The confluence of Cloud computing, 5G, and IoT in the Fog

Tärneberg, William

2019

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Tärneberg, W. (2019). *The confluence of Cloud computing, 5G, and IoT in the Fog*. [Doctoral Thesis (monograph), Department of Electrical and Information Technology]. Department of Electrical and Information Technology, Lund University.

Total number of authors:
1

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

The confluence of Cloud computing, 5G, and IoT in the Fog

William Tärneberg

LUND UNIVERSITY

Doctoral Dissertation
Electrical Engineering
Lund, March 2019

William Tärneberg
Department of Electrical and Information Technology
Electrical Engineering
Lund University
P.O. Box 118, 221 00 Lund, Sweden

Series of licentiate and doctoral dissertations
ISSN 1654-790X; No. 120
ISBN 978-91-7895-010-2 (Print)
ISBN 978-91-7895-011-9 (PDF)

Typeset in Palatino and Helvetica using LaTeX 2ε.
Printed in Sweden by Tryckeriet i E-huset, Lund University, Lund.

Cover designed by William Tärneberg.

Abstract

In the wake of the arrival of cloud computing, future applications are poised to become more resilient and adaptive by embracing elasticity in an osmotic manner. Although cloud computing is a strong attractor for application developers, there are still unconquered performance frontiers. Latency-sensitive and mission-critical applications make up a significant portion of all software systems, and their owners are eager to reap the benefits of cloud computing. However, they are hindered by significant delay, jitter in the delay, and relatively low resilience when operating on traditional, distant, cloud data centres.

Fog computing is emerging as a remedy. Fog computing is a heterogeneous hyper-distributed cloud infrastructure paradigm, ranging from small compute nodes close to the end-users to traditional distant data centres. With greater proximity to the end-users, delay and jitter in the delay can be reduced, and intermediate network reliability improved. Additionally, with increased heterogeneity of resources, applications have a richer tapestry of resources to take advantage of for their objectives. However, managing and taking advantage of this heterogeneity in resources and objectives is a challenge for both the infrastructure providers and application owners alike. Only where to place and scale application components and how to manage system resources to meet the objectives of both parties, is non-trivial. Application placement implies elaborate optimisation objectives, hard-to-find solutions, and operational conflicts.

The objective of this thesis is to investigate the performance-related properties of fog computing, how such an infrastructure can be managed while applications can osmotically take advantage of the infrastructure, and what Fog computing's potential practical performance gains are. These are fundamental topics that need to be answered for providers and application owners alike to be able to invest in fog computing. In general terms, the work in this thesis seeks the trade-offs between infrastructure, applications, and software platform in contrast to the traditional cloud offering.

The thesis provides modelling and simulation tools for evaluating the performance and feasibility of Fog computing. Based on which, the thesis goes on to propose holistic infrastructure management algorithms. The requirements of latency-sensitive and mission-critical applications and use cases are discussed for a fog computing paradigm. These requirements are then translated to Fifth Generation Wireless Specifications (5G) Massive Multiple Input Multiple Output (MIMO) specifications. An original 5G-based fog computing test-bed for time-sensitive and mission-critical applications is implemented. The test-bed is used to evaluate the potential application performance gains of fog computing and to what extent the applications can practically take advantage of a fog infrastructure. The thesis also investigates the architecture of the applications that are proposed to benefit from fog computing and how they perform in traditional cloud offerings.

The included works show that fog computing indeed has a performance advantage over the traditional distant cloud, not only in latency but also in robustness. The benefits of 5G on a time-sensitive application deployed in a fog computing infrastructure are shown to be significant. It is also shown that a fog computing infrastructure with a high degree of heterogeneity and with multiple objectives can be successfully managed scalably. Additionally, the thesis sheds some light on the challenges of implementing latency-sensitive and mission-critical applications with traditional cloud service offerings.

Contents

Preface

This doctoral thesis concludes my work as a PhD candidate at the Department of Electrical and Information Technology at Lund University. The material has either been presented at international conferences or published in international journals. The main overarching contributions presented in this thesis are:

1. Performance modelling and simulation tools for evaluating Fog computing infrastructures.

2. Scalable multi-objective dynamic resource management algorithms.

3. Scheduling algorithms and 5G radio configurations for massive wireless IoT.

4. A test-bed for evaluating opportunities and challenges with time-sensitive and mission-critical in Fog computing.

The publications below are included in the thesis and are grouped by the chapter they contribute to.

Part I - Modelling and managing a Fog computing infrastructure

1. William Tärneberg and Maria Kihl. Workload displacement and mobility in an omnipresent cloud topology. In *Proc. SoftCom (Split, Croatia)*. IEEE, 2014.

   Contributions: Problem definition, model definition, simulation environment, experiments, and analysis.

2. Jakub Krzywda, William Tärneberg, Per-Olov Östberg, Maria Kihl, and Erik Elmroth. Telco clouds: Modelling and simulation. In *Proc. CLOSER (Lisbon, Portugal)*. INSTICC, 2015.

Contributions: Simulation environment, experiments, and analysis.

3. William Tärneberg, Amardeep Mehta, Eddie Wadbro, Johan Tordsson, Johan Eker, Maria Kihl, and Erik Elmroth. Dynamic application placement in the mobile cloud network. *Elsevier, Future Generation Computer Systems*, 2016.

   Contributions: System model, simulation environment, experiments, and analysis. Problem definition and algorithm development efforts shared equally between authors.

4. William Tärneberg, Alessandro Papadopoulos, Amardeep Mehta, Johan Tordsson, and Maria Kihl. Distributed approach to the holistic resource management of a mobile cloud network. In *Proc. International Conference on Fog and Edge Computing (Madrid, Spain)*. IEEE, 2017.

   Contributions: Problem definition, system model, algorithm development, simulation environment, and analysis.

Part II - Smart cities & Internet of Things

6. William Tärneberg, Vishal Chandrasekaran, and Marty Humphrey. Experiences creating a framework for smart traffic control using aws iot. In *9th International Conference on Utility and Cloud Computing*, International Conference on Utility and Cloud Computing (UCC) (Changhai, China). IEEE/ACM, 2016.

   Contributions: Problem definition, system design, and implementation. Evaluation effort shared equally between authors.

7. William Tärneberg, Mehmet Karaca, Anders Robertsson, and Maria Kihl. Cross-layer control for bounded shared state inconsistency in wireless iot devices. In *Proc. Conference on Decision and Control (Melbourne, Australia)*. IEEE, 2017.

   Contributions: Problem definition, system model, simulation environment, experiments, and evaluation. Algorithm development effort shared equally between authors.

Part III - 5G and IoT

8. William Tärneberg, Mehmet Karaca, Anders Robertsson, Fredrik Tufvesson, and Maria Kihl. Utilizing massive mimo for the tactile internet: Advantages and trade-offs. In *Proc. SECON*

Workshops - Robotic Wireless Networks (San Diego, CA, USA). IEEE, 2017.

Contributions: Problem definition and system reliability simulation environment. Evaluation effort shared equally between authors.

Part IV - A Fog computing test-bed

9. Per Skarin, William Tärneberg, Karl-Erik Årzen, and Maria Kihl. Towards mission-critical control at the edge and over 5G. In *International Conference on Edge Computing (EDGE)*. IEEE, 2018.

Contributions: Wireless system integration, PID controller implementation, networking evaluation. Problem definition, system architecture, system implementation, and system evaluation efforts shared equally between authors.

Publications not included in this thesis are:

10. William Tärneberg, Amardeep Mehta, Johan Tordsson, Maria Kihl, and Erik Elmroth. Resource management challenges for the infinite cloud. In *10th International Workshop on Feedback Computing at CPSWeek (Seattle, WA, USA)*, 2015.

11. Meiyi Ma, Sarah Masud Preum, William Tärneberg, Mohsin Ahmed, Matthey Ruiters, and John Stankovic. Detection of runtime conflicts among services in smart cities. In *Proc. International Conference on Smart Computing (St. Louis, MO, USA)*. IEEE, 2016.

12. Jonas Dürango, William Tärneberg, Luis Tomas, Johan Tordsson, Maria Kihl, and Martina Maggio. A control theoretical approach to non-intrusive geo-replication for cloud services. In *Proc. Conference on Decision and Control (Las Vegas, NV, USA)*. IEEE, 2016.

13. Amardeep Mehta, William Tärneberg, Cristian Klein, Johan Tordsson, Maria Kihl, and Erik Elmroth. How beneficial are intermediate layer data centers in mobile edge networks? In *Proc. Foundations and Applications of Self* Systems (Augsburg, Germany)*. IEEE, 2016.

14. Stefan Höst, William Tärneberg, Per Ödling, Maria Kihl, Marco Savi, and Massimo Tornatore. Network requirements for latency-critical services in a full cloud deployment. In *Proc. SoftCom (Split, Croatia)*. IEEE, 2016.

15. Zheng Li, William Tärneberg, Maria Kihl, and Anders Robertsson. Using a predator-prey model to explain variations of cloud spot price. In *Proc. CLOSER (Rome, Italy)*. INSTICC, 2016.

16. Meiyi Ma, Sarah Preum, Mohsin Ahmed, William Tärneberg, Abdeltawaband Hendawi, and John Stankovic. Smart city, data sets, modeling, decision making, real-time, integrating services. *Sumbitted ACM Transactions on Cyber-Physical Systems*, Feb 2018.

17. Karl-Erik Årzén, Per Skarin, William Tärneberg, and Maria Kihl. Control of the edge cloud-an mpc example. In *1st International Workshop on Trustworthy and Real-time Edge Computing for Cyber-Physical Systems (Nashville, TN, USA)*. IEEE, 2018.

18. Lars Larsson, William Tärneberg, Cristian Klein, and Erik Elmroth. Quality-elasticity: Improved resource utilization, throughput, and response times via adjusting output quality to current operating conditions. In *Submitted to International Conference on Autonomic Computing (ICAC) (Umeå, Sweden)*. IEEE, 2019.

I hope you enjoy your reading.

Acknowledgements

At this point, I have already moved on to new endeavours. Nevertheless, the experiences that I have gained throughout the PhD-process and the people how have shaped it, are still dear to me. Somehow, just the right individuals have come into the process at just the right time. Amardeep Metha has been there from the start to almost the very end. We have shared plenty of grief and frustration. Our work gave me many reasons to frequent Umeå. Thank you for everything. Also, in the first year or two, Jakub Krzywda and I quickly realised that all conferences do not always live up to expectations, but that they can be remedied with the presence of a swimming pool. After I had gotten started, Jonas Dürango came along and taught me about model predictive control. A little further along in my studies, Zheng Li dropped in for a post-doc at the department. He taught me about what constitutes an academic contribution. Thank you, Zheng, I wished we had collaborated more than we did.

In 2016, about half way through I had the privilege of spending a good part of that year at the University of Virginia in Charlottesville, VA, USA. What an unforgettable experience. Visiting Professor John Stankovic's research group was very refreshing. Professor Stankovic, thank you for your inclusion and patience with my perspective. I specifically also want to thank Professor Marty Humphrey for our collaboration, your frankness, and through-provoking discussions. After returning to the department, I began working with Per Skarin on what became the final paper in this thesis. Per is a delight to work with, he is diligent, passionate, and he challenges me both technically and intellectually. Thank you, Per, our work propelled me over the edge to completion. In the midst of our test-bed-building, Haorui Peng moved into Zheng's old office. Thank you for being a delightful office neighbour. Over the latter part my PhD studies I frequently interacted with Mohammadhassan Safavi. I truly enjoyed our cultural exchanges and I hope we can finish all the papers we have started. At the very end of my studies, and far too late, Lars Larsson joined the group. I don't think we disagree on anything other than how much more competent you are than I. I miss our daily conversations, they have been genuinely invaluable to me personally, academically, and professionally. I sincerely hope our paths will cross again under similar circumstances.

Because our research group was very small, I often ventured to other departments and institutions in search for collaborations. Therefore, I would also like to thank my friends at Umeå University and at the Department of Automatic Control, they have been like second homes to me. I would also like to thank my supervisors, Maria Kihl for your incredible patience, always assuming that I will make it, and your diligent paper reviews. Erik Elmroth, thank you for including me in your research group and your frankness. Martina Maggio, thank you for unknowingly giving me a new benchmark for academic productivity. Stefan Höst and Jens Andersson, it has been great working with you on countless teaching assignments and labs. I hope all of the material we have produced over the years will come to good use.

As for my close friends. Shiva and Johanna, other than exquisite company, your compassion and determination are permanent way-points on my compass. Mikael Hellberg, you have provided me with much needed intellectual distraction. No one can put technology in new perspectives as you do. Marcus Källgren, we got each other through engineering school with plenty of fu along the way, thank you. My father, Jacob Mannerstråle, and Hans-Göran Nilsson who inspired me to take up engineering in the first place. Thank you.

No matter how unimaginative it may sound, the love from my family has shaped my character. I can always count on their unconditional support. My mother has given me creativity, compassion, and frugality. My father has given me critical and analytical thinking and taught me to not give up. My sister relentlessly inspires me to do more to increase my positive impact on society and challenges me to do more to decrease my negative impact on this planet. Although she was not quite sure in what field, I remember that my grandmother, on my father's side, told me at a young age that she had always known that I one day will study to become a researcher. I will always remember you. My in-laws, Christel and Lars Nilsson, and Jepser Welander have always been there for my and my wife.

Anna, my wife, I have truly enjoyed traversing the PhD-process in tandem with you, both personally and professionally. Although our fields never quite crossed, the lifestyle has brought us closer together like nothing else could have and has changed us profoundly. Who you are and what you do inspires and challenges me in everything I do. Liam, my son who rightfully takes all my attention away from this thesis. This work is somehow both impossible and only possible with you in my life. Liam and Anna, without you, nothing is worthwhile.

This work was funded by the Swedish Research Council (VR) under contract number C0590801, the Lund Center for Control of Complex Engineering Systems (LCCC) also funded by (VR).

William Tärneberg
Lund, March 2019

1 Introduction

With the obliquity of the Internet, virtualisation, cloud computing, and cloud-native platforms, compute capacity, which was once contained in a physical box on our desks is increasingly resembling a utility. Today, compute resources can be accessed almost instantly in the cloud and can satisfy most applications. With this gradual change in infrastructure as developers learn to utilise it, the type and nature of applications are evolving to solve new problems. With a modern infrastructure, there are few reasons for applications to be built as monolithic blocks hosted in immutable compute machines but can be devised using distributed micro-services and scale at will to match demand. As the costs of a digitalised world are becoming clear, forthcoming applications will no longer adhere to dated screen-based User Interfaces (UIs) with rigid real-time performance requirements and huge overheads. Applications will exist omnipresently in an Internet of Things (IoT) world, be less visible, more adaptable, and proactive to each user's expectations, see Figure 1.1.

Staying off the cloud is increasingly indefensible. The breadth and rate of innovation in services and resource management by the large cloud providers is difficult to match by any individual organisation whose core business is not cloud computing. The level of technological and business agility achieved with the rapid scalability of cloud resources is unmatched by privately operated infrastructures.

Cloud computing, as realised today, is however inaccessible to time-sensitive and mission-critical applications. Contemporary clouds are realised with distant DC accessed over the public Internet. The intermediate networks introduce delays and uncertainty the render the cloud an infeasible habitat for time-sensitive applications such as control loops.

Additionally, hauling vast amounts of data to a distant Data Center (DC), promptly, can be both technically and economically infeasible. For a DC operator, Input/Output (I/O) is a precious resource [BCH13]. Therefore, moving vast amounts of data in and

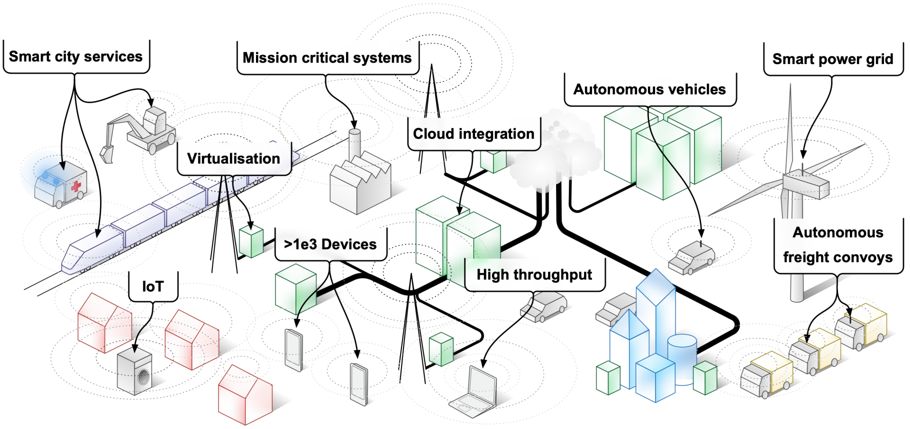


Figure 1.1.: A smart and wireless world

out of a cloud can be costly for its users. Data-heavy IoT applications, in particular, incur significant bandwidth usage.

The cloud can be a hostile habitat for an application. A cloud is a shared pool of resources. Other tenants can have an impact on the performance of your application. Cloud providers implement intricate management policies to meet their internal performance goals. These policies can implicitly conflict with the tenant applications' goals.

In this thesis, it is assumed that the success of the cloud paradigm is founded on generality and frugality. Employing bespoke hardware (HW), resource management principles, and programming models erode the cloud's economies of scale and elasticity. It is therefore assumed that all applications operate in General-purpose Processor (GPP) environments in the DC and that users access such a DC over the public Internet.

To realise the world depicted in Figure 1.1, cloud computing is getting to the point where it needs to spatially and conceptually extend towards its users. This is generally referred to as Fog computing. Fog computing is essentially a heterogeneous hyper-distributed cloud infrastructure. Its resources span the Radio Base Stations (RBSs) at the network's edge to traditional distant DCs. With the Fog, delay and jitter in the delay can be reduced, and reliability can be improved with proximal placement. Not only that, with a richer tapestry of resources, applications now have a broader spectrum of resources to satisfy their capacity needs, given any constraints, such as cost, location, quality, and RTT.

Fog computing infrastructures are vast and intricate, managing them is non-trivial. The underlying Fog computing platform must be able to satisfy the needs of heterogeneous infrastructures while managing the needs of the applications, at scale, and staying elastic. The intermediate wired and wireless networks play an essential role in realising the Fog. To be successful, the Fog infrastructure must be aware of the state of the network, while the network must be aware of the application's needs. Furthermore, applications should be made quality elastic in order to be able to take advantage of dynamic and heterogeneous resource offerings. The above challenges come with non-trivial trade-offs. These challenge are the primary topics of this thesis.

This thesis addresses the challenges mentioned above with the Fog computing in the following manner. The remainder of this chapter provides a background and an intuition for cloud computing, its challenges, and an overview of Fog computing to frame the scope of the thesis. Part I discusses a set of fundamental challenges with Fog computing. Chapters 2 and 3 introduces a set of Fog computing performance models. The models are used to study the effects of User Equipment (UE) mobility on a Fog computing infrastructure. In extension, and primarily, the models are used to design and evaluate two resource management algorithms for the Fog. In Chapter 4, a centralised, optimal, algorithm is provided for an upper performance bound followed by a tractable distributed multi-objective algorithm in Chapter 5. Part II explores the challenges of designing cloud-native IoT applications. Chapter 6 presents an architecture for an IoT-based mission-critical smart city application using existing cloud services in a traditional cloud. The resulting evaluation reveals significant limitations and performance challenges with existing platforms. In Chapter 7, a method for limiting the amount deferred shared state information amongst a set of IoT devices over a 5G link is presented. The addition of 5G in Fog computing is explored further in Part III. Here, the properties of Massive MIMO are studied to argue for the trade-off and challenges with using Massive MIMO for Ultra-Reliable and Low-Latency Communication (URLLC) in IoT. Finally, Part IV compounds the presented work on Fog computing, IoT, and 5G. In Chapter 9, a 5G Fog computing test-bed for mission-critical and time-sensitive IoT applications is presented. The test-bed is used to control a physical plant over 5G with the control-loop implemented as a cloud-native IoT application and executed distributively in the Fog. Experimentation shows that the Fog can bring real performance enhancement to mission-critical and time-sensitive IoT applications.

1.1. CLOUD COMPUTING

This thesis deals with cloud computing technologies not as they exist today but as how they are evolving. This section gives a general introduction to cloud computing to convey an intuition for the cloud computing dynamics and challenges addressed in this thesis.

The term *cloud computing* supposedly first appeared in 1996 [Reg11], came into fashion in the early 2000s, and has become increasingly prevalent in the 2010s. *Cloud* is a debatable metaphor. The image of a cloud has for a while been used as a metaphor for the Internet. In this context, a cloud is supposedly meant to symbolise computing accessed over the Internet as opposed to off-line computing, i.e. local computing or on-premise computing. As with meteorological clouds, clouds that do computing are vague, open to broad interpretation, distributed, and like the weather, are unpredictable. Recently, cloud computing has become a layman hypernym for any higher-level software (SW) service accessed over the Internet where its customers are agnostic of where and how the service is hosted. In contrast to for example a desktop Personal Computer (PC) or a campus cluster, cloud computing organises computing so that a seemingly abundant amount of resource can be accessed almost instantly. With these properties, arguably, cloud computing promises to move us towards ubiquitous computing, where computing is available, procured, and consumed as a utility.

From a technical perspective, in this thesis, we adopt the following definition of cloud computing. Cloud computing is the spatial and temporal pooling of general-purpose Information Technology (IT) resources and the expedient management of those resources. The resulting abstract IT resources and services are offered publicly or privately and are often accessed over the public Internet, like utilities, in what resembles a marketplace. Typically, these resources and services are packaged with a certain degree of abstraction, ranging from a form of a VM or container to specialised HW to a SW platform, and anything in between. A critical technological enabler of cloud computing has to this point been HW and Operating System (OS) virtualisation. With virtualisation, physical resources can be partitioned, shared, and isolated into VMs. A VM can be packaged for expedient portability and replication. Recently, container have emerged as a nimbler alternative. A cloud customer is ordinarily able to specify the properties of the virtual resource or service it wishes to procure but is generally unaware of the actual nature of the underlying physical resource, its physical location, to the extent in which it is shared, or for how long it will be retained.

As with any utility, a Service Level Objective (SLO), between a cloud provider and its customers reassures the customer that a resource superficially equivalent to that initially procured will remain available and responsive. In other words, a cloud provider and its customer agree upon an expected Quality of Service (QoS). The cloud providers' ability to meet these expectations is in no small part a reflection of how they manage those resources. It is these management principles that is the primary focus of this thesis. In contrast to off-line computing, the performance of resources and services in a cloud are non-deterministic, fuzzily governed by the resource management objectives of the cloud provider. Furthermore, in return for relinquishing control, a cloud customer receives access to seemingly abundant low-cost computing resources with little or no maintenance effort, see Figure 1.2.

An playful metaphor for cloud computing is:

> 'Using the cloud is like celebrating your birthday at a bar. You can entertain lots of guests, and they can all get as much as they want, but it is going to cost you. It would have been cheaper to have bought the goods ahead of time, but if you don't know how many guests are coming, you would likely rather have the bar take the risk of having to deal with half-empty bottles when it's all said and done. Plus, it is nice to let someone

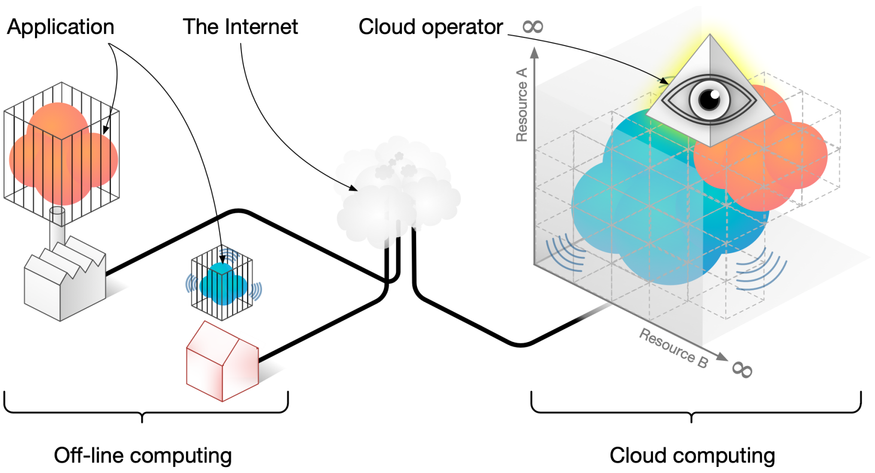


Figure 1.2.: Cloud computing vs. off-line computing. Imagine an application as a multidimensional blob undulating in all resource dimensions. In off-line computing, that blob is contained in an immutable cage. The blob's owner is solely responsible for the health of the blob and cleaning up the cage. In the cloud, across the Internet, the blob can swell and contract at will but contends with other blobs and is chaperoned by a supervisor. The blob's owner needs not to worry too much about cages but gets a bill at the end of the month.

> else deal with cleaning up. However, the bar is a shared space, and your assumption that it has an infinite resource supply may prove to be incorrect. In spite of this uncertainty, it is still highly preferable to having to buy your own virtually infinite supply yourself. ' - Lars Larsson, 2018

For clarity, in this thesis, a *service* is a service provided by a cloud provider. From here onwards, a *cloud* is the physical and legal entity from which you procure cloud resources. Additionally, a *cloud customer* is an entity that procures cloud resources and services and has a QoS expectation on those resources and services. A cloud provider hosts *applications* for cloud customers, utilising the cloud providers services and resources.

1.1.1. WHAT MAKES A CLOUD

Clouds are realised in large-scale compute, storage, and networking warehouses referred to as DCs [BCH13]. Contemporary cloud providers excel at running and maintaining these DCs. Essentially, DCs enable computing at economies of scale and is in many aspects competitive to off-line computing.

A cloud's service offering can be public, such as the services provided by AWS,

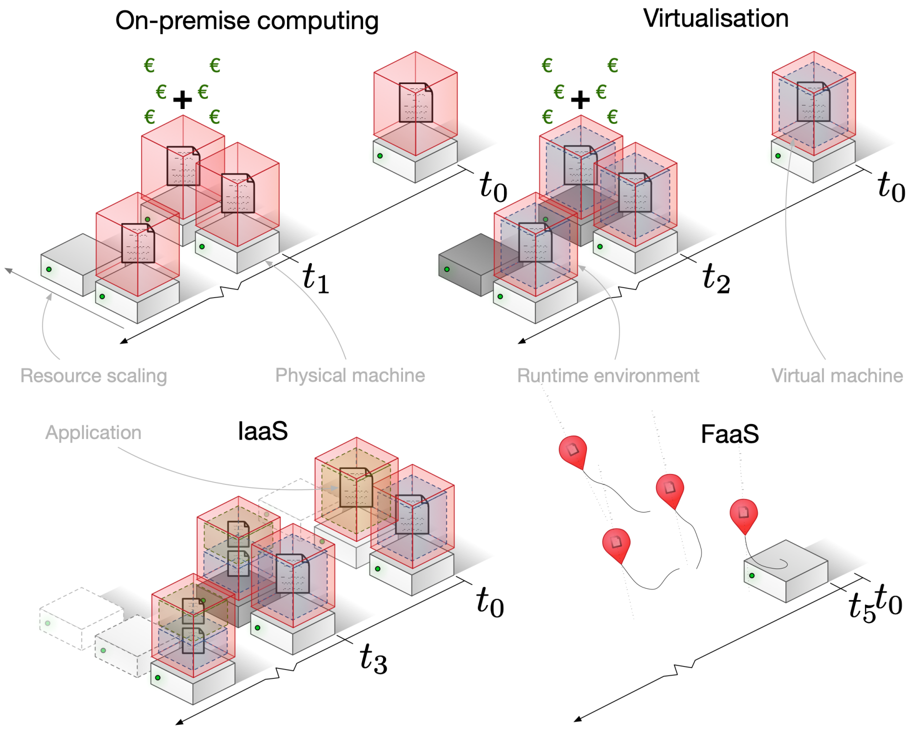


Figure 1.3.: Progression of utility computing implementations.
At t_0 the system has no load. At just after t_0 the application is subjected to a load. Here $t_1 >> t_2 > t_3$ and t_5 is slightly larger than t_0

Microsoft, Google, and IBM. In a so-called public cloud, anyone can seemingly indiscriminately procure cloud resources and services at a market price. Private clouds, on the other hand, are DCs managed as a cloud but its services are offered only to a private group of customers, e.g. a corporation.

The services offered by a cloud ranges from direct representations of servers, VMs in Infrastructure as a Service (IaaS), containers in Constainer-as-a-Service (CaaS) to hosted SW services, to hosted functions (Function-as-a-Service (FaaS)). They are all inherently different SW execution environments. Virtualisation has played a key role in the realisation of the cloud. See Figure 1.3 for an intuitive overview. The primary service abstractions covered in this thesis are the following:

IaaS IaaS was the embryotic resource offering and is still cloud computing's most fundamental. Here, a cloud customer procures resources in the form of VMs and containers (CaaS). The customer installs any SW on the VM and manages it as a server independently of the cloud provider. The performance of an application hosted on the VM is predominantly a reflection of the quality of that application, the customer's ability to manage the instance, the operator's ability to schedule collocated VMs and the characteristics of the other VMs' workloads. Performance guarantees for applications are therefore not provided. Each SW instance type comes with a set of expected I/O throughput and cloud-provider-defined performance abstraction. In AWS, this abstraction is called EC2 Compute Unit (ECU).

With IaaS, the customer scales its application by adding and subtracting VMs or containers. Selling containers in this manner is sometimes referred to as CaaS. VMs have a non-negligible start-up time, containers less so. Additionally, on any individual server, a VM might be collocated and interfere with other VMs from other customers [VKF+12]. In this paradigm, a cloud customer needs to make conscious decisions about the type and quantity of VMs it procures and for how long. In this regard, an application owner faces similar challenges as with off-line computing, but with the benefit of having access to an abundance of resources and the ability to rapidly procure and relinquish resources.

PaaS Platform-as-a-Service (PaaS) offer customers the ability to host their bespoke applications on a hosted platform in the cloud. Typically used for web services, the customer builds an application using an Application Program Interface (API) provided by the cloud provider and its own code. The cloud provider then scales and manages application given that it adheres to the confines of the API. In contrast to IaaS, here the customers do not need to procure and manage VMs and containers but are instead confined to the abilities of the platform.

FaaS In the FaaS paradigm, the cloud provider hosts a program language runtime, often for dynamic and interpreted languages, enabling execution of predominantly stateless compute functions. An application can be contained within one of these functions or constructed as a composition of many. A function is limited to a set of execution time and the amount of memory it can consume. The customer can instantly execute hundreds of instances of its function, paying only per execution. Performance guarantees are not provided, see Section 1.2.2. Although a developer can verify the functionality and stability of the code, it is non-trivial to estimate how it will perform in the cloud. Within these confines, the customer is wholly responsible for the stability of the SW. This paradigm focuses on compartmentalising applications into its fundamental functions and fits into the realm of micro-services. The major cloud providers offer the means to trigger functions from a plethora of other services and to pass messages between them. Messaging between micro-services is often asynchronous and does not come with any tangible performance guarantees.

SaaS In a Software-as-a-Service (SaaS) offering, an application in the form of SW is hosted in the cloud and provided as a subscription to the customer. The SW is often accessed over a web interface and requires no maintenance on behalf of the customer. In the back-end, the cloud provider scales the SW according to the clients' needs. Microsoft Office 365 and Gmail are examples of a SaaS.

1.1.2. ELASTICITY

The expedient management and flexible procurement of cloud resources are the enablers of elasticity. Compared to off-line computing, it should now be clear that elasticity is cloud computing's distinguishing quality. Presented below is a distinction between infrastructure elasticity and process elasticity.

INFRASTRUCTURE ELASTICITY

While off-line infrastructures can adapt in the time scale of days and weeks and are immutable, the cloud is elastic in the time-scale of minutes, see Figure 1.4. As a consequence, resource needs can be matched more responsively. Nevertheless, cloud resources are not perfectly elastic. VMs, containers, and SW platform have start-up times and are affected by overlying management principles. In a perfectly elastic system, deterministic resources on a continuous scale would be available immediately. Figure 1.4 illustrates the dynamics of infrastructure elasticity.

PROCESS ELASTICITY

An elastic infrastructure allows for elastic processes [DGST11]. Cloud is not only resource-scalable but is also cost elastic, and both enables and requires quality elasticity. With elasticity in three dimensions, cloud computing enables application owners to set operating criteria that closely match their business needs. Given the volatility of the cloud infrastructure and the applications' ingress workloads, this is a non-trivial task.

Today, cloud-based applications are designed to serve its customers at a fixed quality level. Applications serve the workload oblivious the state of their execution environments. The execution environment includes; CPU core count, I/O, and RAM as well the level of contention in the infrastructure. Applications indiscriminately produce outputs of fixed quality, without regard to either current operating conditions or what utility each customer receives at that quality level.

Quality-elasticity is defined as letting applications adapt their mode of operation to current operating conditions by dynamically adjusting their output quality accordingly. Quality concessions are achieved through both *basic properties* of their execution environments, such as opting for a more memory-intensive algorithm when memory is more readily available than CPU time, and on *current conditions* such as system load and instantaneous contention effects by other execution environments. Crucially, ap-

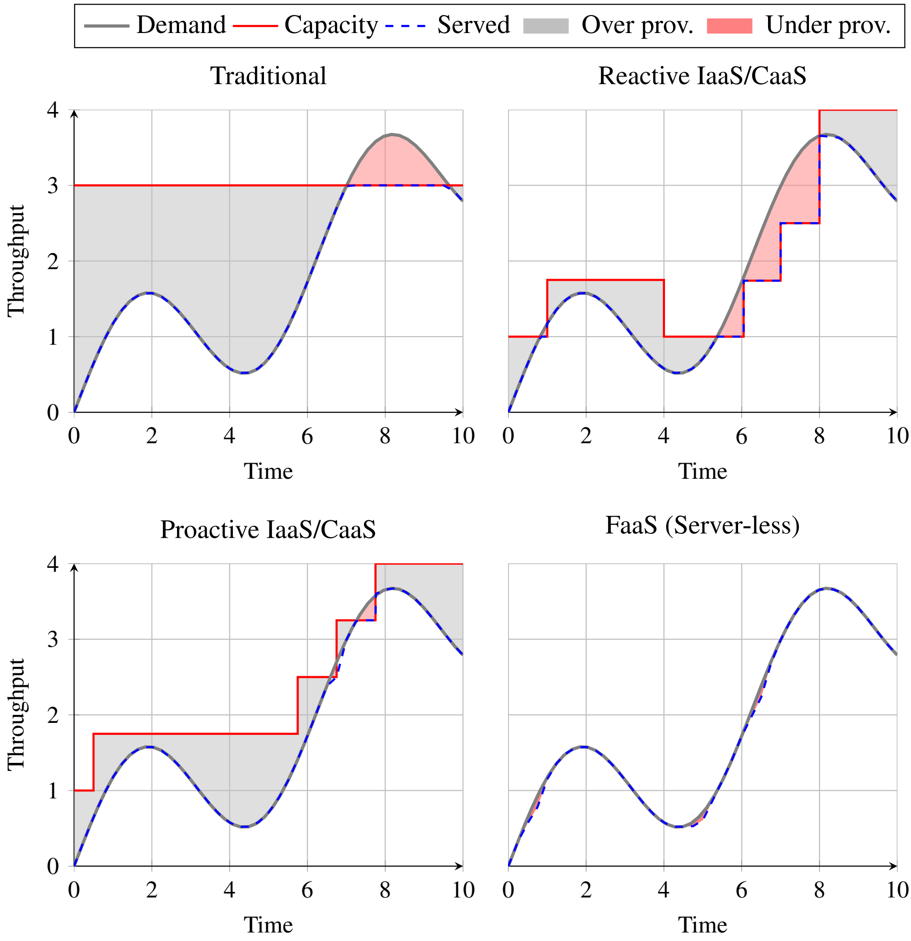


Figure 1.4.: Illustration of resource matching to demand, as seen from the cloud customer. Note that for the sake of simplicity, the systems in the figure are not work-preserving. In traditional provisioning, excessive over-provisioning is common practice. Representing a lost opportunity. In IaaS, matching demand can be challenging. In FaaS, the cloud customer does not nor can not control the underlying resource provision, but can occupationally notice when demand is not matched.

plications shall provide lower-quality results in cases of resource scarcity, and higher-quality ones when resources are abundant.

For the applications, the implication is that service instances can more predictably handle load spikes and avoid client time-outs by reducing output quality of responses. This instantly reduces the load and thus helps maintain the desired throughput.

For the cloud providers, higher utilization and an improved ability to adapt deployed services to resource contention mean that redundant capacity can be reduced. Output quality *reductions*, a subset of quality-elasticity, have been shown to optimize cloud infrastructure in this way [XDB16].

ELUSIVE ELASTIC POTENTIAL

When fully utilising the heterogeneity and elasticity of cloud resources, it can help application owners to meet their business goals. Over-utilisation and thus overspending can be capped. The quality of the output can be scaled to match the actual prevailing requirements on timeliness and granularity.

For a process to take full advantage of process and infrastructure elasticity, the process owner needs to have rich insight into the nature of the resource at hand. It also needs an Elastic Reasoning Mechanism (ERM) of some sorts, such as [CMTD13], and a definition of what the optimal use of those resources is. Finding that momentary optimal configuration is reminiscent of the Vector Packing Problem, which is NP-Hard [CKP03]. There are several related works on adaptive processes, but few generally consider decisions involving cost, resources, and quality.

Because the cloud is not perfectly elastic, when resources do not match demand, applications can enter a state of resource contention. Applications' ability to serve requests is significantly impeded when resources are scarce. Due to imperfect elasticity, there is no practical way to scale up the underlying infrastructure instantly. In this state, requests are either served at a lower rate or, due to client time-outs, perhaps not even at all. Thus, service utility, resilience, and throughput suffer.

When operating with excessive resources, the problem is two-fold. Firstly, the execution environment's resources, that have been paid for are poorly utilised. Secondly, the underutilised resources represent a missed opportunity to provide higher quality results at potentially no additional cost.

Low resource utilisation is costly for both application owners and cloud infrastructure providers alike. Average utilisation is reaching only around 15% [SSH$^+$16]. In a sense, resources are wasted to essentially keep the lights on [RTG$^+$12, BH07]. Cloud providers are also under market pressure to provide seemingly infinite resources, which requires large buffers of available resources to cope with request peaks. This is costly and constitutes a barrier to entry for smaller providers. Thus, both service providers and cloud infrastructure providers have clear incentives to make better use of available resources.

Eventual consistency, explored at length in [CIL$^+$15], is a form of quality-elasticity for databases. This mode of operation has taught users to not expect consistent results to imprecise queries: what products are recommended changes based on factors and proprietary algorithms that users cannot inspect. Thus, they cannot judge whether these results are accurately modelled to their particular preference, instead just trusting that they have been generated "somehow". This non-strictness is ripe for optimisation. For service delivery, we can use this to our advantage, as intuitively, *some* result (of some quality) is better than *none*.

1.1.3. HIGH-LEVEL CONCERNS

Although the cloud has elusive potential, it is not without a set of challenges for its customers.

Availability Despite all its redundant systems, a DC is a single-point-of-failure. Power black-outs and brown-outs can hit DCs and intermediate infrastructure. SW bugs, human error, and malicious attacks can render DCs inoperable. Additionally, a customer's Internet Service Provider (ISP) can go down and vary significantly in quality over tine.

Latency The best-effort networks separating the cloud providers' DCs with its customers provide no performance and availability guarantees. Cloud DCs are spread far and wide. Network latency delay and jitter in the delay can be significant. Additionally, the response times from cloud platforms vary greatly. These two properties are discussed further in Section 1.2.2.

Vendor lock-in The rate of service innovation and differentiation means that applications are becoming less portable between clouds, especially when employing PaaS. The lack of standards across cloud providers is making it difficult to switch cloud provider or spread across multiple cloud providers.

Data security Once in the cloud, there is almost no transparency as to how and where data is stored. It is virtually impossible to prove that data cannot and is not leaked, given the complexity and rate of change of cloud platforms and systems.

Cost Although the cloud is a potential cost saver, it does require a conscious effort. Cloud deployment costs can quickly escalate if left unchecked. Recent cloud-native offerings might appear cost-effective but can be costly if not used frugally or as intended, [VGO+17].

1.1.4. WHO IS THE CLOUD FOR TODAY?

Start-ups have embraced the scalability of the cloud as the ability to snowball. The cloud is very suitable for both small companies that want to snowball and that cannot afford large up-front investments as well as large enterprises that want to become more agile and cost-effective with their IT infrastructure. Today, the cloud is primarily used to host web-services, databases, and for Content Delivery Network (CDN)-like content hosting. The next chapter looks at the next cloud frontier.

1.2. TOMORROW'S APPLICATIONS AND THE CLOUD FRONTIER

Regardless of the shifting merits of cloud computing, enough momentum has gathered around this massive compute immigration to grab everyone's attention. In the midsts of the hype [Gar], cloud business models go in and out of fashion. However, new and existing applications still find it challenging to deploy and make the migration to the cloud, respectively. In this section, emerging computer needs and existing reluctant cloud incumbents are examined. Their potential success and challenges in the traditional and in emerging cloud computing paradigms are discussed and contrasted with the backdrop of general empirical observations.

1.2.1. EMERGING APPLICATION TYPES

Internet access is both kinetically and conceptually increasingly mobile. Humans interact with services from the cognitively arduous confines of mobile devices as they traverse the surface of the earth. Mobility is a challenge for cloud services. As users move through the network, switching from mobile to WiFi infrastructures with varying quality, applications, content, and cloud platforms must adapt and reorganise to meet the user's QoS expectations. Additionally, users accessing bandwidth-intensive and latency-sensitive cloud applications can significantly strain mobile networks. Today, humans downloading content constitutes a much ten times the amount of bandwidth as they upload. That ratio is expected to inverse in the coming years as machines communicate more and more.

Analogously, a great deal, if not the vast majority, of computing workloads are initiated or conceived by humans. Humans submit queries, tune parameters, create and remove content, and constitute content popularity churn. From a birds-eye view, most processes are not only initiated by humans but also have humans in the loop. Some argue for a formalisation; an introduction or ratification of a *social compute unit* [DB11]. These human resources will and are doing cognitive tasks that computers still cannot satisfactorily complete. For example, labelling data for Machine Learning (ML) models, resolve ambiguous conflicts and approve significant state transitions. However, perhaps more commonly, humans act as interfaces between SW or human systems. Waiting an arbitrary amount of time for a human response or interpretation can be a significant performance bottleneck for any system. Often these responses are predictable and can thus arguably be made by Artificial Intelligence (AI). A current example is Google Duplex [LM18]. Here, a AI-based assistant can successfully make arduous phone calls on behalf of its human client to another human-based system interface, without its client's involvement. The Google Duplex team provide two primary examples; booking a haircut appointment and making a restaurant reservation. More systems like these that handle human-to-machine human-to-constraint arbitration might eliminate some need to for our current app-based work-flows. These types of applications require that the cloud can close the communication loop with the IoT-world. Here both delay and data volume is a challenge. Traditional distant DCs is proving to be a

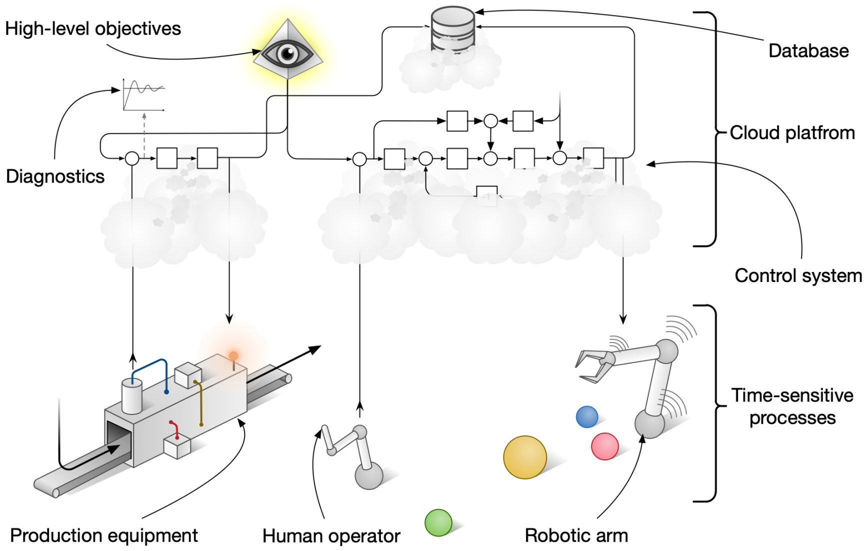


Figure 1.5.: Industry 4.0

challenge. Hauling vast volumes of data over the intermediate network is both costly and incurs congestion. The latency to distant DCs is often significant, see Section 1.2.2.

Industry 4.0 or industrial IoT is an example that reflects the challenges above. Here, sensors, actuators, and machines connect to an omnipresent cloud platform. There, control loops, ordering systems, and analytics systems co-exist, scale, and share data. Pervasive IoT and micro-services play an essential role in realising industry 4.0, see Figure 1.5. Real-time automation systems are time-sensitive. At an industrial scale, they require significant network bandwidth. Such systems require I/O performance symmetry, something which current cloud providers don't provide. I/O performance symmetry refers to the ratio of quantity and timeliness at which that data can be inputted and outputted to and from the cloud. Contemporary cloud services are primarily designed to consume data. Moving data out of the cloud in a timely manner is both expensive and technically challenging. Most cloud providers' IoT solutions are good at hauling data into the cloud but don't provide the means to create a near real-time communication loop. Any downstream communication is typically best-effort over Message Queuing Telemetry Transport (MQTT) topics and is primarily intended for device management. Section 1.2.2 presents a set of performance challenges for such applications.

Collaborative computing is another emerging paradigm where computation needs to be coordinated or even consolidated [GLPL14]. In collaborative computing, intelligent entities or data sources collaborate to improve the quality of the application's output or their performance. The collaboration can be either event-based or continuous. It is triggered either by the devices themselves or by a third party. The resulting process or task can either be shared amongst the entities or completed by a third party.

An application domain ripe for collaborative computing is autonomous and semi-autonomous vehicles. Here, no single vehicle has the data nor the computational capacity to handle every situation. Additionally, as they operate in a shared physical space, no one vehicle can resolve contention. For example, with collaborative computing, the intersections of tomorrow might not need traffic lights [RTM17]. For example, an entity in an intersection detects approaching vehicles, retrieves relevant data from the affected vehicles, arbitrates the use of the intersection, and finally acts on the vehicles. One can also imagine situations where the vehicles themselves initiate the collaboration. The sensors in a vehicle only 'see' so far. If the autonomous entity in a vehicle determines that its actions are too uncertain, it can plausibly both supplement its data from proximal cars and augment its computational capacity using other vehicles or a third party to increase the quality of its output.

These concepts are not necessarily new. They have been around since the dawn of Vehicle to Vehicle (V2V) communication, but with the arrival of abundant computing and vehicle technology, they just might become a reality. The cloud is an alluring third entity in such systems. Its resources are abundant and quickly scalable. However, as such systems deal with human lives, the level of uncertainty in contemporary clouds and wireless networks renders such applications inviable.

1.2.2. LATENCY AND UNCERTAINTY CHALLENGES

Traditional cloud computing relies on distant DCs, shared amongst a large number of applications. Although DCs operate at a low level of utilisation, multi-tenancy has a significant impact on an application's performance [BCH13]. Additionally, to be cost-effective, cloud operators implement resource management mechanisms in their DCs that balance Operational Expenditure (OPEX) with perceived performance. Analogously, both these factors impact the achieved RTT between a customer and the services in a cloud provider's DC and the services therein. Below, RTT includes both network delay and system delay from the cloud platform.

As a reference, an RTT of $\geq 100ms$ makes real-time and time-sensitive applications, as we know them today, infeasible. Achieving an RTT of 1 ms, along with carrier-grade robustness and availability, enables these time-sensitive applications to run in the cloud. These applications constitute what is known as the *Tactile Internet*.

A distributed system like the Internet and the global cloud infrastructure can incur long-tailed delays and jitter in the delay. These are often due to congestions in the systems and policies implemented to mitigate over-utilisation and cost savings. Even with a low mean delay, the occasional proportionally larger delay can equate to a catastrophic interruption of an application. Because the delays have many sources, they can even be multi-modal and therefore both correlated and uncorrelated. As a simple example, network delays within a short time frame are typically not correlated while for example the delays incurred by warming up a container are correlated.

NETWORK PERFORMANCE

The network separating a cloud customer from a cloud provider's DC is not always reliable but also incurs delays and jitter in that delay. Figure 1.6 shows the RTT between the Lund University campus in Lund, Sweden and to all AWS's availability regions in the autumn of 2017. The RTT was measured by sending 100 ICMP pings to each availability region's lambda endpoint every 30 minutes from August to December in 2017. Considering the quality of the campus network and the ISP, the achieved RTT is viewed as ideal.

The geographically nearest availability region is *eu-central-1*, located in Frankfurt, Germany. Between Lund and *eu-central-1* a very consistent RTT of just short of $20ms$ is observed. When staying in Europe, the RTT is between $20-35ms$. Although a RTT of $20ms$ can seem low, it is a challenge for real-time applications. For example, for a process operating at $50Hz$, with a RTT of $20ms$, every action is applied with a delay equivalent of one period. Persistent delays can be compensated for using the practices of networked control [ZBP01] in some systems. Not compensating for delay can have a significant, long-lasting impact on process performance and even make the process unstable. Variance in the delay, i.e. jitter in the delay, is non-trivial and requires remedies that are bespoke to the application, such as process switching and quality elasticity.

Going beyond Europe sees the tails grow significantly. Crossing the Atlantic Ocean adds another $70ms$. The network quality to the AWS availability zones in Asia is less than favourable, and the RTT is well above $250ms$. Inter-regional connectivity is relatively weak, while intra-regional connectivity is acceptable in the U.S. and Europe [clo18a]. Interesting to note from Figure 1.6 is that the number of hops does not significantly increase with spatial separation.

Connectivity over Ethernet is preferred but stationary. In near all industrial IoT scenarios, devices are connected either over a fixed or wireless links. Figure 1.7 shows the same experiment but conducted over a public Long Term Evolution (LTE) network. Here, the variance, or jitter in the delay, is significant. The mean RTT to *eu-central-1* is now above $50ms$, with some instances close to $80ms$.

A large number of IoT devices need to be connected wirelessly. It is evident that LTE is not suited for massive IoT deployments, not only with regards to latency but also energy efficiency and reliability. In many cases, Ethernet performance is not enough. The results presented in Part III show how Massive MIMO in 5G can facilitate industrial IoT and the tactile Internet. In short, the increase in spectral efficiency in Massive MIMO can be utilised to archive URLLC and massive Machine Type Communication (mMTC), essential for industrial IoT.

