AN ABSTRACT OF THE DISSERTATION OF

Sherif Abdelwahab for the degree of Doctor of Philosophy in Electrical and Computer Engineering presented on August 4, 2017.

Title: Architectures and Algorithms for Dynamic Overlay Networks

Abstract approved: _____

Bechir Hamdaoui

Most of today's Internet of Things (IoT) applications assume that data will be moved off devices into centralized cloud platforms. While existing IoT systems leverage cloud-based analytics for meaningful data reasoning, the assumption that data should always be moved off the devices is problematic. The amount of data to be moved from devices over Internet gateways to cloud platforms is huge which potentially make it cost inefficient. In other scenarios, privacy concerns of customers or organizational rules complicate the process of transferring data to third-party data centers.

This dissertation proposes architectures and dynamic overlay network algorithms for in-network and edge processing of data offered by the globally available IoT devices and provides a global platform for meaningful and responsive data analysis and decision making. The proposed techniques shift IoT analytics from a "collect data now and analyze it later" scenario to directly providing meaningful information from the in-network processing of devices data at or near the devices. The techniques serve future IoT use cases including distributed context awareness, on-demand data analysis, and in-network decision making. The dissertation comprises three main components.

The first component is a device management protocol for cloning devices' data in proximate Edge Computing platforms. Unlike existing application-layer IoT management protocols the proposed protocol uses the LTE/LTE-A radio frame structure, device-to-device communication, and IoT data properties to avoid excessive network access latency in existing technologies.

The second component realizes distributed IoT analytics as overlay networks of devices clones. By means of virtual network embedding, it selects and interconnects devices' clones to efficiently realize applications' virtual topologies to achieve goals such as minimum latency, minimum infrastructure cost, or maximum infrastructure utilization.

Finally, the dissertation presents a communication middleware that allows autonomous discovery, self-deployment, and online migration of devices' clones across heterogeneous Edge computing platforms. The middleware ensures that communication latency between clones is kept minimum despite the uncontrolled variability of the network and hosting platforms conditions.

We evaluate the proposed architectures and algorithms through simulations and prototype implementation of various components in controlled testbed environments, which we evaluate using real user applications. We explore the feasibility of the proposed techniques from both theoretical and practical perspectives.

Architectures and Algorithms for Dynamic Overlay Networks

by

Sherif Abdelwahab

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented August 4, 2017
Commencement June 2018

Doctor of Philosophy dissertation of Sherif Abdelwahab presented on August 4, 2017.

APPROVED:

Major Professor, representing Electrical and Computer Engineering

---

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Sherif Abdelwahab, Author

I would like to express my gratitude for my advisor Bechir Hamdaoui for all the feedback, guidance, and support that he has given me during the Ph.D. journey. This work was made possible by Bechir's NPRP from the Qatar National Research Fund, his career grant from the National Science Foundation, and the assistantships of the EECS department of Oregon State University. I would also like to thank Mohsen Guizani, Taeib Znati, and Ammar Rayes for their valuable support in this work.

I would love to express my thankfulness to the EECS Head, V John Mathews, for his valuable support and words of wisdom. I will also never forget how kind the EECS Senior Associate Head, Bella Bose, was to me and how much he helped me. No words can express my appreciation to the mentorship and actions of Christine Kelly, Jim Lundy, Jon Dorbolo, Stephanie Bernell, and Tori Byington. I would like to acknowledge the technical contributions of my committee members Ben Lee, Attila Yavuz, Amir Nayyeri, and Margaret Niess for their invested time reviewing and polishing this thesis with their recommendations. I wanted to thank Nicole Thompson, Tarrigon Van Denburg, the Graduate School, International Programs, and Human Resources staff for their assertive, timely, and humanly assistance.

I am very thankful to my friends and colleagues at Oregon State, Hewlett-Packard, Etisalat, and Alcatel-Lucent for their emotional support, technical insights, and encouragements. I will always show an indebtedness to my brothers and sisters Ahmed, Ayman, Asser, Heba, and Alaa. I am also grateful to my mother-in-law Aisha and my uncle Hatem for their unconditional support. To my kids, and my sibling' kids I will remain thankful for the joy they give us during this Ph.D. journey.

