http://www.xen.org`, 2018.

[132] Ming Xia, Meral Shirazipour, Ying Zhang, Howard Green, and Attila Takacs. Network function placement for nfv chaining in packet/optical datacenters. *Journal of Lightwave Technology*, 33(8):1565–1570, 2015.

[133] Sami Yangui, Pradeep Ravindran, Ons Bibani, Roch H Glitho, Nejib Ben Hadj-Alouane, Monique J Morrow, and Paul A Polakos. A platform as-a-service for hybrid cloud/fog environments. In *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–7. IEEE, 2016.

[134] Chenying Yu and Fei Huan. Live migration of docker containers through logging and replay. In *2015 3rd International Conference on Mechatronics and Industrial Informatics (ICMII 2015)*. Atlantis Press, 2015.

[135] Ya-Ju Yu, Te-Chuan Chiu, Ai-Chun Pang, Ming-Fan Chen, and Jiajia Liu. Virtual machine placement for backhaul traffic minimization in fog radio access networks. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2017.

[136] Quan Yuan, Haibo Zhou, Jinglin Li, Zhihan Liu, Fangchun Yang, and Xuemin Sherman Shen. Toward efficient content delivery for automated driving services: An edge computing solution. *IEEE Network*, 32(1):80–86, 2018.

[137] Michael C Yurko. A parallel computational framework for solving quadratic assignment problems exactly. *ScienceBuddies*, 2010.

[138] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[139] Aleksandr Zavodovski, Suzan Bayhan, Nitinder Mohan, Pengyuan Zhou, Walter Wong, and Jussi Kangasharju. DeCloud: Truthful decentralized double auction for edge clouds. 2019.

[140] Aleksandr Zavodovski, Nitinder Mohan, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. ICON: Intelligent container overlays. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, HotNets '18, pages 15–21, New York, NY, USA, 2018. ACM.

[141] Aleksandr Zavodovski, Nitinder Mohan, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. ExEC: Elastic extensible edge cloud. In *Proceedings of the 2Nd International Workshop on Edge Systems, Analytics and Networking*, EdgeSys '19, pages 24–29, New York, NY, USA, 2019. ACM.

[142] Aleksandr Zavodovski, Nitinder Mohan, and Jussi Kangasharju. eDisco: Discovering edge nodes along the path. *arXiv preprint arXiv:1805.01725*, 2018.

[143] Aleksandr Zavodovski, Nitinder Mohan, Walter Wong, and Jussi Kangasharju. Open infrastructure for edge: A distributed ledger outlook. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[144] Guoqiang Zhang, Yang Li, and Tao Lin. Caching in information centric networking: A survey. *Computer Networks*, 57(16):3128 – 3141, 2013. Information Centric Networking.

[145] Lei Zhao, Jiajia Liu, Yongpeng Shi, Wen Sun, and Hongzhi Guo. Optimal placement of virtual machines in mobile edge computing. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE, 2017.

[146] Wenwu Zhu, Chong Luo, Jianfeng Wang, and Shipeng Li. Multimedia cloud computing. *IEEE Signal Processing Magazine*, 28(3):59–69, 2011.

[147] Yibo Zhu, Xia Zhou, Zengbin Zhang, Lin Zhou, Amin Vahdat, Ben Y Zhao, and Haitao Zheng. Cutting the cord: A robust wireless facilities network for data centers. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 581–592. ACM, 2014.