Exiting ISP's network and crossing a continent's backbone network can result in the traversal of more than 20 nodes. Figure 1.8 shows the mean latency per hop to all AWS availability zones from Lund University in Sweden. The figure shows after how many hops the ISPs' networks end, on average, and how the jitter in the delay becomes

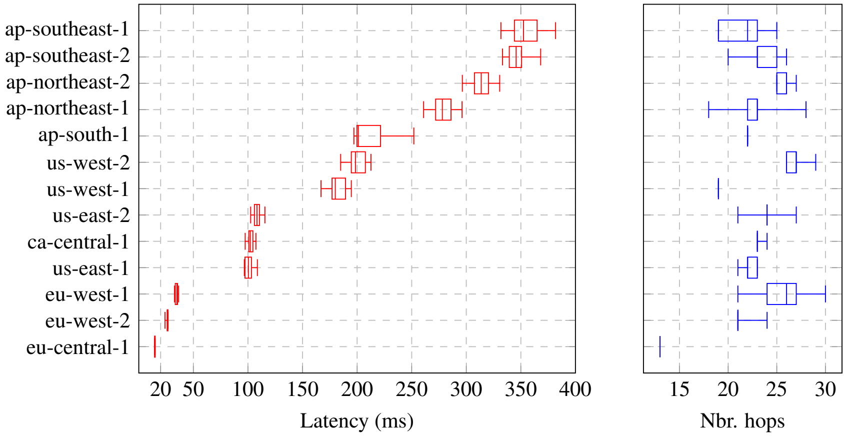


Figure 1.6.: Network delay over campus network to AWS availability zones.

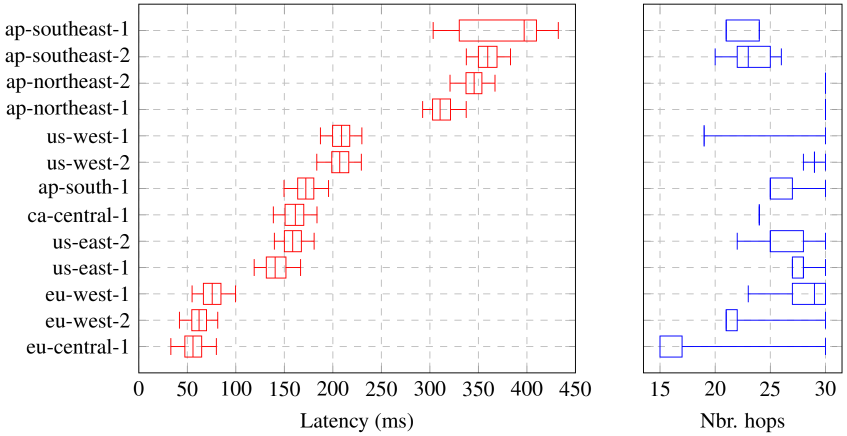


Figure 1.7.: Network delay over public LTE network to AWS availability zones.

significant as you leave the ISPs' networks.

CLOUD NATIVE SERVICE PERFORMANCE

Compounding tens of *ms* of network delay with system delays at the cloud provider's DC, the RTT can extend well into $1s$. In this section, an IoT-like application is built using a composition of AWS services to study cloud platform system delays. The application is then used to gauge the RTT of the application as a composition of AWS services and its constituent AWS services.

An IoT application is assumed to consist of at least one connected device that sends

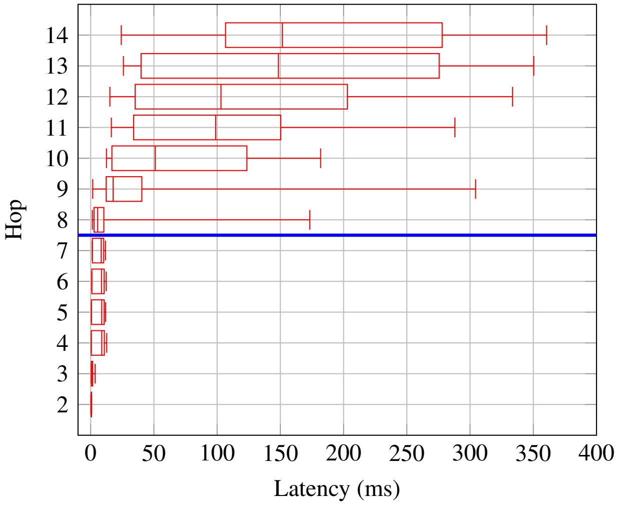


Figure 1.8.: Number of hops to all AWS availability zones. Blue line indicates median end of ISP's network.

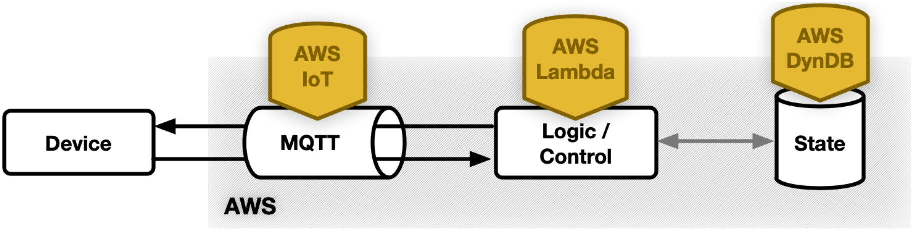


Figure 1.9.: Rudimentary IoT application architecture using AWS services.

data to the cloud. The data is processed in the cloud, a result or action is then returned to the device and saved in the cloud to represent the state of the device or the system. The application is built using AWS IoT for device management and communication, AWS Lambda for computation, and AWS DynamoDB for storing state. The device is, in this case, a desktop PC running Ubuntu Linux 17.04, connected to the Lund University campus network. The device posts a value to an uplink AWS IoT MQTT topic at a set rate. A rule in AWS IoT triggers an instance of an AWS Lambda function on the uplink post. The AWS Lambda function reads a value from an AWS DynamoDB table, combines it with the reported value, updates the Database (DB), and returns the value. The Lambda function publishes the resulting value on downlink AWS IoT MQTT topic, to which the device subscribes, thus closing the communication loop. The AWS Lambda function was implemented in Python and Boto 3 was used throughout. An overview of the architecture is provided in Figure 1.9.

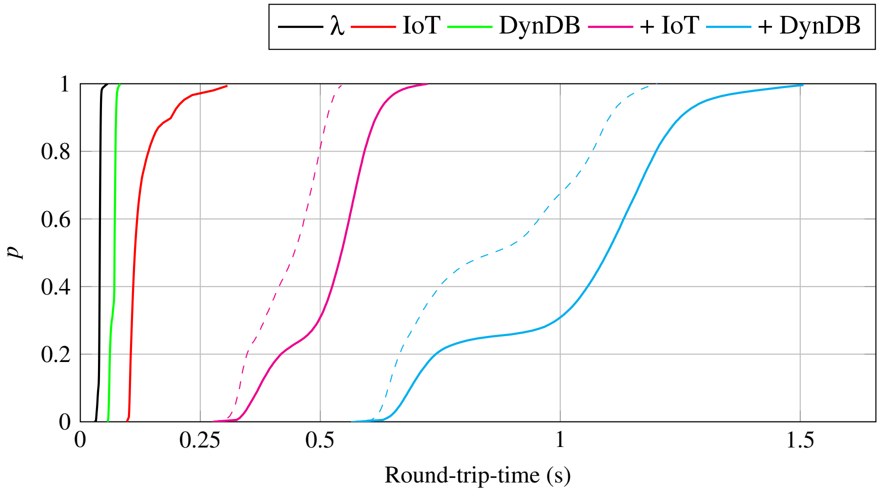


Figure 1.10.: RTT of AWS services common in cloud-based IoT-applications. Dashed line represents the 'brute-force' approach. + represents the addition of that service, i.e. '+ DynDB' therefore includes all λ, IoT, and DynamoDB.

Figure 1.10 shows the eCDF of the RTTs for the targeted IoT application, measured with 1000 samples every 15 minutes from August to October 2018. Note that the mean RTTs for the individual services, namely AWS IoT, AWS Lambda, and AWS DynamoDB are 129ms, 40ms, and 69ms, respectively. It is clear from Figure 1.10 that communicating over an AWS IoT MQTT queue incurs a significant delay and a heavy tail. Note that the samples captured during the warm-up period, for each sampling instance, have been removed. The warm-up time can be up to 300ms for AWS Lambda, 5s for AWS IoT, and 72s for AWS DynamoDB. The values presented in Figure 1.10 should, therefore, be considered to have been produced by a continuously operating system, under ideal circumstances.

AWS IoT and AWS Lambda are combined in the eCDF labelled '+ IoT'. The complete application with the DB component is presented in the eCDF labelled '+ DynDB'. The solid lines show the synchronous RTT. Notice that the incurred RTT for the entire application (+ DynDB) is significantly greater than the sum of the individual services. From Figure 1.10 it is clear that accessing DynamoDB in a lambda function comes with a substantial time penalty.

The dashed lines show the lowest RTT out of 20 near-concurrent asynchronous calls. The cost of each additional concurrent call grows linearly, but the benefits diminish exponentially, in both scenarios, as seen in Figure 1.11. The disparity shows that there are gains to be made with this 'brute-force' approach and hits at the innate quality variance of AWS's services. This effect is an indication of the performance variance

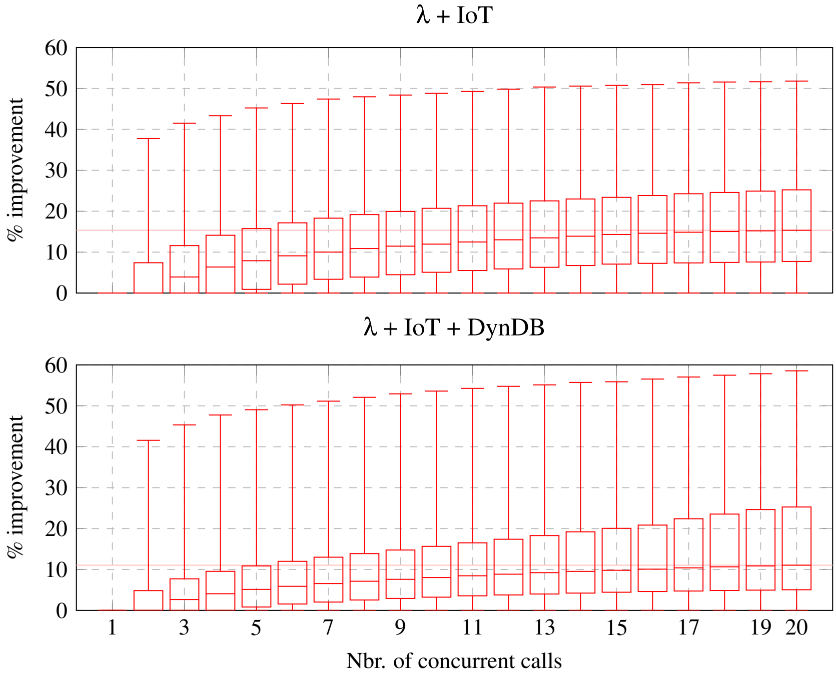


Figure 1.11.: Benefit of 20 concurrent asynchronous calls. The pink line demarks the highest median.

one might encounter with cloud-native or micro-services in the cloud.

1.3. FOG COMPUTING

Fog computing is emerging as a new cloud paradigm in the wake of IoT and an increasingly distributed cloud infrastructure. IEEE 1934-2018 [83818] defines a Fog computing system as a system of heterogeneous and distributed cloud resources. Before IEEE's draft standard what we now know as Fog computing was known by many different names. For example, half a decade ago; omnipresent cloud, distributed cloud, infinite cloud, mobile edge cloud, and edge computing were all emerging and partially overlapping definitions.

The premise of Fog computing is to make cloud computing accessible to time-sensitive, and mission-critical applications where traditional DCs don't suffice. By introducing cloud capacity proximal to the end-users, applications are accessed at lower delay with less jitter in the delay and with greater network reliability. User-proximal nodes also permit applications to consider data and processing-locality constraints, as well as more elaborate fault-tolerate mechanisms relying on relative geographic ad-

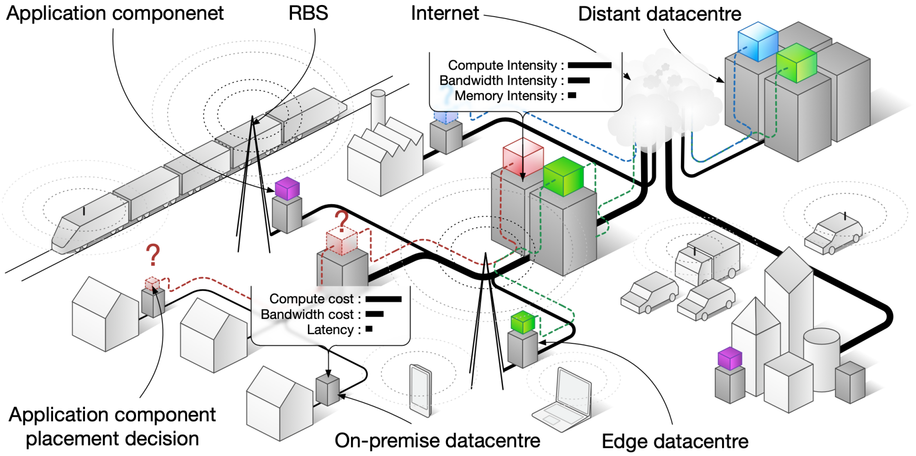


Figure 1.12.: Schematic overview of a fog computing infrastructure

vantages. Here, proximal resources are resources near the end customers in contrast to traditional DCs.

Recently, a distinction has emerged between Edge and Fog computing, [AZTS18]. Edge computing is becoming synonymous with technologies and systems that take advantage of IoT and mobile devices at the edge of the network. These devices are managed in a cloud-manner and applications can span across or move between devices and traditional DCs.

In this work, a Fog computing computing infrastructure is defined as:

A system of a set of heterogeneous hyper-distributed general-purpose cloud computing nodes with varying degrees of proximity to the system's end-users. The system's resources can be federated and opportunistic, meaning that they are not necessarily purposefully for Fog computing. Applications deployed to such an infrastructure can take advantage of the infrastructure's heterogeneity to meets their performance targets. Likewise, the infrastructure is actively managed to meet its performance goals as well as those of the applications.

Figure 1.12 provides a general overview of a Fog computing infrastructure. Here, cloud resources are distributed in a network ranging from traditional distant DCs to the edge of the mobile network, e.g. in RBSs. The resources vary in capacity and ability across the network. Computational capacity arguably decreases with network depth as DCs further down the network will serve fewer end-users. Conversely, the cost per compute and time unit arguably increase with network depth as economies of scale diminish with successively smaller DCs. On the other hand, aggregate bandwidth cost will likely decrease with depth. In this thesis, the mobile network plays an integral part of the Fog computing infrastructure. With the arrival of 5G, the networks are capable enough to support Ultra-Reliable and Low-Latency Communication (URLLC) and massive Machine Type Communication (mMTC) applications which would justify Fog computing.

Numerous proposed use-cases have emerged in the Fog computing research sphere. Massive IoT is a reoccurring theme [BMNZ14] that has gained traction amongst cloud providers. If an over-engineered IoT-world materialises, it arguably needs the support of Fog computing. In such a setting; compute, storage, and networking are subject to data gravity [YIJ17b]. Hauling data to distant DCs is often infeasibly when concerning both cost and performance [YIJ17a]. Some non-controversial use-cases are; mobile offloading, caching, and Network Function Virtualisation (NFV). More targeted use cases include building smaller DCs in areas with computing needs but with poor connectivity. One can imagine both ships and oil rigs that continuously produce large amounts of data and that want to have that data timely analysed for operational efficiency. Oil rigs and ships often rely on expensive, low throughput, and high latency satellite or radio links. The idea is here to process the data where it is needed and only communicate what has value to the outside world. More technically challenging use-cases revolve around collaborative autonomous vehicles and vehicle platooning [TRA$^+$17]. These use cases fall under the notion of serendipitous or augmentative computing, meaning that the vehicles do not necessarily rely on Fog computing but will take advantage of it whenever and wherever it is available. Due to heterogeneity in both infrastructure and performance requirements, applications in a Fog computing infrastructure will have to be process-elastic on a conceivably broad range. Quality elasticity is not just technically challenging, objectively. Subjectively, users' expectation has to be appropriately managed.

A more straightforward but very relevant use case is when obsolete user-proximal HW is structurally or serendipitously augmented by Fog computing resource. Imagine a piece of production machinery. Modern production environments are dynamic; processes are adapted and integrated to meet particular performance targets. The compute HW that was delivered with the machinery will not suffice for long in such an environment. Neither is it in the production machinery manufacturer's business interest to supply and support computing capabilities in their machines that will quickly outgrow their customer's needs. Instead, ultimately all process can be executed in a proximal cloud node, sharing data amongst each other and peripheral ordering, monitoring, and management systems. For a recent comprehensive survey see [YFN$^+$18].

1.3.1. INFRASTRUCTURE CONVERGENCE AND FOG COMPUTING ATTRACTORS

Although an actual Fog computing infrastructure does not currently exist, many industry-lead developments are resulting in a more distributed end-user-proximal cloud infrastructure. In many aspects, new and incumbent cloud providers are leaping towards the edge of the network. For example, investments are being made in edge-aware services and infrastructure relatively closer to the end-users. 5G and the use-cases it promises to deliver is often an instigator for these investments. The primary factors driving the convergence towards Fog computing are detailed in this section.

With exponential growth in cloud computing usage, providers are continuously increasing their availability presence. The dominant cloud providers are investing in more and larger DCs in more regions. The result is a geographically denser network of DCs. Having a DC in the backbone network near to where you operate can both significantly reduce communication delay and increase availability. Not to mention, your data can be stored where it can more easily be stored in the originating geographical region. In some regions, this is required by law. These investments should not necessarily be seen as a deliberate encroachment towards the edge. These DCs are superficially equivalent and are meant to increase general availability and not specifically to satisfy edge use cases. Recently, AWS has begun to offer AWS infrastructure on-premise through a service named AWS Outposts [1]. This allows customers to run AWS-native application on their premises and managed as if a AWS DC.

To satisfy classic edge use cases, AWS and Microsoft have begun to offer specialised edge solutions. They come either in the form of CDN-type caches that can host limited dynamic content (Edge Side Includes (ESI)), to simple on-premise IoT devices running a SW-platform provided by the cloud operator. These offerings allow developers to deploy code to the edge or on-premise, typically as a FaaS. Although these locally hosted platforms are well integrated with the cloud operator's service offerings, they are primarily intended as a means to extract and haul data to a distant DC for storage, processing, and a ML model. Furthermore, the platforms don't scale as DC-based cloud computing, and the application owner is responsible for deciding where to place application components.

CDNs such as Akamai and Cloudflare have extensive networks of user-proximal DCs. Although not at the magnitude of the big cloud providers' DCs, they scale well to the demand in their network vicinity. Typically only hosting static content, CDNs such as Cloudflare have started to change their infrastructure to allow it to host simple applications [Clo18b] (ESI). They are primarily intended to intercept and personalise web responses dynamically and are therefore not equatable to the FaaS services offered by the big cloud providers. However, ESIs are a significant step towards intercepting, routing, and manipulating in-transit web and IoT traffic.

Starting with 4G, telcos and Telecom Original Equipment Makers (OEMs) have had the ambition to virtualise some to all functions of the Radio Access Network (RAN). NFV involves deploying network functions previously hosted on dedicated and specialised hardware to cloud-like DCs with GPPs. The idea is to aggregate the functionality to a set of geographically proximal RBSs in one common DC or processing node. With full virtualisation, a RBS is split into a Remote Radio Head (RRH) and a Baseband Unit (BBU). The RRH and the BBU can be spatially separate. The RRH relays any baseband signals over what is called a front-haul network to the BBU for processing. This paradigm is often referred to as Cloud-RAN [CCY$^+$15], CRAN, or virtualised RAN. The prospected benefits are the ability to load-balance workloads

[1] https://aws.amazon.com/outposts/

across cells, more flexible and scalable HW independent deployments, and reduced cost. Again, the draw of elastic GPP computing resources is a major attractor. Additionally, virtualising the telco's infrastructure would also allow equipment owners to slice their infrastructure.

Network functions are very latency sensitive, especially lower level Medium Access Control Layer (MAC) and Physical Layer (PHY) functions [CCY+15]. Latency sensitivity is set to increase as throughput requirements and Radio Access Technology (RAT) processing increases with each successive 3rd Generation Partnership Project (3GPP) iteration. Higher level functions such as billing and location registry are already deployed to GPP platforms. Furthermore, routing and fire-walling have been running successfully in the cloud for some time. However, MAC scheduling decisions, channel coding, and estimation occur at a very high rate, proportional to the channel throughout. Therefore, the delay and jitter in the delay subjected to the baseband signal would have to be much lower than what can be achieved over the public Metropolitan Area Networks (MANs). Coupled with GPP-based cloud platforms many RAN functions' real-time constraints cannot be satisfied with the traditional cloud paradigm. Full stack virtualisation might require bespoke front-haul networks and bespoke hardware in the BBU. Again, moving away from GPP erodes the benefits of the cloud. Analogously, these requirements limit the potential geographical reach of virtualised telco systems.

Telcos are new to the cloud domain. They are accustomed to providing availability rates that are much higher and latencies that are much lower than traditional cloud providers. Telcos are currently implementing or investigating business-models for selling any excess cloud capacity embedded in the network. Such services range from CDN-type caching to dynamic applications, to NFV. This is often referred to as the *telco cloud* [SGP+15]. The advantage, a telco cloud can potentially offer application owners deeper integration into the network by for example exposing users' locations and prevailing network conditions. The telco-operated DCs will arguably not be at the scale of traditional DCs and therefore not achieve the same level of economies of scale. Conceivably, they need to operate with a higher utilisation factor.

The positive effects of the fog computing paradigm might be undercut by developments in RAT, RAN, and Wide Area Network (WAN) technologies. With 5G, RAT latency is proposed to go down to $1ms$, in what is called Ultra-Reliable and Low-Latency Communication (URLLC) [SMS+17]. The backbone networks are also continuously getting upgraded. The star and ring topologies are giving way to more point-to-point topologies and Software Defined Networks (SDN). Meanwhile, optical fibre is becoming the norm for the last mile. All of these improvements contribute to lower latency communication on less congested networks. On the other hand, Network Function Virtualisation (NFV), SDN, and slicing might contribute to increased aggregate network delay and jitter in the delay.

1.3.2. ELASTICITY IN THE FOG AND APPLICATIONS

A fog computing infrastructure provides to a large extent the same degrees of freedom as traditional cloud computing, namely; cost, resource, and quality, but also adds spatiality and heterogeneity. An application deployed to a fog computing infrastructure can disaggregate, place, and scale individual components at different geographical points in the network, in whichever constellations it may achieve its desired performance, at that point in time. Although a Fog computing infrastructure has a wide pallet of resources, they will arguably not be as resource elastic as a traditional DC. Smaller DCs will have fewer resources on which to scale and will come at a higher cost.

Responding and mitigating to a ephemeral system state and a mobile demand is non-trivial. Continuously migrating full applications or application components around a massive fog computing infrastructure, at every whim, is not desirable. Migrating and starting up VMs and containers, and reconnecting data paths incurs a significant performance overhead and can have a profound negative impact on smaller DC and on the application itself. With higher load factors and to some degree a constant churn, applications will arguably be exposed to more execution jitter. With dynamic application placement, a highly mobile user base, and resource-constrained edge applications the Fog computing infrastructure can quickly become confined to an undesirable or even inoperable state.

To take advantage of and cope with increased heterogeneity and uncertainty, applications in a Fog computing infrastructure should be encouraged to embrace an osmotic existence. *Osmotic computing* [VFD$^+$16] is an application paradigm that is driven by the emergence of distributed heterogeneous cloud infrastructures. Operating conditions frequently change because resources can be scarce and are shared by many. Additionally, a Fog computing infrastructure owner's management policies also contribute uncertainty. Applications, therefore, need to have the ability to adapt to a contracting environment. An osmotic application takes advantage of the ephemeral and heterogeneous nature of such infrastructures by continuously practising resource, cost, and quality elasticity. Application components scale vertically and horizontally in the infrastructure to where they incur the least cost, perform the best, or where they can take advantage of a unique resource. Furthermore, applications do not necessarily have the necessary quantity or type of resources available in the part of the network to meet lofty or ideal performance targets. Even though the current operating conditions are undesirable, moving an application or one of its components might not always be cost-effective or feasible.

This trade-off is non-trivial. For an Elastic Reasoning Mechanism (ERM) to achieve elasticity in every dimension require a great deal of insight into the infrastructure and the dynamic properties of the resident applications. Attempts have been made at designing an ERM for IoT workflows [NND$^+$17].

The principles of osmotic computing do not singularly apply to fog computing. The service offering by traditional cloud providers is also becoming more heterogeneous.

Additionally, an elastic application is a robust application. For example, a quality elastic robust web application is proposed in [KMÅHR14].

1.3.3. FOG COMPUTING DETRACTORS

Fog computing bows to the same detractors as cloud computing, only exacerbated by heterogeneity and distribution. Given the scale of these new resources, one can then argue there won't be enough elasticity at the edge to allow for the dynamic osmotic effect that we desire in a fog computing infrastructure. The relative cost of executing, maintaining, and deploying small edge devices will conceivably only increase as large DC approach ever greater economies of scale. Moving to a hyper-distributed infrastructure also implies a software development paradigm shift. Once the dust has settled, and applications have found their rightful place in the infrastructure, how likely is it that they will ever have to move and will application developers or operators even want them to, given the uncertainty this adds?

Part I.

Modelling and managing a Fog computing infrastructure

2

Mobility

End-user mobility is a key differentiating factor between traditional DC -centric clouds and Fog computing. In Fog computing, an end-user's location, within a few meters or kilometres, determines in which DC an application executes. The rate at which the end-users move determines to what extent and to where an application needs to be migrated to achieve its performance target.

The relationship between application performance and geographic location has received little research attention [SNM10, ZDZQ13]. There is thus comparatively little research bridging state of the art cloud hosting research and a cloud's ability to operate in a mobile network with mobile end-users. What is explicitly lacking is how the mobile end-user's generated workloads will vary and be displaced between Fog computing DCs as a consequence of end-user mobility and a study of the associated resource cost. The authors of [GHMP08] investigate DC latency in geo-distributed networks in the context of the operational cost of transmitting and operating the intermediate network at the desired performance level. The authors of [ADJ$^+$10] studied the effect of migrating end-user instances geographically to existing geo-distributed DCs, in response to a end-users location on a global, inter/intra-continental scale. However, in Fog computing, end-user movement is potentially significantly more granular.

This work explores the fundamental dynamics of workload displacement as a result of end-user mobility between independent DCs adjacent to and associated with an RBS. The paper then proceeds with examining the proportion of workload being displaced to adjacent DCs, and the proportion of resources the act of migration consumes in a DC proportional to the work it completes. A simulation model is proposed, that includes the basic building blocks of Fog computing, in conjunction with a mobility model aimed at provoking and exploring basic system workload displacement vulnerabilities and the dynamic effects on an application's performance as a result of mobility.

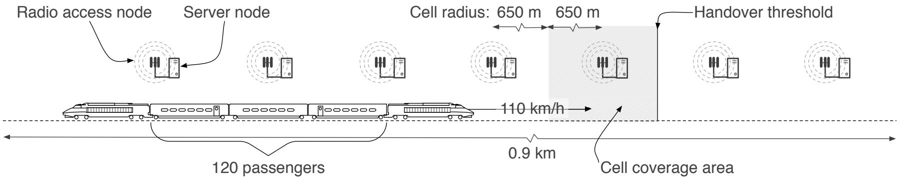


Figure 2.1.: One dimensional simulation scenario

The results show that end-user mobility in a Fog computing topology prompts a cumulative spatial displacement of the workload in successive DC. Additionally, when the DCs are over-provisioned, the simulation reveals that the DCs, at an increasing rate, spend more time migrating VMs than executing them. As a result, a stable system-wide waiting time is only attainable with a system load of less than 80%. The simulations also reveal that despite a stable system, the waiting time still increases in the spatial domain as a result of end-user mobility. The paper also investigates an end-user's utility in subscribing to a Fog computing node.

Section 2.2 details which aspects and abstractions of a Fog computing topology that are included in the experiments. Furthermore, the resulting simulation model and its constituent parts are specified in Section 2.3 followed by Section 2.4, which accounts for the specifics of the simulation experiments. Lastly, Sections 2.5 and 2.6 present the results and consultations drawn from the experiment.

2.1. TARGETED SYSTEM

An application is geographically migrated with the end-user(s) to the closest DC, to maintain proximity to its end-user(s) as it moves around the network. Moving to the closest DC is a naïve approach and should be seen as a lower performance bound. Proposedly, where Fog computing infrastructure is available an application instance can be migrated from a distant DC where it traditionally resides to a Fog computing DC in the mobile network. As mobile end-users move through the network, and when it is deemed feasible to migrate an application given a geographic discrepancy, the concerned VM is migrated to where latency and congestion are minimised. However, doing so will incur an additional load both on the receiving and sending DC, and the intermediate WAN. Moreover, migration and its overhead are minimised when the amount of work completed in each DC is maximised during a end-user's residency, and when inter-DC transmission is minimised.

2.2. TARGETED SCENARIO

To explore an extreme scenario, in this paper, to strictly minimise the proximity to the end-user, each abstract RBS will host a cloud server entity, a DC, see Figure 2.1. From now on, an RBS DC pair is referred to as a *node*. To be able to observe consecutive workload displacement, end-users are displaced according to a train model at a constant speed along a linear path through a one-dimensional space. Furthermore, throughout the one-dimensional space, RBSs are positioned equidistantly.

In the proposed model, end-user movement and network resources are homogeneous. As a result, the proportional displacement of workload between comparable nodes, as end-users move between nodes, can be observed. Additionally, this will also show the subsequent proportional degradation of perceived application quality, experienced by the end-user over the whole network. One will also be able to discern the rate of which an application needs to be migrated, which can be seen as an abstract measure of the scale of a resulting VM or container migration. The simulation will show how mobility affects the proportion of sessions that will be migrated between consecutive nodes, consecutive degradation of waiting time, and the potential resulting VM migration burden imposed on the system.

2.3. SIMULATION MODEL

The simulation model is discrete-time and contains multiple independent end-users N_u, each with a unique location determined by a train mobility model. A end-user's location within a network determines which singular RBS it is associated with.

The modelled network contains multiple, equidistant RBSs. Each RBS or cell has a fixed coverage radius, r_{cell}. The network re-evaluates end-user and RBS association at a specific rate throughout the simulation. All end-user-generated requests are sent to its current associated RBS. The RBS forwards subsequently all incoming requests to a single node which processes the incoming requests at a particular service rate $T_{service}$.

2.3.1. APPLICATION MODEL

The adopted application model is based on the open-loop, one tier, long tailed, HTTP request model detailed in [BC98]. The modelled traffic is consistent with web surfing on mobile devices, where end-users access mobile-adapted web pages with very little in-line dynamic content, revisited at a high frequency. Additionally, the duration of the resulting sessions is proportional to the radius of the networks cells. Each session spawns some requests proportional to the File size (S_f) and the Request size (S_r) in KB, both Pareto-distributed. Each request is separated by an Inter-request Weibull-distributed delay (D_r). Moreover, each session is separated in time by a Pareto distributed inter-session delay (D_s).

2.3.2. NETWORK MODEL AND TOPOLOGY

Each RBS is bounded by a cell coverage radius, r_{cell}. Given that an end-user is within the aggregated cell coverage of the network, that end-user will always be associated with the RBS closest to it. The network periodically evaluates each end-user's proximity to all the RBSs in the network. If an end-user moves closer to another RBS, at that threshold, a handover will occur, and the RBS association will be updated, see Figure 2.1.

2.3.3. MOBILITY MODEL

The simulation model uses a train mobility model in one dimension with clusters of N_u end-users, and a constant velocity, V_{train}. This train model presents an extreme mobility condition where the total end-user population and thus traffic is displaced in concentrated groups from node to node, progressively and permanently abandoning RBSs in rapid succession.

2.3.4. DC MODEL

Each DC in each node is modelled as a single server queue that processes requests from its deferred queue with an exponentially distributed service time $T_{service}$. Furthermore, when a end-user is handed over from one RBS to another, all deferred requests from that end-user in the active node queue are instantly migrated to the newly associated node. More precisely, this occurs when the current process is completed and incurs no additional load to the network or the server. The migrated requests are placed at the end of the receiving node's queue. Any ongoing processing is completed before the migration procedure begins.

The mechanisms that govern the provisioning of network resources and cloud resources are, in this model, independent. The association and connection between RBS and a DC is not specific to any particular mobile system generation topology.

2.4. EXPERIMENTS

The adopted simulation model was implemented as a discrete-event Java simulator using simjava [HM98] as the event engine. To be able to evaluate geographic load displacement and the subsequent application performance degradation in relation to server load scenarios, using the model above, server load levels at 50% to 150% were deployed in the simulation model. Furthermore, server load is defined as the inverse percentage of the request service time $T_{service}$. Moreover, the request service time is defined as the quotient of the total arrival rate at full end-user residency, see Equation 2.1, where λ_i is the arrival rate for the ith end-user. For example, a 50 % load is when $T_{service}$ is twice as high as the aggregate inverse arrival rate.

$$T_{service} = \frac{1}{\sum \lambda_i} \tag{2.1}$$

To ensure that the system is subject to multiple migrated sessions, the mean application session duration is set proportional to the radius of a cell, r_{cell} and the velocity of the train. As a result, all requests equal to and below the mean session length will on average be completed in one node, while those above, will on average, be subject to migration. Given the previously mentioned node displacement and end-user spatial density, each end-user will be associated with and reside within the domain of each

Parameter	Value
r_{cell}	650 m
N_u	120
V_{train}	110 km/h
$T_{service}$	50-150% of 0.0039 seconds
T_{sim}	8,8 minutes (7 nodes)

Table 2.1.: Simulated environment parameter values

Component	Distribution	Parameters
S_f	Pareto	K=133000 α =1.1
S_r	Pareto	K=1000
D_r	Weibull	α =1.46 β =0.382
D_s	Pareto	K=1 α=1.5

Table 2.2.: Application model parameter values

RBS for 40 seconds.

The simulation runs for T_{sim} minutes, through which the train of passengers pass through 7 RBS domains. Given the application model described in Section 2.3.1, the simulation reaches its steady state after 3.6 simulation minutes, at which point the first end-user gets in range of the first RBS. Consequently, the total steady-state simulation time amounts to 5,2 simulation minutes. The steady-state simulation time is sufficient to allow each end-user to spawn several open-loop sessions and thus to reveal the fundamental dynamics of the system. Designedly, the first node will not be subject to migrated requests.

The simulation scenario includes several node load levels. Feasibly, homogeneous nodes subject to a load higher than 100% results in an unstable system with a transient workload growth. Given a certain end-user velocity, an unstable system will experiance varying application response times with displacement. Note that, as the system is modelled without signalling latency, the waiting and service times can be regarded as the server response time.

Furthermore, application model parameters are sampled from the distributions in Table 2.2 in accordance with [BC98]. Similarly, Table 2.1 details the global simulated environment parameters.

Each node is sampled for; queue length, waiting time, and processed and migrated request sizes per session. These parameters allowed us to reveal how mobility affects the proportion of sessions that will be migrated between consecutively nodes, consecutive degradation of waiting time, and the potential resulting VM migration burden placed on the system. The resulting data is comprised of the mean of 10 independent replications.

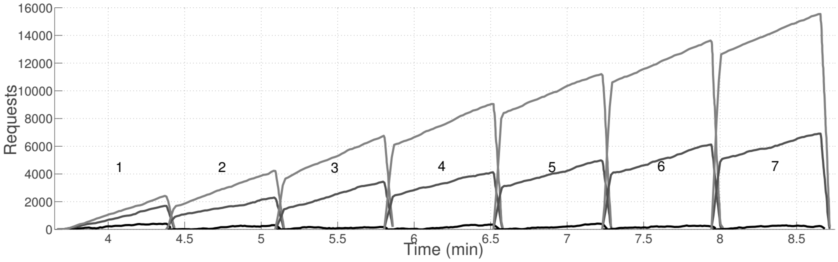


Figure 2.2.: Queue length displacement at 100%, 110%, and 120% load, respectively. Each node is marked with its corresponding number.

2.5. RESULTS AND DISCUSSIONS

In this section, the results from the simulations and their implications are presented. Figure 2.2 shows how the workload is spatially displaced when nodes are subject to a load greater then 100%. As end-users move out-off and in range of the subsequent node, any incomplete requests will be migrated to the subsequent node. The mean deferred queue length exhibits growth according to $c \cdot n_i^l$, where n_i is the ith node and l the load quotient, e.g. 120% = 1.2. Additionally, given that the sessions are longer than the duration a end-user spends in a node, the subsequent nodes will need to, on average, be able to absorb the additional migrated load.

Figure 2.2 reveals the load point where the system becomes unstable. Any node loaded greater than 100%, results in an unstable system with progressive degradation of waiting times. As a consequence of end-user mobility, a cumulative amount of workload is migrated to the subsequent nodes to the point where the system is unable to recover.

Furthermore, note that Figure 2.2 shows how the deferred queue length at 100% load grows during maximum end-user residency to the point where sessions are not completed and are thus migrated to the subsequent node. Nevertheless, both the sending and receiving nodes can recover during the transitions between nodes, and thus maintain stability.

2.5.1. WAITING TIME DEGRADATION

Degradation of waiting time is another consequence of the above-mentioned progressive workload build-up. Degradation occurs when the nodes are subject to loads greater than 80%, which is shown in Figure 2.3. As can be seen, the mean waiting times during max residency increase linearly for each consecutive node. An end-user will thus experience a linear degradation of the mean response time in space. Also, the mean waiting times for each node as a function of the load level grows quadratically with increased load.

As illustrated by Figure 2.2, at the maximum stable load (100%), beyond which, the

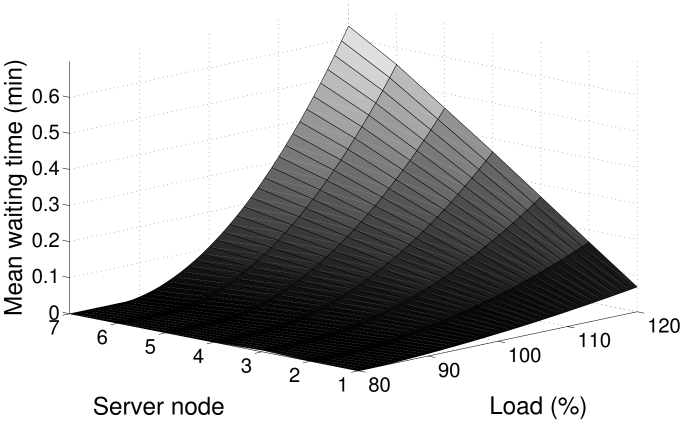


Figure 2.3.: Waiting time degradation

queue length diverges, the system is able to maintain a consistent deferred queue length and session residency, but because of migration and the resulting session migration effort, waiting time degrades 5 fold across the span of the network. Only at a load of less than 80% is the system able to recover the incurred migration effect and thus maintain a consistent waiting time. This implies that to maintain system stability the individual nodes can never be provisioned to utilise 100% of its resources.

2.5.2. SESSION AND VM MIGRATION

It was shown above that request migration incurs a degraded response time. Furthermore, each of those requests constitutes a subset of a session. As detailed earlier, each session is regarded as a VM instance in a generic cloud server. As such, observing the residence and migration of sessions reveals how often VM migration occurs and the potential load a VM migration can incur.

The investigations show that at 100% load, 90% of the VM are completed in one node and are not subject to migration. On the other hand, at 120% load, on average, a VM in the last of the 7 nodes only completes 10% of its request, the corresponding value for the first node is 20%. Moreover, at a 120% server load, on average 65% of the incoming requests receive 0% of that node's compute cycles. In other words, some VMs do not receive any resources to complete any of its requests despite the system spending resources migrating these VMs to the next node. At this point, the paradigm is contributing far more latency than it is eliminating.

2.5.3. VM MIGRATION TIME

Concerning the VM migration time, to maintain a consistent waiting time and allow a migration to recover, VM migration needs to be performed within the period of the

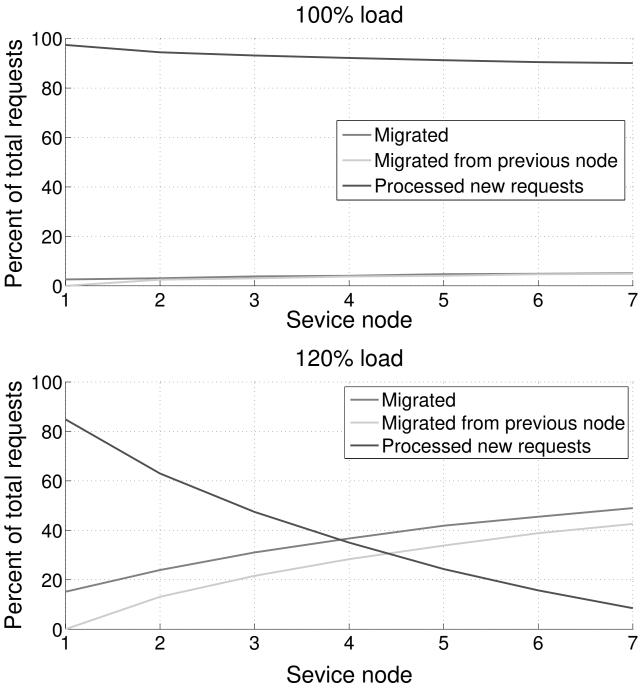


Figure 2.4.: Migrated vs. processed packets

mean waiting time. The simulation discloses that waiting time recovery is only feasible at less than 80% load, and is only fully able to do so when the system is subject to a load less than 50%.

2.5.4. REQUEST MIGRATION

In contrast to sessions or VMs, the rate at which requests are processed versus end-user node residency is a metric of utility. Figure 2.4 displays the proportion of processed requests that were generated in the domain of that node. The figure reveals that the total received requests decays exponentially with each subsequent cell. At 120% load, the first node processes 90% of the requests generated in, while associated with that node. The rate diminishes to 8% in the final node. Feasibly, the utility of subscribing to that node is negligible. Moreover, at 100% workload, the amount of requests being processed that were generated while subscribing to that node decays faster than the number of migrations, which quickly converges. This behaviour is a contributing factor to why the waiting time is decaying in an otherwise stable system, as discussed above.

Consequently, the amount of time spent processing migrated requests by each node grows exponentially, converging to where no intra-node generated requests are processed. At this point, the migrated VMs contribute more requests than what is generated within the domain of the server node. It would arguably be more efficient to eliminate much of the migration by consolidating multiple nodes and spend those resources on processing tasks.

Furthermore, the mean waiting time in proportion to the time spent in the domain of a node gives you one metric of how much that node is contributing work. At the far node, at 120% load, almost the whole residency is rewarded with, on average, 1,11 processed requests. As such, using that node carries very little return. The effect of the diminishing return of the time spent in a node is shown by Figure 2.4.

2.5.5. SESSION MIGRATION VERSUS NODE RESIDENCY TIME

Another relevant comparison is that of session migration versus node residency, which corresponds to the general scale requirements of the resources. As one can expect, in a stable system the number of VMs will remain relatively constant over time, given a 100 % workload. It is made evident by Figure 2.2 that the system can recover from temporary overloads in one node, as any excess workload is gradually spread to the adjacent vacant nodes. This self-balancing effect is, of course, proportional to the distribution of end-users, the speed of which they are moving in and the dimensions of the RBS cells.

2.6. CONCLUSIONS

The Fog computing model and simulation reveal the challenges facing mobility in Fog computing. The simulation results made it apparent how mobility incurs severe progressive workload accumulation, and that VM migration will contribute to a large overhead, depending on the topology. The incurred VM migration load on the system consumes such a large proportion of the system's resources that it will require the system administrators to greatly over-provision the system to maintain consistent performance.

It was also made clear that the return of subscribing to the closest Fog computing node has a diminishing utility with node order and server load. At the simulated extremes, slightly more than 1 request is processed during the time an end-user on average spends in a cell. Thus the cost of migrating the session far exceeded the amount of work it contributes.

Complementary, it will conceivably be relevant to determine network topological placement of Fog computing DCs and determine the effects of applications and VMs migrating to and from a distant DC and horizontally in the network, and through other network access media, such as 802.11, as a means to load balance the system of distributed DCs.

3

Modelling and system architecture

Despite the interest in the Fog computing paradigm, there are no simulation models capable of simultaneously modelling the dynamics of UEs, placement and capacity of DCs, and network infrastructure. Understanding these relations is essential for Fog computing stakeholders, e.g., Infrastructure Providers (IP) can use that knowledge to reduce infrastructure costs, while still delivering competitive performance.

In this chapter, a comprehensive Fog computing meta-model is proposed, which facilitates experimentation and evaluation of possible configurations, such as placement and capacity of DCs. The meta-model uses existing, well-established simulation models, e.g., for RANs or DCs, for modelling of the aforementioned individual parts of the infrastructure behaviour. The model describes the dynamics of a Fog computing infrastructure, including QoS, and the associated costs of this paradigm. Additionally, a meta-model is presented that captures the described dynamics using existing and composite models.

3.1. EXISTING FOG COMPUTING MODELS

To support the creation of a meta-model that incorporates workload, set-up, and objectives of the Fog computing described in the previous section, existing models are surveyed in the following categories: application request generation and resource requirements, UE mobility, networks, DCs, and infrastructure costs.

Most of the models and simulators are assigned to only one of the categories mentioned above. However, the capabilities of the four surveyed simulators extend to many categories. They are summarised in Table 3.1.

Table 3.1.: Overview of surveyed simulators.

Framework	RG	RR	M	N	DC
NS-3	✓		✓	✓	
OMNeT++	✓		✓	✓	
CloudSim		✓			✓
GreenCloud		✓			✓

RG – Request Generation, RR – Resource Requirements,
M – Mobility, N – Network, DC – Data Centre.

3.1.1. WORKLOAD MODELS

Applications running in the Fog consist of a set of User Equipments (UEs), users, and a server processing offloaded computations. Therefore, they should be modelled from two perspectives: *request generation* that describes how requests are created and sent to the DCs; and *resource requirements* that describes how many computational resources are needed to process the requests.

TRAFFIC

Traffic models capture a user's behaviour by primarily representing interaction times, or the timing clicks through a stochastic process, often Poissonian. A user behavioural model can be further refined by introducing a stochastic model for the duration of time a user consumes a particular type of content. The transition between types of content is conventionally modelled as a Markov process.

Furthermore, the traffic characteristics are commonly modelled with multiple stochastic processes, encompassing the number of packets in a session, and the size of each packet. Traffic models are either closed or open looped. In an open loop model, the generation of each new session is typically a Poisson process independent of the resulting DC action. Conversely, in a closed loop model, the generation of new sessions is dependent on the timing of the response from the DC and thus the properties of the previously generated session.

In the packet-level event driven network simulators; NS-3 [RH10] and OMNeT++ [V$^+$01], a node can act as either a client or server. Either by sending packets provided by a stochastic model, at a given rate, within a specified period, and at a specified interval, or processing received packets from a buffer, at a given rate. Both server and client models can be augmented with a more complex system of queues to such an extent that they can represent an abstract DC that hosts multiple applications.

RESOURCE REQUIREMENTS

CloudSim [CRB^{+}11], which is a simulator of cloud infrastructure, provides an application model that describes computational requirements – the number of resources that needs to be available (e.g. number of cores, memory and storage); and communicational requirements – the amount of data that needs to be transferred. GreenCloud [KBK12], which is a packet level simulator based on NS-2, apart from computational and communicational requirements, also describes QoS requirements, expressed by an execution deadline. The application model may also include the size of the code that has to be offloaded and dependencies on other services, e.g., regarding the amount of data that has to be sent or received [Kov12].

MOBILITY

The NS-3 and OMNeT++ nodes described above can be set into motion given a specific stochastic mobility model. They can, for example, traverse the space as pedestrians, or automobiles, with corresponding velocity and rate of change. The spatial relationship between nodes and RBS affects the current channel properties and RBS-to-node associations. Node mobility will also result in handover between RBSs, which in turn will alter the paths of the node-generated workload in the network.

3.1.2. SET-UP MODELS

Described below are existing models and simulators of networks and DCs, which can be used to configure the setup of the Fog computing meta-model.

NETWORK

Several well-established event-driven frameworks model computer networks, mobile networks, applications, packet-level network traffic, infrastructure, and independent mobile users. The two primary examples are NS-3 and OMNeT++. These two are commonly deployed in academic network research and provide detailed results on network utilisation, throughput, congestion, and latency.

Both NS-3 and OMNeT++ are comprehensive packet-level network simulation frameworks that include wired and wireless standards and can simulate communication channel conditions. Furthermore, both frameworks have detailed models for channel definition, such as propagation delay, interference, data rate, and medium access schemes. Also, to a varying degree, NS-3 and OMNeT++, by default or through extension, support control plane signalling for many wireless standards and complex network topologies.

Both frameworks have support for modelling different types of network nodes, ranging from computers to routers and switches. Each edge and node pair has a defined communication and medium access standards, such as TCP/IP and Ethernet. Each packet that is sent over the network is treated in accordance with the current network and transport protocols and routing standard. In both, the event of arrival and departure of packets drives the simulation clock.

Furthermore, they require detailed configuration of all communication modes as well as node behaviour, making it very time-consuming to implement and verify systems with different levels of abstractions, and are thus cumbersome to model systems that cannot yet be described in such detail.

A Fog computing infrastructure topology is yet to be defined with unspecified control planes, it would thus be counter-intuitive and time-consuming to implement Fog computing infrastructure topologies in either NS-3 or OMNeT++. In some instances, some modules would have to be redesigned entirely, and others would have to be specified to much greater detail than the Fog can offer at this stage.

DATA CENTRE

The purpose of this section is to survey the DC models that are the most suitable for inclusion in the Fog computing meta-model. An extensive list of mathematical models, simulation approaches, and test beds can be found in [SL13], while [AS14] provides a survey of twelve cloud simulators. After careful examination, a handful are further reviewed below.

DC models and simulators are compared based on descriptions provided by the authors of the simulators. For each model the following is described: *Resource Provisioning* – what resources are included and how they are modelled; *QoS* – what performance indicators are measured; *Costs* of computation in the DC; *Performance* of simulator – an estimation of the time needed to perform a simulation.

CloudSim is an event-based simulator implemented in Java, for simulation of cloud computing system and application provisioning environments.

Resource Provisioning. The CloudSim simulation layer offers dedicated management interfaces for Central Processing Unit (CPU), memory, storage and bandwidth allocation, as well as, defining policies in allocating hosts to VM – VM provisioning. Hosts are described by their processing capabilities (in MIPS) and a core provisioning policy, together with an amount of available memory and storage. A model supports time-sharing and space-sharing core provisioning policies on both host and VM levels.

Latency (QoS). The latency model is based on conceptual networking abstraction, where the communication delays between each pair of entity type (e.g. host, storage, end-user) are described in a latency matrix as a constant value expressed in simulation time units (e.g. milliseconds).

Costs. CloudSim provides a two-layered cost model, where the first layer relates to IaaS, with costs per unit of resources, while the second one relates to SaaS, with costs per task units (application requests). This model allows calculation of the costs of using the cloud from the end-user perspective or the revenue from the IP perspective.

Performance. CloudSim can perform large-scale simulations, e.g., it can instantiate an experiment with 1 million hosts in 12 seconds. Moreover, memory usage grows linearly with the host number and even with 1 million hosts it does not exceed 320 MB.

CloudAnalyst [WCB10] is a simulator of geographically distributed large-scale cloud applications, developed with Java and that utilises CloudSim and SimJava.

Resource Provisioning. Cloud Analyst uses the same resource provisioning model as CloudSim.

Latency (QoS). A latency model allows configuration of network delays, available bandwidth between regions, and current traffic levels. CloudAnalyst facilitates experiments with latency by producing the following statistical metrics: the average, minimum, and maximum response times of all user requests; and response time grouped by time of the day, location, and DC.

Costs. CloudAnalyst supports the calculation of costs for using cloud resources, such as cost per VM per hour and cost per Gigabit of data transfer.