My wife, Enas has been extremely patient, forgiving, and responsible to go hand-in-hand through the entire Ph.D. process with all its down and up moments. Last, and most importantly I will always carry my dad's name as my last name, and I dedicate this dissertation to my mother, Kawthar, she will always deserve more credit in my work than I do.

*

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Page

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF APPENDIX FIGURES

To mom.

Chapter 1: Introduction

The scalability and reliability of IoT applications require rethinking system and end-to-end network architecture designs that differ from existing systems such as enterprise applications [15, 11, 118].

The goal of this dissertation is to develop cloud networking technologies for on-demand, dynamic, trusted, and reliable IoT analytics. Analytics for IoT can be done in conventional cloud computing platforms by moving data from devices into centralized data lakes. The emerging paradigm of Edge computing is also getting attention to executing such analytics near the network edge without the need to move data from devices to conventional clouds. Finally, IoT analytics can be done with in-network processing with IoT devices as devices communicate through device-to-device communication. We explore existing networking technologies that can enable such diverse requirements, and dive deep in network virtualization technologies that realize IoT analytics as overlay networks of virtual topologies.

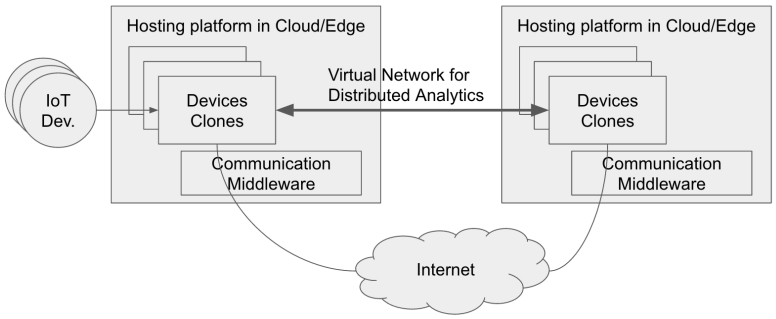


Figure 1.1: High level architecture.

1.1 Background

1.1.1 Edge computing in LTE/LTE-A

Mobile-edge Computing augments already-existing Radio Access Network (RAN) with cloud computing resources in close proximity to mobile subscribers. For developers, Mobile-edge

Computing provides an environment of ultra-low latency and high-bandwidth as well as direct access to real-time radio network information.

The LTE all-IP architecture, built-in security and spectral efficiency nominate LTE to become the dominant connectivity technology for the Internet of Things (IoT), while IoT services and applications create unprecedented traffic growth for 3GPP LTE/LTE-A networks [4]. For IoT applications, it is becoming a global consensus that cloud computing technology is an essential driver for IoT computation speedup, energy consumption, and service realizations [9, 144]. IoT applications include, for example, connected vehicles, smart grids and cities, and wireless sensors and actuators networks [35]. In such applications, IoT devices offload its computations by replicating small-sized (tiny) memory objects and transferring these memory replicas through LTE networks to a back-end cloud computing infrastructure that enables the IoT applications to scale its computing resources on elastic infrastructure instead of resource limited devices besides many other benefits. Despite the potential of LTE to efficiently transport these memory replicas to the scalable cloud infrastructure, the massive number of devices per cell, which is projected to reach 50,000 devices by 2020 (see [5]), renders an LTE network as the major communication bottleneck for cloud offloading that can significantly limit offloading performance gains.

We propose REPLISOM - a memory replication architecture and protocol - based on the emerging mobile edge computing paradigm [68], the Device-to-Device (D2D) communication technology [23], and the compressed sampling theory [62].

1.1.2 Virtual Network Embedding

Network Virtualization and Application overlay networks have emerged as key technologies to effectively use network resources in support of diverse services and applications [96, 10]. Network virtualization consolidates hardware and software network resources under a single administrative authority. In general, network virtualization involves interconnecting virtualizable hardware in a network that performs a certain task. The tasks can be permanent - as in enterprise applications - or on-demand as we explore in this dissertation. Network virtualization can be done at any layer of the networking stack. In this work, we refer to virtualization at the application level as application overlay networks. We also refer to virtualization of inter-networked IoT devices as virtual sensor networks.