Performance. To improve the performance of simulation entities are grouped at three levels: clusters of users, a cluster of requests generated by users, and clusters of requests processed by VM.

GreenCloud is a packet level simulator based on NS-2, for simulation of energy-aware clouds.

Resource Provisioning. Servers are modelled as a single core node with a defined processing power limit (in MIPS or FLOPS), size of memory and storage, and implementing different task scheduling mechanisms.

Latency (QoS). Full support for the TCP/IP protocol reference model is provided and thanks to that the simulator can calculate communication latency with high accuracy.

Costs. GreenCloud allows detailed modelling of energy consumption by implementing energy models for every DC element.

Performance. Given that GreenCloud has to simulate the full stack of Internet protocols, each simulation only takes in the order of tens of minutes for a DC with a few thousands of nodes.

3.1.3. COSTS MODELS

The DC models mentioned above focuses mostly on the costs of running applications in DCs from the end-user perspective. To be able to investigate Fog computing resource management challenges models for Capital Expenditure (CAPEX) and OPEX are needed.

CAPEX includes costs of peripheral infrastructure and the servers.

Infrastructure Costs. Costs of building, power distribution, (and cooling can be estimated using a following equation: $\$200M \cdot (1+c_m)/a_i$, where c_m is the cost of money[1], and a_i is the time of infrastructure amortisation [in years] [GHMP08].

Server Costs. Costs of servers can be modelled as $n_s \cdot p_s \cdot (1+c_m)/a_s$, where n_s is the number of servers, p_s is the price of one server [in \$], c_m is the cost of money, and a_s is the time of server amortisation [in years] [GHMP08].

OPEX consists of utilities and personnel costs.

Power Costs. Here, power is assumed to be the primary utility expense common to all DC-types. To estimate costs of power, the following equation can be used, $n_s \cdot pc_s/1000 \cdot PUE \cdot p_{KWH} \cdot 24 \cdot 365$, where n_s is the number of servers, pc_s is the power consumption of one server [in W], PUE is Power Usage Efficiency, and p_{KWH} is the price of electricity [in \$ per KWH] [GHMP08].

Personnel Costs. Costs of personnel can be calculated using $M_1 \cdot C_1 + M_2 \cdot C_2 + M_3 \cdot C_3$, where M_1 is the number of IT personnel per rack, M_2 is the number of facility personnel per rack, M_3 is the number of administrative personnel per rack, and C_1, C_2, C_3 are the average costs per person for each of the above mentioned categories [PS05].

3.2. FOG COMPUTING META-MODEL

In this section, the models surveyed above are composed into a Fog computing meta-model. Figure 3.1 visualises of the proposed meta-model. UEs, such as cell phones or laptops, are carried by end-users, who are in motion. The UEs generate requests which are sent over the network to a DC. It is also possible that requests are generated by sensors that may be static (e.g. traffic cameras) or mobile (e.g. trains). The requests are processed in the DC and the response is sent back to the UE or sensor. Processing requests, in case of state-full applications, generate a user state, that has to be migrated with the end-user if he moves to another DC.

The primary objective of the meta-model is to capture the interactions between application workload, UE mobility, network topology, DC characteristics, and their influence on QoS and costs of a Fog computing infrastructure. The parameters that define the meta-model are presented in Table 3.2 and described in detail below.

[1] Cost of money is the rate of interest or dividend payment on borrowed capital.

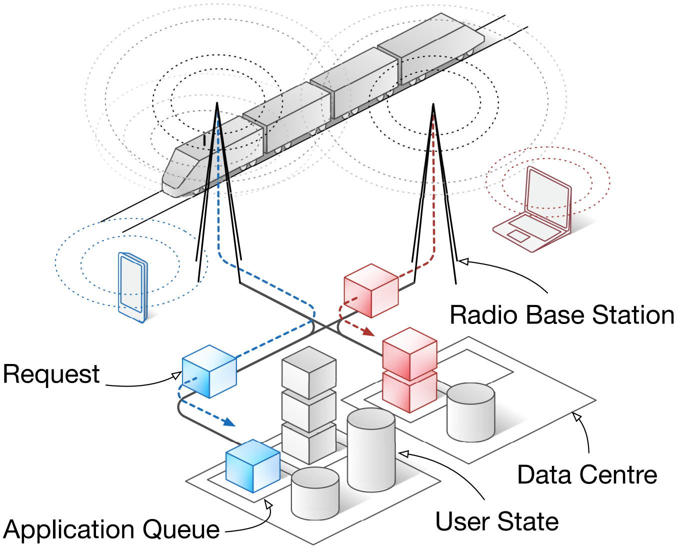


Figure 3.1.: Visualisation of the proposed Fog computing meta-model.

3.2.1. WORKLOAD MODEL

The first group of parameters in Table 3.2 describes the mobility of end-users carrying UE and the characteristics of requests generated by these UEs.

REQUEST GENERATION

N_{ser} applications may concurrently be deployed in the Fog computing. An application is modelled as a state-full web application. Each session is separated in time with a Poisson process λ_{ses} [RLGPC^{+}99]. Each session produces N_{req} requests, sampled from an inverse Gaussian distribution, where each request is separated in time by Log-Normal distributed delay D_{req} in seconds. The size of each request is given by S_{req} KB and is drawn from a Pareto distribution.

RESOURCE REQUIREMENTS

Application resource requirements are modelled using a linear model specifying the needed amount of resources, both for an idle application and per processing each request. An idle application uses CPU_{idle} CPU operations, mem_{idle} amount of memory, and $disk_{idle}$ amount of storage. Additionally for each processed request, the application uses CPU_{req} CPU operations, mem_{req} amount of memory, and $disk_{req}$ amount of storage. The amount of user's state data created by each request is defined by $state$ and expressed in absolute value or percentage of request size S_{req}.

Table 3.2.: Fundamental meta-model parameters.

Type	Parameters	Unit	Description
WORKLOAD			
Request generation	N_{ser}		Total number of applications
	λ^i_{ses}, where $i = 1,2,\ldots,N_{ser}$	s	Session arrival rate to DC
	N^i_{req}, where $i = 1,2,\ldots,N_{ser}$		Number of requests per user session
	S^i_{req}, where $i = 1,2,\ldots,N_{ser}$	KB	Size of requests for a given application
	D^i_{req}, where $i = 1,2,\ldots,N_{ser}$	s	Inter-request time
Resource Requirements	CPU^i_{idle}, CPU^i_{req}, where $i = 1,2,\ldots,N_{ser}$	MI	CPU cycles used by application
	mem^i_{idle}, mem^i_{req}, where $i = 1,2,\ldots,N_{ser}$	MB	Size of memory used by application
	$disk^i_{idle}$, $disk^i_{req}$, where $i = 1,2,\ldots,N_{ser}$	MB	Size of storage used by application
	$state^i$, where $i = 1,2,\ldots,N_{ser}$	MB	Size of user's state produced per req.
Mobility	N_{UE}		Number of UEs
	s^i_t, a^i_t, θ^i_t, ω^i_t, where $i = 1,2,\ldots,N_{UE}$		Movements of UEs
SETUP			
Network	N_{RBS}		Number of RBSs
	d_{RBS}	m	Dimensions of an RBS cell
	D_{net}	s	Cumulative network delay
Data Centre	N_{DC}		Number of DCs
	N^i_S, where $i = 1,2,\ldots,N_{DC}$		Number of servers in DC
	N^j_{CPU}, where $j = 1,2,\ldots,N^i_S$		Number of CPUs per server
	s^j_{CPU}, where $j = 1,2,\ldots,N^i_S$	MIPS	CPU's speed
	$memory^j$, where $j = 1,2,\ldots,N^i_S$	MB	Amount of memory per server
	$storage^j$, where $j = 1,2,\ldots,N^i_S$	GB	Amount of storage per server
	$network^i_{bw}$, where $i = 1,2,\ldots,N_{DC}$	Mb/s	Network bandwidth
	t_{init}, t_{idle}, t_{term}	s	Times of VM transitions
Service Placement	$placement = \{every, n\text{-}closests\}$		Service placement policy
OBJECTIVES			
Quality of Service	RT^i, where $i = 1,2,\ldots,N_{ser}$	s	Application response time
	TP^i, where $i = 1,2,\ldots,N_{ser}$	req/s	Application throughput
Costs	$Cost$	$	Total costs of infrastructure

MOBILITY

N_{UE} UEs populate the network, each subscribing to a subset of the N_{ser} available applications. The 2-dimensional, multimodal, mobility model detailed in [Bet01] provides us with an on-average uniform distribution of users, with movement proportional to the duration of a session and the scale of the mobile network. The model mentioned above defines the properties of a UE's movement. A UE's momentary movement is defined by its velocity constituted by the current speed s and current direction θ. Changes in mobility are defined by multiple stochastic processes that describe the duration of its state. An entity's speed s is independent of direction θ and is maintained for T_s seconds, after which acceleration a is applied between a_{min} and a_{max} for time T_a, until it reaches s_{min} or s_{max}. Furthermore, direction θ is maintained for time T_θ until the next change-event where the direction θ is altered for T_ω seconds with at the rate of ω radians per second. T_s, T_a, T_θ, and T_ω, describing the timing of each change-event, are set for each mobility mode and are each defined by a probability distribution bounded by maxima and minima.

3.2.2. MODEL PARAMETERS

The second group of parameters in Table 3.2 characterise the network and DCs.

NETWORK

In the proposed model, the core network introduces a cumulative propagation, switching, and routing delay. This delay is modelled with a Weibull distribution D_{net} in multiples of the number of network nodes between the source and the destination [PMF^{+}03].

The network distance between RBSs is equal to the cell dimension d_{RBS}. The associated RBSs are equidistant to their common DC and are for the sake of simplicity assumed to be separated by one network edge.

Furthermore, forthcoming cell planning practices aim to increase area energy efficiency by favouring smaller cells in urban areas [SKA13, FRF09]. The model employs a small homogeneous mobile network composed of N_{RBS} equidistantly distributed RBSs.

Agnostic to a specific mobile generational standard, a UE is handed over between RBSs at the geographic point where they cross the cell boundary distinguishing two independent RBSs defined by the width of the rectangular cells d_{RBS}.

DATA CENTRE

The DC model captures the influence that its capacity has on performance and costs of computation. To model DC performance, the quantity and quality of each DC resource is described. A DC consists of N_S servers. N_S is a function of the DC's scale. A server contains N_{CPU} CPUs capable of executing s_{CPU} operations per second. Values of *memory* and *storage* specify the total amount of available memory and storage, respectively. The network bandwidth is $network_{bw}$. The DC model also includes a provisioning model, that describes how available resources are shared among several applications, e.g., time-sharing or space-sharing.

In this work, for the sake of simplicity, a DC hosts applications in VMs or containers. Again, for the sake of simplicity, we refer to these discrete compute units simply as VMs. An application can be distributed over multiple VMs. The incoming workload is load-balanced by either a method of round-robin, random selection, or placed in the VM with the lowest load. However, a user's requests are always forwarded to the VM that served its first request. An application can specify a minimum and a maximum number of VMs it requires. The DC scales the application within these bounds based on the load-balancing outcome.

To emulate the life-cycle of a VM six VM states are defined. The states and their corresponding transitions are illustrated in Table 3.3. The transitions between the states are presented in Figure 3.2. In the beginning, all VMs are in the INACTIVE state. A VM is initiated when the first request arrives at a DC. It takes t_{init} seconds before

Table 3.3.: States of Virtual Machine.

Name	Description
INACTIVE	VM is turned off.
INITIATING	VM is booting up.
PROCESSING	VM is serving requests.
IDLE	VM is waiting for requests.
MIGRATING	VM is transmitting data.
TERMINATING	VM is shutting down.

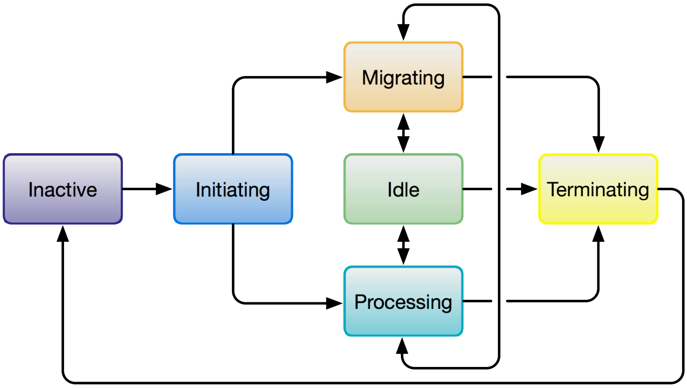


Figure 3.2.: Transitions between Virtual Machine states.

a VM is ready to start processing requests or receiving migrated requests and user state from other DCs. It is assumed that a VM is not able to process requests and handle migrations at the same time, so it changes state between PROCESSING and MIGRATION over the time. Moreover, migrations are given a higher priority than processing, so processing is paused if there are any migrations to perform. When there are no requests to process and no migrations to handle a VM goes into the IDLE state. A VM is terminated if IDLE state lasts for longer than t_{idle} seconds, and the VM termination takes t_{term} seconds.

SERVICE PLACEMENT

Service placement policies define in what DC(s) an application should be hosted, what number of replicas should be running, and when an application should be migrated between DCs. These decisions depend on the mobility of users, the size of the user's state that has to be migrated, and Service Level Agreements (SLAs). For example, an application can be hosted in n Proximal DCs closest to the majority of its users (n-*closests*), or in the case of latency-sensitive applications in every Proximal DC that is needed to provide acceptable QoS (*every*).

3.2.3. OBJECTIVES MODEL

The third group of parameters in Table 3.2 describe QoS and costs of a Fog computing infrastructure.

QUALITY OF SERVICE

Combining the resource requirements model, which describes the number of resources an application needs, with a DC model, allows simulating how collocation of different applications in a DC influences their response times RT^i and throughputs TP^i.

COSTS

Cost models available in the literature and described in Section 3.1.3 are arguably "country dependent", because of the inclusion of variable parameters such as salaries, costs of energy or costs of property. They are also not taking into account parameters important from the perspective of Fog computing, such as the size of DC. Therefore, the costs of a Fog computing infrastructure are modelled using a basic heuristic based on the observation that dispersion of infrastructure causes additional costs, e.g.: increase of administrator travel time between locations, and higher unit costs of computation in proximal DCs because of smaller scale and high initial costs.

$$\text{Cost} \propto \frac{N_{\text{DC}}}{\sum^{\text{DC}} N_S} \tag{3.1}$$

As shown in Equation 3.1, the total cost of a Fog computing infrastructure is directly proportional to the number of DCs and inversely proportional to the total number of servers in all DCs. This implies that; distributing the same number of servers among many DCs is more expensive than placing them in one DC.

3.2.4. LIMITATIONS

The proposed meta-model has several limitations. The application model assumes that all requests generated by one application are homogeneous and each of them consumes the same amount of resources. The mobile access network model does not take into account the physical layer, channel provisioning, and cell load balancing. Additionally, the radio access network functions as a mechanism to associate UEs with DCs propagation and system processing delays are thus not modelled. However, the model is granular enough to be efficient and includes sufficient dynamics to run a meaningful experiment to study a Fog computing infrastructure's performance. Additionally, at this point, not enough is known about Fog computing infrastructures to add more details to the model.

3.3. SIMULATION SHOWCASE

A coarse-grained simulator was implemented using SimJava [HM98] as the underlying event-driven simulation framework. All modules are implemented from scratch and are based on the meta-model presented in Section 3.2. The simulator fully implements the proposed request generation and network models but implements more abstract mobility, resource requirements, DC, and service placement, models.

To demonstrate the scope of the Fog computing meta-model and the simulator elementary showcase scenario is introduced below. The scenario is designed for studying the fundamental relationship between workload – UE mobility, set-up – Proximal DC catchment, and objectives – the aggregate utilisation of a Fog computing infrastructure.

3.3.1. EXPERIMENTS

A simple scenario with one application is presented below. The size of the simulation is reduced from a full-scale Fog computing infrastructure. The VM scalability and placement models shall be seen as proofs-of-concept. The goal is to obtain clear conclusions about the relation between UE mobility, and DC catchment and avoid the interference of other elements. The scenario is described in detail below.

The telecommunication infrastructure is composed of 16 RBSs, in a 4x4 layout, as presented in Figure 3.3. The cells, depicted with dashed lines, are tangent but not overlapping and are dimensioned as a typical LTE micro-cell at 750 m, as detailed in [SKA13]. The number of DCs varies between the experiments and thus so, also the DC catchment, represented with the solid lines, and defined as the ratio between DCs and RBSs, changes between (1:1) and (1:16). Note that a (1:16) catchment covers the whole simulation area with one DC. In abstract terms, the (1:1) catchment represents a set-up with one Proximal DCs per RBS. In contrast, the (1:16) catchment approaches a more traditional case of Remote DC serving all users in the domain.

To study the effects of DC catchment, all DCs are of the same capacity. The number of VMs in each DC is scaled proportionally to the number of users they serve. The DC in the (1:16) catchment scenario has 16 VMs, while the DC in the (1:1) scenario has just one VM. The workload is balanced among available VMs; new sessions are forwarded to the least loaded VM. To observe the full extent of the effect of user mobility, user states, and requests are strictly migrated to the geographically nearest DC.

The request generation model has a session arrival rate of λ_{ses} described by a Log-Normal distribution with the parameters $\mu = 3$ and $\sigma = 1.1$. The number of requests per session N_{req} is taken from an Inverse Gaussian distribution with the parameters $\lambda = 5$ and $\mu = 3$. Inter-request time is D_{req} seconds and is modelled with with an Exponential distribution with $\lambda = 0.1$. The simulation domain is populated by 480 UEs, all subscribing to the same service. Due to the size and simplicity of the network topology in the proposed scenario, a Markov-based mobility model is deployed. The mobility mode is based on a car and is as specified in Section 3.2.1, with parameters from

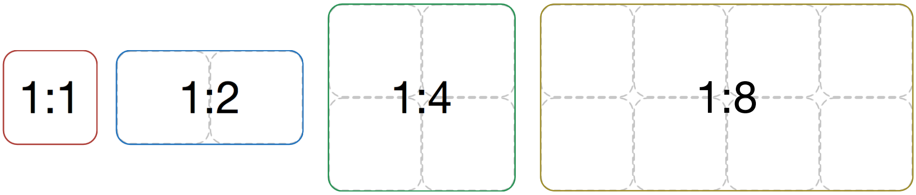


Figure 3.3.: RBSs ranges and DC catchments.

[Bet01]. To allow the mobility and workload models to reach a steady state jointly, the simulation is run for 8 simulated hours. This results in an average processing load of 30%; this level should give enough margin to for example migrations to complete successfully.

The user state is proportional to the aggregate size of that user's sessions with the application it subscribes to and is defined by a 5th order AR-process with linearly decaying parameters. The initialisation of a VM takes $t_{\text{init}} = 81s$, similarly as for m1.small VM type in Amazon EC2 [OIY+10]. A VM is terminated if it remains in the IDLE state longer than t_{idle} which is equal to the mean inter-session time. It takes $t_{\text{term}} = 21s$ to terminate a VM.

To investigate the influence of Proximal DC, catchment on the aggregate performance of a Fog computing infrastructure the simulator observes the life cycle of the VMs that run within the DCs by recording the amount of time each VM spends in each state. Two sets of experiments are conducted. In the first set, end-users are static. The second set of tests introduces mobility. In both sets, the variations in the distribution of time that VMs spend in each state is investigated.

3.3.2. RESULTS

Figure 3.4 shows the breakdown of the mean time spent in each VM state in the system per DC catchment. With a (1:1) DC catchment the utilisation suffers from the proportion of time spent in IDLE state due to the relatively low request arrival rate generated by one-sixteenth of all users. The inefficiency is caused by the time the system spends in the IDLE, INITIATING, and TERMINATING states. The composition of time spent in these states changes with DC catchment, and is a reflection of the number of VMs in a DC and load-balancing effort. Reducing the time spent on starting and terminating VMs would free up more resources and perhaps also make the system more reactive to sudden workload changes. The intelligent management of VM scalability and placement is something that needs to be optimised.

Figure 3.4 reveals the overhead of user mobility and the migration effort it incurs. Depending on the DC catchment, different migration dynamics come into play. As migrations are more frequent in the (1:1) case than in the (1:8) case, user states do not have the time to grow as much between migrations in the former case. The migration effort is therefore not a factor eight lower in the (1:8) case versus the (1:1) case, but

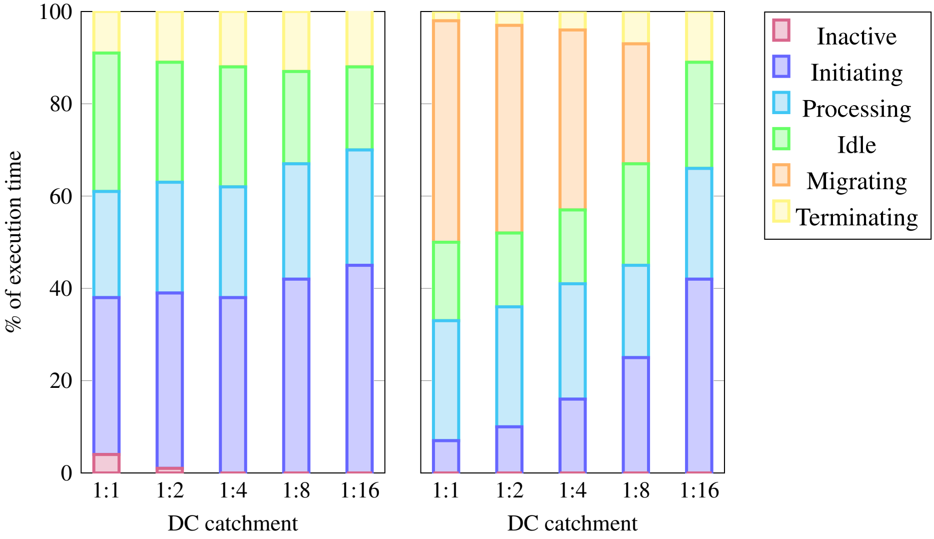


Figure 3.4.: DC catchment vs. time spent in each VM state.

rather, they spend 26% and 47% of their time in the MIGRATING state, respectively. The system dynamics revealed by Figure 3.4, where at worst, 47% of the execution time is spent migrating users, points to the need to find scaling mechanisms for Fog computing that take into account mobility and inactivity, so that resources can be freed dynamically for other revenue generating applications. A policy of strictly migrating user states and requests to the geographically closest DC, regardless of DC catchment, to obtain minimal propagation and communication latency, is suboptimal.

4

Centralised Fog computing resource management

Meeting the objectives of a set of heterogeneous applications and a heterogeneous Fog computing infrastructure, is non-trivial. The highly distributed and heterogeneous nature of the Fog introduces several interesting resource management challenges arising from a highly dynamic workload, heterogeneous energy costs and resources, rapid user mobility, and multi-component applications, [KET+13]. The topology depicted in Figure 1.12 reflects the union of a Mobile Network Operators (MNO)'s network and a federated cloud infrastructure and should be seen as an abstraction of a Fog computing infrastructure topology proposed in [BDPW11].

The Fog computing paradigm will proposedly enable and drive new types of services and applications that exploit the increased proximity to the end-users and critical infrastructure components. Contemporary cloud resources are housed in centralised DCs that are separated from the end-users by the intermediate WANs, core, and access networks. The added latency and weak-backhaul introduced by those networks has proven to inhibit the performance of cloud-based applications [BS10]. Furthermore, there is a large and growing set of mission critical real-time applications such as telerobotic surgery [Bal02], RBS baseband signalling [CCY+15], gaming [HTÖ+16], and Augmented Reality (AR) [CKW13] that are unable to operate in such a latency-, jitter-, and throughout-uncertain environment, provided by a centralised cloud paradigm. The decreased distance between the cloud infrastructure and the end-users, provided by a Fog computing infrastructure, reduces the RTT and jitter, increases availability, and fault-tolerance [SBL15] for the infrastructure's resident cloud applications.

To operate a viable Fog computing infrastructure, its operator needs to administer the admitted applications and the system's resources such that resources are not over-provisioned, the total operational cost is minimised, and that all applications' performance requirements are met. When managing a Fog computing infrastructure, its operator's primary degree of freedom is the placement of the system's resident applications. Continuously and scalably evaluating the placement of a vast set of heterogeneous applications over a set of heterogeneous nodes is non-trivial and is the fundamental problem addressed in this paper.

In this chapter, the feasibility of a Fog computing infrastructure is evaluated by studying application placement algorithms in such an infrastructure. Because the workload is highly mobile, this work focuses on where to run applications in the network and how to continuously evaluate that decision as an instrument to fulfil the holistic management objectives of a Fog computing infrastructure. To this effect, an objective function is proposed that minimises the global system cost to manage the DCs and the network resources of a Fog computing infrastructure when hosted in a tree-structured network topology. Furthermore, experiments designed to study the validity of the placement algorithms in the Fog paradigm are provided.

4.1. RESOURCE MANAGEMENT CHALLENGES

One of the foremost challenges in the Fog computing paradigm is how to manage the highly heterogeneous and distributed resources in a complex system. The sheer size of the infrastructure and the number of management parameters renders a fully centralised resource allocation strategy infeasible [AS07]. As a result, a decentralised collaborative resource management approach needs to be considered. Before we can begin to design a distributed management approach we explore the theoretical optimal solution. A centralised system can provide an optimal management solution but might be practically infeasible due to, for example, computational complexity, scalability, and fault tolerance. With the hypothetical and experimental scope of this work, it is free from such constraints.

4.1.1. SERVICE PARADIGM

The nodes in a Fog computing infrastructure can be viewed as nodes in a federated cloud [RBL$^+$09] whose resources are brokered globally but are for example sought after for their locality to a specific group of users, sensors, or actuators. Application components are submitted by application owners from beyond and within the network to serve a subset of the network's population. Application owners impose performance, availability, latency, and locality requirements on the Fog platform in the form of a SLO or a Service Level Agreement (SLA). Here, Fog platform refers to the SW platform that runs on the Fog computing infrastructure. Additionally, both end-users and application owners alike are agnostic to where the application is hosted and how the network and cloud infrastructure is managed. The end-users cannot impose requirements on the network or the applications' performance. Performance requirements, SLOs, for applications originating from an end-user device, are defined by the application owners. Additionally, there is no resource competition between applications and a Fog computing infrastructure honours no priorities, but rather, applications have global performance objectives where a Fog computing infrastructure might augment performance and facilitate scalability. The decision to deploy an application to the Fog is therefore assumed to be made by the application owner.

A Fog operator's overall management objective is to ensure that the fundamental circumstances for an application to perform according to its SLO or SLA are met. As such, a Fog operator practices admission control and can reject new applications if the application will compromise its internal management objectives and the integrity of the other applications' SLO or SLA. Once an application component has been placed, the performance of an application is the result of the applications' properties and the internal resource management policies of the DC the application is running on, and is beyond the scope of this work. Note that this work is thus not concerned with VM or application to Physical Machine (PM) mapping. Application components hosted in the Fog are assumed not to have a scheduled deadline but are instead being terminated based on the application's internal management objectives.

4.1.2. RESOURCE MANAGEMENT OBJECTIVES

The management objectives for a Fog infrastructure operator are similar to those found when operating a wireless network, such as user mobility and limited network capacity. Nevertheless, there is a clear paradigm chasm between Telecom and cloud services. Telecom provided services such as voice have very well defined and strict SLAs. Depending on the service type, cloud service SLOs on the other hand are more loosely defined, where the service offering is multidimensional and given the nature of the resource offering, performance responsibility is more opaque [Bas12]. Additionally, operable core and access networks are prerequisites to hosting and accommodating cloud resources and traditional Telecom services in the network. As such, the successful operation of the access and core networks are therefore prioritised over the cloud services. The scope of this work does therefore not cover MNO services and network infrastructure virtualisation, as their objectives overlap with that of a Fog computing infrastructure.

It is assumed that a Fog computing infrastructure is managed on top of the existing Telecom infrastructure. The Fog computing infrastructure management process is agnostic the momentary load and objectives of the Telecom network. The objective of the Fog computing's management entity is, therefore, to minimise the resource usage and thus the resulting operational cost and incurred load on the shared Telecom network, and to provide a service with a finite set of resources.

4.1.3. CHALLENGES

The internal Fog computing management challenges are found in the union of cloud and mobile infrastructure. a Fog infrastructure operator has only a few degrees freedom to control the operations of the infrastructure. An operator can alter:

- The number of applications in the network.

- The pallet of applications and

application's heterogeneity.

- Which pieces of infrastructure to run.

- Where to run the application components.

Continuous evaluation of application component placement is the common denominator. No assumptions are made about how specific applications or set of users behave. It is therefore assumed that any application can behave in any manner in the realm of what is physical and computationally possible. When re-evaluating an application placement, the management process determines if the energy, computing, network, and latency costs fall short of any of the possible placement possibilities that qualify, for a certain period. The systems' rate of change determines the duration under which a decision is valid. The number of possible placement combinations and the rapid rate of change means discrete placement decisions need to rely on the prevailing workload, resource availability, and user location. The system needs to re-evaluate the placement of application components, whenever workload changes for an application, when new applications arrive or are terminated, when applications scale up or down, or when foreground traffic volumes change. The triggers and decisions need to be at the granularity of individual application components.

Operating a profitable network relies on an operational network and the ability to cost-manage that network. The resource heterogeneity of a Fog computing infrastructure, the mobility of the users in the network introduce and the heterogeneity of the applications to which they subscribe introduce a complex set of management decisions. A Fog infrastructure operator will need to manage the placement decision and the subsequent placement revaluation of application components in a manner that minimises the overall resource usage.

4.2. EXTENDED FOG MODEL

Here, although analogous, the model presented in Chapter 3 is augmented and reworked to fit the extended problem addressed in this chapter. Also, the notation has been modified and made more compact to better fit the formulations below.

The placement and scale of a Fog computing infrastructure's DCs is dependent on the degree virtualisation of a MNO's infrastructure [ZZGTG10], the degree of convergence of core and access networks, and the prevailing geographic demand for proximal compute capacity. Although some bounds can be identified, these properties are not yet defined as the design of forthcoming mobile access network standards and topologies are far from being finalised [SS12]. Additionally, no assumptions are made of DC placement or scale but the models are generic enough to handle many possible next-generation infrastructure topologies.

As a whole, a Fog computing infrastructure is modelled as an undirected forest or tree graph [RMK$^+$09, BDPW11, JPE$^+$11, MPZ10], where the vertices are DCs and the

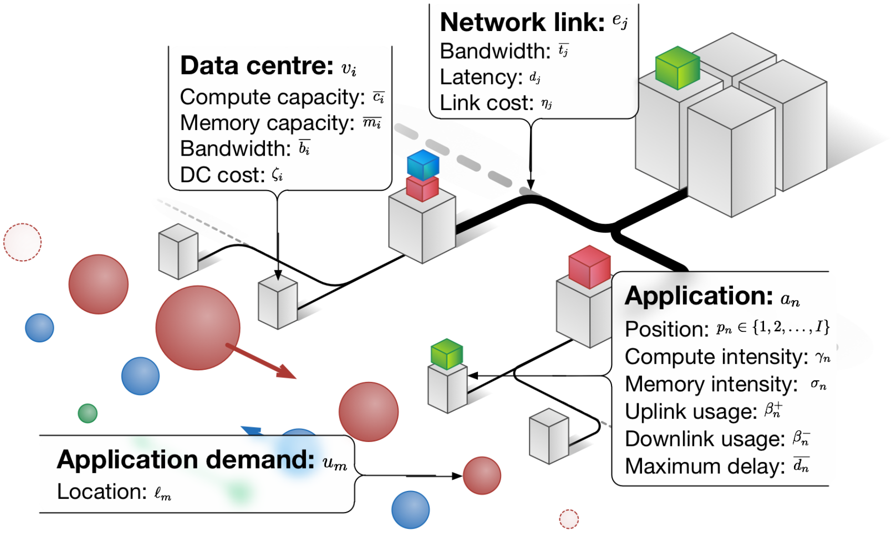

Data centre: v_i
Compute capacity: $\overline{c_i}$
Memory capacity: $\overline{m_i}$
Bandwidth: $\overline{b_i}$
DC cost: ζ_i
Network link: e_j
Bandwidth: $\overline{t_j}$
Latency: d_j
Link cost: η_j
Application: a_n
Position: $p_n \in \{1, 2, \ldots, I\}$
Compute intensity: γ_n
Memory intensity: σ_n
Uplink usage: β_n^+
Downlink usage: β_n^-
Maximum delay: $\overline{d_n}$
Application demand: u_m
Location: ℓ_m

Figure 4.1.: Model overview

edges are network links, each with a set of finite resources, see Figure 4.1. Applications are hosted in a DC and are subject to demand through the network links, originating at the leaf nodes. The graph, $G = (V, E)$ denotes a tree depicting a Fog computing network topology, where

$$\begin{aligned} V &= \{v_i \mid i = 1, 2, ..., I\}, \\ E &= \{e_j \mid j = 1, 2, ..., J\}, \end{aligned} \tag{4.1}$$

where v_1 is the root node. The subscript i of the node v_i is named such that all the nodes v_k between v_i and v_1 follows

$$\mathrm{dist}(v_k, v_1) \leq \mathrm{dist}(v_i, v_1), \quad \forall k \leq i, \tag{4.2}$$

where the distance between nodes $v \in V$ and $w \in V$, $\mathrm{dist}(v, w)$ is measured in number of vertices that is on the shortest path from v to w. RBSs connect the leaf vertices from which the end-users access the network. Thus, the leaf vertices are geographic aggregation points of application demand.

4.2.1. DATA CENTRE MODEL

The Fog compute resources will proposedly reside in existing MNO infrastructure [BDPW11], such as in a MNO's regional offices or what remains of previous generation network infrastructure. Furthermore, the compute capacity is proportional to the aggregate demand of applications from their users that have access to it, thus successively decreasing with depth. Henceforth, the computing cost will increase with depth.

Each vertex is a DC and hosts applications using a set of finite resources. Vertex v_i, $i \in \{1,2,\ldots,I\}$ in the graph has the following features:

- **Compute capacity** $\overline{c_i}$, a number describing the total compute capacity of the DC.

- **Memory capacity** $\overline{m_i}$, a number describing the amount of memory on the DC.

- **Bandwidth**, $\overline{b_i}$, a number describing the maximum throughput that the DC can handle.

In addition to the features above, vertex v_i, $i \in \{1,2,\ldots,I\}$ is associated with the following operational cost

- **DC cost** ζ_i, a function of resource usage (compute, memory, and bandwidth) that returns the DC's running cost per time unit.

In general, the leaf vertices of the graph correspond to smaller DCs and thus the compute costs are arguably more significant at the leaf vertices than at vertices at lower depths in the tree, [BCH13].

4.2.2. NETWORK MODEL

Existing 3rd and 4th generation mobile access networks are generally tree-structured [EEsS14]. Future mobile infrastructure generations will feasibly inherit this structure. Furthermore, bandwidth availability as well as communication latency and jitter decrease with tree depth. Additionally, the network topology is modelled as a tree-structured graph, where each edge has network resources and exhibits latency and congestion.

Each edge e_j, $j \in \{1,2,...,J\}$, in the graph has the following features

- **Bandwidth** $\overline{t_j}$, a number specifying the maximum throughput over the edge.

- **Latency** d_j, a function of the throughput that returns the delay caused by that link's resource utilisation and length.

In addition, each edge has the following operational cost

- **Link cost** η_j, a function of throughput that returns the link's running cost per time unit.

The communication latency of an application is dictated and maintained by the applications' relative locations to its demand and the level of congestion on the links it employs. As the demand mobility from one edge node to another can be highly dynamic, the size and location of applications' demand can vary with time. Latency is modelled as a function of propagation delay and network congestion [FTD03]. In general, the bandwidth cost increases with the distance to the root vertex.

4.2.3. APPLICATION MODEL

A Fog computing infrastructure hosts applications a_n, where $n = 1, 2, \ldots, N$. Let $A = \{a_n \mid n = 1, \ldots, N\}$ denote the set of all applications hosted in the Fog. Application a_n, $n \in \{1, 2, \ldots, N\}$, see Figure 4.2, has the following features:

- **Position** $p_n \in \{1, 2, \ldots, I\}$, a number specifying that the application component is running on the DC at vertex v_{p_n}.
- **Compute intensity** γ_n, an increasing function of the demand of the application component that describes the number of computational resources required by the application component.
- **Memory intensity** σ_n, an increasing function of the demand of the application component that returns the amount of memory required by the application component.
- **Uplink usage** β_n^+, an increasing function of the demand of the application component as well as the locations of the application component's end-users that returns the uplink throughput associated with the application component,
- **Downlink usage** β_n^-, an increasing function of the demand of the application component as well as the location of the application component's end-users that returns the downlink throughput associated with the application,
- **Maximum delay** $\overline{d_n}$, the longest delay (latency) that provides the users of the application component a satisfactory experience. This delay is specified in the SLA between MNO and application owner.

Applications thus scale vertically in a DC, as a function of demand.

Note that in the experiment section of the current work, only single-tier applications are considered. However, the model can be generalised to account for multi-tier applications. In the multi-tier setting, application a_n can be considered as being composed of s_n stages or sub-applications. Each of these sub-applications will have the same features as a single tier application. The relationship between application components is expressed with a demand affinity as described in [UPS$^+$05]. As an alternative to the viewpoint of handling multi-tier application would be to include general affinity and anti-affinity constraints between application components.

4.2.4. USER MODEL

Finally, let $U = \{u_m \mid m = 1, 2, \ldots, M\}$ be the set of users of the Fog computing infrastructure. Each user u_m, $m \in \{1, 2, \ldots, M\}$ has the following features

- **Location** $\ell_m \in \{1, 2, \ldots, I\}$, a number specifying that the user is currently served by the DC at vertex v_{l_m}.

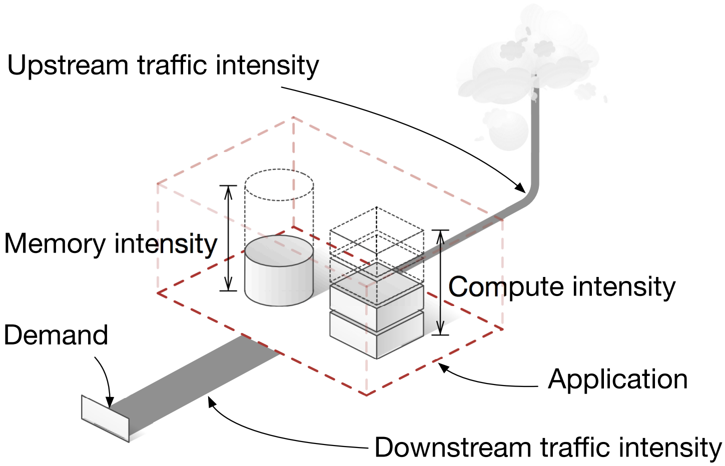


Figure 4.2.: Application model

- **Active applications** $A_m \subset A$, a set of application components that the user currently runs.

For future notation, $U_n = \{u_m \in U \mid n \in A_m$ is the demand for an application component n and let $U_{n,i} = \{u_m \in U_n \mid \ell_m = i\}$. Note that for the numerical experiments in this chapter, users are not explicitly tracked, only the demand of each application component at each leaf vertex.

4.3. OPTIMISATION FORMULATION

In this section the proposed Fog application placement algorithms are detailed. The infrastructure is restricted by resources with finite capacity and can thus not host an infinite number of applications. Furthermore, not all placement constellations can meet all application constraints. Accommodating an application's constraints and meeting its SLA/SLO is a prerequisite to generate revenue. Therefore, analogously, when searching for possible placement options applications' constraints are the primary concerns. Applying the application's resource and performance constraints, prunes the search tree. After that, the algorithm searches for a set of placement options that incurs the least overload in that point, over time.

4.3.1. RESOURCE UTILISATION METRICS AND CONSTRAINTS

The primary objective of the optimisation in this paper is to decrease the running costs of a Fog computing infrastructure by placing/moving the applications on different DCs by normalising the usage of the infrastructure's DC resources and minimising the incurred network usage. Here, the control or decision variable is the vector

$\mathbf{p} = (p_1, p_2, \ldots, p_N)$, that holds the position of all applications. For full generality, the set of admissible placements are

$$\mathcal{A} = \left\{ \mathbf{p} \in \mathbb{Z}^N \mid p_n \in \{1, \ldots, I\} \right\}. \tag{4.3}$$

To detail the relations resources and how they are utilised, the following objects are defined: Let $P_{i,i'}$ denote the path from DC i to DC i', that is, the set of edges that connects the two DCs. Moreover, let $E_i = \{e \in E \mid v_i \in e\}$ be the set of edges that represents links that are connected to DC i.

The throughput t_j over link j is the total usage (uplink plus downlink) for all pairs of users and applications such that the link is on the path connecting the DC serving the user and the DC hosting the application. More precisely, the throughput is given by

$$t_j = \sum_{\left\{ n,m \middle| \substack{u_m \in A_n, \\ e_j \in P_{p_n,\ell_m}} \right\}} \left(\beta_n^+ \left(|U_{n,\ell_m}| \right) + \beta_n^- \left(|U_{n,\ell_m}| \right) \right). \tag{4.4}$$

For each application $a_n \in A$ and user $u_m \in U_n$ of that application, the latency $d_{n,m}$ experienced by the user is the sum of all latencies on the path connecting the DC serving the user and the DC hosting the application. Thus, the latency $d_{n,m}$ can be computed as

$$d_{n,m} = \sum_{\{j \mid e_j \in P_{p_n,l_m}\}} d_j(t_j). \tag{4.5}$$

Any additional latency incurred by intermediate DCs is not included. The computing and memory usage at DC i is the sum of the corresponding usage by the applications that are running at the DC, that is,

$$c_i = \sum_{\{n \mid p_n = i\}} \gamma_n(|U_n|) \tag{4.6}$$

and

$$m_i = \sum_{\{n \mid p_n = i\}} \sigma_n(|U_n|). \tag{4.7}$$

The throughput over vertex i is the sum of the throughputs of all edges that are connected to the corresponding DC. Thus,

$$b_i = \sum_{\{j \mid e_j \in E_i\}} t_j. \tag{4.8}$$

Each of the resource metrics is associated with a constraint connected to the features of the DC, application, and network models. These constraints are

$$c_i / \overline{c_i} \leq 1, \qquad i = 1, 2, \ldots, I, \tag{4.9}$$

$$m_i / \overline{m_i} \leq 1, \qquad i = 1, 2, \ldots, I, \tag{4.10}$$

$$b_i / \overline{b_i} \leq 1, \qquad i = 1, 2, \ldots, I, \tag{4.11}$$

$$t_j / \overline{t_j} \leq 1, \qquad j = 1, 2, \ldots, J, \tag{4.12}$$

$$d_{n,m} / \overline{d_n} \leq 1, \qquad n = 1, 2, \ldots, N \text{ and } \{m \mid u_m \in U_n\}. \tag{4.13}$$

All constraints above are written on the form $a/\overline{a} \leq 1$, that is that the relative usage (or latency) of a particular resource should be at most 1. The choice of a common form simplifies the discussion about the formulation of the optimisation problem below.

4.3.2. OPTIMISATION PROBLEM

The objective function in Equation (4.14) is designed to capture the application execution cost and the overload penalty on the node as well as edge resources in the system. Here, the objective function is constructed from the infrastructure providers' viewpoint. Primarily, they want to minimise the overall running cost.

$$J(\mathbf{p}) = \sum_{i=1}^{I} \zeta_i(c_i, m_i, b_i) + \sum_{j=1}^{J} \eta_j(t_j). \tag{4.14}$$

In this work, it is assumed that the cost for running the DCs is linearly proportional to their compute resource usage and that the cost for the network is linear to the throughput over each link. Remark that adding a constant background cost that represents the cost for when the links and DCs are idle does not influence the possible savings by migrating the applications. Thus, in principle, the optimisation problem is formulated as

$$\min_{\mathbf{p} \in \mathcal{A}} J(\mathbf{p}), \tag{4.15}$$

subject to constraints (4.9–4.13).

The problem formulation above is straightforward and intuitive. A Fog computing infrastructure is subject to a highly volatile workload. Even small changes in the location and quality of demand can render any previously optimal solution obsolete. In the worst case, significant migrations would be required to resolve the infeasibility. To ensure feasibility of the constraints stating that the relative usage should be smaller than 1 and to avoid link or vertices becoming overloaded, therefore a penalty initialisation point $\tilde{x} < 1$ is introduced, and a penalty–barrier function is defined as;

$$f_{\tilde{x}}(x) = \begin{cases} 0, & \text{if } x < \tilde{x}, \\ \frac{1}{1-x} + \frac{2\tilde{x}-x-1}{(1-\tilde{x})^2}. & \text{if } x \geq \tilde{x}. \end{cases} \tag{4.16}$$

In the equation above, x should be viewed as the relative usage (or latency) that is the quotient on the left hand side in constraints (4.9–4.13). For the case when $x \geq \tilde{x}$, the first term is selected to ensure that $f_{\tilde{x}}(x) \to \infty$ as $x \to \infty$ and the second term is selected so that $f_{\tilde{x}}(\tilde{x}) = f_{\tilde{x}}(\tilde{x}) = 0$, that is, to guarantee that $f_{\tilde{x}}$ is continuously differentiable in the interval [0,1).

By construction, the function $f_{\tilde{x}}$ acts as both a penalty and a barrier function; it is a penalty function for constraints of the type $x \leq \tilde{x}$ and a barrier function for the constraint $x \leq 1$. In essence, this makes it easy to modify the point where the penalisation starts for each constraint separately. The algorithm utilises this versatility by having different penalty initialisation points for constraints corresponding to different features. A more elaborate set-up could, for example, have a penalty initialisation point for the computing resource usage that depends on the level in the tree that the corresponding DC is located. To compute the overall penalty G, the penalty–barrier function corresponds to all constraints with their respective penalty initialisation points. That is,

$$\begin{aligned} G(\mathbf{p}) = &\sum_{i=1}^{I} f_{\tilde{c}}\big(c_i/\overline{c_i}\big) + \sum_{i=1}^{I} f_{\tilde{m}}\big(m_i/\overline{m_i}\big) + \\ &+ \sum_{i=1}^{I} f_{\tilde{b}}\big(b_i/\overline{b_i}\big) + \sum_{j=1}^{J} f_{\tilde{t}}\big(t_j/\overline{t_j}\big) + \\ &+ \sum_{n=1}^{N} \sum_{\{m|m\in U_n\}} f_{\tilde{d}}\big(d_{n,m}/\overline{d_n}\big), \end{aligned} \tag{4.17}$$

where $\tilde{c}$, $\tilde{m}$, $\tilde{b}$, $\tilde{t}$, and $\tilde{d}$ are the penalty initialisation points corresponding to the constraints for compute usage, memory usage, DC throughput, link throughput, and application latency, respectively. Here, the individual penalty–barrier functions for all constraints are added. This corresponds to taking the 1-norm of the vector whose elements hold the values of the individual penalty–barrier functions for all constraints. An alternative overall penalty–barrier method is obtained by taking another vector norm of the constraint vector.

To conclude, rather than solving problem (4.15) subject to constraints (4.9–4.13), we solve the following problem

$$\min_{\mathbf{p}\in\mathcal{A}} J(\mathbf{p}) + \mu G(\mathbf{p}), \tag{4.18}$$

where μ is a positive penalty parameter and J and G are defined in expressions (4.14) and (4.17). Dimension for J is in terms of monetary cost per time unit, whereas G is dimensionless. Hence, the unit for μ is the monetary cost per time unit.

4.4. PROPOSED APPLICATION PLACEMENT METHOD

One approach to solving the optimisation problem (4.18) is to perform a so-called exhaustive search, that is to evaluate the objective function for each admissible placement. The computational complexity of this approach is exponential, since $|\mathcal{A}| = N^I$. For example, to evaluate the objective function in a setting with 22 applications and 12 DCs would require $O(10^{18})$ summation operations (each evaluation of the objective function requires $O(10^2)$ summation operations). Thus, even for relatively small problems, this approach is computationally intractable. Here, a local search algorithm is employed, described below, to find approximate solutions to optimisation problem (4.18).

The objective function is re-evaluated, $J + \mu G$, during runtime to compute the total cost for running all the applications inside the infrastructure. The re-evaluations are triggered either by an internal event in the infrastructure, entity, or application or is triggered by a periodic heartbeat signal, agnostic to the state of the system.

For each $\mathbf{p} \in \mathcal{A}$, a k-neighbourhood of $\mathbf{p}$ is defined as

$$\mathcal{N}_{\boldsymbol{p}}^{k} = \{\boldsymbol{q} \in \mathcal{A} \mid \|\boldsymbol{p} - \boldsymbol{q}\|_{\mathcal{A}} \leq k\}, \tag{4.19}$$

where $\|\cdot\|_{\mathcal{A}}$ is a measure of the network distance for elements on $\mathcal{A}$. A larger k gives more freedom to migrate, replicate, and consolidate application components; however, we need to select k such that $|\mathcal{N}_{\mathbf{p}}^{k}|$ is not too large. The network distance can be computed as,

$$||p-q||_{\mathcal{A}} = \sum_{n=1}^{N} \delta_{\mathcal{A}}(p_n, q_n), \tag{4.20}$$

where a possible choice of function $\delta_{\mathcal{A}}$ is:

$$\delta_{\mathcal{A}}(p_n, q_n) = \begin{cases} 1, \text{ if } p_n \neq q_n, \\ 0, \text{ else }. \end{cases} \tag{4.21}$$

An alternative choice is:

$$\delta_{\mathcal{A}}(p_n, q_n) = \begin{cases} |P_{p_n, q_n}|, \text{ if } p_n \neq q_n, \\ 0, \text{ else }, \end{cases} \tag{4.22}$$

where $|P_{p_n,q_n}|$ can be computed considering the latency in all the traversed edges.

We employ a depth-first search algorithm to find the neighbourhood nodes in the tree. We start with finding all the solutions with neighbourhood depth k and compute the local optimal solution. From the local solution, we construct its subsequent $k-$neighbourhood solutions and compute the local optimal solution. We repeat the process until we achieve the specified maximum number of iterations or the cost starts increasing. The algorithm is described as,

Algorithm 1 Local Optimisation or Local Search

```
 1: Input : current placement vector of applications ($\mathbf{p}$), maximum number of iterations ($\mathbf{maxIter}$), depth ($\mathbf{k}$)
 2: Output : new placement vector for applications
 3: $C_{\mathbf{p}} \leftarrow J(\mathbf{p}) + \mu G(\mathbf{p})$
 4: nIter ← 0
 5: while nIter < maxIter do
 6:     $\mathcal{N} \leftarrow$ get $\mathcal{N}_{\mathbf{p}}^{k}$, the $k$-neighbourhood of $\mathbf{p}$
 7:     $C_{\mathbf{q}} \leftarrow \min_{\mathbf{q} \in \mathcal{N}} J(\mathbf{q}) + \mu G(\mathbf{q})$
 8:     if $C_{\mathbf{p}} \leq C_{\mathbf{q}}$ then
 9:         break
10:     end if
11:     $C_{\mathbf{p}} \leftarrow C_{\mathbf{q}}$
12:     $\mathbf{p} \leftarrow \mathbf{q}$
13:     nIter ← nIter + 1
14: end while
15: return p
```

For larger neighbourhoods, that is when k is large, branch and bound techniques can be used to efficiently solve the local optimisation problem in Line 6 in Algorithm 1.

4.4.1. EXHAUSTIVE SEARCH

To yield the optimal solution to problem (4.18), so-called exhaustive search is performed, that is to evaluate the objective function for each admissible placement. Doing so provides an upper performance bound of the system at each time it re-evaluates the system. However, the computational complexity of this approach is exponential, since $|\mathcal{A}| = I^N$. Thus, this approach is not feasible for large-scale systems.

The algorithm re-evaluates the objective function, $J + \mu G$, during run-time to compute the total cost for running all the applications inside the infrastructure. With a-priori knowledge of the system's rate of change, one could adjust the rate at which the applications' placements are re-evaluated, accordingly. However, the rate of re-evaluation is beyond the scope of this work; therefore, in this chapter, the algorithm is designed to re-evaluate periodically.

As applications are admitted to the system, an initial placement is performed by searching for the globally optimal location for all applications, which minimises the objective function detailed in Section 4.3. In other words, there is no formal distinction between initial and continuous placement.

4.4.2. ITERATIVE LOCAL SEARCH

To make the algorithm more scalable, the number of evaluations can be reduced by limiting the spatial search domain for each application, in the following called the Iterative local search approach. This algorithm employs a depth-first search approach to find the neighbourhood nodes in the tree. The algorithm starts by finding all the solutions with neighbourhood depth k and computes the local optimal solution. From the local optimal solution, the algorithm constructs its subsequent $k-$neighbourhood and computes a new optimal solution. The algorithm then repeats the process until it achieves the specified maximum number of iterations or a locally optimal solution is found. The algorithm is described as in Algorithm 2,

For larger neighbourhoods, that is when k is large, branch and bound techniques can be used to efficiently solve the local optimisation problem in Line 6 in Algorithm 2.

Each applications distance to its optimal placement determines how many iterations the algorithm will converge. It is therefore vital to have a strategy for placing admitted application's. In the simplest form, the algorithm can, for example, start by randomly placing applications in the network and then let the placement algorithm converge over a certain number of iterations. Starting in a random location can produce situations where the application's placement might never converge to its optimal point. In this chapter, the algorithm places applications as they are admitted optimally by doing an exhaustive search of the entire network.

4.4.3. RE-EVALUATION INTERVAL

At each time stamp of evaluation, all applications are simultaneously evaluated to find the constellation with the lowest global overload and cost. A local or exhaustive search is performed over the possible placement options to find the placement. No discrete events are identified, such as spikes or spatial shifts in demand. Nor does it make any predictions on the workload or resource availability. The system only acts on the momentarily incurred overhead.

Algorithm 2 Iterative local search algorithm

```
Input : current placement vector of applications ($\mathbf{p}$), maximum number of iterations ($\mathbf{maxIter}$), depth ($\mathbf{k}$)
Output : new placement vector for applications
$C_{\mathbf{p}} \leftarrow J(\mathbf{p}) + \mu G(\mathbf{p})$
$\mathbf{nIter} \leftarrow 0$
while $\mathbf{nIter} < \mathbf{maxIter}$ do
    $\mathcal{N} \leftarrow$ get $\mathcal{N}_{\mathbf{p}}^{k}$, the $k$-neighbourhood of $\mathbf{p}$
    $C_{\mathbf{q}} \leftarrow \min_{\mathbf{q} \in \mathcal{N}} J(\mathbf{q}) + \mu G(\mathbf{q})$
    if $C_{\mathbf{p}} \leq C_{\mathbf{q}}$ then
        break
    end if
    $C_{\mathbf{p}} \leftarrow C_{\mathbf{q}}$
    $\mathbf{p} \leftarrow \mathbf{q}$
    $\mathbf{nIter} \leftarrow \mathbf{nIter} + 1$
end while
return $\mathbf{p}$
```

4.5. EVALUATION MODEL

In this section, the simulation set-up is detailed. The behaviour of the placement algorithms is studied by subjecting them to a set curated configuration and workload scenarios. The performance of each algorithm is evaluated in Section 4.6 using a set of heuristics detailed below.

Section 4.5.1 outlines the premise of the experiments and their objectives. Section 4.5.6 details the simulation software framework used to execute the experiments.

4.5.1. EVALUATION METHOD

This section describes the experiment platform used to study the properties and behaviours of the algorithms. First, a description of the different degrees of freedom of the experiments is provided, such as the simulator's parameters, workload properties, and the parameter space for the placement algorithms.

To identify the system's performance boundaries, the placement of each application is evaluated at run-time in a discrete-time manner, employing a set of placement algorithms to regularly re-evaluate the placement of all applications to minimise the total cost of the tree-structured system. To evaluate the performance of the placement methods, a number of performance metrics are observed, including the overall heuristic system cost, resource utilisation, and application RTT. Primarily, the overall cost as defined in Equation (4.14), will contrast how well each placement algorithm deals with a specific scenario or configuration, the system's operational objectives. Secondly, the resource utilisation will show how well the method is meeting the system's objective of minimising the incurred network usage. Lastly, the measured application RTT intends to show to what extent the application's performance will be penalised by a specific placement method.

4.5.2. APPLICATION DEMAND

The application demand model is constructed using four parameters that capture each scenario: time variation, spatial variation, popularity distribution among applications, and resource usage diversity among applications.

To examine the system's behaviour, three relevant example scenarios are considered. The first scenario has a high degree of *User Mobility*, where demand for each application is concentrated to a few leaf nodes and is highly mobile. Second, a diurnal-centric *University Campus* scenario is presented, where the demand of all applications is concentrated at the University campus during daytime and disperses geographically to a uniform distribution at night. Furthermore, the third scenario captures a *Sporting Event* where an application becomes popular in a small group of nodes due to a very high local demand.

User Mobility In the *User Mobility* scenario, the spatial demand distribution is modelled as a normal distribution over a few consecutive nodes to show demand from a batch of users for an application. Time-variation is modelled as a random walk, traversing the consecutive nodes next to the nodes where demand is currently residing. The quantity of demand is not varied, but the spatial distribution changes with time. Furthermore, the popularity distribution among applications is modelled using a uniform distribution. This workload captures the behaviour of a small number of groups of users subscribing to one application each, on consecutive nodes and their independent movement in their locality. This situation resembles that of applications such as Augmented Reality, autonomous vehicles applications, or cloudlets.

University Campus In the *University Campus* scenario, the demand for an application across the network's leaf nodes is modelled as a normal distribution. To capture the students' pronounced diurnal migration patterns [SB04], the standard deviation (σ) is varied diurnally to achieve a night-time near-uniform distribution with higher σ, to a noon distinctly normal distribution with lower σ. As a result, the demand for all applications is higher at the nodes near the university campus during the daytime. The popularity among applications is modelled using a Zipf distribution [Zip49, BFF$^+$10]. The demand variation from a uniform distribution during the night to a normal distribution during the day is modelled as linearly process as a function of time.

Sporting Event During a *Sporting Event*, demand on the nodes corresponding to the places where the teams are based or where they are playing is likely to increase as fans and spectators start browsing for scores or stream the game to their

Mobile Devicess (MDs).

An unexpected surge in demand for these applications is modelled as a spike. In [MDTE15], spikes were formally analysed and defined as a significant shift in the workload pattern. In this work, the time variation for the growing phase of a spike are modelled as,

$$y_\tau = y_{\tau-1}(1 + \alpha_g(1 - \lambda_g y_{\tau-1})), \tag{4.23}$$

where y_τ is a time series of demand for an application, α_g and λ_g are coefficients, α_g denotes the growth rate and $lambda_g$ denotes the slope in the log scale. The inverse of λ is the maximum of the popularity. The decreasing phase of an spike is modelled as,

$$y_\tau = y_{\tau-1}(1 - \alpha_d(\lambda_d y_{\tau-1} - 1)), \tag{4.24}$$

where α_d and λ_d are coefficients. Using the model defined above, a spike similar to the one experienced during the Fifa 1998 world championship [AJ00] is constructed using the parameters, α and λ for each stage of the spike. The growing phase has two stages, one stage for the decreasing phase for the first peak followed by a stage each for the growing and decreasing phases for the second peak. The spike is only present in one node and is stationary.

4.5.3. INFRASTRUCTURE AND TOPOLOGY

In the experiments, the infrastructure is composed of a set of DCs, distributed amongst the veracities of a tree with a certain depth. The capacity and cost of the DCs vary with depth. The veracities are connected with links. The links' capacities progressively diminish with depth.

Application demand originates from the leaf nodes in the network, and it propagates to the DC where the application is hosted. Application demand incurs a proportional resource usage on the DC its applications are hosted on and on the intermediate links.

For each workload scenario, the total amount of resources required for all the applications at each leaf node at each time instance is computed. Based on the aggregate demand, the nodes and links of the system are allocated a proportional amount of resources, so that no application is denied admission to the system due to lack of resources. The remaining nodes are allocated a proportion of the aggregate demand from its children.

4.5.4. APPLICATION TYPES

Applications consume a heterogeneous quantity of resources proportional based on an empirical resource consumption profile and the size of its demand. In the presented model, applications can be CPU, or I/O intensive [WSVY07, QKP$^+$09, QJM$^+$09, KRDZ10, CSW$^+$12]. An example of a CPU intensive benchmark application is Sysbench, while PostMark is a benchmark for I/O intensive applications.

Because the infrastructure is cost heterogeneous and capacity heterogeneous, each application type is expected to have a bias towards a certain depth of the network where it incurs the least operational cost globally or locally. For example, I/O intensive applications are likely to gravitate towards the capillaries of the network where they incur less aggregate network traffic and where I/O is relatively cheaper.

4.5.5. PLACEMENT ALGORITHM PARAMETRISATION

In this section, the exhaustive- and local-search algorithms are parametrised. Details are provided on how they are configured in the experiments. The experiments encompasses the following placement principals/algorithms:

Random static Each application component is initially placed at random in the network and is not dynamically moved.

Random continuous Each application at each re-evaluation is relocated to a random node in the network.

Local continuous Finds local minimal cost search with 4 iterations.

Global static Applications are initially placed in the globally optimal node but are not continuously re-evaluated and relocated. The search depth is set to 4.

Global continuous Each application's placement is continuously re-evaluated and is relocated accordingly. The search depth is set to 4.

The methods that minimise the global cost are bound by the parameters of the cost function and on the penalty function defined in Equation (4.16) and can be configured in many different manners to accommodate different objectives. The optimisation function Equation (4.18) contains a weight μ that combines the overall running cost and the overall penalty (or overload cost).

The Iterative local search was specified in Section 4.4.2. This algorithm minimises the cost function specified in Section 4.3, iteratively, over a confined neighbourhood k, a maximum of **maxIter** times per evaluation.

4.5.6. SIMULATOR

An event-driven simulator is used to validate and evaluate the topology and the presented placement methods. Existing simulation frameworks such as NS-3 [HLR+08] and CloudSim [CRB+11] offer competent network and data centre models, respectively. Nevertheless, neither framework offers a complete solution at the desired abstraction level nor do they scale adequately for large networks. As a result, a coarse-grained event-driven simulator in Python was developed. The simulator utilises SimPy [Mat08] as the underlying event-driven framework. Furthermore, the simulator is constructed around the system model detailed in Section 4.2 which represents a Fog computing infrastructure topology with DCs, a network, and time-variant demand. The input to the simulator is a time series workload composed of a quantity of demand for each application in each leaf node for each time instance. The workload is propagated throughout

Time instance	Parameters
0	$\mu = 10, \sigma = 50, size = 100$
1	$\mu = 15, \sigma = 40, size = 100$
2	$\mu = 20, \sigma = 30, size = 100$
3	$\mu = 30, \sigma = 20, size = 100$
4	$\mu = 40, \sigma = 10, size = 100$
5	$\mu = 50, \sigma = 1, size = 100$

Table 4.1.: University campus scenario parameters

the network to the DCs in which the application is hosted, incurring a resource usage on the host resources proportional to the demand.

To manage the experiments and its outputs, the simulator features a centralised management unit that places new applications as they arrive and updates the network with resources consumed by the applications. One or multiple parallel controller modules, in each DC, can trigger placement re-evaluations, either periodically or at an arbitrary event. The placement algorithms are those specified in Section 4.3. The simulator monitors and outputs the momentary total cost, application RTT, and resource utilisation levels.

4.6. EXPERIMENTS

This section presents the parameter values for the experiments.

4.6.1. WORKLOAD SCENARIOS

Three workload scenarios are presented, *User Mobility* scenario, *University Campus*, and *Sporting Event*. For the *User Mobility* scenario, three nodes are selected at random and users are distributed amongst them according to a normal distribution ($\mu = 20, \sigma = 10$). The demand is spatially moving according to a random walk.

For the *University Campus* scenario, users are distributed among all the leaf nodes according to a normal distribution. The mean and standard deviation of the distribution are varied with time as shown in Table4.1. A histogram is then generated with bin sizes equal to number of leaf nodes, 6 in this work. The time stamps 1 and 5 represent midnight and mid-day workload for the *University Campus* scenario respectively.

For the *Sporting Event* scenario, application popularity is initially uniformly distributed with parameters $a = 0.1, b = 0.6$. An application is made popular by assigning a higher probability after time stamp 25. Parameters $\alpha = 0.4, \lambda = 0.001$ are used for Equations (4.23) and (4.24).

	Property	**Value**	**Description**
Topology	Depth	3	Network depth
	DCs	9	Number of DCs in the network
	Links	16	Number of links in the network
DC capacity	Large	1	Large DC prop. capacity (Reference)
	Medium	$\frac{1}{2}$	Medium DC prop. capacity to Large DC
	Small	$\frac{1}{6}$	Small DC prop. capacity to Large DC
DC cost	Large	1	Large DC prop. cost (Reference)
	Medium	$\frac{5}{9}$	Medium DC prop. cost to Large DC
	Small	$\frac{2}{9}$	Small DC prop. cost to Large DC
Link capacity	-	-	Link capacity is homogeneous
Link cost	-	-	Link cost is homogeneous

Table 4.2.: Topology and infrastructure parameters

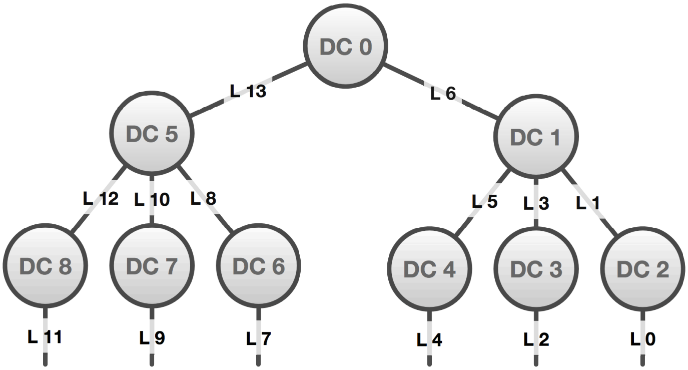


Figure 4.3.: Experiment infrastructure topology

4.6.2. INFRASTRUCTURE

The infrastructure employed in the experiments is distributed in a tree structure as in Figure 4.3 whose parameters are presented in Table 4.2.

The topology has a depth of 3, see Figure 4.3. This topology yields a set of 9 DCs and 14 links. Furthermore, the demand for each application originates at the tree's leaf nodes and is propagated to the node that hosts that particular application. This scale is sufficient to reveal the dynamics of the system. This structure is persisted throughout all experiments.

Let x be the resources consumed by an application. The operational cost is proportional to $\frac{x}{1.33}$ for a large DC, whereas it will be proportional to $\frac{x}{1.2}$ and x for a medium and small DCs respectively.

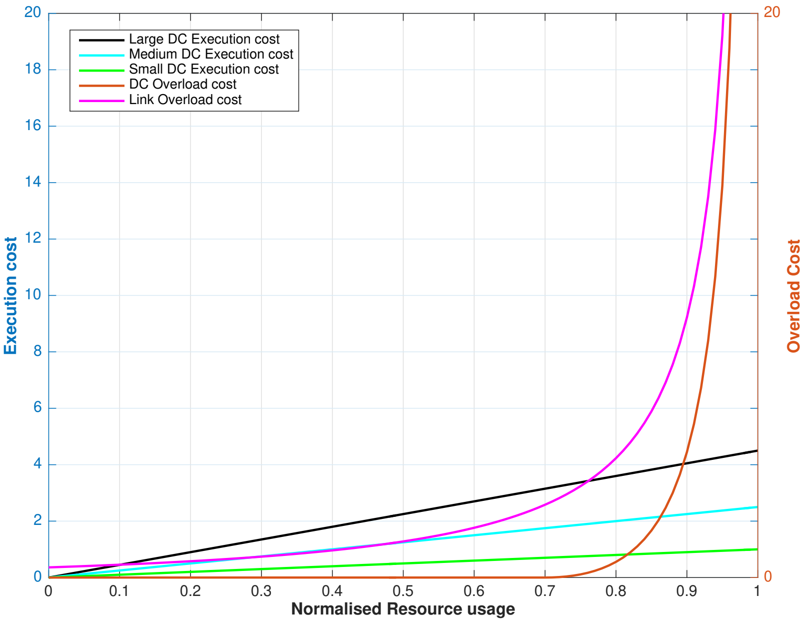


Figure 4.4.: Overload function vs. Resource Execution cost of large and small DC and link

4.6.3. APPLICATION TYPES

The application model described in Section 4.5.4 is used to classify the applications based on their heterogeneous resource requirements as shown in Table 4.3. For example a compute intensive application's requests (of Type 1 in the Table 4.3) spends 5% of it execution time using compute resources and 95% time using I/O resources.

Table 4.3.: Heterogeneous application Model

Type	Compute (%)	I/O (%)
1	5	95
2	50	50
3	95	5

4.6.4. PLACEMENT ALGORITHMS

In the experiments, $\tilde{x}$ in Equation (4.16) is set to 0.7. The execution cost and overload cost for the reference simulation system is illustrated in Figure 4.4. The meeting point of the curves in Figure 4.4 determines the dynamics for the cost for running applications in the system.

4.7. RESULTS

This section presents and discusses the results of the experiments specified in Section 4.6. Firstly, the resulting cost for each placement algorithm in Section 4.7.1 is discussed. Sections 4.7.2 and 4.7.3 contrasts the mean RTT experienced by all applications and the mean utilisation level of each DC as a result of each placement policy, respectively.

4.7.1. COST

Observing the system's aggregate cost as defined in Equation (4.14) reveals how well the set of placement algorithms can meet the systems management objectives, at each point in time. The cost time series of the *Mobile users* scenario is presented in Figure 4.5, followed by the *University Campus* and *Sporting Event* represented by Figures 4.6 and 4.7 respectively.

Starting with the cost-minimising methods, Figure 4.5 illustrates that for the *Mobile Users* scenario, the continuous globally optimal method achieves an average of 25% lower cost than when applications are statically placed in a globally optimal manner. Additionally, the contrast between the static and the continuous approaches reveals that no matter how deliberately you place the applications as they are deployed, there are significant gains to be made if their placement is continuously re-evaluated.

The stochastic nature of the *Mobile users* scenario makes it particularly challenging to manage. In that scenario, the algorithm has no a-priori knowledge of the demand's location and therefore attempts to move the application at the rate at which the demand is changing. Furthermore, the workload in the *Mobile user* scenario is transient, and thus, the volume and constellation of demand will never return to the same quantity and location. Consequently, the cost incurred by an application placed permanently in a DC, optimally or otherwise, is not likely to be either equal or greater to the cost achieved at initial placement. As illustrated by Figure 4.5, on average, when applications are randomly statically placed, the cost is 5% higher than when employing the optimal static placement scheme but comes at a much lower computational cost.

The *University Campus* and *Sporting Event* workload scenarios presented in Figures 4.6 and 4.7, respectively, are more structured since changes follow a less uncertain trajectory. In these scenarios, with a-priori knowledge of workload characteristics, the random placement scheme is particularly ineffective. Therefore, the results for the random placement scheme are not included as it incurs a cost way beyond the scale of the globally and locally optimal solutions, which are the primary methods evaluated in this paper. Because of a-priori knowledge, the static and continuous solutions perform nearly identically, with only a slight advantage to the continuous solution.

As illustrated by Figure 4.5, the locally optimal algorithm, with a maximum of 4 iterations, performs near-optimal on the *Mobile Users* workload. The cost difference

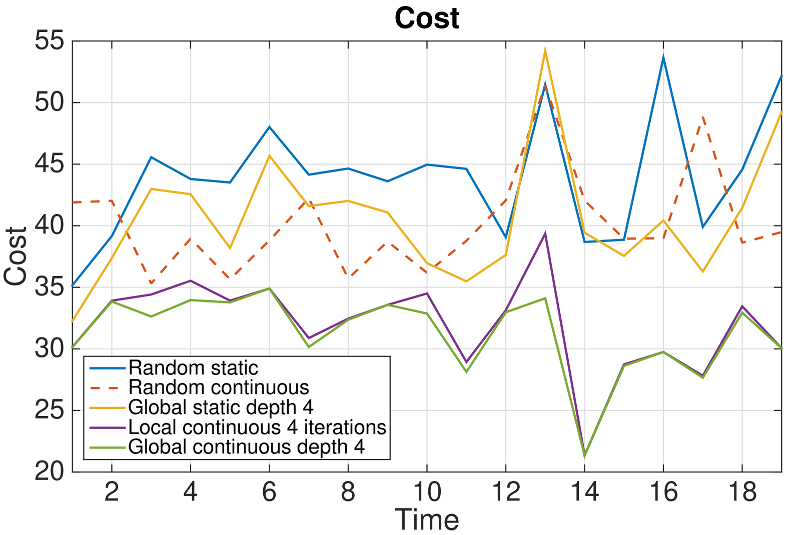


Figure 4.5.: Cost time-series for all placement methods for the *Mobile users* workload

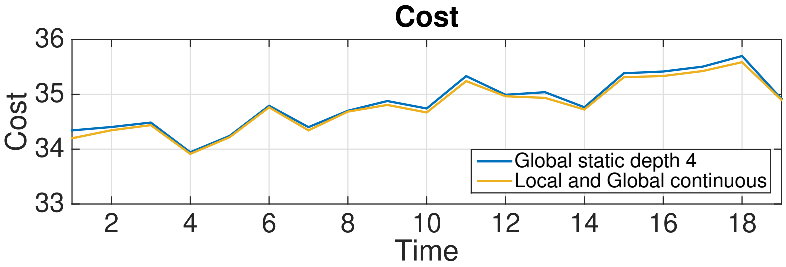


Figure 4.6.: Cost time-series for all placement methods for the *University Campus* workload

is on average 1%. The discrepancy is due to the temporal and spatial diversity of the workload and the resulting multiple minima. This is confirmed by Figures 4.6 and 4.7 where the workloads are less spatially and temporally complex. In the *University Campus* and *Sporting Event* workloads, the cost difference between the locally optimal and globally optimal is on average $< 0.5\%$.

Furthermore, the experiments also show that the search depth has little impact on the system's ability to minimise costs. A search depth of less than 4, the maximum depth of the network, introduces an occasional lag when contrasted with the optimal that is recovered in the next iteration.

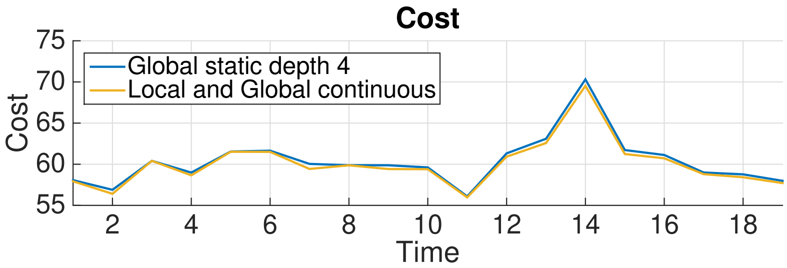


Figure 4.7.: Cost time-series for all placement methods for the *Sporting Event* workload

4.7.2. RTT

Although none of the applications in this scenario have SLA constraints dictating RTT, it is worthwhile to explore the effect the cost minimisation effort has on the mean RTT across all applications. The effort of minimising the aggregate cost and managing each application's RTT are not entirely mutually exclusive. For example, minimising the cost in the manner presented in this paper, applications that are I/O intensive will tend to converge towards the origin of its demand in an attempt to reduce network usage. Moving closer to the origin of its demand also potentially reduces overall contention and thereby also latency.

Figure 4.8 reveals in the *Mobile Users* scenario, that applications on average experience a 35% lower RTT when applications' placements are continuously evaluated than when they are statically placed on admission. Intuitively, placing the applications at random results in a significantly higher RTT for each application on average as the mean network distance tend to be higher between the application instance and its demand. Furthermore, in the *University Campus* workload case, as the geographic discrepancy between the application instance and its demand increases, RTT increases by 10% throughout the scenario, see Figure 4.9. The uniformity of demand across all applications and the stationarity of the spatial centre of demand, leave little room to significantly alter the placement of the applications, resulting in a gradual decline in RTT as demand concentrates around a few nodes. Running the *Sporting Event* workload shows how the affected applications can relocate to accommodate a spatial surge in demand and mitigate some of the incurred latency, see Figure 4.10.

4.7.3. RESOURCE UTILISATION

As stated in Section 4.1, the primary objective of any application placement algorithm is to minimise network usage and to mitigate skewed resource usage of the MNOs' DCs, while meeting the resident applications' SLAs. Attention is now directed at how well the methods mitigate skewed resource utilisation. The algorithms work towards avoiding overloading individual components in the system by progressively penalising high utilisation levels through the penalty component in the objective function, defined

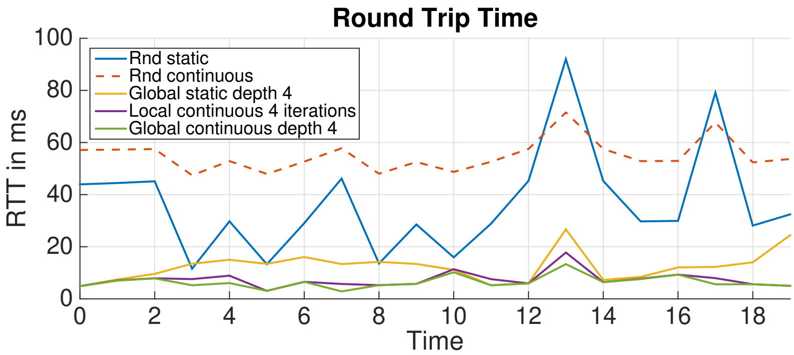


Figure 4.8.: RTT time-series for all placement methods for the *Mobile Users* workload

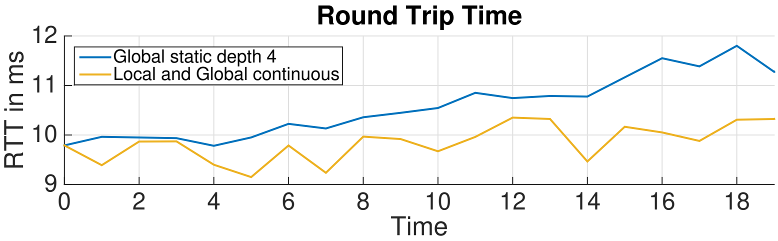


Figure 4.9.: RTT time-series for all placement methods for the *University Campus* workload

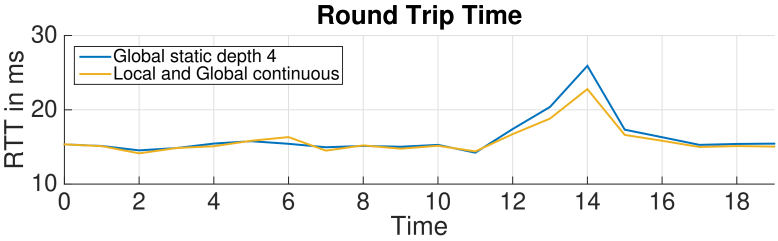


Figure 4.10.: RTT time-series for all placement methods for the *Sporting Event* workload

in Equation (4.16).

Figures 4.11 to 4.13 show that network utilisation is reduced when applications' placements are continuously evaluated. The jagged shape of the DC utilisation CDF exhibited in Figure 4.11 can be explained by the stochastic discrete movement of the demand, from one branch of the tree to another. The stochastic nature of the *Mobile users* workload can also be attributed the more gradual ascent of the link utilisation CDF. Although it exhibits discrete levels, the depth of the network and the number of

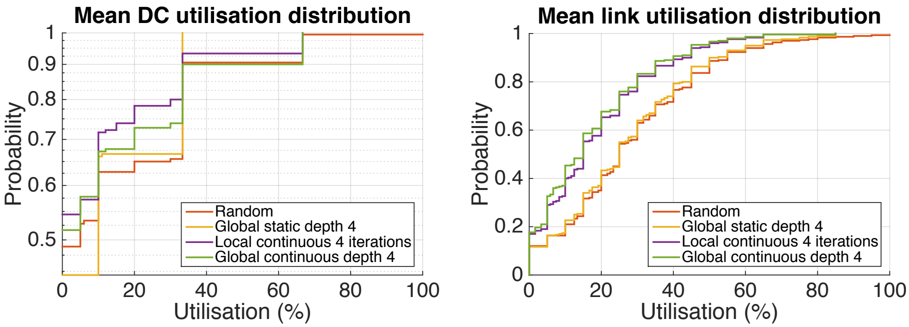


Figure 4.11.: DC and Link utilisation for the *Mobile users* workload

links increase the granularity of the utilisation levels.

Looking closer at the outcome when employing continuous re-evaluation, the globally optimal algorithm achieves lower mean utilisation levels than the locally optimal algorithm. Figure 4.11 shows that the mean link resource utilisation in the Mobile user case is reduced by 5% in favour of the optimal search over the local search method. On the other hand, mean DC utilisation increased by 5%. Figure 4.11 shows that the continuous methods persistently achieve lower utilisation levels but with a similar gradient. This property can be attributed to the reasonably persistent mean spatial offset to the optimal placement. A more extensive network would thus exhibit a greater separation. Furthermore, the random static, random continuous, and globally optimal static placement algorithms produce the same utilisation levels within a range of 2%, with a slight advantage to the globally optimal static placement algorithm.

Having observed the properties of the *Mobile User* scenario, attention is now directed to the structured workloads of *University Campus* and *Sporting Event*. Figures 4.12 and 4.13 illustrate that no algorithm has a mean link utilisation advantage. The results are thus represented as one graph. Again, this can be attributed to the structured nature of the workload where demand is confined in space. Additionally, the steep utilisation level ascent in Figure 4.13 is a consequence of the high concentration of demand in the *Sporting Event* scenario, which confines the application to a small group of DCs. This effect also manifests itself as a higher mean utilisation level in that scenario. Nevertheless, the mean DC utilisation level is more clustered and is confined to a narrow range when using the continuous globally and locally optimal algorithms over the static globally optimal placement scheme, see Figures 4.12 and 4.13. In either case, the utilisation levels are kept at a desirable level.

4.8. RELATED WORK

In this section, related works are surveyed, and a discussion is provided on how the result can be employed in Fog computing research and some of the challenges that

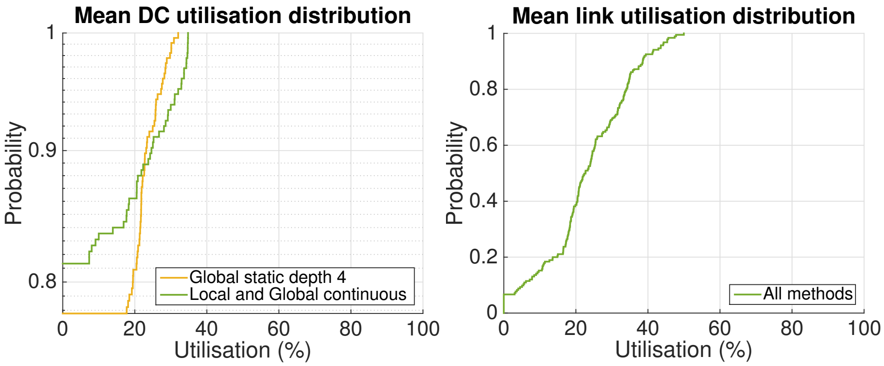


Figure 4.12.: DC and Link utilisation for the *Campus* workload

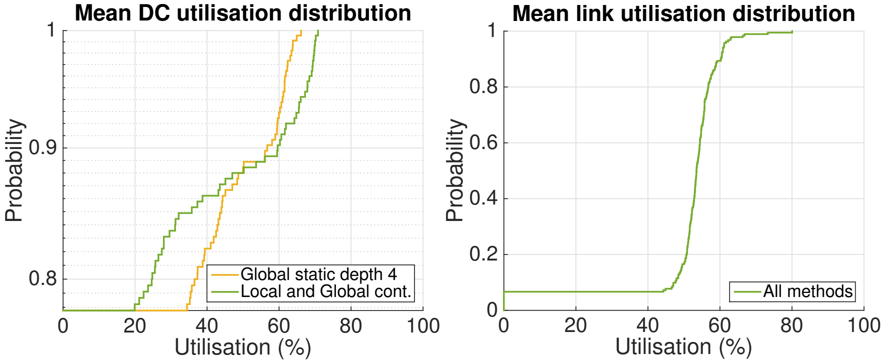


Figure 4.13.: DC and Link utilisation for the *Sporting event* workload

remain unanswered by the literature. There exists an extensive body of work in the field of content and service placement in distributed compute and content delivery systems. Existing research results address many of the challenges facing a Fog computing infrastructure.

4.8.1. REPLICA PLACEMENT

There have been numerous efforts at developing algorithms for replicas in a network of computers [KK04]. In general, their objectives are to either optimise the application's performance in existing infrastructure, to reduce the computational complexity of the decision or minimise the infrastructure cost while meeting the application's SLA. The suboptimal algorithm proposed in [SPvS05] attempts to minimise the communication latency between the application and its clients by identifying and placing replicas in regions based on the relative latencies between nodes. Similarly, [RGE02] proposes a topology-aware replica placement. The two algorithms achieve near-optimal placement of replicas, but do not address the mobility of the service's users and thus the continuous evaluation of the service's placement in the network.

4.8.2. CDN AND CACHING

Many challenges facing a Fog computing infrastructure, such as application placement in a distributed Telco-infrastructure and stochastic workload/demand are also found in CDNs. Research on CDNs has yielded mature centralised [RGE02, QPV01] and distributed [ZG11] methods for initial and churn-driven continuous placement of content in caching infrastructure core networks to mitigate network contention, and ensure content availability. The methods often employ a Mixed Integer Programming (MIP) approach to compute a global or local placement optimum. The minimisation functions often incorporate system properties such as demand churn, network topology, and foreground traffic to meet the MNO's objectives. Due to the nature of CDN infrastructures, the methods mentioned above do not take into account user mobility, application/content component affinity and execution, and finite heterogeneous resources.

4.8.3. INTER-AND-INTRA DATA CENTRE VM-PLACEMENT

Research in intra-DC VM placement and continuous placement evaluation has yielded methods [SLW$^+$14, MPZ10, THOP13] to primarily consolidate applications, improve locality, and to minimise network and energy usage inside and across DCs. The literature often assumes a tree-structured topology with resource contentions similar those found in a Fog computing infrastructure. Methods that actuate Intra-DC VM placement are often agnostic to the intermediate topology between the DCs, relying primarily on observed performance. Although the surveyed methods adequately take into account network topology and heterogeneity, compute resources are assumed to be homogeneous, with no application component affinity. A high degree of resource heterogeneity and application component affinity are two fundamental properties of a Fog computing infrastructure.

These works often resort to Mixed Integer Programmings (MIPs) to resolve a complex set of constraints and relationships between DC resources and application requirements. The objective is often to reduce/minimise Intra-DC network contention and to reduce cost and energy consumption by improving application locality and more appropriate VM to PM mappings. As the placement domain is either within a DC, or in a set of DCs, [KFCM12] the research fails to take into account end-user locality and rapid spatial and temporal changes in demand.

A method to increase individual end-user proximity to their user data by migrating and duplicating user instances on a planetary scale with the objective to reduce RTT is presented in [ADJ$^+$10]. The work highlights the inherent challenges of what to migrate, duplicate, and replicate, and how to evaluate the action's performance return. The work does however not take into account demand churn and replicating whole instances of an application, nor does it consider a fine-grained network topology, such as an MNO's access network. In [LSYR11], the authors have devised an approach to migrate and duplicate data across continental distances while taking into account the intermediate network's availability and the contention the transfer incurs.

The challenge of placing applications in a cloud environment has been addressed in the literature for content routing [MPZ10], intra- and inter-DC application placement [BB12, LSYR11], and in optimal content distribution in CDNs [BGW10]. To the best of our knowledge, none of the presented approaches simultaneously and holistically consider a set of criteria that are synonymous with vast resource cost- and capacity-heterogeneous infrastructures with a high geographic granularity.

The broad challenges of VM placement across DCs are taxamonised in [Man15] and formalised in [PM15]. The literature contains work on the placement of applications and their constituent VMs in DCs, to minimise cost [BB12], energy consumption [BAB12], data-locality [PSLJ11], and network usage [BCF$^+$12]. These methods primarily address the internal objectives of a DC and therefore inherently disregard the geographical discrepancy to the end-users and the heterogeneity in both applications and infrastructure. Thus, they cannot be applied to this problem. Additionally, the internal administration of a Fog computing infrastructure's DCs is beyond the scope of this work.

Furthermore, CDNs share much of the same distributed topological properties of a Fog computing infrastructure but operate with the objective of maximising the hit-rate of a set of content over a finite set of resources as a function of the content's popularity. In a CDN, content is static, and resource usage is often not proportional to the demand and is confined to storage. Additionally, no performance guarantees can be given for all applications, and immediate scalability needs are not a concern. In contrast, in a Fog computing infrastructure, resources are heterogeneous, and applications are highly dynamic with heterogeneous performance requirements that all must be accommodated.

4.9. CONCLUSIONS

One of the foremost challenges in the Fog computing paradigm is how to manage the highly heterogeneous and distributed resources in a complex system. This work contributes with a system model for a Fog computing infrastructure and an objective function to minimise the global system cost as a means to manage the computing and network resources in a Fog computing infrastructure. Based on this model, globally optimal placement of static and mobile applications was designed. Further, a locally optimal placement scheme with at a fraction of the computational cost was designed. Additionally, a set of near-optimal and intuitive methods are used to contrast the performance of the presented algorithms. Also, based on the presented model, a simulation environment was developed on which the algorithms are evaluated using a set of challenging Fog computing workloads.

The results reveal that when user demand is highly mobile and stochastic, significant resource utilisation and cost gains can be made when one employs any method that attempts to map the location where the application is executed with the location of that application’s demand. Furthermore, the experiments also reveal that the globally optimal and locally optimal schemes can achieve near equal performance with workloads that have a spatially uniform distribution of demand. A difference in performance between the algorithms was uncovered when subjecting the system to a workload with a spatially non-uniform demand. Here, the globally optimal approach outperforms the locally optimal. Nevertheless, with a transient workload, system cost and resource utilisation are with either algorithm non-divergent.

5

Distributed Fog computing resource management

Optimally placing the resident applications in a Fog computing infrastructure, given the constraints addressed in this thesis, is NP-hard [SG76]. The optimal placement of the Fog's resident applications was studied in Chapter 4, where it was concluded that a centralised solution is not scalable because it fundamentally fails to keep up with the system's rate of change. A centralised agent can not feasibly manage a massive heterogeneous infrastructure hosting a vast number of applications. Scalability is a significant challenge for a Fog resource manager. The work in this chapter extends upon the work in Chapter 4 with a distributed scalable algorithm that takes into account the objectives of the Fog computing infrastructure and the resident applications, that solves the application placement challenge. The algorithm takes a holistic approach by accommodating the system's primary objectives over a neighbourhood of DCs. By doing so, the algorithm can accommodate the heterogeneous resources and applications in the system without incurring additional cost and application placement oscillations. The algorithm is defined in Section 5.2. The algorithm is evaluated over a set of infrastructure topologies and contrasted with an optimal and a naïve method, detailed in Section 5.3. The results of the evaluation presented in Section 5.4 show that the algorithm can quickly and consistently converge while meeting all constituent entities' objectives. It is also shown that the algorithm approaches the system's optimal cost point within 8% and in a reasonable amount of time. Furthermore, the algorithm also outperforms the naïve method, both in term of convergence and cost. The evaluations also reveal some of the distinct challenges with the different topologies.

5.1. EXTENDED FOG COMPUTING MODEL

In this section, the models in Chapters 3 and 4 are extended upon to accommodate the challenge addressed in this chapter. Note that the model notations from Chapter 4 will

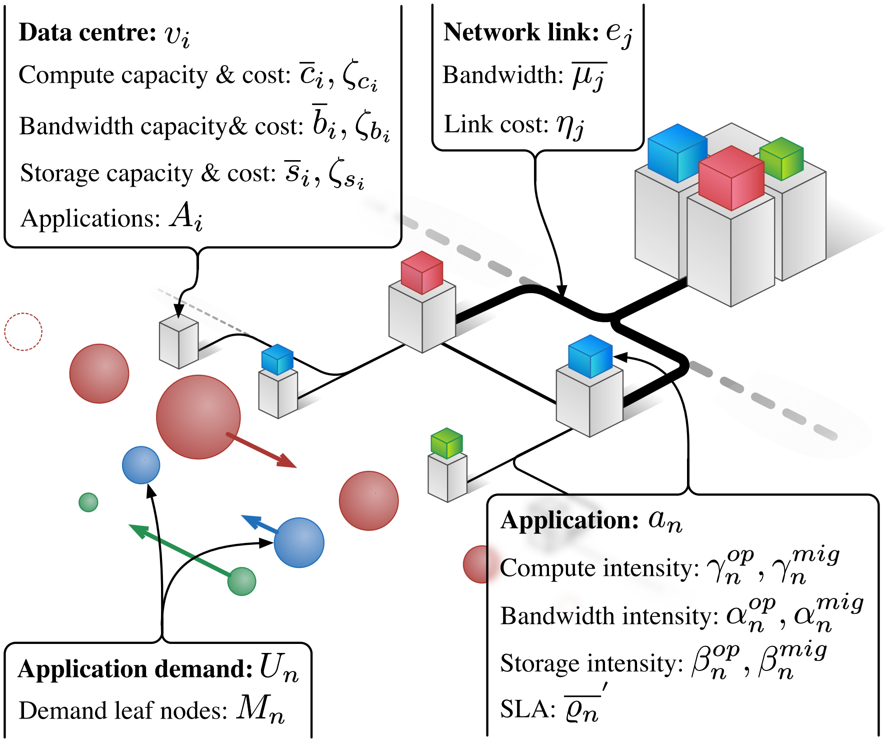

Data centre: v_i
Compute capacity & cost: $\overline{c}_i, \zeta_{c_i}$
Bandwidth capacity& cost: $\overline{b}_i, \zeta_{b_i}$
Storage capacity & cost: $\overline{s}_i, \zeta_{s_i}$
Applications: A_i
Network link: e_j
Bandwidth: $\overline{\mu_j}$
Link cost: η_j
Application: a_n
Compute intensity: $\gamma_n^{op}, \gamma_n^{mig}$
Bandwidth intensity: $\alpha_n^{op}, \alpha_n^{mig}$
Storage intensity: $\beta_n^{op}, \beta_n^{mig}$
SLA: $\overline{\varrho_n}'$
Application demand: U_n
Demand leaf nodes: M_n

Figure 5.1.: Model overview with entities and their properties

feel mostly familiar, but have been extended to incorporate the richer system dynamics addressed in this chapter. The presented model is used for defining the presented algorithm as well as for constructing a simulated environment for evaluating the algorithm. An overview of the model's components can be seen in Figure 5.1.

5.1.1. TOPOLOGY

A Fog computing infrastructure is modelled as an undirected graph where the vertices are DCs and the edges are network links, each with a set of finite resources. Applications admitted to the Fog are hosted in DCs and are subject to demand through the network links, originating at the graph's leaf, i.e. vertices with degree one. Thus, let the graph $G = (\mathcal{V}, \mathcal{E})$ denote a Fog computing infrastructure topology, where

$$\mathcal{V} = \{v_i \mid i = 1,2,...,I\}, \tag{5.1}$$

$$\mathcal{E} = \{e_j \mid j = 1,2,...,J\}, \tag{5.2}$$

See Figure 5.1 for a visualisation of the system's topology.

5.1.2. DATA CENTRE MODEL

In a Fog computing infrastructure, traditional, centralised DCs are supplemented by a large set of geographically dispersed DCs that are embedded in an MNO's infrastructure. The DCs within a Fog computing infrastructure are both capacity- and cost-heterogeneous.

A vertex v_i in the graph has the following capacities, expressed as real positive numbers: compute capacity $\overline{c_i}$, storage capacity $\overline{s_i}$, and bandwidth $\overline{b_i}$. A DC's resource requests are aggregated into resource units. These units can be seen as VMs or containers. A resource unit is defined by a compute capacity $\overline{c}_{VM}$, a storage capacity $\overline{s}_{VM}$, and a bandwidth $\overline{b}_{VM}$, expressed as real numbers, where $\overline{c}_{VM} \ll \overline{c_i}$, $\overline{s}_{VM} \ll \overline{s_i}$, $\overline{b}_{VM} \ll \overline{b_i}$. The momentary utilisations of these resources are expressed as real numbers; compute utilisation c_i, storage utilisation s_i, and bandwidth utilisation b_i. A DC is assumed to be able to accommodate any set of applications that aggregately do not exceed its capacity.

Additionally, a vertex v_i is associated with an operational cost per resource and time unit. These operational costs are defined by the following real number functions of utilisation: compute cost ζ_{c_i}, storage cost ζ_{s_i}, and bandwidth cost ζ_{b_i}.

5.1.3. NETWORK MODEL

In a Fog computing infrastructure, the links that join the DCs have different cost and capacity depending on the depth they are at in the network, who owns them and their communication media type. A link in the network is modelled as an edge e_j, $j \in \{1,2,...,J\}$ in G and has a non-directional capacity expressed as a bandwidth $\overline{\mu_j}$. Additionally, a network resource e_j has a link cost η_j, which is a function of throughput that returns the link's running cost per time unit.

5.1.4. APPLICATION MODEL

The set of applications hosted by a Fog computing infrastructure, $\mathcal{A} = \{a_n \mid n = 1,\ldots,N\}$, are assumed to be wholly managed by the Fog. The resident applications' owners are therefore agnostic to where and how their applications are executing.

Each application a_n, where $n \in \{1,2,\ldots,N\}$ is served by a DC, v_i. Each DC v_i hosts a set of applications $\mathcal{A}_i \subseteq \mathcal{A}$. An application a_n is defined by the following increasing functions of the demand for the application's operational compute intensity γ_n^{op}, operational storage intensity α_n^{op}, and operational bandwidth intensity β_n^{op}.

Migrating an application between two DCs incurs additional resource usage for both the recipient and the host. The additional load is defined by a migration compute intensity γ_n^{mig}, migration storage intensity α_n^{mig}, and migration bandwidth intensity β_n^{mig}, all functions of the application's aggregate demand.

DEMAND

The applications' end-users subject the applications to a quantitatively and spatially time-variant demand. An application a_n is subject to an aggregate demand from a set of demand sources $\mathcal{U}_n = \{u_{n,m} | m \in \mathcal{M}_n\}$ where $\mathcal{M}_n \subset \mathcal{V}$ is the set of leafs from which the demand originates. Each source of demand $u_{n,m}$ is represented by a function of time that returns a real number specifying the demand for application a_n at time t from leaf m.

PERFORMANCE REQUIREMENTS

Application owners can impose a set of performance requirements per application that the operator of a Fog computing infrastructure is obliged to accommodate, an SLA. In this work, an application's SLA is expressed as by convention; the maximum of the 95th-percentile of the network delay distribution [SBDR07]. Furthermore, network delay is proportional to the number of links separating an application's end-user from the current hosting DC. Thus, an application's SLA $\overline{\rho_n}'$ is defined as the upper limit of the 95th percentile of the mean network distances between its set of sources of demand U_n and the DC it is hosted v_i. The set of network distances for application a_n is defined as:

$$l_n = \{|\sigma_{\mathcal{V}}(v_i, v_m)| \mid m \in \mathcal{M}_n\} \tag{5.3}$$

where $|\cdot|$ denotes the cardinality of a set, $\sigma_{\mathcal{V}} : \mathcal{V} \times \mathcal{V} \to \mathfrak{P}(\mathcal{V})$, with $\mathfrak{P}(\cdot)$ being the power set operator, is a function that determines the minimum path between two nodes. See Figure 5.1 for an illustration of the relationship between the applications and their demand.

5.2. DISTRIBUTED RESOURCE MANAGEMENT ALGORITHM

The presented algorithm scalably solves the challenge of placing a set of highly heterogeneous applications in a cost-heterogeneous and capacity-heterogeneous distributed cloud infrastructure while meeting both the DCs' operational cost and the infrastructure's resident applications' performance objectives.

Central to the algorithm are two types of reactive agents, a DC-agent and an application-agent. The agents represent the objectives of the two primary stakeholders in the system, namely DCs and applications. The agents act independently based on the performance of their respective objectives, namely operational cost and application SLA.

To achieve the objectives in a tractable manner, each agent reacts to a violation of a system objective, by re-evaluating the placement of a set of applications over a subset of DCs in a neighbourhood. The neighbourhood of depth at most k for v_i is defined as the set:

$$\mathcal{N}_i^k := \left\{|\sigma_{\mathcal{V}}(v_i, v_j)| \leq k+1 \mid j = 1, \ldots, I, \, j \neq i\right\}. \tag{5.4}$$

The resulting placement decision is reached using a common heuristic objective function $\mathcal{R}$ that is formalised in Section 5.2.1. The two agent types react to a system objective violation by re-evaluating the common heuristic objective function $\mathcal{R}$ over a subset of the system's resources and applications. To meet their objectives, the common heuristic objective function is applied differently for each agent. The fundamental properties of the algorithm are illustrated in Figure 5.2.

Strict caps on resource utilisation and costs do not accommodate variations in demand across the system and might either put the system in an unstable state or require a much finer granularity of evaluation, at a significant cost. Therefore, in this algorithm, a budget for each DC is adopted to represent its desired resource utilisation or cost level, over time. The long-term objective of the algorithm is to maximise the mean budget surplus across the system. A DC's budget surplus or deficit history is distributed amongst its peers and is used to evaluate its suitability when re-evaluating the neighbourhood's application's placements. More on the budget-mechanism in Section 5.2.2.

5.2.1. COMMON OBJECTIVE FUNCTION

In a distributed heterogeneous system, such as the Fog, an application can incur very different loads and costs in different DCs in a neighbourhood. Similarly, applications' SLAs might be accommodated with varying success amongst a set of neighbouring DCs. Thus, migrating a set of applications from one DC to mitigate its budget violation may violate the applications' SLAs or incur budget violations in the recipient DCs. They, in turn, might incur additional violations and application placement oscillations due to subsequent mitigation actions.

Application migrations are preferably avoided as they incur additional resource usage. Consequently, because a migration incurs a cost, a placement decision should preferably be long-lasting. Thus, the objective function should take into account both budget constraints, SLA constraints, and the additive cost of link usage in a holistic manner.