A virtual network is described by application developers as a network of virtual nodes interconnected by virtual links. Successful on demand network virtualization involves three main

	Bird-VNE	RADV	Flock
Scalability	Polynomial	Polynomial	Linearithmic
Closeness to Optimal	$\frac{1}{2}$ far from optimal	N/A	$1+\epsilon$
Architecture	Centralized	Hybrid (conjectured)	Centralized or Distributed
Network Tomography	Must maintain consistent network measurements	Cooperative measurements	local measurements
Service Discovery	Centralized	Relies on Gossip Methods	N/A
Implementation Complexity	3k lines of code	3k lines of code	1k lines of code

Table 1.1: Comparison of the proposed technuiques and why flock is *the key technuiqe*.

methods. First, a discovery of the resources of the substrate network is essential for a virtual network embedding algorithm to have a complete of a partial view of the network resources. Second, a virtual network embedding algorithm selects the substrate nodes and the substrate links that realize the virtual network. The virtual network itself can realize enterprise application, on-demand distributed algorithms, or applications that span several geographical locations as in IoT analytics.

We explore architectures and algorithms for Network virtualization at different levels. We propose a centralized algorithm that we name BirdVNE. BirdVNE embeds on-demand virtual networks onto substrate networks using the theory of constrained optimization. Centralization of BirdVNE constraints its scalability, hence we propose RADV as another solution for distributed resource discovery and network virtualization using gossip protocols and distributed optimization. Although performing well, in practice, we found RADV properties hard to formally prove and complex to implement as a single unit as it involves several components. Finally, we propose Flock algorithm, which we demonstrate its superior performance in terms of stability, closeness to an optimal solution, ease of implementation, and applicability to satisfy diverse design goals. Table 1.1 shows a comparison between Bird-VNE, RADV, and Flock and shows how Flock is the key technuiqe proposed.

1.1.3 Implementation, Testing, and Usage

Today, emerging applications of overlay networks require experimentation in geographically distributed testbeds. Distributed analytics near the network edge is one example application in which several hundreds of microservices and Internet of Things devices execute distributed computation on data from devices close to access points [144, 146, 10]. Existing cloud computing infrastructures best suit the needs of such experimentations.

For example, to experiment our virtual network embedding algorithms we need to deploy cloud resources that are spread over several geographical locations that mimic real substrate

Figure 1.2: Implementation types for evaluation and testing.

Technuiqe	Implementation Type
REPLISOM	Numerical Simulation and Prototyping for Device-To-Device communication at Layer-7
Bird-VNE	Emulation
RADV	Emulation and Prototyping for the Service Discovery mechanism
Flock	Simulations and Prototyping for the entire technuiqe

Table 1.2: Comparison of the proposed technuiques and why flock is *the key technuiqe*.

networks with shared traffic, workload, and overhead. To remain cost-effective we shall be able to dynamically create, interconnect, and configure such virtual testbeds, run experiments for a predefined time period, collect results, and then destroy virtual testbed as soon as the results are available for analysis. In some experiments, we also need to measure and account for real Internet and shared cloud resources variations. Experiments in such variable uncontrolled and real environments allows credible prototyping of new architectures and systems that face todays real-world challenges such as scalability, multi-tenant support, and handling of resource shortage. Figure 1.2 shows the implementation phases that we folow for testing and evaluating the proposed techniques and Table 1.2 shows a taxonomy of the differnet implementation types we have for these technuiqes.

We propose Beelet; A geographically distributed virtual testbed management API that simplifies large-scale virtual testbed creation for research experiments in network function virtualization, distributed systems, and large-scale overlay networks. Beelet uses Layer-3 overlay networks to interconnect application services that Beelet deployed in virtual machines. Beelet can use Amazon EC2 or Google cloud platforms to host the virtual machines and automatically interconnect them according to a predefined topology given by Beelet users. We demonstrate the effectiveness of Beelet through a usage model in which we deploy FogMQ for functional experiments, and present its use for integrating RADV and Bird-VNE implementations and evaluation.