The common objective function $\mathcal{R}$ is formulated for DC $v_q \in \mathcal{V}$ and the running applications $\overline{\mathcal{A}}$ at time t as,

$$\begin{aligned} R(q,\overline{\mathcal{A}},t) := \sum_{i\in\mathcal{N}_q^k}\sum_{n\in\overline{\mathcal{A}}} \mathcal{P}_{i,n}^{q}(\\ + \phi_b \frac{1-(\vartheta_{n,i}^{\mathrm{op}}(t)+\vartheta_{n,i}^{\mathrm{mig}}(t))}{\xi_i(t)} \\ + \phi_s \frac{1-\rho_{n,i}^{95th}}{\overline{\rho_n}'} \\ + \phi_l \mathcal{L}) \end{aligned} \tag{5.5}$$

where $\vartheta_{n,i}^{\mathrm{op}}$ is the momentary operational cost for each application in DC i,

$$\vartheta^{\text{op}}_{n,i}(t) = \psi_n(t)\zeta_i^{VM}, \tag{5.6}$$

with $\psi_n(t)$ being the unitary resource allocation cost of application n in any DC that is defined as:

$$\psi_n(t) = \left\lceil \max\left(\frac{\gamma_n(t)}{\bar{c}_{VM}}, \frac{\alpha_n(t)}{\bar{s}_{VM}}, \frac{\beta_n(t)}{\bar{b}_{VM}}\right)\right\rceil, \tag{5.7}$$

where ζ_i^{VM} is the cost of each resource unit in DC i and is defined as:

$$\zeta_i^{VM} = \bar{c}_{VM}\zeta_{c_i} + \bar{s}_{VM}\zeta_{s_i} + \bar{b}_{VM}\zeta_{b_i}. \tag{5.8}$$

where $\mathcal{L}$ is the aggregate system-wide link and is formally defined as,

$$\mathcal{L} = \sum_{i\in V}\sum_{n\in\mathcal{A}_i}\sum_{m\in\mathcal{U}_n} \eta_m \beta_n^{op} u_{n,m}(t) \tag{5.9}$$

and $\mathcal{P}^q_{i,n}$ being the elements of a binary matrix $\mathcal{P}^q$ of size $|\mathcal{A}| \times |\mathcal{N}^k_q|$, here called *application placement decision matrix*. In particular, $\mathcal{P}^q_{i,n}$ is equal to 1 if and only if application $a_i \in \overline{\mathcal{A}}$ is placed in node $v_n \in \mathcal{N}^k_q$, 0 otherwise. Note that, by construction, each row sums to 1, since an application cannot be placed in more than one node. Finally, $\{\phi_b, \phi_s, \phi_l\}$ are weights in the interval $[0,1]$. The i^{th} row of the matrix represents the placement of the i^{th} application $a_i \in \bar{\mathcal{A}}$ amongst the DCs in the neighbourhood $\mathcal{N}^k_q$, represented by the columns.

The placement decision is expressed as follows,

$$\mathcal{P}^{q,\star}(t) := \underset{\mathcal{P}^q}{\arg\max}\, \mathcal{R}(q, \overline{\mathcal{A}}, t) \tag{5.10}$$

constrained by each evaluated DC's budget surplus ξ_i and the operational cost ε_i, formalised as,

$$\sum_{n\in A_q} \mathcal{P}^q_{i,n}\vartheta^{\text{op}}_{n,i}(t)t_e \le \xi_i(t) \tag{5.11}$$

$$\sum_{n\in A_q} \mathcal{P}^q_{q,n}\vartheta^{\text{op}}_{n,q}(t)t_e \le \varepsilon_i \tag{5.12}$$

Because the algorithm is applied iteratively and is not evaluated over the entire network, hard constraints cannot be applied to individual application's performance. In the worst case, an application might have to traverse a set of DCs where its SLA will be violated to reach a DC where it can run in compliance with its SLA.

When maximising the objective function $\mathcal{R}(q, \bar{\mathcal{A}}, t)$, the remaining budget across all its neighbours reduces the number of additional violations in the neighbourhood and the process also takes into account any pending SLA violations, when the system is in a stable state. By normalising each component in the objective function with their quantitative targets, the algorithm can indiscriminately evaluate the placement of $\bar{\mathcal{A}}$ across a highly heterogeneous set of DCs and applications. As the incurred is not

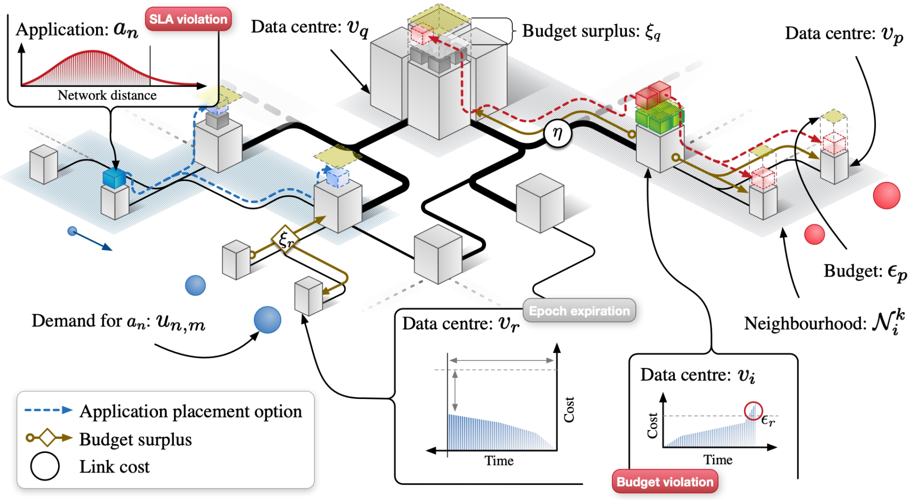


Figure 5.2.: The algorithm's mechanism; agents, actions, and evaluation domains.

accommodated in a DC's budget, the link cost is treated independently. The above-detailed mechanisms of the algorithm and its parameters are illustrated in Figure 5.2.

The use of a budget to represent the state of a DC and the limited evaluation domain imposed by the neighbourhood decouples the algorithm allowing it to be implemented in a distributed manner. Additional information and state granularity would require significantly more synchronisation between agents and states and information passing, marking the implementation intractable.

5.2.2. DATA CENTRE AGENT

Each DC in the network is governed by a DC agent. The objective of a DC agent is to contain the operational cost of a DC and is realised by the budget monitor process which is continuously run in each DC $v_i \in \mathcal{V}$.

Essential to the algorithm and the DC agent is a budget that is assigned to each DC. A DC's budget is the maximum allowed operational expenditure over a period and is a heuristic for a DC's capacity and desired maximum utilisation over that period. In practice, the budget allows the Fog operator to set coarse-grained holistic objectives for the system's resources that do not interfere with the internal management of each DC. Additionally, the budget also allows the algorithm to integrate temporary costs, over time, such as migration overheads and smaller workload variations, within the confines of the budget over an epoch.

BUDGET MONITOR PROCESS

The budget monitor process in each DC is assigned a budget ε_i for its operational cost over a period, referred to as an epoch Δt_e. The operational cost of a DC is defined as,

$$\zeta_i(t_1,t_2) = \int_{t_1}^{t_2} \sum_{n\in\mathcal{A}_i} \vartheta_n^{\mathrm{op}}(t)\,\mathrm{d}t, \tag{5.13}$$

In runtime, the operational cost ζ_i of each DC is evaluated over an epoch of length Δt_e. If the budget is violated before the end of the epoch, i.e., $\zeta_i(h\Delta t_e, t_2) \geq \varepsilon_i$, with $h \in \mathbb{N}$, and $h\Delta t_e \leq t_2$, the placement of the resident applications $\mathcal{A}_i$ is evaluated over the neighbourhood $\mathcal{N}_i^k$ using the objective defined in Section 5.2.1.

When a budget is violated or when an epoch expires without a budget violation, the budget is renewed for another epoch. For each such event, the budget surplus $\xi_i(t) = \varepsilon_i - \zeta_i(h\Delta t_e, (h+1)\Delta t_e)$ of v_i is passed to all its neighbours $\mathcal{N}_i^k$. The resulting vector of the last reported neighbours' budget surpluses for DC i is denoted as,

$$\mathcal{B}_i(t) = \{\xi_q(t) \mid q \in \mathcal{N}_i^k\} \tag{5.14}$$

The budget monitor process is summarised in Algorithm 3.

Algorithm 3 Budget monitor process for DC v_i,
for each epoch.

```
Input : Budget $\varepsilon_i$, application set $\mathcal{A}_i$, current placement $\mathcal{P}^i$, and budgets in neighbourhood $\mathcal{B}_i$,
Output : Budget surplus $\xi_i$,
application placement matrix $\mathcal{P}^{i,\star}$
$t \leftarrow 0$, $\zeta_i' \leftarrow 0$
$\mathcal{P}^{i,\star} \leftarrow \mathcal{P}^i$
while $t < \Delta t_e$ do
    $\zeta_i' \leftarrow \zeta_i' + \zeta_i(t, t+\Delta t)$
    if $\zeta_i' \geq \varepsilon_i$ then
        $\xi_i \leftarrow 0$
        $\mathcal{P}^{i,\star} \leftarrow \arg\max_{\mathcal{P}^i} \mathcal{R}(i, \mathcal{A}_i, t)$
        break
    end if
    $\xi_i \leftarrow \varepsilon_i - \zeta_i'$
    $t \leftarrow t + \Delta t$
end while
return $\{\xi_i, \mathcal{P}^{i,\star}\}$
```

Algorithm 4 SLA monitor process for application n.

```
Input : Hosting DC $v_i$, SLA $\overline{\rho_n}$
Output : Application placement matrix $\mathcal{P}^{i,\star}$
while true do
    if $\overline{\rho_{i,n}} \geq \overline{\rho_n}'$ then
        $\mathcal{P}^{i,\star} \leftarrow \arg\max_{\mathcal{P}^i} \mathcal{R}(i, \{a_n\}, t)$
        break
    end if
    $t \leftarrow t + \Delta t$
end while
return $\mathcal{P}^{i,\star}$
```

STATE

The state of a DC agent is defined by its budget surplus ξ_i, its resource unit cost ζ_i^{VM}, its resident applications $\mathcal{A}_q$, and the budget surplus of its neighbours.

5.2.3. APPLICATION AGENT

An application agent monitors the performance of each resident application in the infrastructure. The objective of an application agent is to ensure that the observed application meets its SLA. This is realised by an SLA-monitoring process which is continuously run in parallel to each application a_n in each v_i.

SLA MONITORING PROCESS

An application's performance is measured in terms of its SLA, $\overline{\rho_n}'$. An application's placement is re-evaluated when its SLA is violated, $\overline{\rho_{i,n}} \geq \overline{\rho_n}'$. The SLA monitoring process is summarised in Algorithm 4.

Note that in the case of an SLA violation, only one application is evaluated, i.e. $\bar{\mathcal{A}} = a_n$.

STATE

The state of an application is defined by its demand's location, quantity $\hat{\mathcal{U}}_n$, and current latency performance $\rho_{i,n}^{95th}$.

5.3. EXPERIMENTS

The experiments detailed below are designed to examine the viability of the algorithm as a tractable holistic Fog computing resource management approach. Given the distributed nature of the algorithm, the evaluation is primarily focused on stability and on how closely it performs to optimal, as defined in [TMW$^+$16]. The experiments

Table 5.1.: DC categories, their capacity and costs.

	Small	Medium	Large	Huge
Capacity ψ_n	250	500	1000	2000
Unit cost [2]	150%	125%	112.5%	100%

are designed to reveal the algorithm's convergence time from a random state as well as its step response. For comparison, both a random placement method and a naïve method are included. They are defined in Section 5.3.6. Additionally, to evaluate how the distributed algorithm performs in both current and forthcoming network topologies, the experiments employ both fat-tree and random graph network topologies. The parameters in the experiments are based on the findings in [MTK+16].

As there are no Fogs yet in existence, the experiments are conducted in a simulated environment. The simulator is written in python [1] around a SimPy's time-driven core using MILP solvers from PuLP [MOD11] to represent and solve the objective function.

5.3.1. INFRASTRUCTURE

Similar to the model presented in Chapter 4, a Fog computing infrastructure is represented by a set DCs and links in a network. The model is defined in Section 5.1. To add cost- and capacity-heterogeneity to the infrastructure, a set of categories for each resource categories are defined. Each category has a unique capacity and cost, reflective of their position in the infrastructure, these are specified below.

5.3.2. DATA CENTERS

A DC's resources are partitioned into and provisioned as discrete units. DCs are categorised as either Small, Medium, Large, or Huge. The DC capacity is halved for each following category, proportional to its depth in the network, while the operational cost grows linearly with depth. For example, a Huge DC is 8 times larger than a Small DC and cost 44% less to operate per resource unit. The properties of each DC category are summarised in Table 5.1. The DCs are assigned a budget ε_i that is proportional to 80% of the total cost of all resources over an epoch, as advised by [PBM+07].

5.3.3. LINKS

The links are categorised by capacity $\overline{\mu_j}$ as either Small, Medium or Large. A definition of each link category's properties can be found in Table 5.2.

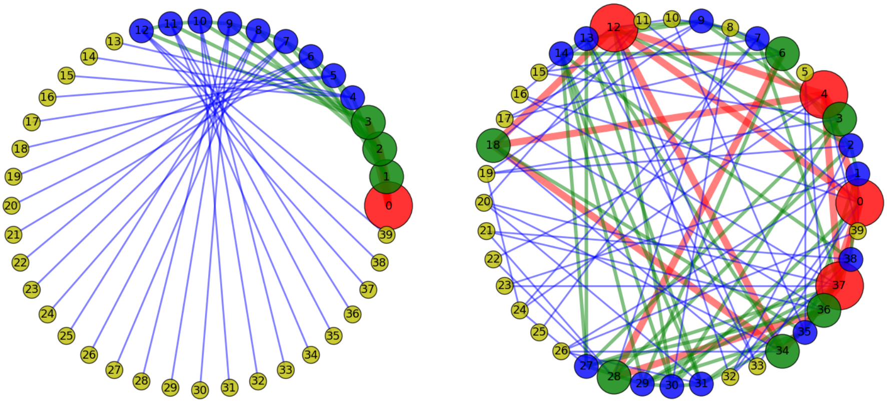
5.3.4. TOPOLOGY

The DCs and links specified above are situated in a network. In this work, a fat tree and an Erdös-Rényi random graph are used to evaluate the performance of the distributed algorithm, representing a current and forthcoming network topology, respectively. The

[1] Code and experiments available at: `gitlab.com:eit-wit/mcn\_placement\_simulator.git`

Table 5.2.: Link categories, their capacity and costs.

	Small	**Medium**	**Large**
Capacity $\overline{\mu_j}$	3000	5000	80000
Unit cost [3]	77%	87%	100%



(a) Fat-tree with depth 4 and branching factor 3.

(b) Erdös-Rényi random graph with branching probability of 1.1.

Figure 5.3.: Network topologies used in experiments, each with 40 nodes. DC assignment: Huge, Large, Medium, Small. Link assignment: Large, Medium, Small.

topologies used in the evaluation both consist of 40 nodes ($I = 40$), which corresponds to the typical size of a regional Fog computing infrastructure. The topology types have an equivalent depth and total DC capacity, see Figure 5.3.

FAT TREE

Mobile core and access networks often take the shape of fat trees, with the middle tiers having the highest degree of interconnectivity [Bed14]. The network is assigned a Huge DC in the root node, Small DCs are assigned in the leaf nodes, and remaining nodes are assigned a DC category per its depth in the network. Figure 5.3a illustrates the structure and resource assignment of the fat-tree topology.

RANDOM GRAPH

Access and core networks are becoming more and more interconnected, through multiple carriers and with the addition of new disaggregated network technologies [BHL+14]. To imitate this type of topology, an Erdös-Rényi random graph of $I = 40$ nodes is used. The graph is generated using a branching probability of 1.1.

The nodes in the network are assigned to a DC category based on their number of connections. The tier of nodes with the fewest connections is assigned a Small DC. The top 10% of the nodes with the highest number of branches are assigned a Huge DC. Intermediate nodes are assigned either Large and Medium DC in proportion to their network distance to one of the Huge DCs. The topology's structure and resource assignment are illustrated in Figure 5.3b.

A critical difference between the two topologies is that a random graph is more heterogeneous than a fat-tree in the sense that a fat-tree is symmetric and that the depth of the network strongly correlates with the mean distance to the demand, DC capacity, and degree of connectivity. Furthermore, in a random graph, a set of neighbours do not have to be of similar capacity and with very different degrees of connectivity.

5.3.5. WORKLOAD AND APPLICATIONS

To model the spatial- and quantitative-heterogeneity of the applications and users in a Fog computing infrastructure, the system is subjected to a workload that is composed of a set of applications and their respective demand, as defined in Section 5.1. In this work, 400 heterogeneous applications are hosted in the infrastructure. The applications' aggregate requested resource needs equal to a time-average of 50% of the system's resources. A system load of 50% is reasonable for this type of system, yet high enough to cause resource contention.

Three properties define an application; its demand, its performance requirements (SLA), and its resource usage profile. They are as defined below.

DEMAND SPREAD AND QUANTITY

The demand of an application is spread over a set of leaf nodes. In this work, the spread of an application's demand is categorised as either local, regional, or global. The demand spread of an application is linked to the branching factor of the network's DCs. Local demand is associated with small DCs, regional demand is associated with medium and large DCs, and global demand is associated with huge DCs. The demand spread types are uniformly distributed across the 400 applications. Furthermore, the quantity of demand is proportional to the capacity of the DC type, which is associated with their demand spread.

PERFORMANCE REQUIREMENTS

The performance requirements or SLA for an application is a real number upper limit of the 95th percentile of the network distance distribution of the number of hops from all users of an application to where the location of the DC in which the application is hosted. An application's SLA is associated with the type of DC that the application's demand spread is associated with. The SLA is sampled from a uniform distribution in the range from the minimum to the mean network distance between all leafs nodes to

Table 5.3.: Application SLA range as the maximum of the 95th percentile of the distance distribution

	SLA range
Local	$[1,2]$
Regional	$[2,3]$
Global	$[3,5]$

Table 5.4.: Application resource utilisation characteristic types with utilisation intensities.

	Compute	**Storage**	**I/O**
CPU Intensive	0.95	0.5	0.05
I/O Intensive	0.05	0.75	0.95
Symmetric	0.5	0.5	0.75

all DCs of the corresponding type. The ranges are specified in Table 5.3.

RESOURCE USAGE

An application's resource usage profile is categorised as either compute, storage, or I/O intensive. For example, a compute-intensive application is characterised as using relatively more compute resources than storage and I/O resources, in proportion to its total demand. The resource usage intensity classes used in this work are detailed in Table 5.4. In this work, the resource usage profile types are uniformly distributed over the 400 applications and assigned independently of the application's SLA and demand spread.

5.3.6. COMPARISON METHODS

In the experiments, the performance of the proposed algorithm is compared with an optimal, a random, and a naïve placement method. They are defined below.

RANDOM SELECTION AND PLACEMENT

This method utilises the agents of the presented algorithm, but the decision is applied randomly. To be more precise, if the budget is violated in DC v_i, one application out of $\mathcal{A}_i$ is uniformly randomly selected and migrated to a DC in $\mathcal{N}_i^k$ with a recorded budget surplus greater than 0, randomly selected from a uniform distribution. Similarly, if the SLA of an application $a_n \in \mathcal{A}_i$ is violated, it is migrated to a DC in $\mathcal{N}_i^k$ with a reported budget surplus greater than 0, randomly selected from a uniform distribution. From now on this method is referred to as the random method.

NAÏVE - MAXIMUM IMPROVEMENT WORST-FIT MITIGATION

This heuristic method utilises the change agents from the presented algorithm, but the decision is applied in a maximum improvement worst-fit approach. The reasoning here is to locally minimise the additional operational cost and load incurred by an application placement change. If the budget is violated in DC v_i, a set of applications are selected for expulsion, based on the cost they incur if they are migrated in relation to how much the application contributes to the aggregate operational cost of the hosting DC, as given by:

$$\overline{a_i} = \underset{a_n \in \mathcal{N}_i^k}{\arg\max} \frac{\hat{\vartheta}_i^{\text{op}}(t_0, t_0 + \Delta t_e)}{\hat{\vartheta}_i^{\text{mig}}} \tag{5.15}$$

The applications are then placed in the DC in $\mathcal{N}_i^k$ with the largest budget surplus. SLA violations are mitigated by placing the application in the $v_k \in \mathcal{N}_i^k$ where the mean distance to the demand is minimised, per:

$$\underset{i \in \mathcal{N}_q^k}{\text{minimize}} \qquad \rho_{i,n}^{95th} \tag{5.16}$$

$$\text{subject to} \qquad \vartheta_{n,i}^{\text{op}}(t) t_e + \vartheta_{n,i}^{\text{mig}} \qquad \leq \hat{\xi}_i(t) \tag{5.17}$$

The method is naïve in the sense that it acts locally without and independently of the system's other objectives.

CENTRALIZED OPTIMAL PLACEMENT

To provide an upper performance bound for the presented algorithm, the centralised optimal placement method from Chapter 4 is also included. All applications are placed where they incur the least amount of cost, meet their performance requirements, given that they do not aggregately exceed any individual DC's desired allocation level.

To increase the potential total utilisation level of the system given a highly heterogeneous workload this method does not have a global load balancing objective. A soft load balancing constraint would contradict the soft cost minimisation constraint. This is contrary to the presented distributed algorithm where a uniform load across a neighbourhood is actively pursued, as it is essential for the algorithm to permutate successfully to find a steady state iteratively.

5.3.7. EVALUATION METRICS

The algorithm is evaluated on its ability to meet the system's management objectives using the following metrics.

Total system cost The total momentary cost of all resources at time t, defined as:

$$\kappa(t) := \sum_{i \in \mathcal{V}} \sum_{n \in \mathcal{A}_i} (\vartheta_n^{\text{op}}(t) + \vartheta_n^{\text{mig}}(t)) + \mathcal{L} \tag{5.18}$$

The system's management objectives seek to minimise the total momentary cost, which means that a low value is desired.

Number of budget violations by any resource, at each point in time. A low number is desired.

Number of SLA violations by any application, at each point in time. A low number is desired.

Resource allocation distribution is defined as the standard deviation of the distribution of DC allocation levels across the infrastructure. The metric shows how well the load is balanced across the system. A low standard deviation is desired.

5.4. RESULTS

In this section, the results from the experiments detailed in Section 5.3 are presented and analysed. This section begins with observing the algorithm's convergence time to a steady state from a random state followed by their step responses and resource utilisation distributions.

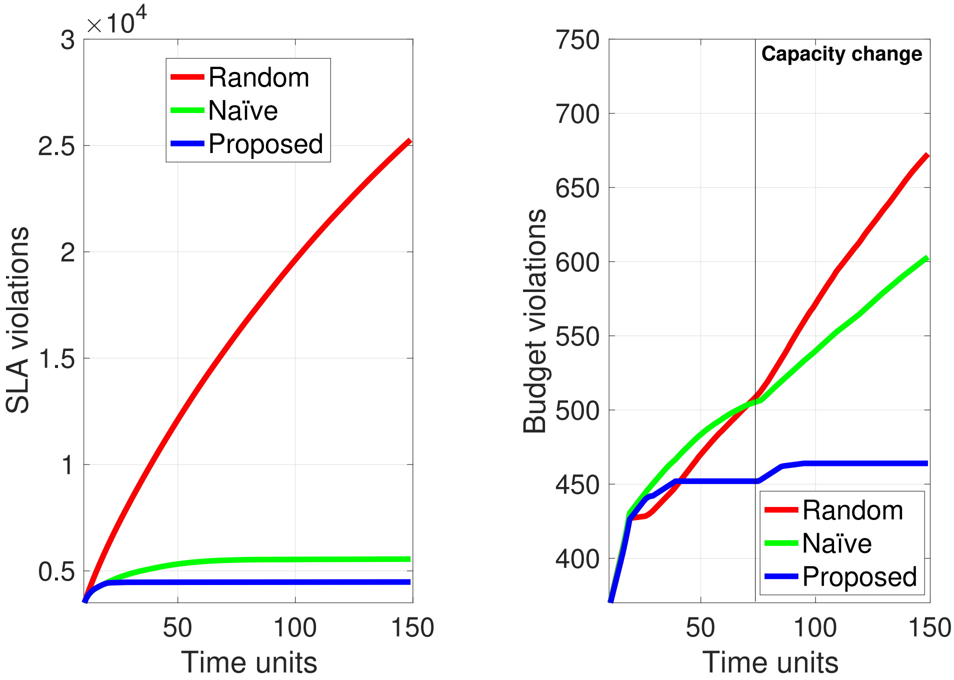
5.4.1. CONVERGENCE

From the onset, at time $t = 0$, all applications are placed uniformly random in the network with a 50% load of the system's DCs. Thus, on average, 65% of the applications violate their SLAs, and 50% of the DCs violate their budgets. Below, the convergence time of the algorithm is evaluated for both its agents' performance objectives, SLA and budget. The convergence time from a random state is representative of how quickly, if at all, an algorithm can reach a steady state.

Note that, when an agent evaluates the objective function the agent uses the last reported budget surplus values from its neighbours. Therefore, neither method begins to act until the first budget surpluses ξ_i are communicated, namely at the end of the first epoch $t = 10$. Furthermore, as the optimal method is already in a steady state, its convergence time is naturally not considered.

SLA

Starting with the traditional fat-tree topology, as illustrated by Figure 5.4a, the distributed algorithm can meet all resident applications' SLAs after 20 time steps. The naïve method does not do so until $t = 70$. This is due to the naïve method's competing actions, the budget violation and SLA processes. To this effect, up until this point, the naïve method has retarded 19% of the applications' SLA deficits while working towards meeting all DCs' budgets. In the random case, the SLA violation process does not converge within the time-frame of the experiment.



(a) Cumulated SLA violations.

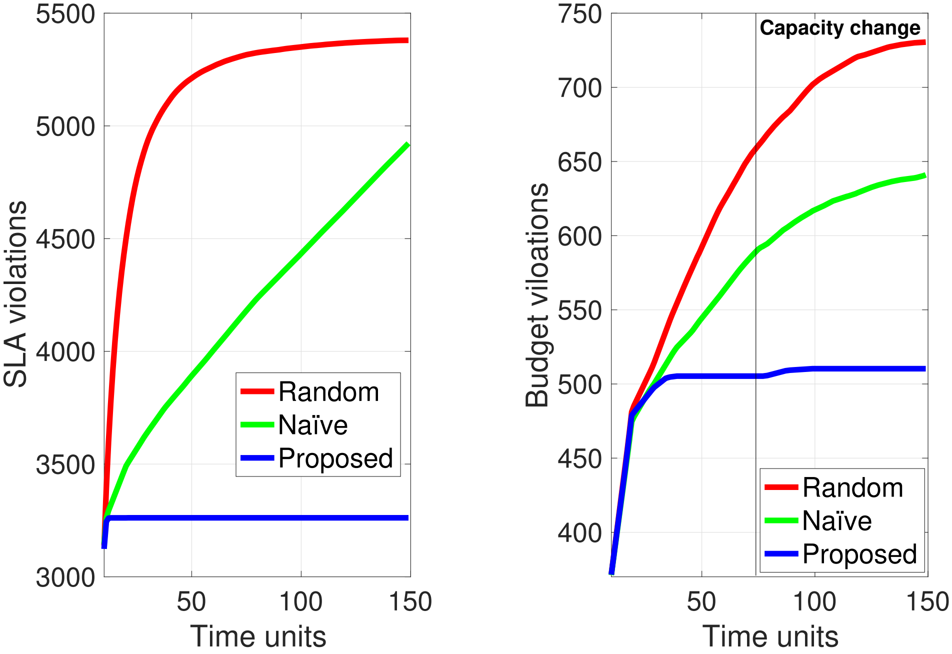
(b) Cumulated budget violations.

Figure 5.4.: Objective violations in a fat-tree topology.

The random graph scenario leads to a different outcome. Due to the higher degree of connectivity, the distributed algorithm's SLA process is now able to converge after 12 time steps, see Figure 5.5a. The naïve and random methods fail to converge because they can at this point no longer be propelled by the differential between the heterogeneous layers in the fat-tree topology. Instead, the naïve method permanently deposits 9% of the applications that violate their SLA's in DCs from which it cannot find more suitable hosts. Interesting to note is that the random method is well suited to handle this degree of heterogeneity. Although it does not converge until $t = 140$, the random method can reach a steady state.

BUDGET

The distributed algorithm's budget violation process converges after 40 time-steps when deployed in the fat-tree topology scenario, see Figure 5.4b. The naïve method converges at $t = 70$, at the expense of an additional 100 budget violations. When considering the SLA deficit/surplus of the applications, none of the applications with a small SLA surplus are migrated up until the point the SLA process converges at $t = 20$.



(a) Cumulated SLA violations.

(b) Cumulated budget violations.

Figure 5.5.: Objective violations in a random graph topology.

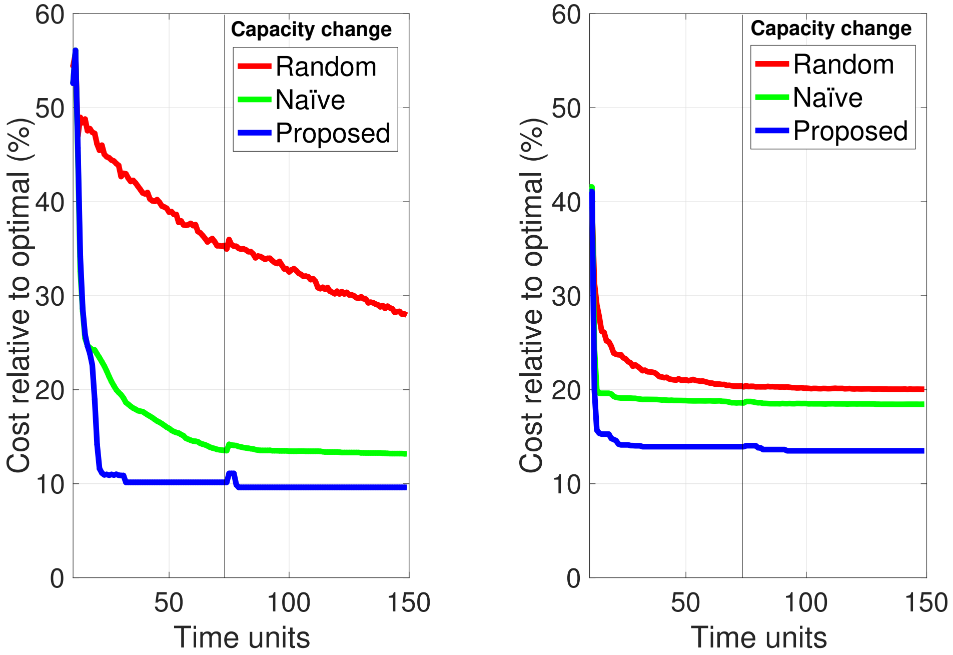
Again, the random method does not converge within the time frame of the experiment.

A similar outcome can be found in the random graph topology, see Figure 5.5b. Again, the algorithm is assisted by the higher degree of connectivity, and now converges at $t = 35$.

OPERATIONAL COST

Starting with the fat-tree topology; once converged, the distributed algorithm incurs a total system cost within 9% of the operational cost achieved by the optimal method, see Figure 5.6a. The method's ability to approach the optimal cost point is a reflection of the system load or budget. A smaller budget forces the methods to find a lower cost point but at the cost of the ability to permutate. Despite failing to meet all DCs' budgets, the naïve method incurred cost converges to 13% of the optimal. As with the previous scenarios, the random method fails to converge.

The outcome for the random graph topology is illustrated in Figure 5.6b. The total system cost achieved by the distributed algorithm when employed in the random graph topology converges to 13% within the cost incurred by the optimal approach. In this



(a) Fat-tree topology.

(b) Random graph topology.

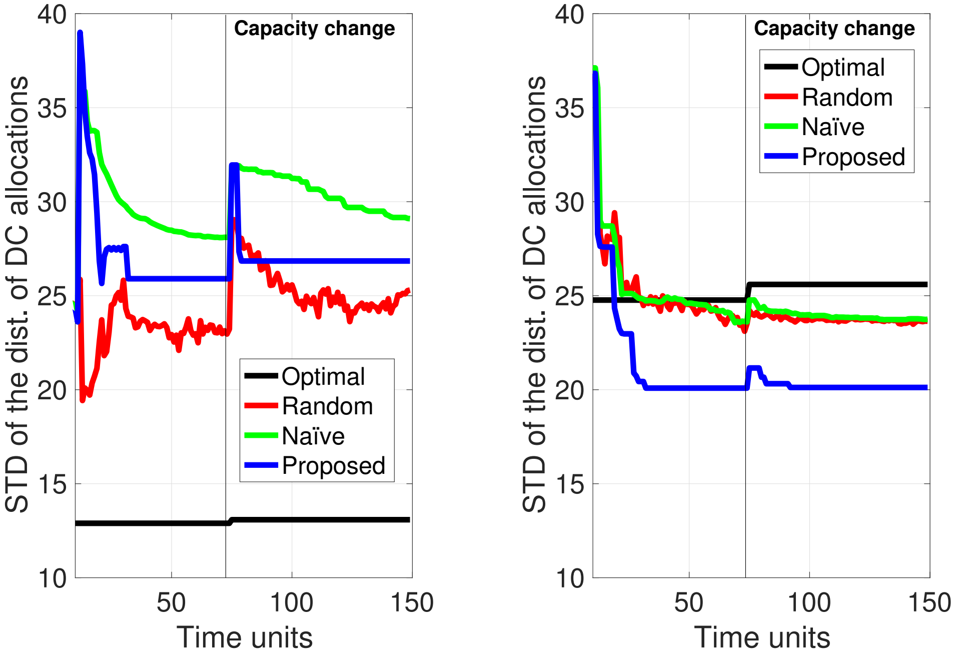
Figure 5.6.: Total operational cost relative to the cost incurred by the optimal approach.

case, both the naïve and random methods converge to an incurred system cost of 18% and 20% from the optimal, respectively.

5.4.2. STEP RESPONSE

Observing the algorithms' step responses reveal how well they can respond to changes from a steady state. To subject the system to a change, the capacity of a random medium-sized DC in the network is instantly halved at $t = 75$. The budget of that DC is reduced accordingly.

In the fat-tree topology, the distributed algorithm can spread the affected DC's excess demand to its neighbours, who then propagate any excess to their neighbours while attempting to balance the load throughout the system. Because the objective function considers the SLA deficit/surplus of the applications, only applications that would reduce the net load in the neighbourhood or improve its SLA deficit, are likely to be affected. The distributed algorithm's budget process thus reaches a new steady state after 7 time steps. The naïve method, on the other hand, fails to spread the excess load over DC 2's neighbours, and instead creates a bottleneck in the middle of the net-



(a) Fat-tree topology.

(b) Random graph topology.

Figure 5.7.: Standard deviation of the distribution of DC allocation levels across the system.

work. From this point on, the naïve and random methods' budget process diverges and therefore fails to load balance the system, which will inhibit the system to handle any forthcoming changes in load or capacity.

In the random topology, with a larger number of neighbours per DC, the distributed algorithm can converge to a new steady state with only 20% of the violations compared to the fat-tree topology. Due to the increased interconnectivity of the random graph, it can handle a significant change in capacity. Furthermore, the naïve and the random methods have still not converged at $t = 150$, with a significant number of both SLA and budget violations.

5.4.3. ALLOCATION DISTRIBUTION

Although resource allocation is not one of the system objectives, as explained earlier, it does provide an idea of the state to which the algorithms converge to. A more uniformly load balanced system is desirable as it leaves the system in a better state to accommodate changes uniformly in the network. If the change and quantity in changes followed a particular distribution, then perhaps a uniform load balance might not be what is sought after. Analogously, the optimal solution does not have this objective. A wide allocation distribution across the system's resources implies that specific resources are less able to permutate in the event of a change in capacity or demand. In these experiments the load distribution imposed on the system by the applications' demands is uniform; thus a narrow distribution is desired.

Figure 5.7a shows the standard deviation of the distribution of DC allocation levels across a fat-tree topology. The figure reveals that all the non-optimal methods achieve a very similar level of allocation distribution. Note that each algorithm converges to its previous level despite a significant reallocation of resources.

For the random graph topology, presented Figure 5.7b, all the non-optimal solutions achieve a lower allocation variance than the optimal solution. Additionally, in the random graph topology, all non-optimal methods converge significantly faster and are less disrupted by the change in the capacity at $t = 75$ than the fat-tree topology. This can be attributed to the greater variety of resources available to any given node in the random graph topology.

5.5. CONCLUSIONS

In this chapter, a distributed algorithm to holistically manage a large set of heterogeneous DCs and applications with different objectives was presented. The main challenge has been to reach a steady system state and while accommodating a set of entities with heterogeneous objectives hosted in a cost- and capacity-heterogeneous network. The distributed algorithm was evaluated over two different types of topologies with varying degrees of heterogeneity and compared to both a centralised optimal solution, and two naïve methods. The results reveal that the distributed algorithm presented in this chapter can quickly and consistently converge despite a high degree of heterogeneity in the system. The evaluations also reveal some of the properties in a heterogeneous topology that can be used to extend this work.

A possible investigative extension of this work is a thorough investigation of the distributed algorithm's convergence performance under a transient workload and resources with time-variant capacity and cost. Possible extensions to the algorithm include elastic horizontal scaling of applications and multi component applications.

Part II.

Smart cities & Internet of Things

6

Realising smart city services with IoT and FaaS

A long-held goal of many scientific projects has been real-time manipulation of sensing devices and actuators. For example, a system to monitor animals could be constructed with a lattice of mobile sensing devices, and the discovery of a particular animal could alert a scientist to dynamically reconfigure both sensor positions/techniques and gating/routing mechanisms to improve data collection or improve the animals' lives. However, in general, because of cost and technology constraints, such devices usually lack network capabilities and thus are unable to be read/controlled in real-time. Instead, scientists are typically forced to collect the data sometime after the fact manually. This data is then analysed on lab computers, perhaps leading to modifications to the sensing algorithms/positions and control logic used in subsequent re-deployments. These constraints have significantly limited the rate of scientific progress.

Two recent developments have brought such real-time science scenarios closer to reality. First, device manufacturers have begun to believe that there is a market for network-enabled devices in many scenarios (e.g., a "connected" light bulb for the home). Such cost reductions and broadly-available capabilities can be extended or applied to scientific devices, thereby significantly reducing the cost to deploy such devices. Second, public clouds have recently added specific support for IoT. For example, in October 2015, AWS introduced AWS IoT, to "support billions of devices and trillions of messages". While these two developments are promising, there are many open issues that must be addressed before large-scale scientific experiments based on real-time sensors and actuators are feasible. Long-lasting power supplies and the availability of networking infrastructure (e.g., cell towers) for devices/actuators must be addressed in general. Additionally, while early public cloud IoT success stories have focused on smaller-scale scenarios such as connected houses, it is unclear to what extent these new public cloud mechanisms and abstractions are suitable and effective for larger-scale or scientific scenarios, which often have a different set of constraints or requirements.

This chapter addresses the challenge of implementing a scalable IoT infrastructure test-bed in the public cloud for scientific experimentation. There are two main contributions to this chapter. The design and implementation of a representative cloud-based IoT infrastructure in AWS are presented and the evaluation of that system. The system created is for dynamic vehicle traffic control based on vehicle volumes/patterns and public transport punctuality. Specifically, in-road induction sensors and vehicle GPS positioning comprise the input to a control algorithm to regulate the red-green patterns of traffic lights with the goal of increasing safety and minimising wait/idle times. The targeted system operates in real-time and is both data-driven and stateful. In this first phase, the system has been designed and implemented based on a simulated system and sensors; as the system matures, these simulated sensors can be replaced seamlessly with real sensors that report to the cloud, without needing to modify any of the control logic. Three primary non-trivial challenges when developing such an application in AWS are presented:

1. Designing and defining scalable stateful data-driven IoT services that operate asynchronously with real-time constraints.

2. Long tail latency performance barriers.

3. Practically managing and extending the infrastructure components in a scalable manner.

This chapter is structured as follows. Section 6.1 discusses related work, and the suspends the research gap. Section 6.2 details the requirements and properties of the targeted system. Section 6.3 describes the infrastructure's components and presents the design. Section 6.4 presents the evaluation, and Section 6.5 concludes the chapter.

6.1. RESEARCH GAP

In this work, a smart city-like traffic control service is used to evaluate the feasibility of deploying a scientific test-bed in the cloud. Below a review of the state-of-the-art in the field of traffic control and how this thesis contributes to that field.

In [DDMM15], Djahel et al. present the requirements of a future traffic management system in the IoT era. These requirements include providing real-time road traffic simulation and visualisation to help authorities more efficiently manage the road infrastructure and ensuring the integration of existing systems and new technologies and managing the evolution of these systems. In [GGD$^+$07], Gradinescu et al. present an adaptive traffic light system based on wireless communication between vehicles and fixed controller nodes deployed at intersections. They prove that total time delay experienced at intersections can be significantly reduced using their system. In [HPL15], Hu et.al. propose an intelligent Transport Signal Priority logic based on connected cars. Transit Signal Priority (TSP), also referred to as bus priority, is a collection of techniques that provide preference to transit buses at intersections. By adjusting the traffic signal plan according to bus arrivals, the delay that transit buses experience at intersections is reduced. This system aims to improve the overall transit service quality.

The above findings have to various degrees been evaluated either in a simulator or small real-world systems. There is to the author's knowledge no work done on how such policies and systems interact with tangent systems in a Smart City or at scale. Additionally, neither the proposed systems nor the evaluations take much consideration to the supporting sensor and orchestration platforms that would be required in a real-world deployment.

Inspired by the past work in smart traffic management, this work looks at a cloud-based infrastructure to support such systems. In particular, the work examines the use of contemporary cloud services and platforms to scale experiments and create a hybrid simulated and real-world experiment environment.

This work aims to leverage AWS IoT services to build the cloud infrastructure necessary for such a forthcoming traffic management system. In this work, the effectiveness of using AWS IoT is evaluated. Additionally, the challenges faced during development and suggestions for improvements are presented.

6.2. TARGETED SYSTEM

In smart cities, a Traffic Signal Control (TSC) system incorporated into a general traffic and public transport subsystem will employ a wide range of sensor types with heterogeneous availability, data types and quantities, and outputs. The aggregate system state is represented by the state of the individual connected vehicles and devices as well as historical and real-time data external and internal to the system. Several parallel, event-driven, real-time, and periodic processes will orchestrate devices and vehicles, collate and aggregate data, and provide feedback control based on the system's objectives. The system's many objectives, such as actuating traffic lights to meet a specific deadline are in their nature, real-time. Accommodating a data-flow to achieve real-time decisions in a distributed system at scale is a challenge on its own. Ingress data to such a system and its various processes is arguably heterogeneous both in terms of volume, velocity, variety, and veracity.

A scientific IoT TSC test-bed will need to be able to orchestrate both real-world as well as virtual objects scalably and in real-time [ZDZ12]. A system state shall be able to be defined by any subset of the system's inputs and states. Processes, administrative actions, and system states shall be able to be triggered by devices and vehicles and by observing data flows. Data generated by the system and its constituent components, therefore, needs to be made available in real-time to those processes and states.

The scale of the set-up and the duration of the experiments vary from experiment to experiment and even during runtime. The system, therefore, needs to be able to scale to a large number of devices without affecting the real-timeliness of the control processes nor limit the number of concurrent scientific data analysis processes. It is also desirable for a scientific team not to have to commit to, develop, and maintain their infrastructure.

This work begins to address this challenge by evaluating and exploring the possibility of using an emerging cloud IoT PaaS, namely AWS IoT and AWS Lambda. To provide a platform for developing and evaluating such a system, the TSP solution proposed in [HPP14] is employed. In the remainder of this section, the system components are described and then the system properties are enumerated.

6.2.1. SYSTEM COMPONENTS

The Transit Signal Priority with Connected Vehicles (TSPCV) system presented in [HPP14] relies on connected public transit vehicles and wireless TSC sensors [YAKS10] to fulfil its objective of reducing mean commuter waiting times. The paper proposes to do so by manipulating the en-route traffic lights to allow the en-route buses to maintain their schedules while considering their current ridership and the impact on peripheral traffic in the affected intersections.

More specifically, the triggers in such a system are realised with a set of sensors in bus stops reporting the arrival of buses. In parallel, buses report their location, speed, and ridership. The state of the en-route traffic lights can be manipulated and observed in real-time. The punctuality of each bus is monitored by a process that compares the reported arrival of buses at bus stops and a set timetable. A change in the traffic light program will be considered when the bus approaches the intersection and is behind schedule. The evaluation process queries the punctuality process, the state of the affected traffic light, and the prevailing peripheral traffic conditions from the nearby induction loops, cameras, or collaborative, connected vehicles [TRL^{+}09]. The resulting control decision is then relayed to the affected traffic light(s). The consequences of that decision will be picked up by separate processes that continually produce, analyse, store, and monitor prevailing traffic conditions.

The following components are needed to construct such a system (see Figure 6.1):

Sensors and actuators The system will collect data from a large set of sensors. The sensors produce and attempt to report data at a specific rate. The sensors are geographically distributed throughout the evaluation domain. Due to intermittent connectivity, sensors might not be able to report at the desired rate successfully. Additionally, sensors report data with an error. The traffic lights are the system's actuators. A traffic light can be queried for its last reported state, and the state can be changed with a control signal.

Device orchestration The devices, sensors and actuators are orchestrated in such a manner that enables them to register, validate, and securely communicate with the system.

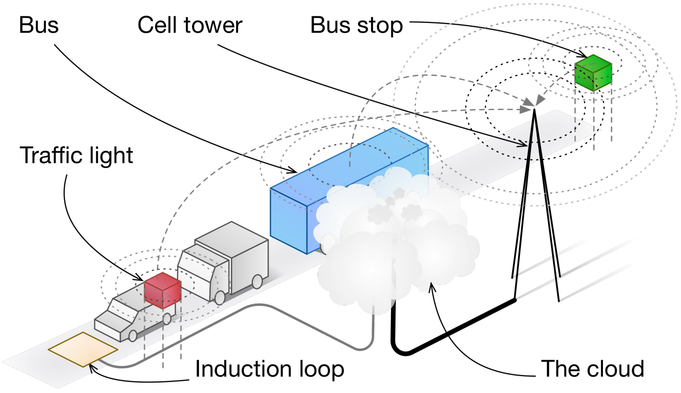


Figure 6.1.: Targeted system and application scenario

Scientific analysis To determine the effectiveness of a strategy, an analysis is performed on the data. The set of running analyses can be changed in runtime to reflect the ongoing experiments. The intent and output of analysis can either be preprocessing for the controller or scientific evaluations of the system.

Storage The data collected by the devices, the control decisions, the current state of the systems processes and entities, and the analysed data is recorded and stored indefinitely for concurrent and future analysis.

Controller The system operates a set of controllers that act on multiple inputs from both real-time, stored, and preprocessed sensor data to each produce a set of control signals that are then relayed to their constituent traffic lights.

Monitoring The performance and availability of the system's components and devices are monitored in runtime.

The components of the targeted system are illustrated in Figure 6.2.

6.2.2. SYSTEM PROPERTIES

The targeted scientific IoT test-bed has the following properties and requirements.

Real-time The system shall be able to forward, process and store information as well as run control loops in real time.

Multiple inputs The system shall support MIMO by exposing the entire set of sensors to the set of controllers.

Scalable To cover large geographic and densely populated areas, the systems need to scale from 10's to 1000's of devices with a comparable number of controllers.

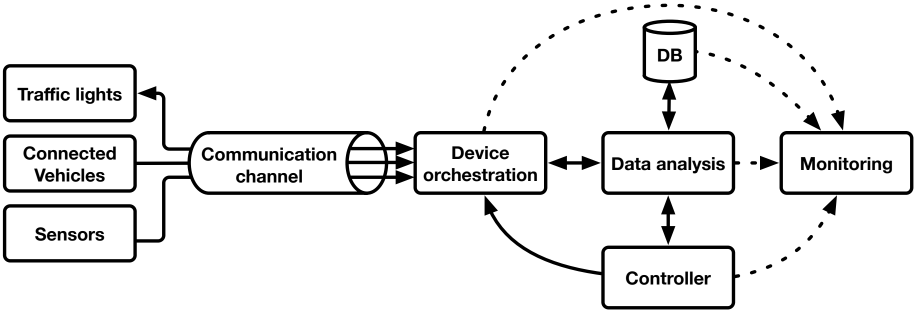


Figure 6.2.: System components of the targeted system

Survey-able The system and its component's states shall be made available to the operator of such a system

No-Operations (NoOps) The system shall not require active provisioning, software maintenance, nor extensive software development

6.3. IMPLEMENTATION

In this section, the implementation of the targeted system using AWS components is presented. AWS was chosen because it has at this point the most comprehensive portfolio to construct a scalable, extensible, and NoOps cloud IoT infrastructure. Additionally, the AWS service offering allows the architecture to bridge real and simulated paradigms by interacting with both virtual and physical devices. The relevant individual AWS components are first described, followed by how they were used to construct the real-time vehicle control system.

6.3.1. AWS COMPONENTS

To realised the components and requirements of the targeted system, the following AWS are employed.

IOT

AWS IoT is a platform that enables connected devices to securely communicate and relay information to and from the AWS platform using MQTT [BG14]. Alongside ZigBee [A+06], MQTT has become one of the prevailing home-IoT messaging protocols. In addition to message passing, AWS IoT also offers data stream endpoint connectivity and message routing from a simple state-less rule engine with an Structured Query Language (SQL)-like syntax. Simple system logic can be achieved with rules, using stateless thresholds and trigonometric functions.

To connect a physical infrastructure of devices to the AWS IoT platform, the accompanying AWS IoT Software Development Kit (SDK) supports a number of embedded systems such as Embedded C and Arduino Yún. Additionally, AWS IoT supports RESTful communication to ensure that virtually any device of any capability can be connected to the system, as long as it is connected to the Internet.

An entity in AWS IoT is referred to as a *Thing*. AWS IoT maintains the state of each *Thing*, through what is referred to as a *Shadow state*. A *Shadow state* can be queried through services external and internal to AWS IoT. The relationship is maintained regardless whether the physical device is connected or not. The targeted system will require a stateful logic that goes beyond the capabilities of the AWS IoT. Therefore, AWS IoT is in this work used to scalably and securely orchestrate the resident devices' communication and authentication, in real-time. All external entities are connected to the cloud-hosted infrastructure over AWS IoT.

LAMBDA

In addition to its traditional collection of Virtual Machines, AWS is offering a highly scalable server-less micro-computing platform called Lambda. A Lambda function is a state-less piece of code, with an input and an output that can be triggered by a wide array of sources internal and external to AWS. In contrast to an Elastic Compute Cloud (EC2) instance, a Lambda function has one dedicated purpose and deliberately only runs for up to a few minutes. Arguably, instead of running an entire application in a single VM, it can now be broken up into a set of redundant asynchronous sub-functions. Lambda functions scale instantly to hundreds of instances, with almost no platform maintenance. In this work, Lambda functions constitute all computational instances used for evaluating bus punctuality, aggregating and maintaining data in databases, sensors fusion, and traffic control loops.

DYNAMODB

AWS offer a number of DB services. AWS DynamoDB is a low-latency No-SQL schema-based DB. In this work AWS DynamoDB is used for storing collected data and maintaining shard states.

KINESIS

AWS Kinesis is a highly scalable aggregating streaming data buffer. Kinesis is scalable in the sense that it can achieve high throughput by forwarding ingress data to a practically infinite pool of parallel end-points. Its ability to maintain high throughput and thus ensure that what is beyond the endpoints is updated promptly is what makes it real-time. This property also ensures that additional end-points can be introduced non-intrusively, without interrupting or throttling the existing end-points. In this work

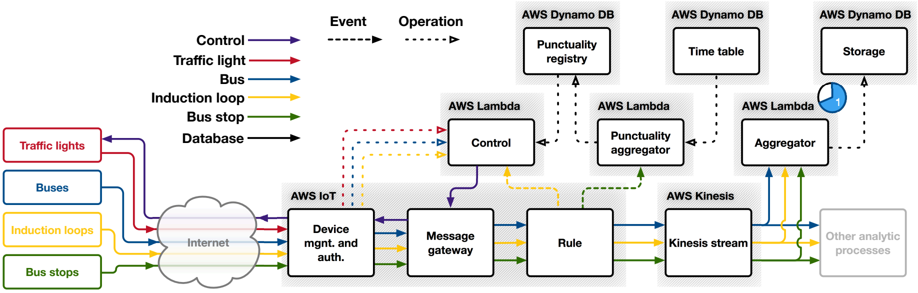


Figure 6.3.: AWS components and data flow for the test-bed

AWS, Kinesis is used to scalably aggregate and process the flow of reported sensor values. In practice, the Kinesis stream is also used as a means to expose the ingress stream of data to any future end-point or experiment.

CLOUDWATCH

AWS CloudWatch is a platform for monitoring AWS services through logs as well as predefined and custom metrics. AWS CloudWatch is the primary outlet for debugging AWS services in run-time. Additionally, AWS CloudWatch provides a set of primitive plotting capabilities for monitoring existing and user-defined metrics. Cloud watch also allows the setting of time-based triggers.

6.3.2. TESTBED ARCHITECTURE

The system's various entities such as traffic lights, buses, and induction loops are connected to the cloud infrastructure as *Things* via AWS IoT (see Figure 6.3). The communication between the sensors and the cloud infrastructure is encrypted using Transport Layer Security (TLS). The credentials for each device are stored locally on the device. This implies that each device is registered and trusted by the system. This satisfies the security and reliability requirements. Additionally, entities, or *Things*, can be added dynamically to the system in runtime.

Data from all traffic and bus monitoring sensors are published to a shared data channel for analysis in real time. A rule is deployed as an end-point to each upstream channel to route the message to a AWS Kinesis stream. The received messages are in their entirety deposited into the Kinesis stream. From this point on, any 3rd service can be set-up to access the incoming streaming data without altering the set-up of running AWS IoT flow provided that they have been granted access. In other words, new data stream endpoints can unobtrusively and trivially be added. Furthermore, a channel in AWS IoT is achieved by an MQTT topic.

Both AWS IoT and Kinesis have a high enough throughput to receive and route hundreds of sensor readings per second successfully. The number of concurrent Lambda functions automatically scales to meet the number of ingress sensor values. In theory, this ensures that the information flow expedited end-to-end in near real-time.

To aggregate and demultiplex the streaming data, AWS Lambda functions are attached as end-points to the streams. A Lambda function is employed to introduce a simple mean value calculation of traffic throughput, that observers the traffic flow data and aggregate the samples from each sensor over an epoch, arguably at the scale of the duration between control loop evaluations. The values are aggregated in an AWS DynamoDB entry for each epoch and are used in the analysis and classification of long-term behaviours. Moreover, at the arrival of a bus at a bus stop, an event message is sent on the `bus_stop_event` MQTT topic containing the bus stop id, the bus number and route, and the time of its arrival. The message is intercepted by a nAWS IoT rule which internally spawns a Lambda function that assesses the punctuality of that bus by comparing the ingress data with a timetable in DynamoDB. The result is stored in a DynamoDB table. Subsequent Lambda functions can be added to evaluate the state of the entire route or the transit system as a whole. The benefit of a Lambda function as compared to an EC2 instance is made clear in this scenario. The evaluation of a bus's punctuality is event-driven and does not happen continuously, but multiple evaluations might run concurrently for multiple buses. An equivalent EC2 instance would have to be run continuously, maintained, and the resident software would have to be able to scale to multiple evaluations while guaranteeing real-timeliness on one machine and one socket. Lambda functions and Kinesis thus contribute to the system meeting its scalability, real-timeliness, and NoOps requirements.

At this point in the system, data is accumulated in a set of DynamoDB tables that are structured in such a way that they are suitable for both long-term storage and the controller. There are several ways to access the data from both AWS-internal and -external services. Access privileges can be established to regulate who and what services can access the DB.

When a bus approaches an intersection, the adjacent induction loops register the presence of a bus and report the intersection id, bus number, and the time of the event to a dedicated AWS IoT topic, `intersection_event`. The rule performs two operations – the data is forwarded to Kinesis for record keeping and further analysis. Primarily, the rule triggers a Lambda function intended to evaluate the state of the traffic lights in the affected intersection. The resulting Lambda function begins by querying the punctuality DB entry for that specific bus to determine the extent of the need to assist the bus by altering the traffic pattern. If so, then the AWS IoT *Thing* shadow states of the induction loops in the affected intersection are queried for their last reported state. In addition to forwarding the relevant data to the control loop, the state update is intercepted by a AWS IoT rule that forwards the data aggregation and processing to the Kinesis stream. The control output is acted on by sending the new states of the affected traffic lights; this is accomplished by setting the Shadow State. The process is repeated for each such incident. Multiple such processes can run concurrently. Any interactions with AWS services are done through Python Boto3.

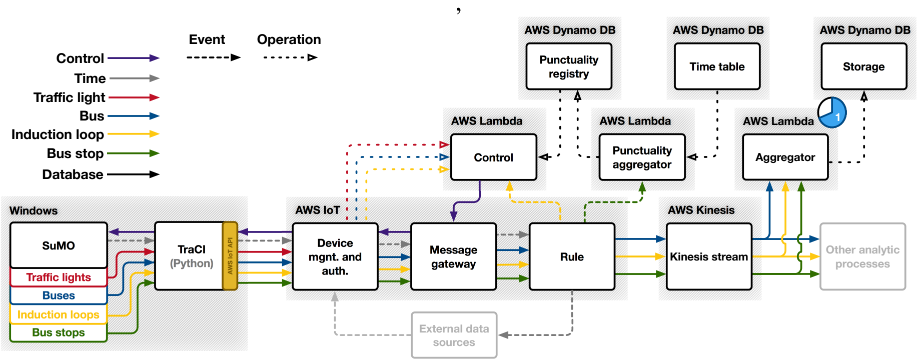


Figure 6.4.: AWS components and data flow with simulated/emulated endpoints

If the controller was designed to be triggered on multiple events, an intermediate state would have to be created in for example Dynamo DB. Although, Lambda function can be triggered from a variety of data driver sources, such as an update in a table, and another Lambda function via an intermediary AWS Simple Notification Service (SNS) message, it cannot be atomically triggered based on a state composed of multiple inputs. This is because data is submitted and processed asynchronously in the system, there is no one active entity in the information flow that can make the atomic decision to iterate the control loop. Founded in the fundamental concept of the Kinesis and Lambda, AWS cannot be ensured that a Kinesis entry will only be processed once. Multiple entries can be due to an error in the input or an error in the Lambda code. If AWS were able to guarantee that a table entry was atomic, then multiple controller Lambda functions that each trigger on an update for a subset of reported sensors over the past epoch could be deployed. As a result, the controller is therefore activated asynchronously and independent of the data source using a CloudWatch Scheduled Event. This ensures that only one instance of the controller Lambda function is called, and allows the controller to act independently of the data source.

6.3.3. SIMULATED TESTBED ARCHITECTURE

To validate the infrastructure and to provide a portable proof-of-concept, a simulated environment was developed based on the scenario presented in Section 6.2. Evaluating and experimenting with the infrastructure with actual sensors and real traffic at a real-world rate is neither safe nor allows the stressing of the limitations of the infrastructure. The simulated environment, therefore, employs a simulated traffic environment with virtual entities. The primary architectural discrepancies between the real-world test-bed and the simulated environment is illustrated in Figures 6.3 and 6.4. The information flow and control logic in the simulator are maintained with a few additional states and signals.

Simulation of Urban MObility (SUMO) [BBEK11] supplies the simulated traffic environment. SUMO is an inter- and multi-modal, space-continuous, and time-discrete traffic flow simulation platform. A SUMO simulation scenario is at the minimum specified by a network of roads, traffic-monitoring induction loops, traffic lights, and vehicle arrival rates, speeds, and entry points. SUMO is accompanied by a visualisation tool that renders the environment and the changes within it. The tool also allows real-time interaction with the simulation.

SUMO provides an extensible interface, Traffic Control Interface (TraCI), that allows researches and developers to interact with the simulator, environment, and the visualisation tool over a socket in real-time. This decoupling enables external entities to control the simulator's clock, traffic lights, and extract the prevailing traffic conditions. In this work the TraCI module is used to expose the induction loops, buses, cars, bus-stops, and traffic lights to the information flow as *Things* in AWS IoT. This is achieved by running the SUMO simulator core and an MQTT enabled TraCI python script on a Windows AWS EC2 instance. Because Boto3 cannot act as an AWS IoT *Thing* a virtual *Thing* module was developed to connect SUMO entities with AWS IoT. As a result, SUMO entities are managed by AWS IoT, have a shadow state, and can be queried in the same manner as their real-world counterparts.

SUMO can run in real time with real-world input, but to be able to scale any experiments the simulator clock needs to be centrally controlled. In this implementation, the simulator clock has therefore been made to drive time progression in the entire system. At each instance that the simulator progresses the time horizon, it sends a time-stamp on a dedicated AWS IoT MQTT topic, `time_tick`. To realise a pseudo-real-time environment, all external data-producing entities subscribe to the `time_tick` topic. At each time update, any data producing entity updates its shadow state. This structure enables the system to execute at almost any rate. Being able to vary both the input rate and the rate of execution allows one to scale the experiments to load the information flow in both time and volume.

6.4. EVALUATION

In this section, the designed architecture is evaluated using a representative scenario based on the proposed TSP-CV from [HPP14] detailed in Section 6.2. Based on this set-up, a basic cost and a latency analysis are presented.

6.4.1. REPRESENTATIVE SCENARIO

To evaluate the designed architecture, a representative scenario was developed in the simulated environment. The scenario is based on an area of roads and intersections in Charlottesville, VA. The area incorporates two intersecting bus routes, namely the Free Trolley and Route 7. The buses follow their regular schedule and are along with all the entireties connected to AWS IoT. Furthermore, the area features four intersections; each intersection implements the TSP-CV policy proposed [HPP14]. The integrated

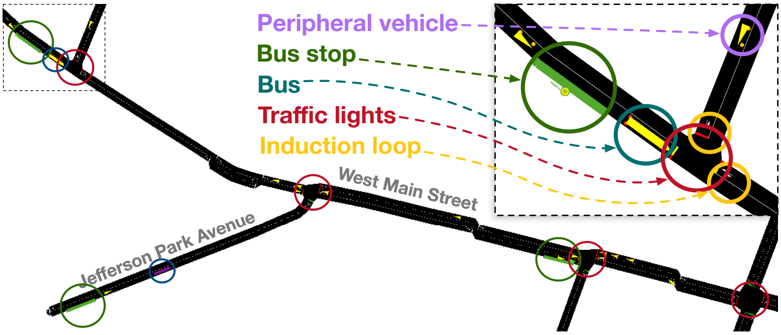


Figure 6.5.: Show case TSP-CV scenario in Charlottesville Virginia.

components and the layout of the scenario are illustrated in Figure 6.5.

6.4.2. PERFORMANCE

THROUGHPUT AND SCALABILITY

AWS IoT provides an interface to add and manage *Things* (sensors) deployed in the test-bed. The developed infrastructure can support thousands of sensors, as AWS can scale on demand. However, the credentials required for sensors to communicate with AWS IoT have to be stored locally in each sensor. This means that overhead is incurred each time a sensor is added to the system. Further, when the number of sensors in the system is very high, managing the sensors via the interface provided by AWS is tedious and error-prone. Thus, the infrastructure can scale to support thousands of sensors, but manually managing the sensors via the AWS IoT interface is challenging.

LATENCY

As the control loop is decoupled from the simulator, it operates asynchronously with the simulation clock. The control loop is the most latency-sensitive component in the system and determines the maximum rate of the system. Although AWS claims to offer their services in real time, running the control loop in a separate entity subjects it to the AWS-internal latencies.

To find the upper bound for the simulation rate, the individual latencies for the operations involved in the control loop, as specified in Section 6.3, were measured. Figure 6.6 presents the distribution of latencies for AWS IoT Shadow State get, and DynamoDB reads from a Lambda function using Boto3. The bimodal distributions for each operation are attributed to the first such operation each time an instance of a control Lambda function is instantiated. This phenomenon is independent of the way the Boto3 instance is initialised and reused.

Figure 6.6 also reveals the total round trip time for the control action, measured

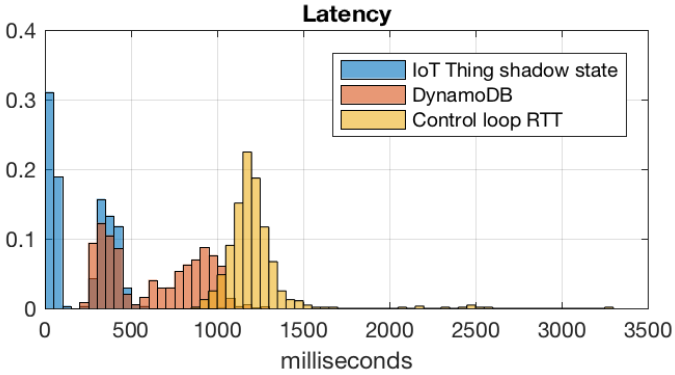


Figure 6.6.: Latency distributions per AWS operation.

from the time an event is sent on `bus_stop_event` until the affected traffic light's Shadow State is updated on the virtual entity. This finding suggests that it will take on average 1.2103 seconds for the control action to take effect. This implies that either the simulation or the real world will need to accommodate 1.2-second delay from the time a bus is detected until the first instance a traffic light can be changed. Moreover, the system, at some point, will need to be aware and compensate for this additive stochastic delay. Furthermore, running with real devices would subject the system to an additional WAN latency, in the range of 30 to 150 ms, depending on your physical proximity to a AWS DC.

COST

The cost of deployment is an essential factor in determining the effectiveness of a cloud service provider in terms of deploying a large-scale scientific experiment. The cost of the developed infrastructure for varying data sampling rate and a varying number of deployed sensors is depicted in Figure 6.7. It is assumed that the number of shards used is 1 and the data sampling rate is the data collected per minute from each deployed sensor.

6.5. CONCLUSIONS

In this work, the emerging IoT support in public clouds was investigated and evaluated for scientific experimentation. The system created provided dynamic vehicle traffic control based on vehicle volumes/patterns and weather conditions. It was found that while AWS IoT performance and performance scalability often do not meet the requirements of many next-generation scientific IoT use-cases. Additionally, manageability/modifications of a scientific IoT scenario can be challenging for moderate- to

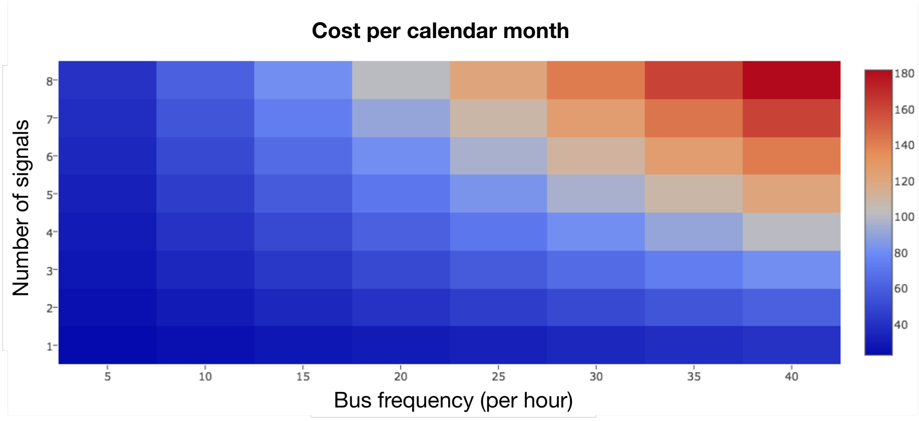


Figure 6.7.: Cost per month for operating one road intersection in AWS.

large-scale deployments. We are currently investigating techniques to replace the simulated sensors and actuators with their corresponding real devices. Also, new control algorithms are being pursued to control traffic efficiently. Such application-specific developments will synergise with any new IoT support from public clouds.

7

Bounding shared state inconsistency in distributed IoT systems

As IoT matures and enters new fields of operation, the notion of independent wireless sensors and actuators is becoming obsolete. Contemporary IoT applications are more focused on data generation and distributed decision making than merely connecting mundane household things for the sake of programmability and remote actuation. Forthcoming IoT applications and services will instead operate across multiple devices to, for example, accomplish a one-time task, operate a continuous process, or collect data. These applications will be composed and provisioned dynamically at runtime in a distributed manner to meet real-time demands. Additionally, IoT devices and their capabilities are discovered and managed in a manner reminiscent of micro-services from the Fog computing domain, see Figure 7.1.

Emerging platforms such as Ericsson's Calvin [PA15] will proposedly enable and orchestrate these types of IoT applications and systems of IoT devices. To do so, these platforms rely on a shared distributed state realised using a Distributed Hash Table (DHT) [PP12]. The DHT contains the capabilities and state of the system's devices and hosted applications. The information shared through the DHT allow devices to autonomously broker resources. This approach removes the need for centralised control and introduces some degree of platform fault tolerance. This also allows services and server-less functions to migrate between IoT devices and Fog computing resources seamlessly. Nevertheless, all autonomous distributed management decisions in such a system rely on a consistent DHT. Although a DHT is inherently fault-tolerant, fine-grained, and mission-critical decisions require near complete knowledge of the state's consistency. The expected consistency of the DHT is thus linked to the operational efficiency of the system and its applications.

An IoT device, being it a thermometer or a multi-purpose robot, quintessentially connects to the world beyond, wirelessly. Devices that have sparse and intermittent power supplies or are entirely self-reliant must be frugal with how they use the wire-

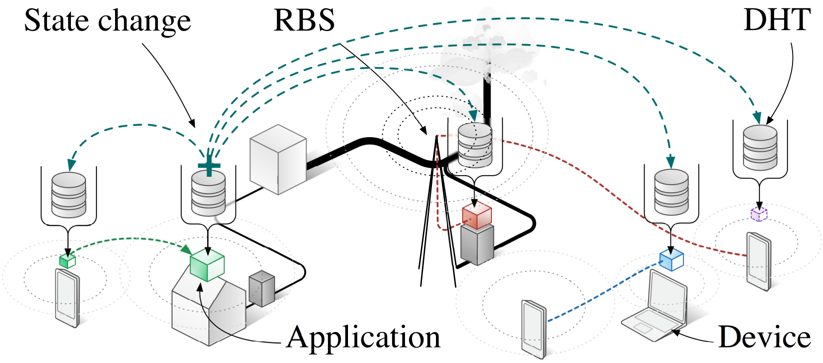


Figure 7.1.: Forthcoming IoT application paradigm.

less medium. Emerging 5G concepts are embracing these challenges faced by IoT devices [PDG$^+$16]. Proposed 5G standards consider device limitations and intermittency, multiple medium access methods [SJ16], Device-to-Device (D2D) communication, and new radio-access technologies [LETM14]. The propositions of 5G are more about IoT and enabling mission-critical applications than higher throughput.

In IoT devices, as applications and internal processes serve requests, they generate both egress application data and updates bound for the DHTs. From now on, the former is referred to as *application traffic* and the latter as *state traffic*. Although significant to the consistency of the system as a whole, the state traffic is not as latency-sensitive as the application traffic.

Application traffic and state traffic contend for the device's wireless resources. The amount of state traffic versus application traffic produced by a service request is stochastic, as is the rate of service requests. Consequently, the state traffic will intrude on the application traffic. The trade-off between application traffic and state traffic is non-trivial. For example, naïvely prioritising the application traffic or capping the state traffic can disrupt the state traffic entirely. Doing so will ensure neither a favourable trade-off between the two traffic flows nor that the system will be stable. As the traffic and the wireless link connecting the device is stochastic and vary significantly over time, no absolute bounds on either traffic flow can be trivially set that accommodates both traffic flows. It is evident that maintaining a consistent DHT comes at a significant performance cost for the applications that the system is intended for. Therefore some level of DHT inconsistency needs to be tolerated. However, uncertainty in the DHT's inconsistency has a detrimental effect on the quality of the system's distributed decision-making.

In this work, a Cross-Layer Control (CLC) is presented that bounds DHT inconsistency uncertainty by bounding the time-average amount of deferred state traffic by using a virtual queue technique [GNT$^+$06]. The proposed method is expressed as a jointly-formulated flow control and scheduling decisions derived using stochastic Lyapunov drift optimisation techniques found in [GNT$^+$06, NML08]. The proposed controller is thus able to accommodate the stochastic nature of the two traffic flows and ensure a bounded amount of deferred state information while maintaining queue and system stability. More precisely, the proposed controller schedules the two traffic flows so that the expected deferred state traffic is bounded at a predefined level without explicitly violating the application traffic. Intuitively, the deferred state traffic represents the amount of state information that would be lost if the device fails or the level inconsistency of the distributed state. This level can be dynamically adjusted to reflect each devices' probability of failure, or a system-wide tolerated level of distributed state inconsistency. The resulting flow control decision is implemented and operated independently in each device while the scheduling decision is implemented centrally in the wireless infrastructure.

The authors of [LMS12] stabilise a network with finite queues. This is not adequate for the problem presented in this chapter as it needs to accommodate a variable target consistency while adequately accommodating the application traffic. In doing so, one must be able to violate the target consistency to maintain a high throughput temporarily. Lyapunov drift analysis has been used extensively when analysing the performance and stability of mesh networks, such as [RYS10]. Although related concerning the characteristics of the devices, these works do not share the same objectives nor do they consider the consistency of a shared data source, such as a DHT. The work in [FLJ13] taxonomies methods for achieving consistent DHT-base routing algorithms in mesh networks. Although such methods are related, they cannot be applied at the scale and rate of change of IoT systems. Additionally, in [DTT$^+$16] a similar trade-off is explored but without explicit consistency objectives using Model Predictive Control (MPC).

Through simulation, it is shown that the proposed controller can bound the time-average shared state inconsistency within a narrow margin of the desired level while keeping the system stable. It is also shown that the controller is also able to do so while accommodating changes in load and wireless channel conditions. Additionally, the proposed controller is also able to more successfully balance the system's two traffic flows than comparable and conventionally used methods.

7.1. SYSTEM MODEL

In this section, the model is presented for a device and the system's infrastructure. As depicted in Figure 7.1 the target system consists of N devices and an RBS. To accommodate the traffic flows' individual objectives, each device operates two egress traffic queues, one for each traffic class; application traffic Q_i^a and state traffic Q_i^s. The state queue has a variable time-average limit $l_i(k)$. The limit $l_i(k)$, constitutes the bound on the deferred amount of state data. Both egress queues share the same sink, the wireless interface. All devices communicate over the wireless interface provided by

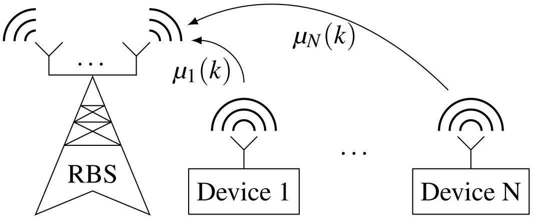


Figure 7.2.: Infrastructure model.

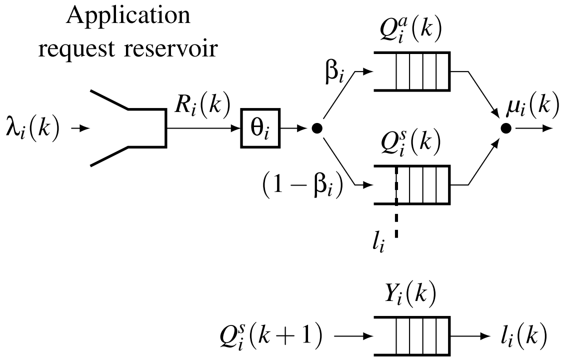


Figure 7.3.: Device model with virtual queue $Y_i(t)$.

the RBS. Each device has a time varying channel capacity of $\mu_i(k)$ in the time interval $[k, k+1]$. $\mu_i(k)$ has a probability distribution. Access to the medium is scheduled by the RBS at each time instance. For each device i, the capacity $\mu_i(k)$ can be shared between the application and state queues. Let $\mu_i^a(k)$ and $\mu_i^s(k)$ be the capacity allocated to the application and state queues for device i at time period k, respectively and $\mu_i(k) = \mu_i^a(k) + \mu_i^s(k)$. The targeted system as a whole is depicted in Figure 7.2. An overview of the device model is illustrated in Figure 7.3.

Each device is subject to ingress application requests at the rate of $\lambda_i(k)$. The application requests are deposited in a FIFO reservoir and processed at a controllable rate $R_i(k)$. As a request leaves the reservoir, the device commits to processing the request, which subsequently results in a change to the DHT. Specifically, $R_i(k)$ is the number of requests committed at time k, and θ_i is the mean of the size of each request. β_i is the fraction of $R_i(k)$ that is application traffic while $(1-\beta_i)$ of $R_i(k)$ is state traffic $\beta_i, \theta_i, \lambda_i(k)$ are constant for a device, but can be different among the devices, and are unique for each application i.

7.2. CROSS-LAYER CONTROLLER

In this section, the premise of the target problem is formalised and the proposed Cross-Layer Controller is presented. From here on, the cross-layer controller is simply referred to as the *controller*. The controller is formulated using the Lyapunov drift-plus-penalty Theorem presented in [GNT$^+$06]. In the resulting formulation, the controller has two control decisions; flow control and user (IoT device) scheduling. The two decisions are jointly formulated but act independently.

Contrary to the predictive approach in [DTT$^+$16], this approach acts only on the system's current state in each time instance. This allows the controller to accommodate the stochastic nature of the system's queuing dynamics. Also, the proposed controller can handle stochastic wireless channel conditions. The time-average horizon objective is achieved by acting on the relative changes in the system's queue sizes, i.e., the Lyapunov drift.

7.2.1. OBJECTIVE

The objective of the controller is to maximise the utility of the application queue while ensuring its stability, i.e., $E[Q_i^a(k)] < \infty$ and bound the time-average occupancy of the state queue, i.e., $E[Q_i^s(k)] \leq l_i$. In a practical sense, the utility achieved by an IoT device is defined as the number of requests (average $R_i(k)$) that is to be handled by that IoT device. The achievable utility depends on the designed flow controller, scheduling decisions, and the network capacity. A strictly increasing utility function is defined for for the application queue i, $\forall i$ as a function of output rate of the application request reservoir $R_i(k)$, as follows: let r_i be the time-average utility of the application queue i,

$$r_i = \lim_{k \to \infty} \frac{1}{k} \sum_{\tau=0}^{k-1} E[\beta_i R_i(\tau)] \tag{7.1}$$

The objective of the controller is formally expressed as,

$$\begin{aligned} \max \quad & \sum_{i=1}^{N} g(r_i) & & (7.2) \\ s.t. \quad & E[Q_i^s(k)] \leq l_i & \forall i \qquad & (7.3) \\ & E[Q_i^a(k)] < \infty & \forall i \qquad & (7.4) \end{aligned}$$

Note that imposing the constraints in Equation (7.3) make the solution of the optimisation problem complicated and the problem is a complex Markov Decision Process (MDP). It is well-known that MDP suffers from the curse of dimensionality since the queue state vectors grow geometrically with the number of IoT devices. Therefore, it is not feasible to find a suitable algorithm in practice. In this paper, instead of finding an optimal solution to the problem Equations (7.2) to (7.4), in this chapter a much simpler sub-optimal controller which is based on Lyapunov drift technique [GNT$^+$06] is developed.

7.2.2. QUEUING DYNAMICS

Firstly, the system's queue dynamics is defined. The application and state queue's dynamics of IoT device i are as follows,

$$Q_i^a(k+1) = \max\{Q_i^a(k) - \mu_i^a(k), 0\} + R_i(k)\theta_i\beta_i$$
$$Q_i^s(k+1) = \max\{Q_i^s(k) - \mu_i^s(k), 0\} + R_i(k)\theta_i(1-\beta_i)$$

To satisfy the constraint in 7.3 a virtual queue (see Figure 7.2) is introduced with the following dynamics,

$$\begin{aligned} Y_i(k+1) &= \max\{Y_i(k) - l_i(k), 0\} + Q_i^s(k+1) \\ &= \max\{Y_i(k) - l_i(k), 0\} + \max\{Q_i^s(k) - \mu_i^s(k), 0\} \\ &\quad + R_i(k)\theta_i(1-\beta_i) \end{aligned} \tag{7.5}$$

The virtual queue $Y_i(k)$ for application i in Equation (7.5) accumulates that application's number of deferred state data in $Q_i^s(k)$ and deducts its bound on the deferred number of state data $l_i(k)$. Consequently, if the virtual queue $Y_i(k)$ is stabilised then the constraint in Equation (7.3) will be satisfied. This can be achieved by using Lyapunov Optimisation Drift, as shown next.

7.2.3. LYAPUNOV DRIFT

In this paper, a quadratic Lyapunov function is used and expressed as,

$$L(k) = \frac{1}{2}\sum_{i=1}^{N} Q_i^a(k)^2 + Y_i(k)^2$$

This Lyapunov function is the scalar measure of the total of both actual and virtual queues in the network. The Lyapunov drift is given by,

$$\Delta L(k) = E[L(k+1) - L(k) \mid \bar{Q}] \tag{7.6}$$

Where $\bar{Q} = \{(Q_1^a(k), Y_1(k)), \ldots, (Q_N^a(k), Y_N(k))\}$. It is a well-established result that minimising the following function [GNT$^+$06],

$$\min \Delta L(k) - \sum_{i=1}^{N} VE[g(R_i(k))] \tag{7.7}$$

maximises Equation (7.2). Clearly, it enables a simple multi-objective optimisation problem where the first term (the Lyapunov drift) satisfies the constraints in Equations (7.3) and (7.4), whereas the second term ($\sum_{i=1}^{N} VE[g(R_i(k))]$) maximises the total network utility as expressed in Equation (7.2). The drift-plus-penalty theorem in [GNT$^+$06] does not require strict convexity assumptions. However, it ensures that the expected of the optimisation problem's primals converge to a solution that is within a factor of optimality, with bounded queue sizes. Furthermore, V is a system parameter which allows for making a trade-off between the achieved utility and the average queue backlog in the queues, as applied in [GNT$^+$06, NML08]. Next, a controller which minimises Equation (7.7) at each period k is presented.

7.2.4. CONTROLLER DESIGN

Since there is no explicit close-form expression for Equation (7.7), to formulate a controller that minimises Equation (7.7) in discrete time, the bounds for Equation (7.6) need to be found. With the following in mind: $\max\{x,0\}^2 \leq x^2$ and $\max\{x,0\} \leq x^2$ to express,

$$\begin{aligned}
Q_i^a(k+1)^2 \leq & Q_i^a(k)^2 + \mu_i^a(k)^2 + (R_i(k)\beta_i\theta_i)^2 \\
& - 2Q_i^a(k)(\mu_i^a(k) - R_i(k)\beta_i\theta_i), \\
Y_i(k+1)^2 \leq & (Y_i(k) - l_i(k))^2 + Q_i^s(k)^2 + \mu_i^s(k)^2 \\
& + 2Y_i(k)(Q_i^s(k) + R_i(k)(1-\beta_i)\theta_i) \\
& - 2Q_i^s(k)(\mu_i^s(k) - R_i(k)(1-\beta_i)\theta_i) \\
& + (R_i(k)(1-\beta_i)\theta_i))^2
\end{aligned}$$

Note that the network capacity at a period k is always upper-bounded due to practical limitations (e.g., transmit power). That is to say, $\mu_i^a(k) \leq \mu_{\max}$, $\mu_i^s(k) \leq \mu_{\max}$ for all i and k. Also, the number of requests that can be admitted is upper-bounded, i.e., $R_i(k) \leq R_{\max}$, for all i and k. $\mu_{\max}$ and $R_{\max}$ are constants. The choice of $R_i^{\max}$ is independent of the stability of the system. However, a choice of $R_i^{\max}$ less than the allocated aggregate mean channel capacity results in a strictly underutilised system. Then,

$$\begin{aligned}
& Q_i^a(k+1)^2 - Q_i^a(k)^2 \\
& \qquad \leq B^a - 2Q_i^a(k)(\mu_i^a(k) - R_i^a(k)\beta_i\theta_i)
\end{aligned} \tag{7.8}$$

where $B^a = \mu_{\max}^2 + (R_{\max}\beta_i\theta_i)^2$.Similarly, a bound for the virtual state queue $Y_i(k)$ is found,

$$\begin{aligned}
Y_i(k+1)^2 - Y_i(k)^2 \leq & \, l_i(k) - 2Y_i(k)l_i + Q_i^s(k)^2 \\
& + 2Y_i(k)(Q_i^s(k) + R_i(k)(1-\beta_i)\theta_i) \\
& + (R_i(k)(1-\beta_i)\theta_1)^2 + \mu_i(k)^2 \\
& - 2Q_i^s(k)[\mu_i(k) - R_i(k)(1-\beta_i)\theta_i] \\
\leq & \, B^Y + Q_i^s(k)^2 \\
& - 2Y_i(k)(l_i - Q_i^s(k) - R_i(k)(1-\beta_i)\theta_i) \\
& - 2Q_i^s(k)(\mu_i^s(k) - R_i(k)(1-\beta_i)\theta_i)
\end{aligned} \tag{7.9}$$

Where $B^Y = l_{\max}^2 + \mu_{\max}^2 + (R_{\max}(1-\beta_i))^2$ and $l_{\max}(k) = \max\{l_i(k)\}$. By using Equations (7.8) and (7.9), Equation (7.7) can now be expressed in its entirety,

$$\begin{aligned}
&\Delta L(k) - \sum_{i=1}^{N} VE[g(R_i(k))] \qquad (7.10)\\
&\leq \sum_{i=1}^{N} B^a - 2E[Q_i^a(k) \mid \bar{Q}](\mu_i^a(k) - R_i^a(k)\beta_i\theta_i)\\
&+ B^Y - 2E[Y_i(k)(l_i(k) - Q_i^s(k) - R_i(k)(1-\beta_i)\theta_i) \mid \bar{Q}]\\
&- 2E[Q_i^s(k) \mid \bar{Q}](\mu_i^s(k) - R_i(k)(1-\beta_i)\theta_i)\\
&+ E[Q_i^s(k)^2 \mid \bar{Q}] - VE[g(R_i(k))]
\end{aligned}$$

Minimising the right-hand side of the inequality in Equation (7.10) ensures queue stability and maximises the objective in Equation (7.2). The controller can now be defined, which jointly minimises the right-hand side of Equation (7.10) at each k.

FLOW CONTROL

The flow control decision's objective is to regulate the flow $R_i(k)$ from the application request reservoir. Collecting the expressions from Equation (7.10) that include $R_i(k)$ which is given as,

$$\begin{aligned}
&\sum_{i=1}^{N} E[2Y_i(k)R_i(k)(1-\beta_i)\theta_i + 2Q_i^s(k)R_i(k)(1-\beta_i)\theta_i\\
&\quad + 2Q_i^a(k)R_i(k)\beta_i\theta_i - Vg(R_i(k))]
\end{aligned}$$

Minimising the above expression for each device i at each k will minimise the right-hand side of Equation (7.10). Each device i solves the following problem, which will determine the flow decision for that device at time k,

$$\begin{aligned}
\arg\min_{R_i} \quad & R_i[Q_i^s(k)(1-\beta_i)\theta_i + Q_i^a(k)\beta_i\theta_i\\
& + Y_i(k)(1-\beta_i)\theta_i] - V_i g(R_i)\\
s.t. \quad & R_i \leq R_i^{\max}
\end{aligned}$$

The choice of $R_i^{\max}$ is independent of the stability of the system. V is a parameter that determines the balance between the controller's desire to back pressure requests and maximise its utility through R_i. Any positive V will result in a stable system.

Furthermore, the flow control is at its most effective when it keeps up with the system's rate of change, $r_{\text{flow}}(k) = \min\{\lambda_i(k), \mu_i^a(k)\mu_i^s(k)/(\mu_i^a(k) + \mu_i^s(k))\}$. However, a lower rate does not violate the stability criterion.

SCHEDULING

The other control decision in Equation (7.10) is the scheduling decision (i.e., determination of $\mu_i^a(k)$ and $\mu_i^s(k)$). The scheduling decision is centralised to the RBS. The idea is that $\mu_i^a(k)$ and $\mu_i^s(k)$ are determined in a way that the right-hand-side of Equation (7.10) is minimised. Grouping and maximising in terms of $\mu_i^a(k)$ and $\mu_i^s(k)$ from Equation (7.10) yields,

$$\arg\max_i \quad \max\{Q_i^a(k)\mu_i^a(k), Q_i^s(k)\mu_i^s(k)\} \tag{7.11}$$

This essentially implies that the largest queue amongst all the devices will receive the channel capacity at that time k.

7.2.5. PARAMETER ESTIMATION

The distribution of the parameters θ_i and β_i are unknown to the system at runtime. The mean of the parameters is estimated using stochastic gradient descent, $\theta_i^{k+1} \leftarrow \theta_i^k - \gamma_i \nabla F(\theta_i^k)$, where $\gamma_i = \gamma \quad \forall i$.

7.3. EVALUATION

In this section, the evaluation scenarios for the proposed CLC are presented and motivated. To evaluate the controller, a simulator was developed in Python using SimPy for queue representations and the simulation core. CVXPY [DB16] is used for solving convex optimisation problems.

The simulations presented below are designed to reveal how the proposed controller can maintain the maximum time-average deferred state information, converge, and handle variations in inputs.

7.3.1. COMPARISON POLICIES

Three different scheduling policies are used to evaluate the proposed controller.

Round Robin (RR) scheduling In RR scheduling, the queues are scheduled in a sequential order regardless of their occupancy, i.e., this approach does not enforce an explicit scheduling objective. RR is a commonly used scheduling approach.

Prioritised scheduling In prioritised scheduling, the queues in the system are partitioned into different prioritisation sets. The set of queues in a prioritisation level are never scheduled unless the queues in the higher prioritisation levels are empty. In this scenario, the $Q_i^a(k)$ is prioritised over $Q_i^s(k)$ to ensure minimum application latency.

Priority Threshold (PT) scheduling In PT scheduling the queues in the system are partitioned into two sets. One of the sets has an occupancy threshold. A queue in that set is only scheduled if its occupancy exceeds the threshold. In the simulations below, the set of $Q_i^s(k)$ are assigned a threshold of $l_i(k)$.

7.3.2. METRICS

The metrics below are used to evaluate the proposed controller.

Utilisation ρ The utility level of a system of queues is expressed as $\rho = \lambda/\mu$. In the simulations below, two utilisation factors are observed; end-to-end utilisation for device i $\rho_i^{\text{e2e}}(k)$ and the resulting level of utilisation due to back pressure from the flow control decision $\rho_i^{\text{flow}}(k)$. The time designed time-average utilization of a device as determined by the load and system parameters in Table 7.1, is define as $\bar{\rho}^{\text{e2e}}(k)$. Furthermore, $\hat{\rho}^{\text{e2e}}(k)$ and $\hat{\rho}^{\text{flow}}(k)$ are the instantaneous time-averages for system and post-controller utilisation levels, respectively, over all N devices. If the system is stable, $E[\rho_i^{\text{flow}}] \leq 1$. Consequently, $\bar{\rho}^{\text{e2e}}(k) \geq 1$ is a system loaded over 100%.

Utility The utility of the system is proportional to R.

Trade-off The trade-off between the competing traffic flows is at the core of the problem addressed in this paper. This trade-off is formally expressed as,

$$\alpha(k) = \begin{cases} \frac{Q_i^s(k) - l_i(k)}{l_i(k)} & \text{if} \quad \rho_i^{\text{e2e}}(k) \geq 1 \\ |\Delta Q_i^s(k) - \Delta Q_i^a(k)| & \text{if} \quad \rho_i^{\text{e2e}}(k) < 1 \end{cases}$$

Where Δ is the forward difference of $f(x)$, $\Delta f(x) = f(x+1) - f(x)$. The first component premiers stability and a bounded $Q_i^s(k)$ when the system is over-utilised. The second component premiers that $Q_i^a(k)$ and $Q_i^s(k)$ change at a similar rate when the system is underutilised. A low value is desired.

Finally, $\hat{\lambda}(k), \hat{\mu}(k), \hat{Q}^a(k), \hat{Q}^s(k)$, and $\hat{R}(k)$ are the instantaneous mean for those values over all N devices.

7.3.3. SYSTEM PARAMETER VALUES

The simulation parameters used to evaluate the proposed controller are presented in Table 7.1. For this scenario, a Poisson arrival process and exponential service times are assumed. In the simulation scenario below, the total system utilisation $\bar{\rho}^{\text{e2e}}(k)$ is intermittently over 100%, $\bar{\rho}^{\text{e2e}}(k) > 1$, see Figure 7.4. Furthermore, for the policies detailed in Section 7.3.1, $R_i(k) = \lambda_i(k) \quad \forall i$.

7.3.4. INPUT VALUES

When evaluating the proposed Cross-Layer Controller, the instances when the system is over-utilised is of particular interest. Therefore, the system inputs are balanced to achieve a desired level of system utilisation as presented by $\bar{\rho}^{\text{e2e}}(k)$ in Figure 7.4. This is achieved by either varying $\lambda_i(k)$ or $\mu_i(k)$. The system utilisation depicted in Figure 7.4 has three different load scenarios recurring over five time periods. The three load scenarios are; strictly underutilised, strictly over-utilised, and transient. They are distributed over five periods, as de-marked in Figure 7.4.

After the epoch of intermittent over-utilisation $t > 600$, system load enters the transient epoch where the designed utilisation drops to 55%, $\bar{\rho}^{\text{e2e}}(k) = 0.55$. Up until this

Parameter	**Value**
Sim. duration	500
N	10
$\lambda_i(k)$	Poisson process, $exp(\lambda = 0.24) \quad \forall i$, see Figure 7.4
$\mu_i(k)$	See Figure 7.4
r_{flow}	0.01
R_i^{Max}	$0.05 \quad \forall i$
θ_i	$\mathcal{N}(\mu = 3, \sigma^2 = 3/2) \quad \forall i$, sampled for each arrival
β_i	$\mathcal{U}(0.35, 0.65)$, sampled once
$l_i(k)$	See Figure 7.4
γ_i	$0.01 \quad \forall i$
V	16
$g_i(R)$	$g_i(R) = \ln \beta R_i(k)$

Table 7.1.: Simulation parameter values of the proposed controller.

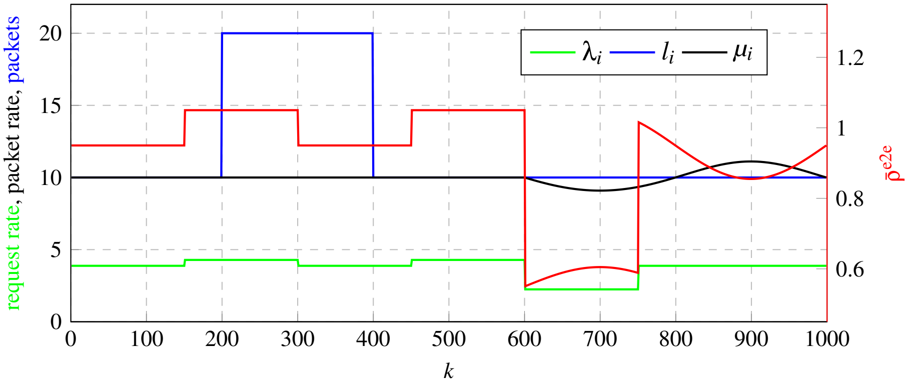


Figure 7.4.: System parameters and designed utilisation.

point the change in $\bar{\rho}^{\text{e2e}}(k)$ was attributed change in $\lambda_i(k)$. After this point $t > 600$, $\lambda_i(k)$ is kept constant and $\mu_i(k)$ is varied as a sinusoid, see Figure 7.4. The intermittent over-utilisation and transit epochs are from now on referred to as epoch 1 and epoch 2, respectively.

7.4. RESULTS

In this section, the results from the simulations detailed in Section 7.3 are presented. Note that the time scale in the simulations is unit-less. It is also worth noting that the estimations for σ_i and β_i have converged after 100 time units.

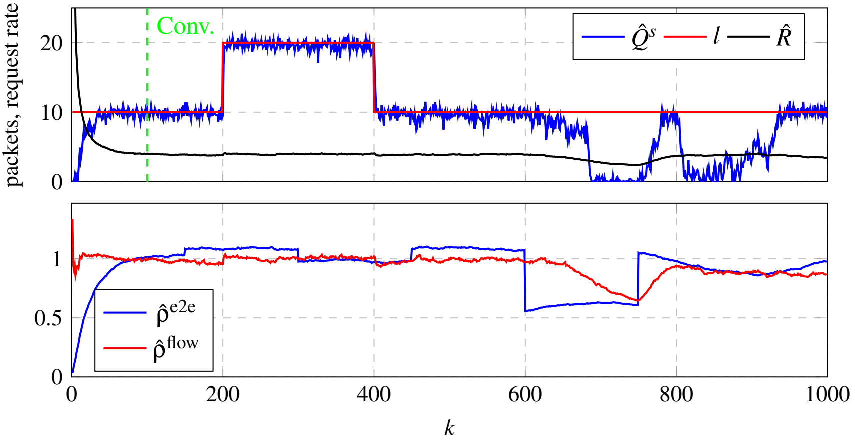


Figure 7.5.: Mean state queue occupancy $\hat{Q}^s$.

7.4.1. EXPECTED DEFERRED STATE TRAFFIC

The design objective of the controller is $E[Q_i^s] \leq l_i$ for each device i. Throughout epoch 1, as the system is over-utilised, the objective of the controller is to keep $E[Q_i^s] = l_i$. Figure 7.5 shows that the controller can maintain a time-average within the 95th percentile $< 1\%$ of $l_i(k)$.

$l_i(k)$ is increased to 20 at $k = 200$ and then decreased back to 10 at $k = 400$. Note that $\hat{Q}^s(k)$ tracks the target $l_i(k)$ practically immediately. The quick adjustment is achieved by momentarily sacrificing instantaneous stability $\hat{\rho}^{\text{flow}}(k) > 1$. This is practically instantly compensated for by a proportional reduction in $R_i(k)$. The expected stability of the system is thus maintained.

At the beginning of epoch 2, $\hat{\rho}^{\text{flow}}(k)$ lags $\hat{\rho}^{\text{e2e}}(k)$ and $\bar{\rho}^{\text{e2e}}(k)$ as the controller accommodates accumulated deferred application requests. Between $k = 600$ and just before $k = 680$ the controller gradually balances the application traffic and state traffic to when $\hat{\rho}^{\text{flow}}(k) < 1$. After this point, there is no contention between the two traffic flows and $\hat{Q}^s$ drops to just above 0, see Figure 7.5.

In epoch 2, $\lambda_i(k)$ and $l_i(k)$ are kept constant and $\mu_i(k)$ is oscillated sinusoidally around $\pm 10\%$ of $\bar{\rho}^{\text{e2e}}(k) = 1$. The scheduling component of the controller will be first to act on the change in $\mu_i(k)$. Consequently, there is small lag in $\hat{R}(k)$ from the change in $\hat{\rho}^{\text{e2e}}(k)$, see Figure 7.5. However, this lag has evidently no practical impact on the controller's ability to track $l_i(k)$ in relation to $\hat{\rho}^{\text{e2e}}(k)$.

Furthermore, PT scheduling has the objective $E[Q_i^s] = l_i$. As Figure 7.6 shows, PT scheduling can meet this objective while the system is over-utilised. However, contrary to the proposed controller, as the system moves into phase 2, $\hat{Q}^s(k)$ does not diminish. This is because none of the state queues will be served if there are any non-empty application queues $\sum_{i=1}^{N} Q_i^a(k) > 0$ or $Q_i^s(k) \geq l_i \forall i$ in the system. This is an

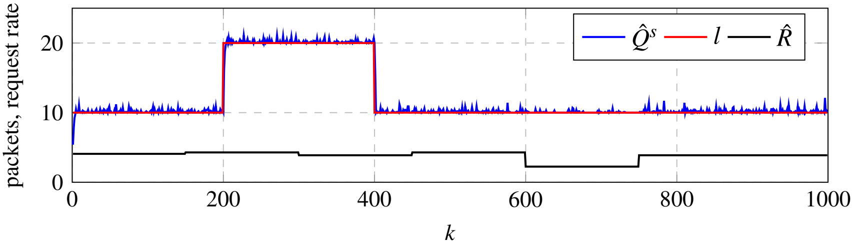


Figure 7.6.: $\hat{Q}^s(k)$ and $l_i(k)$ for PT scheduling.

undesirable trade-off for the targeted system as the intent is to reduce the amount of deferred state information when possible.

7.4.2. STABILITY AND SYSTEM UTILITY

Another objective of the controller is to maintain a stable time-average system of queues, namely $E[Q_i^s] < +\infty$ and $E[Q_i^a] < +\infty$ for each device i. The system's stability is revealed by observing the growth of the queues $\hat{Q^s}(k)$ and $\hat{Q^a}(k)$ as well as the system utilisation after back-pressure is applied, $\hat{\rho}^{\text{flow}}(k)$. As illustrated by Figure 7.7, both conditions: $E[Q_i^s] < +\infty$ and $E[Q_i^a] < +\infty$ are met using the proposed controller.

In Figure 7.5, note that $\hat{\rho}^{\text{flow}}(k)$ persist a level of around 1 even though $\hat{\rho}^{\text{e2e}}(k) \geq 1$. This is because the controlled system administers deferred application requests that have been back pressured. This is as designed and means that the system of N devices, as well as all application queues and service queues, are stable.

Furthermore, the back pressure effect can be directly observed through $\hat{R}(k)$ in Figure 7.5. Note the inverse relationship between $\hat{R}(k)$ and $\hat{\rho}^{\text{flow}}(k)$ in Figure 7.7.

Neither of the methods; RR, PT, nor Prio. have stability as an objective. This is made evident by their growing queue sizes while $\hat{\rho}^{\text{e2e}}(k) \geq 1$ in Figure 7.7. In the case of RR scheduling, as it does not discriminate between two traffic flows, they grow with a similar gradient. Prioritised scheduling, on the other hand, prioritises Q^a and thus lets Q^s grow uncontrollably. PT scheduling inadvertently maintains a stable Q^s at the expense of Q^a stability.

Because the proposed controller practices back pressure, it will analogously achieve a lower utility than the other scheduling methods as R_i is in this case suppressed. RR, PT, and Prio, will on the other hand persist $R_i(k) = \min\{R_{\max}, \lambda_i(k)\}$. This difference is contrasted in Figures 7.5 and 7.6 for the controller and the other methods respectively. The difference is small, but operating with a utility beyond the stability point has a detrimental effect on the latency experienced by each serviced application request.

7.4.3. CHOICE OF V

As discussed in [GNT$^+$06, NML08], although any positive V will result in a stable system, the choice of V does have an impact on the controller's ability to meet its other

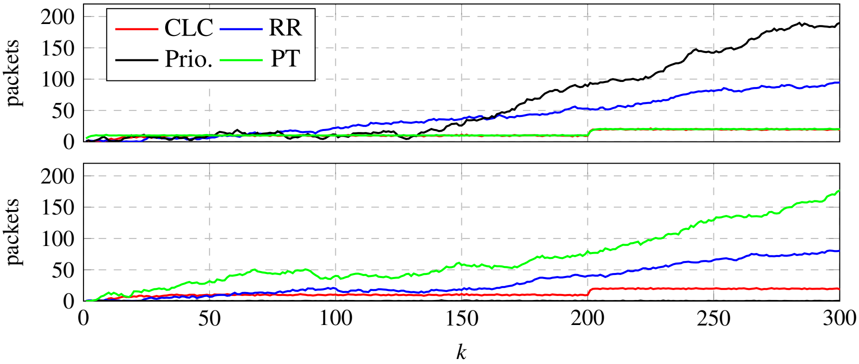


Figure 7.7.: Upper; $\hat{Q}^s(k)$, Lower; $\hat{Q}^a(k)$.