We also propose GROUP; A library to create overlay networks at layer-7. GROUP incorporates an off-the-shelf implementation of the gossip protocols for node discovery in Wide Area Networks and is foundational to the realization of RADV. The library introduces practical methods of overcoming authorization latencies by separating the control plane functionalities from the data-plane functionalities in existing middleware communication.

Finally, we propose FogMQ; an implementation, evaluation, and use case of the Flock al-

gorithm. In FogMQ, we design self-deploying brokering clones that discover cloud hosting platforms and autonomously migrate between them according to self-measured weighted tail end-to-end latency, ultimately allowing us to stabilize clones deployment and achieve a near minimum latency given an existing infrastructure limits. Figure 1.3 shows the development phases for FogMQ prototyping.

Figure 1.3: Development phases of prototypes.

1.1.4 Contributions and Organization

This work comprises nine chapters. Chapter 1 provides and introduction and highlights of our proposed solutions. We organize the chapters and summarize the contributions of this dissertation as follows.

In Chapter 2, we discuss the evolution of existing LTE/LTE-A architecture towards Mobile Edge Computing that supports IoT analytics near the edge. In this chapter, we propose REPLISOM, a protocol that improves mobile edge computing responsiveness for tiny-sized memory replication from a massive number of IoT devices in LTE in specific. REPLISOM Reduce the memory replication delay by diminishing the need of initiating the LTE random access procedure for each replica transfer, which introduces an undesirable delay, increased energy consumption, and risk of instability given a large number of simultaneously active devices. Our proof-of-concept implementation and evaluation shows that REPLISOM enhances the LTE signaling overhead and resource usage for memory replication by avoiding the allocation of unnecessary dedicated control channels per device, which wastes the scarce radio resources and risks the blocking of human communications.

In Chapter 4, we develop a centralized virtual network embedding algorithm, termed BIRD-VNE, for mobile wireless networks. BIRD-VNE is a centralized approximation algorithm that ensures a close to optimal virtual embedding profit and acceptance rate while minimizing the number of virtual network migrations resulting from the mobility of wireless nodes. BIRD-VNE employs a constraint satisfaction framework by which we analyze the constraint propagation properties of the VNE problem and design constraint processing algorithms that efficiently narrow the solution space and avoid backtracking as much as possible without compromising

the solution quality. Our evaluation results show that the likelihood that BIRD-VNE results in backtracking is small, thus demonstrating its effectiveness in reducing the search space. We analytically and empirically verify that BIRD-VNE outperforms existing VNE algorithms with respect to computational efficiency, closeness to optimality, and its ability to avoid potential migrations in mobile wireless networks.

In Chapter 5, we propose an architecture and use case of overlay networks for Sensing as a Service. We conjecture a global architecture, named Cloud of Things, that scales up cloud computing by exploiting the global sensing resources of the highly dynamic and growing Internet of Things (IoT) to enable remote sensing. The proposed architecture scales out by augmenting the role of edge computing platforms as cloud agents that discover and virtualize sensing resources of IoT devices. Our solution enables performing in-network distributed processing of sensing data offered by the globally available IoT devices and provides a global platform for meaningful and responsive sensing data analysis and decision making. We design cloud agents algorithmic solutions bearing in mind the onerous to track dynamics of the IoT devices by centralized solutions. First, we propose a distributed sensing resource discovery algorithm based on a gossip policy that selects IoT devices with predefined sensing capabilities as fast as possible. We also propose RADV: a distributed virtualization algorithm that efficiently deploys virtual sensor networks on top of a subset of the selected IoT devices. We show, through analysis and simulations, the potential of the proposed algorithmic solutions to realize virtual sensor networks with minimal physical resources, reduced communication overhead, and low complexity. RADV overcomes centralization and complexity limitations of Bird-VNE by delegating the embedding tasks to multiple cloud agents and devices. Although performing well in proof-of-concept implementation and evaluation, it is hard to formally prove its superiority and prototype it in practice.