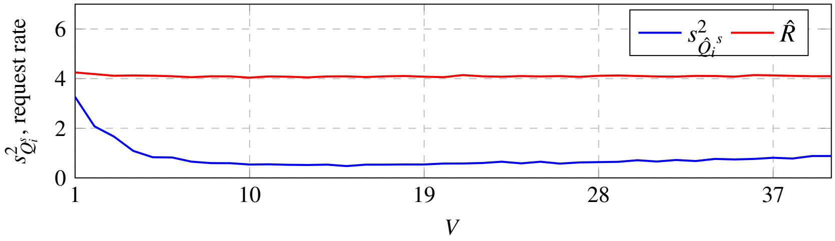


Figure 7.8.: Variance of $Q_i^s(k)$, $s^2_{Q_i^s}$ and $R_i(k)$ as a function of V.

objectives. A large V will promote the utility function, while a small V will promote back pressure. This is reflected in the immediate value of R at each k. However, all values of V will achieve the same time-average R, see Figure 7.8.

Furthermore, as illustrated by Figure 7.8, the choice of V has an impact on the sample variance of Q_i^s, $s^2_{Q_i^s}$. Although the time-average converges at $V > 10$, it is desirable to minimise $s^2_{Q_i^s}(k)$ provided it is not at the expense of system utility $R_i(k)$. Also, note that R is not affected by V. As illustrated in Figure 7.8, $s^2_{Q_i^s}(k)$ is convex to V. For this system, minimum $s^2_{Q_i^s}(k)$ is at $V = 16$.

7.4.4. QUANTIFYING THE TRADE-OFF

It has now been established that the controller successfully bounds the size of $Q_i^s(k)$ to $l_i(k)$ when the system is over-utilised $\bar{\rho}^{\text{e2e}}(k) \geq 1$ and that the controller is able to stabilise the system. The trade-off metric formulated in Section 7.3.2 attempts to summarise the performance of the controller in comparison to the other scheduling methods presented in Section 7.3.1 over time.

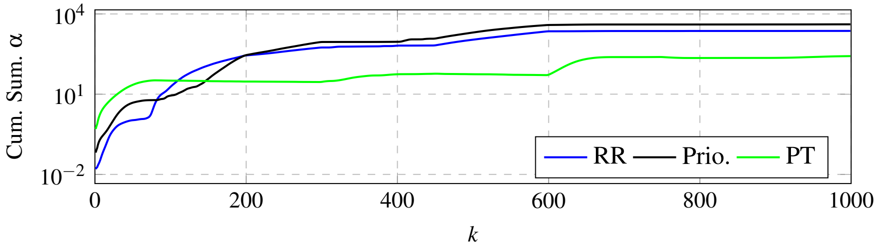


Figure 7.9.: Cumulative difference in trade-off to the controller.

In terms of the metrics in Section 7.3.2, the controller performs strictly better than the comparison scheduling methods. Therefore, for increased contrast, Figure 7.9 shows the cumulative trade-off for each method subtracted by the cumulative trade-off for the controller. PT scheduling tracks the proposed controller well but fails to balance the two flows as $\hat{\rho}^{\text{flow}}(k) < 1$ in the underutilised epoch. Prioritised scheduling allocates, on average, no resources to Q^s which therefore always maintains a high occupancy. RR scheduling indiscriminately schedules Q^a and Q^s and thus achieves a middle-of-the-road value during over-utilisation.

7.5. CONCLUSIONS

In this chapter, a CLC for achieving both a predictable maximum time average shared state inconsistency, and system stability was presented. The proposed controller and system objectives were formulated using Lyapunov Drift optimisation with penalty. The resulting Cross-Layer Controller was verified through simulation. The simulator showed that the proposed Cross-Layer Controller can track the desired level of shared state inconsistency within a narrow margin. It was also shown that the Cross-Layer Controller achieves system stability and can more successfully balance the system's two traffic flows than comparable and conventionally used methods. Additionally, the proposed Cross-Layer Controller can accommodate both momentary stochasticities in the queues and rapid changes in set points l_i while maintaining the desired time average shared state inconsistency.

Possible extensions of this work include a downlink scheduling policy for scheduling entire application processes across the system's IoT devices. To capture the energy constrained nature of IoT devices, such a system controller can include maximum time-average transmission energy for each device.

Part III.

5G and IoT

8

Ultra-Reliable and Low-Latency Communication for the mission-critical applications

The recent uptake in factory automation and robotisation is just the beginning of a more comprehensive adaptation of IoT, machine learning, big data, and Fog computing technologies to automate a large set of the professions that have come to characterise the twentieth century. This change is commonly referred to as the fourth industrial revolution [Sch17].

This technological revolution is still in its infancy. There are significant technological challenges yet to be addressed for the revolution to encompass cognitive and motorically intense professions. The fourth industrial revolution does not only imply that most tasks and professions will be automated at the rate of which technology matures. The ambition is not to indiscriminately eliminate human capital. Technology will instead be used to make use of the human cognitive advantage wherever it might be needed. This notion includes, for example, precise teleoperation, such as telesurgery. One can trivially imagine a future where machine decisions seamlessly inter-operate and complement physical human actions. Haptic feedback is a critical enabling technology in this pursuit [NS91]. It has consequently been argued that the Internet as we know it today will shift from content delivery to labour delivery [Fet14]. The tactile Internet enables this paradigm shift. A tactile Internet can include of amongst other things, a large number of connected tactile surfaces and robotic limbs, accessed remotely at high precision, see Figure 8.1.

Enabling the tactile Internet and haptic feedback for the fourth industrial revolution will require a communication intensive distributed system where a large set of resources are shared, and decisions are made in a distributed fashion in a network with many wireless links. It is a well-known fact that the stability of tactile control loops is particularly sensitive to jitter. The tactile Internet will, therefore, require near-deterministic single-digit millisecond latencies. Additionally, at the rate at which tactile feedback loops operate, there is a minimal margin for error [ADA$^+$15]. Traditional

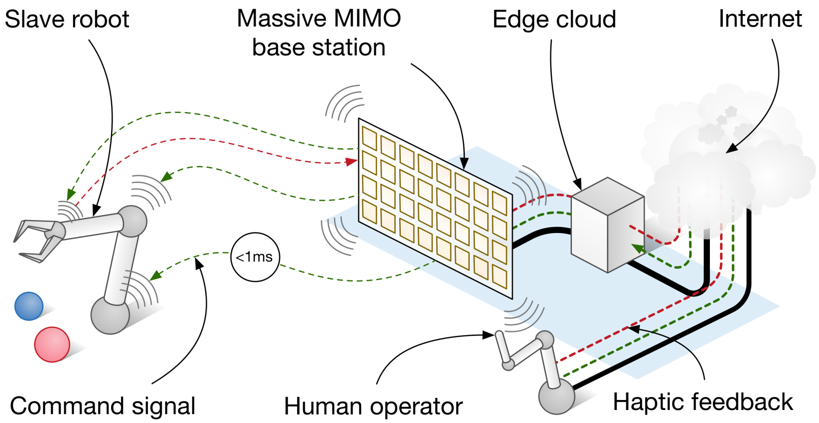


Figure 8.1.: The tactile Internet and massive MIMO.

mechanisms that provide reliability such as Hybrid Automatic Repeat Query (HARQ) and Automatic Repeat Query (ARQ) found in LTE might not be feasible at such a low over-the-air latency. Unlike audio and video content, tactile feedback can currently not be scalably compressed in a lossy fashion. Tactile feedback, contrary to audiovisual content, can therefore not trivially be adapted to prevailing wireless throughput capacity. Furthermore, the surfaces and robotic limbs that are part of tomorrow's tactile Internet do not operate in just one tactile dimension. They each rely on multiple multimodal data inputs and outputs across multiple devices [PCW12], requiring each device to communicate at the same level of reliably and latency simultaneously. Providing wireless URLLC to the devices that constitute these services has been proven to be non-trivial.

Existing mobile specifications, as they are deployed, are unable to provide URLLC cost-effectively at scale [HAS$^+$16]. They cannot also deliver reliable low latency communication to multiple users simultaneously. Consequently, the proposed 5G are focusing on addressing the challenges of scale, device heterogeneity, and mission criticality. In other words, the aim is to provide control communication for closing the global tactile control loop and not just deliver, e.g. audiovisual content.

Work is beginning to emerge in the literature on how to generally realise URLLC and the tactile Internet using 5G. [ADA$^+$15] taxonomises the tactile Internet design challenges facing 5G. The work primarily addresses system design challenges and presents a reasonable foundation of performance requirements and limitations. There are also proposed system architectures for achieving URLLC [SMK$^+$17]. The Fog computing paradigm will arguably also play an essential role and undergo significant changes in realising the services and the infrastructure of the tactile Internet [Mos15]. Although the entire wireless system infrastructure needs to operate at low latency, the over-the-air latency in the physical layer is fundamental in achieving URLLC. Consequently, there are for example proposed physical layer specifications for URLLC wireless networks [WMP$^+$16]. However, none of these works has uniformly looked at the physical layer for closing the tactile feedback loop with the performance gains Massive MIMO can deliver.

This work investigates how to realise URLLC communication for the tactile Internet using Massive MIMO. Requirements of bilateral teleoperation, an application of haptic feedback, is adopted as a baseline, and investigate how its reliability and latency requirements can be fulfilled with Massive MIMO. Furthermore, this chapter contributes to the performance analysis of Massive MIMO under URLLC-conditions which is used to formulate upper performance bounds for such a system. Additionally, the analysis allows us to constructively discuss trade-offs and specifications for a URLLC Massive MIMO system. The results in the analysis are contrasted with what is attainable with the current LTE specifications.

8.1. BILATERAL TELE-OPERATION

In this section, the communication requirements for bilateral tele-operation which is an application of tactile feedback and one of the most promising applications of the tactile Internet is given a closer look. Generally speaking, closing force-feedback control loops for mechanical manipulators with interaction in stiff, as opposed to elastic, environments is challenging and becomes notoriously difficult from a stability point-of-view when uncertain delays are introduced in the loop. Also, in scenarios with less stiff interaction forces, transmission delays and communication jitter deteriorate performance and robustness. The concept of haptic tele-operation with bilateral force-velocity reflection between a "master" (human operator) and a so-called "slave" (slave robot) provides a mean of transparency and experienced interaction of contact forces and end-effector motions for the operator, see Figure 8.1.

Bilateral tele-operation has been studied extensively. The method of 'wave variables', introduced in [NS91], has successfully been extended and applied in remote-controlled mining, dental and medical surgery, and even for space applications with significant delays. In such systems, fundamental limitations will impose a trade-off between stability and quality in terms of experienced transparency of the bilateral tele-operation depending on the properties of the communication channel.

For remote control, a good complement to haptic feedback is streaming video from a remote site, which allows the operator on the master side to visually inspect the interaction. However, for a consistent user experience, it is vital that the different feedback channels, with possibly significantly varying amount of data, are synchronized without unnecessary delays and jitter.

In a typical system, each joint is controlled separately over wireless links. Robotic joints typically update at $250Hz$ [ABB]. A latency of at most $4ms$ is therefore desired, preferably $1-2ms$ to accommodate jitter. Furthermore, the jitter is fundamentally addressed with a high wireless link reliability, Bit Error Rate (BER) $< 1\text{e}{-5}$.

For a good survey of haptic bilateral tele-operation case, see [HS06]. In [Mil14] the quality-latency trade-off for bilateral haptic tele-operation is investigated for different wireless standards. In [MRkFS13], the effect of network quality on bilateral tele-operation was investigated. In that work, the performance metric was how accurate the slave system follows the command of the master system as well as how transparent the environment is to the operator. In that work, it was shown that packet loss (i.e., BER) affects the signal oscillations, while the latency in the network causes the steady-state tracking error increase. Based on this result, one can conclude that the reliability and latency issues must be addressed together.

8.2. RELIABILITY

In this section, it is evaluated how Massive MIMO can be dimensioned to achieve the desired reliability level, BER, presented in Section 8.1. The analysis was done through simulation using MATLAB's communications toolbox executed on Lund Unversity's cluster, LUNARC.

With the reliability challenges detailed in Section 8.1 coupled with the URLLC ambitions in [HWW^{+}16], a BER of 1e$-$5 is targeted. Because the UEs are relatively computationally underpowered and do not require particularly high throughput, as discussed in [BSHD14], we use the Quadrature Phase Shift Keying (QPSK) modulation scheme.

In this work, the Independent and Identically Distributed random variables (i.i.d) Rayleigh fading channel model is adopted. Using i.i.d Rayleigh fading channels will form a reasonable upper performance bound, as a best-case scenario. In reality, due to the correlation between users, the i.i.d assumption will not hold, and one can expect that the minimum requirements for URLLC haptic feedback will be more stringent. For fading channels, to improve the reliability of the channel, it is common to use channel coding such as convolutional or turbo coding depending on the application. Nevertheless, using the coding incurs additional cost in terms of receiver complexity and decoding delay. Note that convolutional coding with rate 1/2 and constraint length 7 is used in this study since it yields a lower delay compared to Turbo coding and Low-Density Parity-Check (LDPC) codes. Concerning diversity combining, the use of both Maximum-Ration Combining (MR) and Zero-Forcing (ZF) is investigated. These are both linear precoding schemes. Although MR is arguably not entirely beneficial in massive MIMO, MR is included as a reference to a low-complexity mechanism. MR, therefore, acts as a lower performance bound and will effectively contract the channel properties with ZF.

As for our targeted haptic feedback system, in the scenario evaluated in [Mil14], a 6-Degrees of Freedom (DoF) robot was considered. The authors of [ADA^{+}15] propose segments of 48 data bits for a 3-DoF set-up. A packet size of 100 bits is therefore adopted with near-constant traffic flow is when the UE is operational. To detect errors, it is also assumed that Cyclic Redundancy Check (CRC) is applied. These parameters can also be considered to be true for other robotics systems.

Modulation	QPSK
Channel model	i.i.d Rayleigh fading
Precoding	MR, ZF
Channel coding	Convolutional code with rate 1/2, constraint length 7
SNR	$(-14,10)$ dB
BER target	1e$-$5
Packet size	100 bits

Table 8.1.: Physical layer parameters.

8.2.1. THE ROLE OF MASSIVE MIMO

Massive MIMO provides the means to significantly reduce the BER over existing LTE Multi-User MIMO (MU-MIMO) specifications in a relatively straightforward manner [MVL^{+}16]. Because of the focusing effect in massive MIMO [LETM14], more UEs can be served with a lower BER at a lower Signal-to-Interference-plus-Noise Ratio (SNR) than in current deployed wireless specifications. The UEs of the tactile Internet operate over a wide range of power requirements. Some UEs are for example battery powered with a targeted lifespan expressed in years. Here massive MIMO offers an advantage over conventional techniques as the UEs can be made relatively simple as much of the complexity can be moved to the RBS. Predictable power consumption is an integral part of the reliability of a UE. A low transmission power typically results in a low SNR. To therefore sweep across SNR levels from as low as -14 dB to 10 dB. The targeted massive MIMO system's parameters are summarized in Table 8.1.

8.2.2. PERFORMANCE OF MASSIVE MIMO

A fundamental differentiating design parameter in a massive MIMO system is the number of RBS antennas, M. A high M/N ratio yields a lower BER or allows the system to operate with a lower SNR. In the targeted Massive MIMO system, the UEs is assumed to operate with one antenna. For the sake of generality, the robotic joints and surfaces in this scenario are referred to as UEs.

We proceed by investigating the relationship between the number of antennas and the system's reliability by finding the minimum number of antennas M required to achieve a BER of 1e$-$5 for a given N simultaneously served UEs at a certain SNR level. Here, SNR is defined as the input SNR where it is defined as the ratio of transmitting and noise power. Since the N UEs in the system are low powered and we do not want to add additional delay as a result of the computational complexity, and since massive MIMO achieves an inherently low BER, we initially proceed without any channel coding. Using the scenario in [Mil14] as our reference. The ABB robotic arms in that scenario have 3-6 Degrees Of Freedom (DOF) [ABB].

Figure 8.2 reveals the difference in performance between MR and ZF for i.i.d chan-

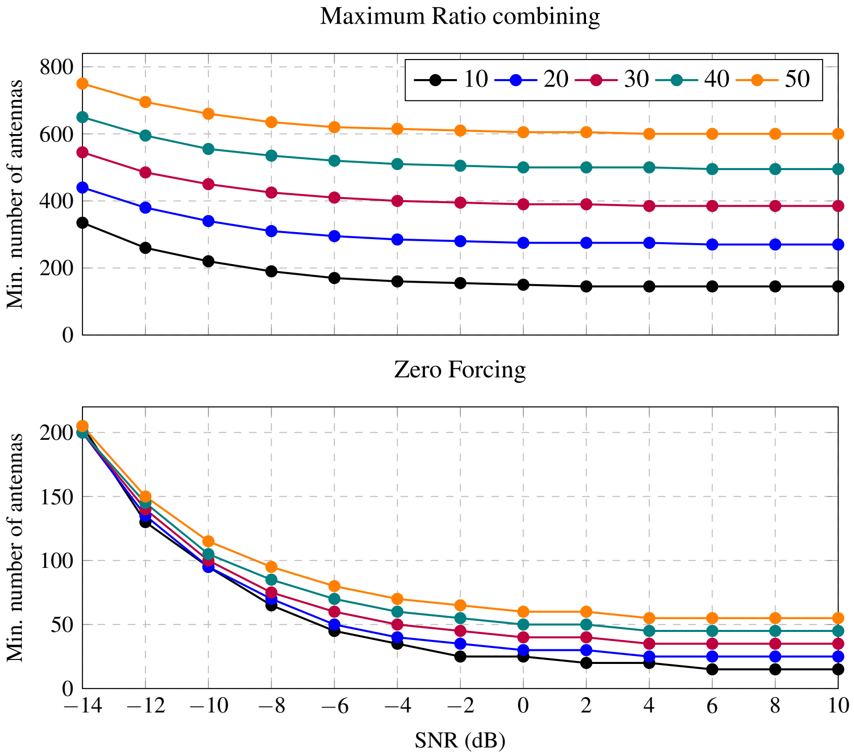


Figure 8.2.: Number of antennas required to simultaneously serve N UEs at a specific SNR with a BER of 1e−5. Without channel coding.

nels. The graphs can be read as either the minimum number of antennas required to achieve a certain BER or the degradation of BER as a function of the number of UEs and SNR. With either pre-coding schemes, there is no significant degradation in BER until SNR$=0$. From this point, ZF's performance degrades at a relatively higher rate with than MR but still performs strictly better than MR. ZF outperforms MR on average a factor of 2 at high SNRs values and with a factor of almost 10 at low SNRs. ZF's gain over MR increases linearly with SNR. However, ZF's gain over MR increases quadratically with the number of UEs, N.

In Figure 8.3, channel coding is used to improve reliability and reveal its relative gain. Again, ZF outperforms MR on average of a factor of 0.85 across all configurations. Contrasting Figures 8.2 and 8.3 shows that adding channel coding to ZF provides on average a factor 0.5 improvement at low SNR levels and practically no improvement for high SNR level when it comes to the minimum number of required antennas. MR, on the other hand, sees a more than four-fold increase in performance.

As suggested by the results in Figures 8.2 and 8.3, with ZF, as long as M is sufficiently high (e.g, $M=100$) we arguably stand to gain very little from channel coding.

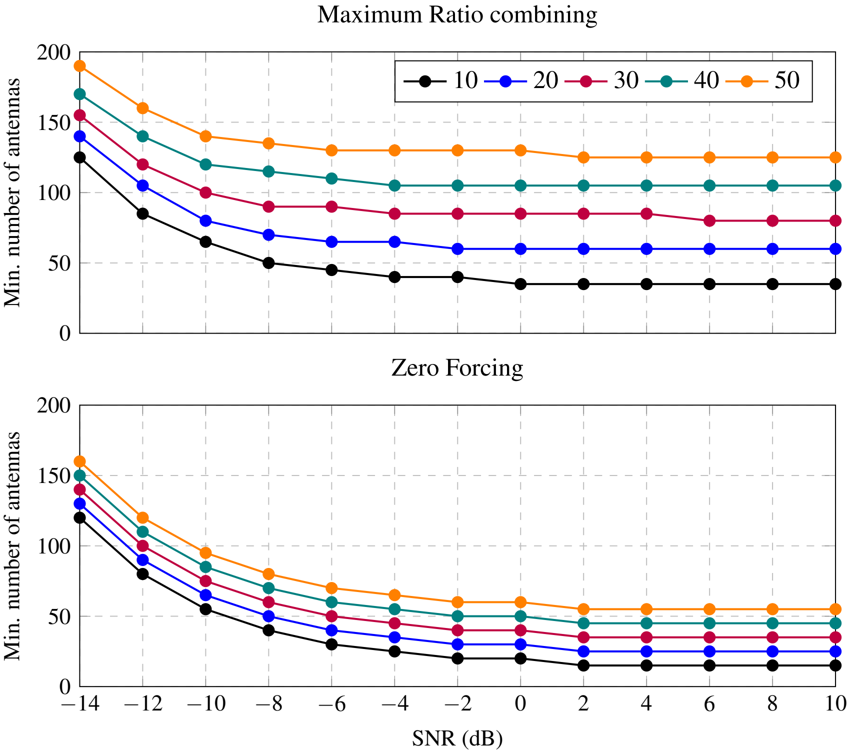


Figure 8.3.: Number of antennas required to simultaneously serve N UEs at a specific SNR with a BER of 1e−5. With channel coding.

As seen in Figures 8.2 and 8.3 the BER is satisfied at SNR= 0, at which point the minimum number of antennas does not decrease with increased SNR. At this point, there is no gain in BER when using channel coding. It is also evident from Figures 8.2 and 8.3 that channel coding is not contributing when the number of UE high and SNR < 0. This is illustrated by the convergence of the minimum number of required antennas for each number of users as SNR is decreased. The gain diminished on average quadratically with diminishing SNR. Channel coding gain is therefore only present for $K \leq 30$ and low SNR conditions. This can be attributed to the fact that the channel coding gain is very low when the intra-UE interference is high, e.g. when $K > 30$.

Furthermore, increasing the number of UEs N will require a factor 1 increase in the number of antennas on the RBS side. However, the number of simultaneously served UEs, N, does not only depend on M, and when the latency requirements are taken into account M and N should be carefully decided as we show next.

8.3. LATENCY

In this section, we evaluate how ultra low latency can be achieved with multi-user massive MIMO. The starting point is a round-trip latency of $1ms$ specified in [ADA$^+$15] and Section 8.1.

The latency contributors can be decompose into three components: i-) processing/coding at the transmitter; ii-) over-the air transmission; iii-) the processing/decoding at the receiver. The first and third components are strongly related to the hardware, software, and coding scheme used at the transmitter and receiver UEs [MVL$^+$16]. The second component is however strongly related to the frame structure used in the communication link. In the current LTE specifications, over-the air latency (i.e., transmission time interval (Transmission Time Interval (TTI)), is in part determined by the symbol duration. This is already at $1ms$, which makes it impossible to achieve the desired low-latency. Therefore, TTI duration should be reduced. One solution is to reduce the symbol duration. More specifically, the current Orthogonal Frequency-Division Multiplexing (OFDM) symbol duration is too long. One way to do decrease it is to increase the sub-carrier spacing which will reduce OFDM symbol duration and consequently TTI, as according to the following relationship,

$$\mathrm{TTI} = \tau \left(\frac{1}{\Delta f} + T_g + T_p \right), \tag{8.1}$$

where τ, Δf, T_g and T_p are the number of OFDM symbols in one TTI, the sub-carrier spacing (in kHz), cyclic prefix duration (in μs) and some processing delay which may depend on the software and hardware on the UE, respectively. Furthermore, one OFDM duration (e.g., symbol duration) is expressed as,

$$T_s = \frac{1}{\Delta f} + T_g \tag{8.2}$$

For example, in the LTE standard, $\Delta f = 15kHz$ and $T_g = 4.76\mu$s. Consequently, one OFDM duration in LTE is, $T_s = T_u + T_g = 71.4\mu s$ where $T_u = \frac{1}{\Delta f} = 66.7\ \mu s$ and there are 14 OFDM symbols in one TTI. Clearly, in order to reduce the TTI duration, Δf should be reduced since the other factors T_g and T_p are not controllable and usually depend on the channel characteristics and the type of the UE, respectively. As it can be seen from Equation (8.1) that another alternative to reduce TTI is to use fewer OFDM symbols at each TTI (i.e., reduce τ) without needing to change Δf. One drawback of using a reduced number of OFDM symbols is that the resulting scheduling cost can increase.

In order for massive MIMO to operate efficiently, the RBS needs to collect Channel State Information (CSI), which is achieved through the pilots symbols transmitted from each UEs to the RBS. In an OFDM based system, each pilot symbol correspond to a sub-carrier in one OFDM symbol. That is to say, some number of OFDM symbols should be dedicated for CSI. If β number of OFDM symbols out of τ symbols are used for pilots, then the number of UEs that can be simultaneously served by a massive

MIMO RBS is,

$$K = \beta S = \beta \left(\frac{1}{\Delta f T_g} \right) \tag{8.3}$$

where $\beta < \tau$ and S is frequency smoothness as defined in [Mar10]. In other words, S is the coherence bandwidth of the channel in terms of number of sub-carriers and over S sub-carriers the channel can be seen as constant and a reliable communication for the CSI transmission can be realized. Note that if β symbols are used for pilots then the actual data communication will be $(\tau - \beta)$ symbols, which are used by all the scheduled UEs at the same time. Here, the system efficiently is defined as $\xi = (1 - \beta/\tau)$. Clearly, with higher β values we can support more UEs. However, the amount of data that can be received or transmitted will be reduced.

Combining Equations (8.1) to (8.3) highlights an interesting trade-off. A higher Δf yields a lower TTI but also lowers the number of UEs K that can be simultaneously served, when $N \geq K$. Generally speaking, we have strict latency requirement, i.e., TTI_{thr} and proceeding to maximize K yields the following optimization problem,

$$\max \quad K \tag{8.4}$$

$$s.t. \quad \text{TTI} \leq \text{TTI}_{\text{thr}} \tag{8.5}$$

The solution is straightforward and given by,

$$\Delta f^* = \frac{1}{\frac{\text{TTI}_{thr}}{\tau} - T_g - T_p} \tag{8.6}$$

and

$$K^* = \beta \left(\frac{\text{TTI}_{thr}}{\tau T_g} - \frac{T_p}{T_g} - 1 \right) \tag{8.7}$$

8.3.1. SYSTEM VIEW

Figure 8.4 depicts the optimal sub-carrier spacing and the number of simultaneously supported UEs with varying T_g values by using Equations (8.6) and (8.7). As an example, when $\beta = 2$ and $\tau = 7$ (i.e., ξ=71%) and $\text{TTI}_{\text{thr}} = \{100, 200\}\mu s$. Less strict TTI requirements can create an opportunity to support more UEs simultaneously. For example, in the targeted scenario we require the TTI duration to be $100\mu s$, i.e., $\text{TTI}_{\text{thr}} = 100\mu s$ which is short enough to provide 1 ms end-to-end delay including encoding/decoding[1] and processing delays [ADA$^+$15]. Thus, each OFDM symbol needs to be $14.28\mu s$, $T_g = 0.9\mu s$, and $T_u = 13.38\mu s$ as a consequence of Equation (8.6) $\Delta f^* = 75kHz$. When $\beta = 2$, by using Equation (8.7) the system can serve $K^* = 28$ UEs simultaneously.

Figure 8.4 can help us design the required frame structure by illustrating the primary system trade-offs. For example, after measuring the channel characteristics (e.g., the

[1] The encoding/decoding delay should be in the same order of one OFDM symbol duration in order to avoid any memory issue.

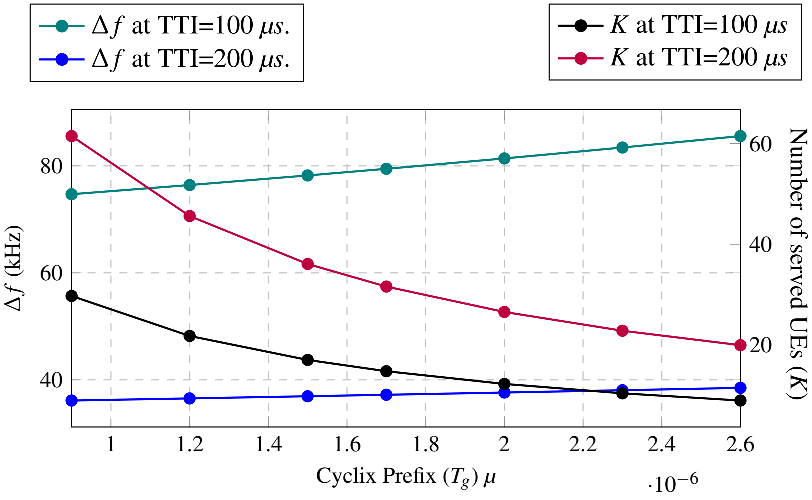


Figure 8.4.: K and Δf with various T_g

delay spread) and deciding the length of the cyclic prefix one can use the results in Figure 8.4 to decide the sub-carrier spacing depending on the latency requirements given in Section 8.1. Then, the number of the supported UEs, K, can trivially be determined by using Equation (8.7). In order to support K UEs, the results in Section 8.2 can be utilized to determine how many antennas are needed at the base station. The design can also be realized in reverse, such that for a given number of RBS antennas, M, one can determine the number of simultaneously supported UEs, K, based on a given latency requirement by using Equation (8.7). If K is not supported with the given number of antennas, a lower K can be considered. However, a lower K will require an increased sub-carrier spacing to reduce the TTI further. The increased sub-carrier spacing comes at the expense of increased bandwidth. If there is a bandwidth shortage then there may not be sufficient number of pilot symbols, and consequently K will decrease.

8.3.2. LATENCY AND RELIABILITY

In the real-time scenario presented in Section 8.1, reliability and latency are inextricably linked. Irregardless of the over-the-air latency, a high BER will result in packet losses. With a BER of 0, the over-the-air latency would be deterministic. However, we are able to achieve BER of $1e{-}5$. Assuming that some degree of error detection mechanism is applied, retransmission mechanisms such as ARQ would contribute with jitter. Packet losses can either be remedied in the controller by compensating for the uncertainty permanently lost information introduces or through ARQ. However, when the TTI is reduced to $100\mu s$, the relative latency of retransmission increases dramatically. ARQ might therefore not be applicable in this scenario.

8.3.3. PRECODING DESIGN

Lastly, we would like to discuss the impact of precoding design. This is an essential part of a massive MIMO system in terms of the end-to-end latency. It has been shown in [PRLE17] that in a typical massive MIMO system with 128 antennas at the base station and 8 UEs transmitting uplink data, the precoding delay due to the required matrix inversion and multiplications for ZF can be up to $150\mu s$. Since ZF has a complexity proportional to K^2M, the precoding latency will dramatically increase as K and M increase. This amount of latency is significant and a challenge for tactile Internet applications. One possible solution to reduce the precoding latency is to use more hardware resources, which however will increase the equipment cost.

8.4. CONCLUSIONS

In this chapter, the potential gains of utilizing Massive MIMO was investigated for realizing ultra-reliable, low-latency communication which is an essential part of the tactile Internet and thus applications that rely on haptic feedback. Although this chapter specifically investigates the requirements for the tactile internet and haptic feedback, the results can generally be applied to any ultra reliable communication scenario in robotics, control, IoT, etc. The minimum reliability and latency requirements for these type of applications and through systematic simulation studied and analyzed the performance of Massive MIMO given these requirements were addressed. The results reveal that depending on the precoding scheme used, the performance may vary but that ZF is highly preferable even without channel coding. Additionally, it arguably would be worthwhile to investigate the performance of Polar Codes in this scenario. Polar Codes [Ari09] have been deemed beneficial for short packet transmission. The latency requirements can be achieved by modifying the frame structure but the trade-off between the latency and the number of simultaneously supportable devices must be taken into account in the design of the system. In the following chapter the above findings are used to control a real-time application over LuMaMi.

Part IV.

A Fog computing test-bed

9

A 5G edge cloud test-bed

This chapter targets the feasibility of running time-sensitive and mission-critical applications in a Fog computing infrastructure. The chatper proceeds by presenting a design and implementation of a Fog computing research test-bed that encompasses a distributed set of compute nodes, a distributed PaaS framework, a 5G cell, and a time-sensitive mission-critical process under control, see Figure 9.1.

A mission-critical system is one in which a failure or interruption comes with an unacceptable business or human cost. Here, a failure may be an arbitrarily small deviation from the desired operation. Naturally, this includes all systems that create a risk of injury but also systems in which failure incurs a very notable inconvenience, such as a means of transportation rendered practically useless because it is making passengers nauseous. Such applications are time-sensitive in the manner that they are unable to cope with delay and jitter in the delay, to the point where they violate their requirements.

The WANs separating a time-sensitive mission-critical system from a traditional distant DC may incur latencies beyond what is operationally acceptable. Fog computing was proposed to mitigate the latent latency, throughput, and availability barriers that separate the end-users from distant DCs. When accessing the Fog over URLLC 5G [SMS$^+$17], the latencies are sufficiently low that time-sensitive mission-critical applications can be deployed in the Fog. An additional benefit of Fog computing is that resident applications can be made spatially redundant and fall-back solutions can be implemented at various geographical points in the infrastructure for additional resilience.

Historically, control systems have been deployed as monolithic SW implementations on carefully tuned HW, adjacent to the plants they control. Deploying monolithic SW on static HW makes such systems undesirably non-modular, less extensible, and limits their ability to self-adapt. Conversely, cloud-native applications are built for the

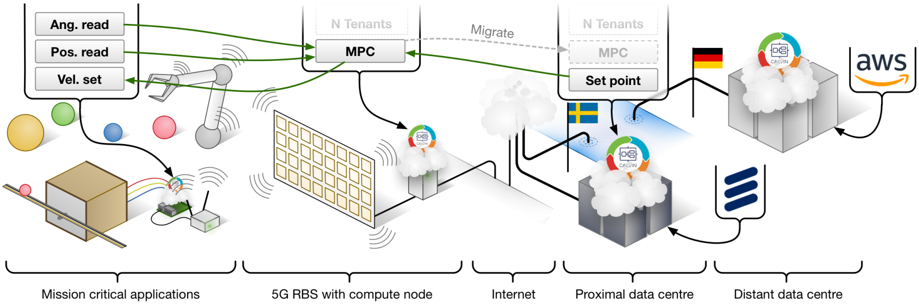


Figure 9.1.: System overview

cloud, offering the prospect of greater flexibility, reuse, availability, and reliability with lower latencies. When applications are implemented in a disaggregated manner, their execution can be distributed across the system's many nodes, migrated, and scaled to meet their individual objectives as well as that of the system as a whole. To adapt to, and prosper in, the Fog, applications will arguably have to adhere to a cloud-native paradigm.

The premise of this chapter is that deploying mission-critical applications over the cloud, with wireless devices, must arguably occur in conjunction with the availability of edge cloud resources, the flexibility of cloud-native applications, and the reliability and low latency of 5G. This thesis argues that such applications can operate in and make use of a distributed Fog computing but that there need to be relevant tools for them to be native to this context. There are many challenges and performance uncertainties in this premise. Therefore, the feasibility of deploying time-sensitive mission-critical applications and their performance when deployed on an actual Fog computing infrastructure is studied. The contributions of this chapter are:

- A state-of-the art Fog computing research test-bed aimed at the study of software autonomy and mission-critical applications.
- An empirical baseline evaluation of the plausibility of deploying latency-sensitive applications in the Fog computing.
- An empirical evaluation of the system's ability to dynamically reconfigure during run-time and the impact this has on the application.
- An empirical evaluation of the benefits of deploying latency-sensitive mission-critical applications at the edge, at the plant, and on a distant DC.

Section 9.1 covers the related work in the field and highlights the research gap. This is followed by a detailed account of the implemented test-bed in Section 9.2. Section 9.3 presents an example automatic control application which is used to evaluate the test-bed. Finally, Section 9.4 highlights the contributions of the chapter and points to new research directions.

9.1. RELATED WORK

In this section, related work and highlight the apparent research gaps left by the literature are surveyed. The works below cover both related attempts at research test-beds and experiments that pursue the viability of a Fog computing infrastructure.

Test-beds spanning UEs, wired and wireless networks, distributed cloud infrastructure and platforms are crucial instruments to realise and study the complexity of a Fog computing infrastructure. The literature contains a number of such attempts. The authors of [KBLG13] present the SAVI test-bed which is an edge cloud test-bed realising NFV with a Field-Programmable Gate Array (FPGA)-cloud. SAVI is used in the investigation of virtualising the wireless access network. Although comprehensive, the test-bed does not provide a general edge cloud implementation for cloud native applications nor does the implementation span multiple tiers of cloud resources, including the device. In [WTS$^+$16], a full test-bed using existing wireless technologies, IoT frameworks, and devices is deployed on an actual production line. The industrial applications targeted in the paper are not time-sensitive and the focus is on framework integration rather than system and application performance. The authors of [HGH$^+$16] implemented a rudimentary edge cloud test-bed to quantify the impact of Edge Computing on Mobile Applications using WiFi and the public 4G network. Their effort reveals significant latency and energy usage improvements compared to distant DCs. Additionally, the author has in a previous work [TCH16] studied the performance of cloud native applications on commercially available platforms in a smart city context.

iFogSim [GVDGB17] offers a platform for abstract modeling of resource management techniques in IoT and edge cloud environments, through simulation. In previous works the authors has developed several simulators for studying the dynamics of the edge cloud, culminating in [TPM$^+$17]. In the absence of a test-bed and in the pursuit of greater flexibility at a higher level of abstraction, simulators are valuable tools. However, we are at the point where a test-bed can be practically implemented and where a simulator cannot capture the complexities of a Fog computing infrastructure.

Other works attempt to characterise and profile the performance of different aspects of a Fog computing infrastructure. For example, the authors of [MSV14] evaluate a generic platform for industrial control, with respect to latency, throughput, and CPU load. Their focus is on the pros and cons of virtualisation in a 'Multi-core' environment rather than the 'cloud'. Similarly, [HK16] implements a water tank control process and evaluates latency over a virtual Software Programmable Logic Controllers (vSoftPLC) on top of LinuxRT. In both of these works, the system implementation rather than the plant under control is evaluated. In [HH15], the authors propose *Industrial automation as a cloud service*. In that paper the authors evaluate a system of time-sensitive control processes in a 1-tier distributed cloud environment. Latency compensation is modelled and redundancy with stability and smooth controller handover is achieved. True for all systems surveyed above, is that they are neither mission-critical nor time-sensitive at the scale addressed in this paper.

9.2. RESEARCH TEST-BED

The presented research test-bed consists of a 5G radio *transmitter* and multiple *receivers* (UEs) (constituting a 5G cell), PC-type *compute nodes* in DCs and adjacent to the radio transmitter (the RBS node), and reduced capacity *input-output* devices and *physical plants* at the radio receiver ends. Here, the physical plant can be a mechanical device that continuously performs a task, for example a robotic arm or an autonomous vehicle. It is assumed that the process which controls the plant is mission-critical and time-sensitive.

A platform with enough knowledge about the application to perform load balancing while allowing an application its own mobility within the network of compute nodes is desired. Further, there is a strong interplay between the edge cloud and the end user equipment such that an application can automatically scale on top of the cloud and provide fall back on local devices. The radio subsystem shall be capable of parallel, synchronised low latency communication with several receivers. Furthermore, the properties of the system's individual components are defined below. From here on the research test-bed is referred to as *the system*.

9.2.1. 5G

A 5G wireless system represents the next generation wireless infrastructure [SMS$^+$17]. The emerging focus of 5G is URLLC and mMTC where a large number of IoT devices, can reliably be served simultaneously at a low latency, $\leq 5ms$. These conditions cannot be replicated with current 802.11 or LTE systems.

A next generation wireless network also implies a deeper integration with associated cloud computing resources. On-demand resources are integrated into the RBS and access networks to off-load the back-haul and eliminate the latency overhead of traversing multiple networks and providers.

Massive MIMO is the emerging RAT for 5G. Fundamentally, massive MIMO is a MU-MIMO scheme, which can simultaneously communicate with multiple UEs on the same wireless resource. Additionally, on the RBS-side, massive MIMO operates with significantly more antennas than existing LTE-based RATs. Massive MIMO is typically configured with an order of magnitude more RBS-side antennas than simultaneously served UEs. Consequently, the system's spectral efficiency is a few orders of magnitude greater than existing RATs. The increased spectral efficiency can be used towards serving more simultaneous UEs, increase throughput, or realising mMTC, beyond what can be achieved with existing RATs.

A 5G wireless network is created using Lund Massive MIMO (LuMaMi). LuMaMi is a Massive MIMO test-bed at Lund University, Sweden. LuMaMi's scope and de-

Node	Device	Location
Plant	Raspberry Pi 3Bs	Plant adjacent
Edge	Intel Core i7 Desktop	LuMaMi adjacent
ERDC	Intel Core i7 VM	Lund, Sweden.
AWS	Intel Xeon VM	Frankfurt, Germany

Table 9.1.: Node types

tailed implementation are found in [MVL+17]. LuMaMi can be seen as one solitary 5G cell that can simultaneously communicate with twelve UEs.

LuMaMi is configured according to [TKR+17]. This configuration premiers low latency and reliability over high throughput. The modulation scheme is QPSK. The resulting throughput is 4.6 Mbps downlink and 9.1 Mbps uplink, per UE, which is more than sufficient to support the application. LuMaMi allows us to directly route traffic through the system, allowing us to place a compute node in the RBS.

9.2.2. FOG COMPUTING AND NETWORK

The system as a whole is tied together by a set of compute nodes joined by a network. A summary of the compute nodes is presented in Table 9.1. The system's network is conceptually configured as depicted in Figure 9.1. Adjacent to each plant is a Raspberry Pi. In order to sample and manipulate the plant, each Raspberry Pi has been equipped with a ADC/DAC shield. They are compute nodes and may service other functions in addition to interacting with the plant. Each Raspberry Pi is also connected to a 5G UE. The 5G cell is isolated in its own subnet. The subnet includes the wireless infrastructure, a Fog computing node, and plant nodes. The plant-adjacent Raspberry Pis are connected to the system's subnet over LuMaMi. The RBS node is adjacent to the LuMaMi RBS. It therefore connects directly to the RBS without traversing additional networks.

A router is connected to the cell's subnet and the larger DCs. The Ericsson Research Data Center (ERDC) resides in Lund, Sweden a few kilometres from the cell. ERDC is a research DC operated by Ericsson (Lund, Sweden), that is open to industrial and academic research efforts within the Wallenberg Autonoms Systems and Software Program (WASP). The VM is run on top of Open Stack Pike and the instance (a c4m16) has four Intel i7 cores registered by Linux as 1.6 GHz, and 16 GB of RAM. The AWS EC2 instance (a c4.large) is hosted on eu-central-1 (Frankfurt, Germany). This node has two Intel Xeon cores at 2.9 GHz and 8 GB of RAM. All cores are not used and therefore expect the latter system to be the best performing. The two VMs on ERDC and AWS connect to the subnet over VPN, allowing direct access between all compute nodes.

9.2.3. CLOUD NATIVE APPLICATION FRAMEWORK

In this work, a cloud-native application is defined as an application that has been disaggregated into logical and independent components connected in a *data-flow graph* and that is hosted on a PaaS framework. The PaaS and its resident applications ubiquitously operate over multiple geographically distributed and heterogeneous compute nodes. An application's data flow graph can be rerouted and extended in run-time, when for example adding a new feature. Additionally, the components shall be able to traverse the cloud and associate with and discover physical input-output devices if the application so requires.

Amazon's AWS, Microsoft's Azure, IBM's Bluemix, and Google's Cloud offer their own flavours of cloud native application platforms, ranging from SaaS to server-less FaaS. However, none of these providers allow their users to define logical data flows nor do they provide necessary performance guarantees. They are typically intended for lifting data from the edge and IoT-devices to the cloud. The services do not provide native support for closing logical loops from edge devices over the cloud and back to the edge device with guarantees on latency and consistency. For the purpose of building an open research test-bed, these platforms are proprietary and cannot be arbitrarily deployed and independently managed across an fog infrastructure. This is a requirement for the system in order to realise the view where one specification of the software can be deployed anywhere.

In this work, Calvin [PA15] is used as the cloud platform. Calvin is distributed, event-driven, server-less, and is based on a data-flow programming model. There are a number of such platforms for different workloads, such as: Nebula [ROCW14], Node-RED[nod], IEC 61499 [Vya11], and Naiad [MMI$^+$13]. The aforementioned systems are targeted for the IoT domain and cater for workloads varying from simple event-driven automation to high-throughput Hadoop jobs, but none of them have been built with the intention to run tight control loops over a dynamic distributed system.

Of the above, Calvin is most similar to Node-RED. However, while Node-RED emphasises the programming model and graphical tools, Calvin puts more focus on runtime dynamics and distributed deployment. The perspective of Calvin is well attuned to the presentation of the Distributed Data-flow model in [GBLL15], where a Distributed-NodeRED (DNR) extension is proposed. A notable operational difference is that DNR employs duplication to realise mobility while Calvin's code migration technique is arguably more efficient and is better suited for computationally intense applications [GBLL15]. Additionally, in Calvin, an application can be migrated using various optimisation criteria to provide for instance load balancing or jitter reduction.

Calvin is conceptually structured as follows. The operational units of Calvin are called *actors* (nodes in data-flow) while a *runtime* is an instantiation of the Calvin application environment on a device. In the present implementation there is a one-to-one mapping between Calvin runtimes and compute nodes and are therefore interchangeably refer to them simply as nodes. An actors' input and output messages, are known as *tokens*. A set of actors and their interconnections constitute an *application*. Each node independently schedules its resident actors in a round-robin manner.

Actors' states can be migrated and horizontally scaled across nodes. What constitutes an actors' state is defined by the developer. The Calvin framework can autonomously migrate and place actors to load-balance nodes and to meet its own performance goals. However, application owners can specify requirements for actors which tie them to a preferred runtime. For example, a sensor reading actor can be required to be placed on the node associated with the physical plant it is observing.

9.3. EVALUATION

In this section, the automatic control application that is the system's time-sensitive mission-critical application is presented. The controller is used to evaluate the performance and plausibility of the test-bed. That is, is mission-critical control over the Fog plausible and does the system exhibit the properties associated with an fog. This section begins by detailing the application in terms of the software implemented on top of Calvin and the plant that it controls. To evaluate its performance a set of experiments are designed to:

1. Reveal the characteristics of the system and the controller by establishing a baseline observation of the performance and behaviour of the controller over long time periods.

2. Verify the adaptability of the system by continuously migrating the controller actor across the system's nodes, in run-time.

3. Explore operating limits of the system's nodes and thus their relative advantage by deploying a well-tuned but computationally demanding and time-sensitive controller on the system.

In order to observe the system's performance potential, the study is limited to normal operating conditions. Notably, the connections to the DCs may at times degrade. It is assumed that these are infrequent, transient behaviours and do not consider how to handle them in this work.

9.3.1. CONTROL APPLICATION

A ball and beam process [Vir04] is the plant under control. The control has to be fast and there are clear limits set by physical constraints, yet enough flexibility for us to modify conditions to create various operating scenarios.Control is critical in that a failure may cause an unrecoverable state.

The objective of the ball and beam process is to expediently move to and maintain a ball on a set location (the set-point) on a beam. The length of the beam is 110 cm. The controller acts on the beam which is manipulated by a motor. The plant *outputs* the

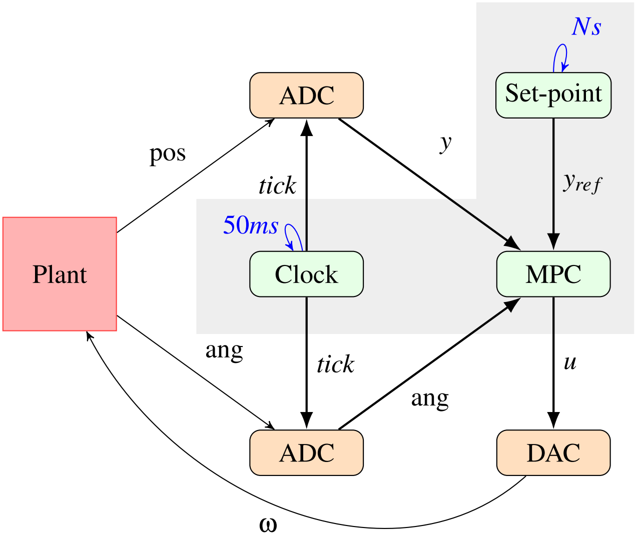


Figure 9.2.: Calvin MPC implementation

angle (α) of the beam and the position of the ball on the beam (x). The control signal, the *input* to the plant, is the radial velocity of the beam (ω). Naturally, the further the ball goes towards the end of the beam the higher the risk that the ball falls off the beam due to network delays, noisy sensors readings, and other system deficiencies.

An MPC [RM09] is used to implement a controller for the ball and beam. The application periodically samples the position of the ball and the angle of the beam. With every sample the MPC interacts with the plant by changing the velocity of the beam. To figure out what velocity to set the MPC performs a numerical optimisation where it takes into account a series of actions that will bring the ball into the desired state. In a primitive and general form this optimisation may be expressed as

$$\underset{u_0,u_1,\dots}{\text{minimize}} \quad \sum_{t=0}^{T-1} L(x_t,u_t) + \phi(x_T) \tag{9.1}$$

$$\text{s.t.} \quad x_{t+1} = f(x_t,u_t) \tag{9.2}$$

$$u_t \in U, x_t \in X \tag{9.3}$$

where $L(x_t,u_t)$ in Equation (9.1) is the *cost* function which puts a value to a state x_t and control input u_t at time step t. The function $\phi(x_T)$ assigns a different value specification to the final (or terminal) state x_T. The number of time steps T is called the horizon and specifies how far into the future the controller anticipates control actions. $f(x_t,u_t)$ in Equation (9.2) is the plant model which specifies the dynamics of how the system states evolve with the time step t. Equation (9.3) is a set of expressions which state limitations to the plant inputs and the state space. All of this is defined to set the operating conditions for the controller. The initial state, x_0, is drawn from measurements and state estimation. The end result is a quadratic program which is re-evaluated every sample.

For the optimisation, a dynamically linked binary created with the use of QPgen [Gis18] is used. The implementation is using sampling period of 50 ms (20 Hz). This choice is a reasonable trade-off between control performance and early observations of the system's latencies. Unlike the well known and often used Proportional Integral Derivative (PID) controller, the execution of an MPC is demanding and the execution time is not constant. The time it takes to solve its optimisation routine varies with disturbances acting on the system and where within its operating range it is currently acting. QPgen is an efficient solver and the optimisation problem has few variables, it will be apparent that the time it takes to find a solution can be considerable. Sometimes there is no solution or one is very hard to find. In such a case, the search is terminated after a fixed amount of iterations in the optimiser.

The processor and memory demands of the MPC optimisation may be considerable and the many ways of tuning it for various situations make it interesting as an fog application. A number of controllers can be designed for the same problem where computational and memory demands are weighted to performance, operational range, stability regions, and erratic behaviour. To handle the presence of plant and sensory noise a Kalman filter is introduced. The filter is also used to estimate the speed of the ball. A simple plant is employed, a basic MPC controller and a standard state estimator but even this rather simple case allows us to study and demonstrate behaviour in the experimental platform and the effects on the control.

The Calvin application graph for the MPC control loop is shown in Figure 9.2. The rounded rectangles represent individual components, implemented as actors, which are deployed onto the systems. The two Analog to Digital Converter (ADC) sensory actors adhere to component reuse and the principle idea that they need not be collocated. However, they are to be read jointly and therefore share a clock tick. The components within the grey area can be freely placed within the system. The ADCs (the position and angle sensors) and the Digital to Analog Converter (DAC) (the motor actuator) have an affinity to the plant-adjacent node.

Note that the scheduling and inter-node communication in the software platform introduce a significant amount of delay and jitter, which affects the performance of the controller and cause oscillations. In extension, much can be done in terms of model and controller tuning, state estimation, delay prediction, system improvements etc., but in this work the focus is on studying the overall performance of the platform. Remedies and improvements are left for later work, here the basic system is characterised and do not focus on details of control performance.

All related software runs on top of Linux and the Calvin runtime is launched using real-time priority and the POSIX FIFO scheduling policy. The edge nodes are allowed

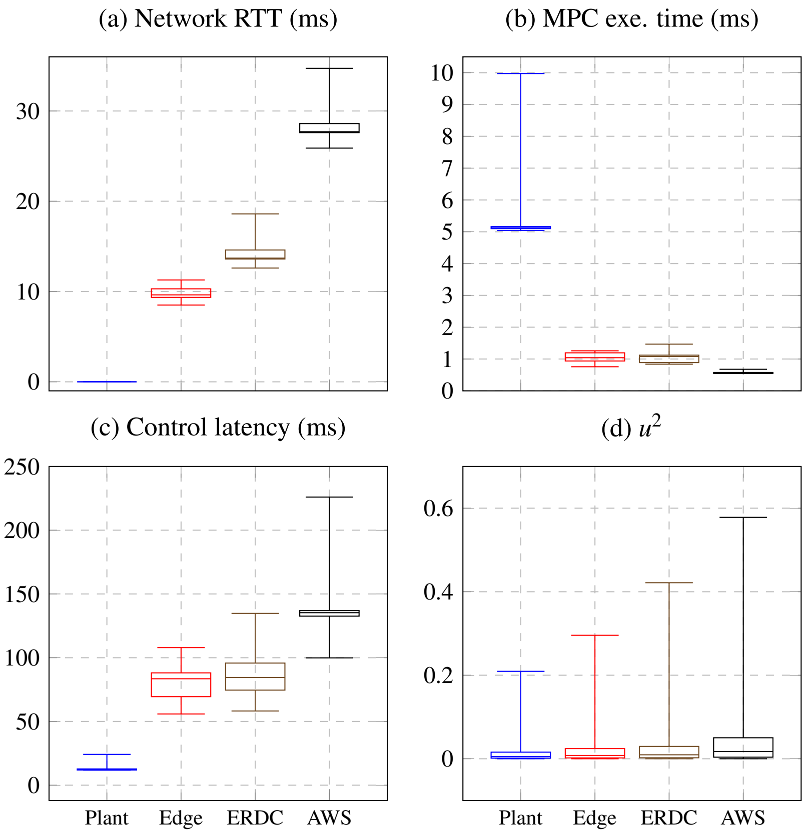


Figure 9.3.: Statistical summary of the MPC baseline measurements.

to take full advantage of this as it runs on bare metal. The kernels have not been patched with the PREEMPT_RT patch set [Fou18].

9.3.2. SYSTEM CHARACTERISTICS

To characterise and verify the basic functionality of the system the MPC is run on each of the nodes in Section 9.2.2. With each test, the MPC controls the beam for 60 minutes while alternating the set-point of the ball between the centre position and one side of the beam. To be robust in this experiment, the set-point is restricted with large margin to the end of the beam. On the other hand, the further out the ball is moved, the more the controller is put to work, which is something that is returned to in Section 9.3.4.

Figure 9.3a shows the RTTs from the Raspberry Pi at the plant to the other systems. Notably the wireless link realised with LuMaMi introduces a latency of 5*ms* one way, as made evident by the 10*ms* RTT between the plant and the edge node. 5G is pushing for even faster RTT but this is good radio link performance compared to commercially available alternatives.

Figure 9.3b shows the MPC execution time. A simple scenario is chosen where all nodes can execute the MPC with a significant margin. However, clearly, the Raspberry Pi at the plant is many times slower than the other systems. The AWS node is faster than edge and ERDC which is to be expected by the specification in 9.2.2. That graph shows that there are large outliers in terms of execution time at the plant. Even though the process is executed in real-time modem, recurrent extended system interruptions to the MPC on the Raspberry Pi are present. This could be the cause of these outliers. On the DCs real-time properties are not expected to apply outside the virtual machine and can therefore expect some outliers. Due to the short execution times, they are expected to be unlikely.

Figure 9.3c shows the aggregate latency from reading the position of the ball to applying the control signal (i.e. adjusting the velocity of the beam). This is an important measure because the controller is designed with the assumption that the input to output is instantaneous and hence, that the state of the system has not changed when the control signal is applied. That figures shows that the differences in delay are not as pronounced as in Figure 9.3a. The execution times in Figure 9.3b and the network latency in Figure 9.3a are not the only contributors to the control latency. This tells us that a significant proportion of the delay in the system is introduced by the software platform or application design and not the network. The system dynamics causing the increasing variance in Figure 9.3c is also an interesting topic for further research. The effect on the process due to these properties are visualised in Figure 9.3d where it can be seen that the energy of the control signal u increases the further the process moves from the plant. Notice that the AWS node performs well but network delays causes it to exhibit a larger mean and variance in the control signal. Such an effect can be part of the heuristics when deciding where to place control in the edge cloud.

9.3.3. SYSTEM ADAPTABILITY

At this point it is established that a controller can successfully be implemented on the edge cloud test-bed and studied characteristics in terms of execution times, latencies and jitter. Essential to the mutability of the system is its ability to migrate applications and actors to respond to the applications' and the infrastructure's changing objectives. During a migration the Calvin cluster performs the necessary modification of the network communication path, recreates the actor at the target node, copies state and handles the transition of token queues. Although the actor moves point-to-point, changing the communication paths may involve many nodes in the cluster. This perspective is now explored by way of relocating the MPC amongst the nodes while balancing and repositioning the ball as in Section 9.3.2.

In Figure 9.4 the MPC actor is continuously randomly migrated across the four compute nodes, in run-time. When doing this, the system must ensure to keep the Kalman filter, the set-point, the previous states, and tracing meta data intact. Delays, data loss or duplication, and incorrect state transfer negatively impacts the control performance. Figure 9.4c shows the placement of the actor in time. Figure 9.4b and Figure 9.4a show the controller inputs and outputs respectively.

Figure 9.4b shows that the process is stable and is able to operate without interruptions. The ball stays on the beam and close to the desired position. Figure 9.4a confirms what is presented in Figure 9.3d, i.e., the control signal increases as a function of the distance to the plant. Set-point changes are clearly visible has high peaks but the migrations in Figure 9.4c are not evident in Figure 9.4a nor Figure 9.4b. However, the peak in the control signal near the set-point change after 650 seconds is likely caused by a coinciding migration. In its current form, the system is not aware of when or to where a migration will occur nor is there any effort to attempted to mitigate its potential effects.

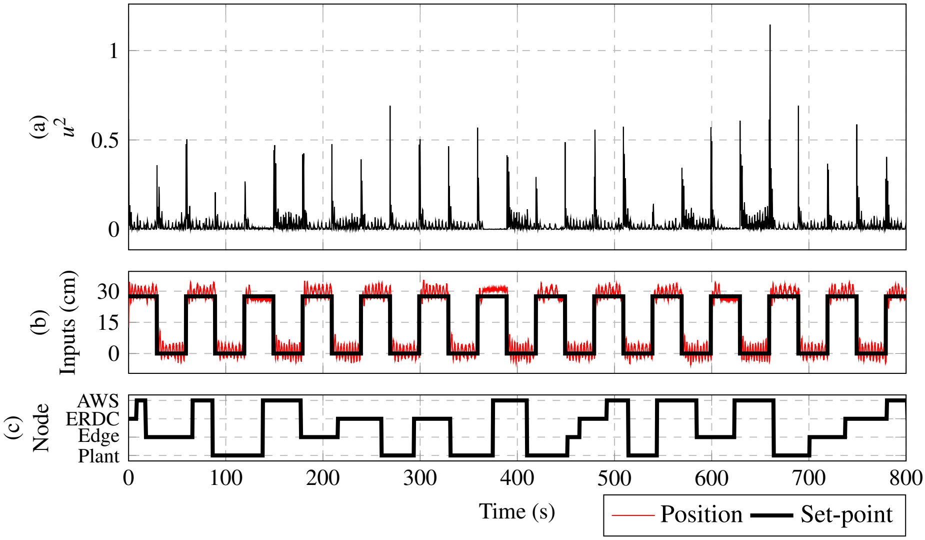


Figure 9.4.: Time-series of MPC being randomly migration between the system's four nodes.

9.3.4. TIGHTENED CONSTRAINTS

Below is an example use case that demonstrates how the controller takes advantage of the fog. Here, relative to the previous example, the control signal to the plant is constrained. Albeit being a synthetic exercise, it is not an unreasonable action since limits in control signal are commonly used to, for instance, reduce actuator wear and to avoid non-linear parts of the operating range. To make the associated optimisation problem harder the ball is moved just short of the end of the beam. In combination with the constraints this will cause a higher load on the MPC host node. Small disturbances may cause the ball to fall off.

The experiment shown in Figure 9.5 applies this configuration. The graphs show time series of the inputs to the controller, the execution time of the MPC, and the total latency from input to output of the measured nodes. The blue circles mark occasions when the MPC fails to find a solution.

As constraints are tightened it becomes increasingly hard to find a control signal sequence which moves the system from the present state to the target state, while staying within the bounds. This manifests in longer execution times for the optimisation. As seen in the execution times in Figure 9.5, when the system has settled around a set-point, the optimisation is easy to solve and computationally light. In these situations the controller on the plant performs well. Latency and jitter of the networked controllers may cause them to deviate more from the set-point.

Eventually however, the computational limitations of the plant cause the ball to falls off the beam as a set-point change occurs. Note that the plant is not unable to move the ball to the position at the end of the beam but eventually, model errors and noise become too large for it to handle. In contrast, the edge node is able to operate without failure. Also note that on the AWS instance, despite its computational capacity, the controller fails to cope with the resulting latency and system jitter - the ball falls off.

The execution time at the plant when the MPC fails to find a feasible solution, is close to an order of magnitude that of the sampling time, represented by a line which extends well beyond the top of the graph. This is representative of the computational problems experienced at the plant due to noise during a set-point change. In contrast, an equal number of iterations consumes 80 ms on the edge node and only 40 ms on the AWS.

With the position closer to the end of the beam and with reduced range in the control output signal, the controller repeatedly experiences non-trivial situations which require additional iterations of the optimisation loop. At times, noisy readings make a tough situation even worse and there may seemingly be no solution that keeps the ball on the beam. When a new evaluation can be made quickly enough then the state of the system may still be such that the ball can be saved. On the AWS node the communication delays increase the frequency of these tough situations and as a result there are repeated problems of finding a solution. However, the speed of the AWS node allows it to cope with many of these situations since the combined execution and communication delay is much less than the compute time at the plant. Only the edge node is in a position where it is able to handle the full range of the system noise.

9.4. CONCLUSIONS

In this chapter, a Fog computing research test-bed for an IoT and heterogeneous cloud environment was presented. The test-bed deployed an automatic control application on the test-bed to act on a time-sensitive and mission-critical process. The controller's viability, performance, and system characteristics were evaluated. The evaluation

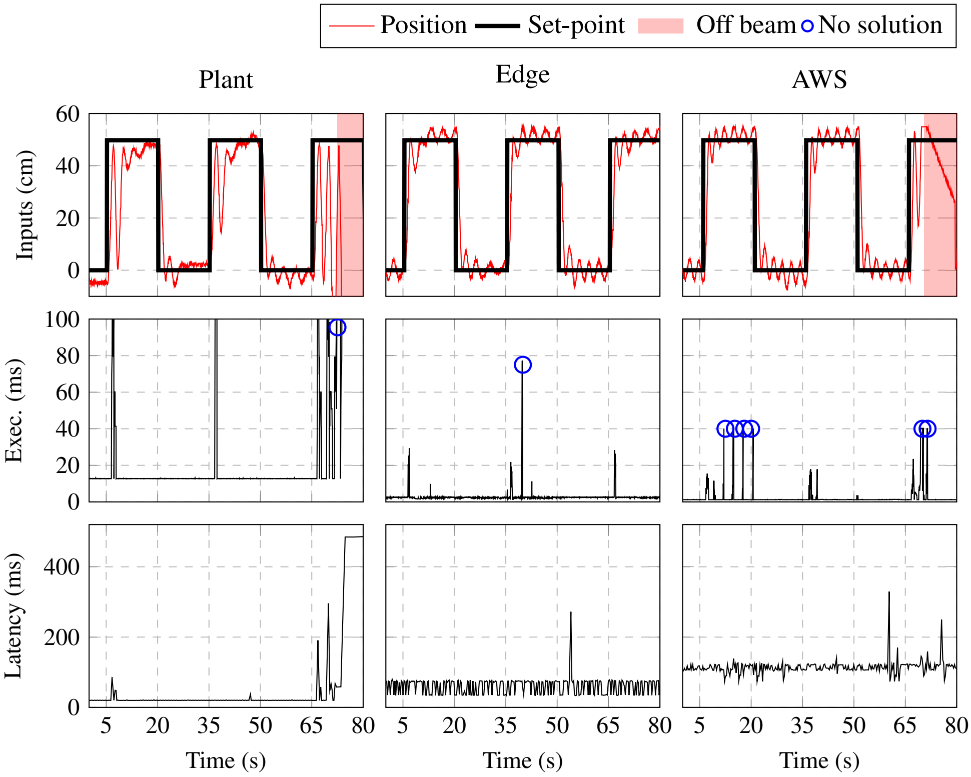


Figure 9.5.: Time-series of experiments run with tightened constraints on the plant, egde, and AWS nodes.

shows that cloud native control loops can viably be deployed on the edge cloud. The system operates at a sampling rate of 20 Hz but there is potential for this to be pushed further in the near future. It was also shown that the controller can benefit from the edge cloud and that the system and the placement of the controller can be dynamically reconfigured in run-time without strictly sacrificing stability.

To further improve the software platform, there remain work to be done on actor scheduling, overheads of message passing, and synchronisation of such a tightly connected system in an inherently uncertain cloud environment. With improvements, the requirements on the application can be increased and the system to move towards anticipated future applications which require such a system to be user friendly, self-adaptive, resilient, high performing, and deliver low latency and low jitter.

Importantly, it can be conclude that the test-bed is observable which enables us to continuously operate mission-critical applications while performing targeted experiments. The test-bes is now in such a state that allows broad experimental research of the interplay between the application and the underlying platform. Continuations to the work targets novel and established techniques within the fields of control theory, distributed systems, and software engineering.

The project source code is available on GitHub [ST18].

Bibliography

[83818] IEEE approved draft standard for adoption of openfog reference architecture for fog computing. *IEEE P1934/D2.0, April 2018*, pages 1–175, Jan 2018.

[A$^+$06] ZigBee Alliance et al. Zigbee specification, 2006.

[ABB] ABB. IRB 140 product specifications. `http://new.abb.com/products/robotics/industrial-robots/irb-140`, accessed on 2017-03-31.

[ADA$^+$15] Adnan Aijaz, Mischa Dohler, A Hamid Aghvami, Vasilis Friderikos, and Magnus Frodigh. Realizing the tactile Internet: Haptic communications over next generation 5G cellular networks. *arXiv preprint arXiv:1510.02826*, 2015.

[ADJ$^+$10] Sharad Agarwal, John Dunagan, Navendu Jain, Stefan Saroiu, Alec Wolman, and Harbinder Bhogan. Volley: Automated data placement for geo-distributed cloud services. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 17–32, 2010.

[AJ00] M. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site. *Network, IEEE*, 14(3):30–37, May 2000.

[Ari09] Erdal Arikan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on Information Theory*, 55(7):3051–3073, 2009.

[AS07] C. Adam and R. Stadler. Service middleware for self-managing large-scale systems. *IEEE Transactions on Network and Service Management*, 4(3):50–64, Dec 2007.

[AS14] Arif Ahmed and Abadhan Saumya Sabyasachi. Cloud computing simulators: A detailed survey and future direction. In *Advance Computing Conference (IACC), 2014 IEEE International*, pages 866–872. IEEE, 2014.

[ÅSTK18] Karl-Erik Årzén, Per Skarin, William Tärneberg, and Maria Kihl. Control of the edge cloud-an mpc example. In *1st International Workshop on Trustworthy and Real-time Edge Computing for Cyber-Physical Systems (Nashville, TN, USA)*. IEEE, 2018.

[AZTS18] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2018.

[BAB12] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[Bal02] Garth H Ballantyne. Robotic surgery, telerobotic surgery, telepresence, and telementoring. *Surgical Endoscopy and Other Interventional Techniques*, 16(10):1389–1402, 2002.

[Bas12] Salman A. Baset. Cloud slas: Present and future. *SIGOPS Oper. Syst. Rev.*, 46(2):57–66, July 2012.

[BB12] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012.

[BBEK11] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. Sumo–simulation of urban mobility. In *The Third International Conference on Advances in System Simulation (SIMUL 2011), Barcelona, Spain*, 2011.

[BC98] Paul Barford and Mark Crovella. Generating representative web workloads for network and server performance evaluation. *ACM SIGMETRICS Performance Evaluation Review*, 26(1):151–160, 1998.

[BCF$^+$12] Ofer Biran, Antonio Corradi, Mario Fanelli, Luca Foschini, Alexander Nus, Danny Raz, and Ezra Silvera. A stable network-aware vm placement for cloud systems. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 498–506. IEEE Computer Society, 2012.

[BCH13] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3):1–154, 2013.

[BDPW11] Peter Bosch, Alessandro Duminuco, Fabio Pianese, and Thomas L Wood. Telco clouds and virtual telco: Consolidation, convergence, and beyond. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 982–988. IEEE, 2011.

[Bed14] Paul Bedell. *Cellular Networks: Design and Operation: a Real World Perspective*. 2014.

[Bet01] Christian Bettstetter. Smooth is better than sharp: A random mobility model for simulation of wireless networks. In *Proceedings of the 4th ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '01, pages 19–27, New York, NY, USA, 2001. ACM.

[BFF$^+$10] Peter Bodik, Armando Fox, Michael J. Franklin, Michael I. Jordan, and David A. Patterson. Characterizing, modeling, and generating workload spikes for stateful services. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 241–252, New York, NY, USA, 2010. ACM.

[BG14] Andrew Banks and Rahul Gupta. Mqtt version 3.1. 1. *OASIS Standard*, 2014.

[BGW10] Sem Borst, Varun Gupt, and Anwar Walid. Distributed caching algorithms for content distribution networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[BH07] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12), 2007.

[BHL$^+$14] Federico Boccardi, Robert W Heath, Angel Lozano, Thomas L Marzetta, and Petar Popovski. Five disruptive technology directions for 5g. *IEEE Communications Magazine*, 52(2):74–80, 2014.

[BMNZ14] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. Fog computing: A platform for Internet of things and analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*. Springer, 2014.

[BS10] Sean Kenneth Barker and Prashant Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 35–46. ACM, 2010.

[BSHD14] E. Björnson, L. Sanguinetti, J. Hoydis, and M. Debbah. Designing multi-user MIMO for energy efficiency: When is massive MIMO the answer? In *2014 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 242–247, April 2014.

[CCY$^+$15] Aleksandra Checko, Henrik L Christiansen, Ying Yan, Lara Scolari, Georgios Kardaras, Michael S Berger, and Lars Dittmann. Cloud ran for mobile networks—a technology overview. *IEEE Communications surveys & tutorials*, 17(1):405–426, 2015.

[CIL$^+$15] Houssem-Eddine Chihoub, Shadi Ibrahim, Yue Li, Gabriel Antoniu, María Pérez, and Luc Bougé. Exploring energy-consistency trade-offs in Cassandra cloud storage system. In *SBAC-PAD'15 — The 27th International Symposium on Computer Architecture and High Performance Computing*. DBLP, 2015.

[CKP03] Alberto Caprara, Hans Kellerer, and Ulrich Pferschy. Approximation schemes for ordered vector packing problems. *Naval Research Logistics (NRL)*, 50(1):58–69, 2003.

[CKW13] Hung-Lin Chi, Shih-Chung Kang, and Xiangyu Wang. Research trends and opportunities of augmented reality applications in architecture, engineering, and construction. *Automation in construction*, 33:116–122, 2013.

[clo18a] September 2018.

[Clo18b] Cloudflare. Cloudflare Workers. `https://www.cloudflare.com/products/cloudflare-workers/`, 2018.

[CMTD13] G. Copil, D. Moldovan, H. Truong, and S. Dustdar. Sybl: An extensible language for controlling elasticity in cloud applications. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)(CCGRID)*, volume 00, pages 112–119, 05 2013.

[CRB$^+$11] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[CSW$^+$12] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade. Autonomic placement of mixed batch and transactional workloads. *IEEE Transactions on Parallel and Distributed Systems*, 23(2):219–231, Feb 2012.

[DB11] Schahram Dustdar and Kamal Bhattacharya. The social compute unit. *IEEE Internet Computing*, 15(3):64–69, 2011.

[DB16] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[DDMM15] Soufiene Djahel, Ronan Doolan, Gabriel-Miro Muntean, and John Murphy. A communications-oriented perspective on traffic management systems for smart cities: challenges and innovative approaches. *Communications Surveys & Tutorials, IEEE*, 17(1):125–151, 2015.

[DGST11] Schahram Dustdar, Yike Guo, Benjamin Satzger, and Hong-Linh Truong. Principles of elastic processes. *IEEE Internet Computing*, 15(5):66–71, 2011.

[DTT$^+$16] Jonas Dürango, William Tärneberg, Luis Tomas, Johan Tordsson, Maria Kihl, and Martina Maggio. A control theoretical approach to non-intrusive geo-replication for cloud services. In *Proc. Conference on Decision and Control (Las Vegas, NV, USA)*. IEEE, 2016.

[EEsS14] Ayman ElNashar, Mohamed El-saidny, and Mahmoud Sherif. *Design, Deployment and Performance of 4G-LTE Networks: A Practical Approach*. Wiley, 1 edition, 5 2014.

[Fet14] Gerhard P Fettweis. The tactile Internet: Applications and challenges. *IEEE Vehicular Technology Magazine*, 9(1):64–70, 2014.

[FLJ13] Ghofrane Fersi, Wassef Louati, and Maher Ben Jemaa. Distributed hash table-based routing and data management in wireless sensor networks: a survey. *Wireless networks*, 19(2):219–236, 2013.

[Fou18] Linux Foundation. Real-time linux. `https://wiki.linuxfoundation.org/realtime/start`, 2018.

[FRF09] AJ. Fehske, F. Richter, and G.P. Fettweis. Energy efficiency improvements through micro sites in cellular mobile radio networks. In *GLOBECOM Workshops, 2009 IEEE*, pages 1–5, Nov 2009.

[FTD03] C. Fraleigh, F. Tobagi, and C. Diot. Provisioning ip backbone networks to support latency sensitive traffic. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 1, pages 375–385 vol.1, March 2003.

[Gar] Gartner. Hype cycle for cloud computing, 2018.

[GBLL15] N. K. Giang, M. Blackstock, R. Lea, and V. C. M. Leung. Developing iot applications in the fog: A distributed dataflow approach. In *2015 5th International Conference on the Internet of Things (IOT)*, pages 155–162, Oct 2015.

[GGD$^+$07] Victor Gradinescu, Cristian Gorgorin, Raluca Diaconescu, Valentin Cristea, and Liviu Iftode. Adaptive traffic lights using car-to-car communication. In *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*, pages 21–25. IEEE, 2007.

[GHMP08] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM Computer Communication Review*, 39(1):68–73, 2008.

[Gis18] Pontus Giselsson. Qpgen. `http://www.control.lth.se/user/pontus.giselsson/qpgen/`, 2018.

[GLPL14] Mario Gerla, Eun-Kyu Lee, Giovanni Pau, and Uichin Lee. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 241–246. IEEE, 2014.

[GNT$^+$06] Leonidas Georgiadis, Michael J Neely, Leandros Tassiulas, et al. Resource allocation and cross-layer control in wireless networks. *Foundations and Trends® in Networking*, 1(1):1–144, 2006.

[GVDGB17] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the Internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.

[HAS$^+$16] Christian Hoymann, David Astely, Magnus Stattin, Gustav Wikstrom, Jung-Fu Cheng, Andreas Hoglund, Mattias Frenne, Ricardo Blasco, Joerg Huschke, and Fredrik Gunnarsson. LTE release 14 outlook. *IEEE Communications Magazine*, 54(6):44–49, 2016.

[HGH$^+$16] Wenlu Hu, Ying Gao, Kiryong Ha, Junjue Wang, Brandon Amos, Zhuo Chen, Padmanabhan Pillai, and Mahadev Satyanarayanan. Quantifying the impact of edge computing on mobile applications. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '16, pages 5:1–5:8, New York, NY, USA, 2016. ACM.

[HH15] Tamir Hegazy and Mohamed Hefeeda. Industrial automation as a cloud service. *IEEE Transactions on Parallel and Distributed Systems*, 26(10):2750–2763, 2015.

[HK16] Christian Horn and Jörg Krüger. Feasibility of connecting machinery and robots to industrial control services in the cloud. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4. IEEE, 2016.

[HLR$^+$08] Thomas R Henderson, Mathieu Lacage, George F Riley, C Dowell, and JB Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 2008.

[HM98] Fred Howell and Ross McNab. Simjava: A discrete event simulation library for java. *Simulation Series*, 30:51–56, 1998.

[HPL15] Jia Hu, Byungkyu Brian Park, and Young-Jae Lee. Coordinated transit signal priority supporting transit progression under connected vehicle technology. *Transportation Research Part C: Emerging Technologies*, 55:393–408, 2015.

[HPP14] Jia Hu, Byungkyu Park, and A. Parkany. Transit signal priority with connected vehicle technology. *Transportation Research Record: Journal of the Transportation Research Board*, 2418:20–29, 2014.

[HS06] Peter F Hokayem and Mark W Spong. Bilateral teleoperation: An historical survey. *Automatica*, 42(12):2035–2057, 2006.

[HTÖ$^+$16] Stefan Höst, William Tärneberg, Per Ödling, Maria Kihl, Marco Savi, and Massimo Tornatore. Network requirements for latency-critical services in a full cloud deployment. In *Proc. SoftCom (Split, Croatia)*. IEEE, 2016.

[HWW⁺16] Bernd Holfeld, Dennis Wieruch, Thomas Wirth, Lars Thiele, Shehzad Ali Ashraf, Jorg Huschke, Ismet Aktas, and Junaid Ansari. Wireless communication for factory automation: an opportunity for LTE and 5G systems. *IEEE Communications Magazine*, 54(6):36–43, 2016.

[JPE⁺11] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible. Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement. In *IEEE International Conference on Services Computing (SCC)*, pages 72–79, July 2011.

[KBK12] Dzmitry Kliazovich, Pascal Bouvry, and SameeUllah Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012.

[KBLG13] Joon-Myung Kang, Hadi Bannazadeh, and Alberto Leon-Garcia. Savi testbed: Control and management of converged virtual ict resources. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 664–667. IEEE, 2013.

[KET⁺13] Maria Kihl, Erik Elmroth, Johan Tordsson, Karl-Erik Årzen, and Anders Robertsson. The challenge of cloud control. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, 2013.

[KFCM12] B. Kantarci, L. Foschini, A. Corradi, and H.T. Mouftah. Inter-and-intra data center vm-placement for energy-efficient large-scale cloud systems. In *Globecom Workshops (GC Wkshps)*, pages 708–713, Dec 2012.

[KK04] Magnus Karlsson and Christos Karamanolis. Choosing replica placement heuristics for wide-area systems. In *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, pages 350–359. IEEE, 2004.

[KMÅHR14] Cristian Klein, Martina Maggio, Karl-Erik Årzén, and Francisco Hernández-Rodriguez. Brownout: Building more robust cloud applications. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014.

[Kov12] Dejan Kovachev. Framework for computation offloading in mobile cloud computing. *Int. J. of Interactive Multimedia and Artificial Intelligence*, 1(7):6–15, 2012.

[KRDZ10] Sajib Kundu, Raju Rangaswami, Kaushik Dutta, and Ming Zhao. Application performance modeling in a virtualized environment. In *IEEE 16th International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–10. IEEE, 2010.

[KTÖ+15] Jakub Krzywda, William Tärneberg, Per-Olov Östberg, Maria Kihl, and Erik Elmroth. Telco clouds: Modelling and simulation. In *Proc. CLOSER (Lisbon, Portugal)*. INSTICC, 2015.

[LETM14] Erik G Larsson, Ove Edfors, Fredrik Tufvesson, and Thomas L Marzetta. Massive mimo for next generation wireless systems. *IEEE Communications Magazine*, 52(2):186–195, 2014.

[LM18] Yaniv Leviathan and Yossi Matias. Google duplex: An ai system for accomplishing real-world tasks over the phone. https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html?m=1, May 2018.

[LMS12] Long Bao Le, Eytan Modiano, and Ness B Shroff. Optimal control of wireless networks with finite buffers. *IEEE/ACM Transactions on Networking (TON)*, 20(4):1316–1329, 2012.

[LSYR11] Nikolaos Laoutaris, Michael Sirivianos, Xiaoyuan Yang, and Pablo Rodriguez. Inter-datacenter bulk transfers with netstitcher. *ACM SIGCOMM Computer Communication Review*, 41(4):74–85, 2011.

[LTKE19] Lars Larsson, William Tärneberg, Cristian Klein, and Erik Elmroth. Quality-elasticity: Improved resource utilization, throughput, and response times via adjusting output quality to current operating conditions. In *Submitted to International Conference on Autonomic Computing (ICAC) (Umeå, Sweden)*. IEEE, 2019.

[LTKR16] Zheng Li, William Tärneberg, Maria Kihl, and Anders Robertsson. Using a predator-prey model to explain variations of cloud spot price. In *Proc. CLOSER (Rome, Italy)*. INSTICC, 2016.

[Man15] Zoltán Ádám Mann. Allocation of virtual machines in cloud data centers; a survey of problem models and optimization algorithms. *ACM Comput. Surv.*, 48(1):11:1–11:34, August 2015.

[Mar10] Thomas L Marzetta. Noncooperative cellular wireless with unlimited numbers of base station antennas. *IEEE Transactions on Wireless Communications*, 9(11):3590–3600, 2010.

[Mat08] Norm Matloff. Introduction to discrete-event simulation and the simpy language. *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, 2:2009, 2008.

[MDTE15] Amardeep Mehta, Jonas Durango, Johan Tordsson, and Erik Elmroth. Online spike detection in cloud workloads. In *2015 IEEE International Conference on Cloud Engineering (IC2E)*, pages 446–451, March 2015.

[Mil14] Victor Millnert. Quality-latency trade-off in bilateral teleoperation. Master's Thesis ISRN LUTFD2/TFRT--5951--SE, Department of Automatic Control, Lund University, Sweden, November 2014.

[MMI$^+$13] Derek G Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: a timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 439–455. ACM, 2013.

[MOD11] Stuart Mitchell, Michael OSullivan, and Iain Dunning. Pulp: a linear programming toolkit for python. *The University of Auckland, Auckland, New Zealand, http://www. optimization-online. org/DB_FILE/2011/09/3178. pdf*, 2011.

[Mos15] Katia Moskvitch. Tactile internet: 5g and the cloud on steroids. *Engineering & Technology*, 10(4):48–53, 2015.

[MPA$^+$18] Meiyi Ma, Sarah Preum, Mohsin Ahmed, William Tärneberg, Abdeltawaband Hendawi, and John Stankovic. Smart city, data sets, modeling, decision making, real-time, integrating services. *Sumbitted ACM Transactions on Cyber-Physical Systems*, Feb 2018.

[MPT$^+$16] Meiyi Ma, Sarah Masud Preum, William Tärneberg, Mohsin Ahmed, Matthey Ruiters, and John Stankovic. Detection of runtime conflicts among services in smart cities. In *Proc. International Conference on Smart Computing (St. Louis, MO, USA)*. IEEE, 2016.

[MPZ10] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9, March 2010.

[MRkFS13] Yaser Maddahi, Ramhuzaini A. Rahman, Wai keung Fung, and Nariman Sepehri. Effect of network quality on performance of bilateral teleoperated hydraulic actuators: a comparative study. *Control and Intelligent Systems*, 41, 2013.

[MSV14] Nesredin Mahmud, Kristian Sandström, and Aneta Vulgarakis. Evaluating industrial applicability of virtualization on a distributed multicore platform. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, pages 1–8. IEEE, 2014.

[MTK$^+$16] Amardeep Mehta, William Tärneberg, Cristian Klein, Johan Tordsson, Maria Kihl, and Erik Elmroth. How beneficial are intermediate layer data centers in mobile edge networks? In *Proc. Foundations and Applications of Self* Systems (Augsburg, Germany)*. IEEE, 2016.

[MVL$^+$16] Steffen Malkowsky, Joao Vieira, Liang Liu, Paul Harris, Karl Nieman, Nikhil Kundargi, Ian Wong, Fredrik Tufvesson, Viktor Öwall, and Ove Edfors. The world's first real-time testbed for massive mimo: Design, implementation, and validation. *arXiv preprint arXiv:1701.01161*, 2016.

[MVL$^+$17] Steffen Malkowsky, Joao Vieira, Liang Liu, Paul Harris, Karl Nieman, Nikhil Kundargi, Ian C Wong, Fredrik Tufvesson, Viktor Öwall, and Ove Edfors. The world's first real-time testbed for Massive MIMO: Design, implementation, and validation. *IEEE Access*, 5:9073–9088, 2017.

[NML08] Michael J Neely, Eytan Modiano, and Chih-Ping Li. Fairness and optimal stochastic control for heterogeneous networks. *IEEE/ACM Transactions on Networking (TON)*, 16(2):396–409, 2008.

[NND$^+$17] Matteo Nardelli, Stefan Nastic, Schahram Dustdar, Massimo Villari, and Rajiv Ranjan. Osmotic flow: Osmotic computing+ iot workflow. *IEEE Cloud Computing*, 4(2):68–75, 2017.

[nod] Node-RED at https://nodered.org.

[NS91] Günter Niemeyer and J-JE Slotine. Stable adaptive teleoperation. *IEEE Journal of oceanic engineering*, 16(1):152–162, 1991.

[OIY$^+$10] Simon Ostermann, Alexandria Iosup, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. A performance analysis of EC2 cloud computing services for scientific computing. pages 115–131. Springer, 2010.

[PA15] Per Persson and Ola Angelsmark. Calvin–merging cloud and IoT. *Procedia Computer Science*, 52:210–217, 2015.

[PBM$^+$07] Tim Püschel, Nikolay Borissov, Mario Macías, Dirk Neumann, Jordi Guitart, and Jordi Torres. Economically enhanced resource management for internet service utilities. In *Web Information Systems Engineering–WISE 2007*, pages 335–348. Springer, 2007.

[PCW12] Yong-Lae Park, Bor-Rong Chen, and Robert J Wood. Design and fabrication of soft artificial skin using embedded micro channels and liquid conductors. *IEEE Sensors Journal*, 12(8):2711–2718, 2012.

[PDG$^+$16] Maria Rita Palattella, Mischa Dohler, Alfredo Grieco, Gianluca Rizzo, Johan Torsner, Thomas Engel, and Latif Ladid. Internet of things in the 5G era: Enablers, architecture, and business models. *IEEE Journal on Selected Areas in Communications*, 34(3):510–527, 2016.

[PM15] Alessandro Vittorio Papadopoulos and Martina Maggio. Virtual machine migration in cloud infrastructures: Problem formalization and policies proposal. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 6698–6705. IEEE, 2015.

[PMF$^+$03] Konstantina Papagiannaki, Sue Moon, Chuck Fraleigh, Patrick Thiran, and Christophe Diot. Measurement and analysis of single-hop delay on an ip backbone network. *Selected Areas in Communications, IEEE Journal on*, 21(6):908–921, 2003.

[PP12] Federica Paganelli and David Parlanti. A DHT-based discovery service for the Internet of Things. *Journal of Computer Networks and Communications*, 2012, 2012.

[PRLE17] Hemanth Prabhu, Joachim Neves Rodrigues, Liang Liu, and Ove Edfors. 3.6 a 60pj/b 300mb/s 128× 8 massive mimo precoder-detector in 28nm fd-soi. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, pages 60–61. IEEE, 2017.

[PS05] Chandrakant D Patel and Amip J Shah. Cost model for planning, development and operation of a data center, 2005.

[PSLJ11] Balaji Palanisamy, Aameek Singh, Ling Liu, and Bhushan Jain. Purlieus: Locality-aware resource allocation for mapreduce in a cloud. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 58:1–58:11, New York, NY, USA, 2011. ACM.

[QJM$^+$09] Xiao Qin, Hong Jiang, Adam Manzanares, Xiaojun Ruan, and Shu Yin. Dynamic load balancing for i/o-intensive applications on clusters. *ACM Transactions on Storage (TOS)*, 5(3):9, 2009.

[QKP$^+$09] A. Quiroz, Hyunjoo Kim, M. Parashar, N. Gnanasambandam, and N. Sharma. Towards autonomic workload provisioning for enterprise grids and clouds. In *Grid Computing, 2009 10th IEEE/ACM International Conference on*, pages 50–57, Oct 2009.

[QPV01] Lili Qiu, V.N. Padmanabhan, and G.M. Voelker. On the placement of web server replicas. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1587–1596 vol.3, 2001.

[RBL$^+$09] Benny Rochwerger, David Breitgand, Eliezer Levy, Alex Galis, Kenneth Nagin, Ignacio Martín Llorente, Ruben Montero, Yaron Wolfsthal, Erik Elmroth, Juan Caceres, et al. The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):4–1, 2009.

[Reg11] Antonio Regalado. Who coined 'cloud compuitng'? MIT Technology Review, October 2011.

[RGE02] Pavlin Radoslavov, Ramesh Govindan, and Deborah Estrin. Topology-informed internet replica placement. *Computer Communications*, 25(4):384–392, 2002.

[RH10] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and Tools for Network Simulation*, pages 15–34. Springer, 2010.

[RLGPC$^+$99] Arcadio Reyes-Lecuona, E González-Parada, E Casilari, JC Casasola, and A Diaz-Estrella. A page-oriented www traffic model for wireless system simulations. In *Proceedings ITC*, volume 16, pages 1271–1280, 1999.

[RM09] J.B. Rawlings and D.Q. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill Pub., 2009.

[RMK$^+$09] Venugopalan Ramasubramanian, Dahlia Malkhi, Fabian Kuhn, Mahesh Balakrishnan, Archit Gupta, and Aditya Akella. On the treeness of Internet latency and bandwidth. *SIGMETRICS Perform. Eval. Rev.*, 37(1):61–72, June 2009.

[ROCW14] Mathew Ryden, Kwangsung Oh, Abhishek Chandra, and Jon Weissman. Nebula: Distributed edge cloud for data intensive computing. In *2014 IEEE International Conference on Cloud Engineering (IC2E)*, pages 57–66. IEEE, 2014.

[RTG$^+$12] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Towards understanding heterogeneous clouds at scale: Google trace analysis. Technical report, Intel Science and Technology Center for Cloud Computing, 2012.

[RTM17] Jackeline Rios-Torres and Andreas A Malikopoulos. A survey on the coordination of connected and automated vehicles at intersections and merging at highway on-ramps. *IEEE Transactions on Intelligent Transportation Systems*, 18(5):1066–1077, 2017.

[RYS10] J. Ryu, L. Ying, and S. Shakkottai. Back-pressure routing for intermittently connected networks. In *2010 Proceedings IEEE INFOCOM*, pages 1–5, March 2010.

[SB04] David Schwab and Rick Bunt. Characterising the use of a campus wireless network. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 862–870. IEEE, 2004.

[SBDR07] Joel Sommers, Paul Barford, Nick Duffield, and Amos Ron. Accurate and efficient sla compliance monitoring. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 109–120. ACM, 2007.

[SBL15] Bart Spinnewyn, Bart Braem, and Steven Latre. Fault-tolerant application placement in heterogeneous cloud environments. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 192–200. IEEE, 2015.

[Sch17] Klaus Schwab. *The fourth industrial revolution*. Penguin UK, 2017.

[SG76] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565, 1976.

[SGP$^+$15] Joao Soares, Carlos Goncalves, Bruno Parreira, Paulo Tavares, Jorge Carapinha, Joao Paulo Barraca, Rui L Aguiar, and Susana Sargento. Toward a telco cloud environment for service functions. *Communications Magazine, IEEE*, 53(2):98–106, 2015.

[SJ16] Tomislav Shuminoski and Toni Janevski. Lyapunov optimization framework for 5G mobile nodes with multi-homing. *IEEE Communications Letters*, 20(5):1026–1029, 2016.

[SKA13] Suhail Shahab, Tiong Kiong, and Ayad Abdulkafi. A Framework for Energy Efficiency Evaluation of LTE Network in Urban, Suburban and Rural Areas. *Australian J. of Basic and Applied Sciences*, 7(7):404–413, 2013.

[SL13] Georgia Sakellari and George Loukas. A survey of mathematical models, simulation approaches and testbeds used for research in cloud computing. *Simulation Modelling Practice and Theory*, 39(0):92 – 103, 2013. S.I.Energy efficiency in grids and clouds.

[SLW$^+$14] Petter Svärd, Wubin Li, Eddie Wadbro, Johan Tordsson, and Erik Elmroth. Continuous datacenter consolidation. Number 2014:08 in Report / UMINF, page 12. Umeå universitet, 2014.

[SMK$^+$17] Philipp Schulz, Maximilian Matthe, Henrik Klessig, Meryem Simsek, Gerhard Fettweis, Junaid Ansari, Shehzad Ali Ashraf, Bjoern Almeroth, Jens Voigt, Ines Riedel, et al. Latency critical IoT applications in 5G: Perspective on the design of radio interface and network architecture. *IEEE Communications Magazine*, 55(2):70–78, 2017.

[SMS$^+$17] Mansoor Shafi, Andreas F Molisch, Peter J Smith, Thomas Haustein, Peiying Zhu, Prasan De Silva, Fredrik Tufvesson, Anass Benjebbour, and Gerhard Wunder. 5g: A tutorial overview of standards, trials, challenges, deployment, and practice. *IEEE Journal on Selected Areas in Communications*, 35(6):1201–1221, 2017.

[SNM10] V. Sarathy, P. Narayan, and Rao Mikkilineni. Next generation cloud computing architecture: Enabling real-time dynamism for shared distributed physical infrastructure. In *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*, pages 48–53, 2010.

[SPvS05] M. Szymaniak, G. Pierre, and M. van Steen. Latency-driven replica placement. In *Proceedings of The 2005 Symposium on Applications and the Internet, 2005.*, pages 399–405, Jan 2005.

[SS12] Sapana Singh and Pratap Singh. Key concepts and network architecture for 5g mobile technology. *International Journal of Scientific Research Engineering & Technology (IJSRET)*, 1(5):165–170, 2012.

[SSH$^+$16] A Shehabi, SJ Smith, N Horner, I Azevedo, R Brown, J Koomey, E Masanet, D Sartor, M Herrlin, and W Lintner. United states data center energy usage report. Technical Report LBNL-1005775, Lawrence Berkeley National Laboratory, Berkeley, California, 2016.

[ST18] P. Skarin and W. Tärneberg. Control over the cloud software. `https://github.com/pskarin/CotC`, 2018.

[STÅK18] Per Skarin, William Tärneberg, Karl-Erik Årzen, and Maria Kihl. Towards mission-critical control at the edge and over 5G. In *International Conference on Edge Computing (EDGE)*. IEEE, 2018.

[TCH16] William Tärneberg, Vishal Chandrasekaran, and Marty Humphrey. Experiences creating a framework for smart traffic control using aws iot. In *9th International Conference on Utility and Cloud Computing*, International Conference on Utility and Cloud Computing (UCC) (Changhai, China). IEEE/ACM, 2016.

[THOP13] Fung Po Tso, G. Hamilton, K. Oikonomou, and D.P. Pezaros. Implementing scalable, network-aware virtual machine migration for cloud data centers. In *2013 IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pages 557–564, June 2013.

[TK14] William Tärneberg and Maria Kihl. Workload displacement and mobility in an omnipresent cloud topology. In *Proc. SoftCom (Split, Croatia)*. IEEE, 2014.

[TKR$^+$17] William Tärneberg, Mehmet Karaca, Anders Robertsson, Fredrik Tufvesson, and Maria Kihl. Utilizing massive mimo for the tactile internet: Advantages and trade-offs. In *Proc. SECON Workshops - Robotic Wireless Networks (San Diego, CA, USA)*. IEEE, 2017.

[TKRK17] William Tärneberg, Mehmet Karaca, Anders Robertsson, and Maria Kihl. Cross-layer control for bounded shared state inconsistency in wireless iot devices. In *Proc. Conference on Decision and Control (Melbourne, Australia)*. IEEE, 2017.

[TMT$^+$15] William Tärneberg, Amardeep Mehta, Johan Tordsson, Maria Kihl, and Erik Elmroth. Resource management challenges for the infinite cloud. In *10th International Workshop on Feedback Computing at CPSWeek (Seattle, WA, USA)*, 2015.

[TMW$^+$16] William Tärneberg, Amardeep Mehta, Eddie Wadbro, Johan Tordsson, Johan Eker, Maria Kihl, and Erik Elmroth. Dynamic application placement in the mobile cloud network. *Elsevier, Future Generation Computer Systems*, 2016.

[TPM$^+$17] William Tärneberg, Alessandro Papadopoulos, Amardeep Mehta, Johan Tordsson, and Maria Kihl. Distributed approach to the holistic resource management of a mobile cloud network. In *Proc. International Conference on Fog and Edge Computing (Madrid, Spain)*. IEEE, 2017.

[TRA$^+$17] Tim Tiernan, Nicholas Richardson, Philip Azeredo, Wassim G Najm, Taylor Lochrane, et al. Test and evaluation of vehicle platooning proof-of-concept based on cooperative adaptive cruise control. Technical report, John A. Volpe National Transportation Systems Center (US), 2017.

[TRL$^+$09] Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts, Samuel Madden, Hari Balakrishnan, Sivan Toledo, and Jakob Eriksson. Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 85–98. ACM, 2009.

[UPS$^+$05] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. An analytical model for multi-tier Internet services and its applications. *SIGMETRICS Perform. Eval. Rev.*, 33(1):291–302, June 2005.

[V$^+$01] András Varga et al. The OMNeT++ discrete event simulation system. In *Proceedings of the European simulation multiconference (ESM'2001)*, volume 9, page 65, 2001.

[VFD$^+$16] Massimo Villari, Maria Fazio, Schahram Dustdar, Omer Rana, and Rajiv Ranjan. Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3(6):76–83, 2016.

[VGO$^+$17] Mario Villamizar, Oscar Garcés, Lina Ochoa, Harold Castro, Lorena Salamanca, Mauricio Verano, Rubby Casallas, Santiago Gil, Carlos Valencia, Angee Zambrano, et al. Cost comparison of running web applications in the cloud using monolithic, microservice, and aws lambda architectures. *Service Oriented Computing and Applications*, 11(2):233–247, 2017.

[Vir04] Marta Virseda. Modeling and control of the ball and beam process. *MSc Theses*, 2004.

[VKF$^+$12] Venkatanathan Varadarajan, Thawan Kooburat, Benjamin Farley, Thomas Ristenpart, and Michael M Swift. Resource-freeing attacks: improve your cloud performance (at your neighbor's expense). In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 281–292. ACM, 2012.

[Vya11] Valeriy Vyatkin. Iec 61499 as enabler of distributed and intelligent automation: State-of-the-art review. *IEEE Transactions on Industrial Informatics*, 7(4):768–781, 2011.

[WCB10] B. Wickremasinghe, R.N. Calheiros, and R. Buyya. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 446–452, April 2010.

[WMP$^+$16] T. Wirth, M. Mehlhose, J. Pilz, B. Holfeld, and D. Wieruch. 5G new radio and ultra low latency applications: A PHY implementation perspective. In *2016 50th Asilomar Conference on Signals, Systems and Computers*, pages 1409–1413, Nov 2016.

[WSVY07] Timothy Wood, Prashant J Shenoy, Arun Venkataramani, and Mazin S Yousif. Black-box and gray-box strategies for virtual machine migration. In *USENIX Symposium on Networked Systems Design and Implementation*, volume 7, pages 17–17, 2007.

[WTS$^+$16] J. Wan, S. Tang, Z. Shu, D. Li, S. Wang, M. Imran, and A. V. Vasilakos. Software-defined industrial Internet of things in the context of industry 4.0. *IEEE Sensors Journal*, 16(20):7373–7380, Oct 2016.

[XDB16] Minxian Xu, Amir Vahid Dastjerdi, and Rajkumar Buyya. Energy efficient scheduling of cloud application components with brownout. *Transactions on Sustainable Computing*, 1(2), 2016.

[YAKS10] Khalil M Yousef, Mamal N Al-Karaki, and Ali M Shatnawi. Intelligent traffic light flow control system using wireless sensors networks. *J. Inf. Sci. Eng.*, 26(3):753–768, 2010.

[YFN$^+$18] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *arXiv preprint arXiv:1808.05283*, 2018.

[YIJ17a] A. Yousefpour, G. Ishigaki, and J. P. Jue. Fog computing: Towards minimizing delay in the Internet of things. In *2017 IEEE International Conference on Edge Computing (EDGE)*, pages 17–24, June 2017.

[YIJ17b] Ashkan Yousefpour, Genya Ishigaki, and Jason P Jue. Fog computing: Towards minimizing delay in the internet of things. In *Edge Computing (EDGE), 2017 IEEE International Conference on*, pages 17–24. IEEE, 2017.

[ZBP01] Wei Zhang, Michael S Branicky, and Stephen M Phillips. Stability of networked control systems. *IEEE Control Systems*, 21(1):84–99, 2001.

[ZDZ12] D. Zhao, Y. Dai, and Z. Zhang. Computational intelligence in urban traffic signal control: A survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):485–494, July 2012.

[ZDZQ13] Xu Zhiqun, Chen Duan, Hu Zhiyuan, and Sun Qunying. Emerging of telco cloud. *Communications, China*, 10(6):79–85, 2013.

[ZG11] S. Zaman and D. Grosu. A distributed algorithm for the replica placement problem. *IEEE Transactions on Parallel and Distributed Systems*, 22(9):1455–1468, Sept 2011.

[Zip49] George Kingsley Zipf. Human behavior and the principle of least effort. 1949.

[ZZGTG10] Y. Zaki, Liang Zhao, C. Goerg, and A. Timm-Giel. Lte wireless virtualization and spectrum management. In *Wireless and Mobile Networking Conference (WMNC), 2010 Third Joint IFIP*, pages 1–6, Oct 2010.