In Chapter 6, we overcome the limitations of both Bird-VNE and RADV by designing Flock algorithm. We recognize the importance of live migration in optimizing the overlay networks in dynamic environments. Flock is a simple and scalable protocol that enables live migration of Virtual Machines (VMs) across the heterogeneous edge and conventional cloud platforms to improve the responsiveness of cloud services. Flock is designed with properties that are suitable for the use cases of the Internet of Things (IoT). We describe the properties of regularized latency measurements that Flock can use for asynchronous and autonomous migration decisions. Such decisions allow communicating VMs to follow a flocking-like behavior that consists of three simple rules: separation, alignment, and cohesion. Using game theory, we derive analytical

bounds on Flocks Price of Anarchy (PoA) and prove that flocking VMs converge to a Nash Equilibrium while settling in the best possible cloud platforms. We verify the effectiveness of Flock through proof-of-concept implementation and evaluation and discuss how its generic objective can simply be tweaked to achieve other objectives, such as cloud load balancing and energy consumption minimization.

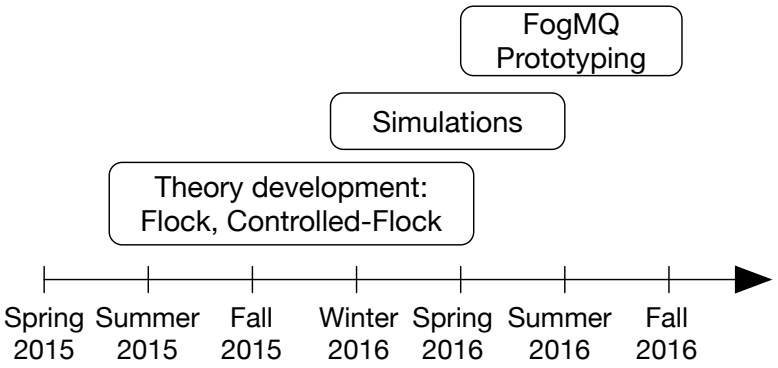


Figure 1.4: Theory and protoyping timeline of Flock and FogMQ.

In Chapter 7, we implement and evaluate Flock in an optimized usage for communication middleware in Edge and Fog Computing that name FogMQ. This prototype implementation is motivated after we show the superiority of Flock algorithm theoretically and with proof-of-concept simulations in Chapter 6. Figure 1.4 shows a timeline for Flock and FogMQ theory, simulation, and prototype development. Excessive tail end-to-end latency occurs with conventional message brokers as a result of having massive numbers of geographically distributed devices communicate through a message broker. On the other hand, broker-less messaging systems, though ensure low latency, are highly dependent on the limitation of direct device-to-device (D2D) communication technologies, and cannot scale well as large numbers of resource-limited devices exchange messages. We propose FogMQ, a cloud-based message broker system that overcomes the limitations of conventional systems by enabling autonomous discovery, self-deployment, and online migration of message brokers across heterogeneous cloud platforms. For each device, FogMQ provides a high capacity device cloning service that subscribes to device messages. The clones facilitate near-the-edge data analytics in resourceful cloud compute nodes. Clones in FogMQ apply Flock, an algorithm mimicking flocking-like behavior to allow clones to dynamically select and autonomously migrate to different heterogeneous cloud platforms in a distributed manner.

As we recognize the possibility to implement overlay networks at different network stacks, we provide frameworks for implementation in Layer-7 and Layer-3 in Chaper 8 and Chapter 9 respectively.

In Chapter 8, we propose GROUP is a dynamic application overlay platform for controlled, trusted, and reliable communication among heterogeneous devices and services in cloud and edge computing environments. GROUP eases development and policy management for such complex environments. It has a control plane that creates application overlay networks. An application overlay network is data plane that is used to exchange messages between application services. Each data plane can also act as a control plane to create and manage additional application overlay networks. Control plane actions are uniformly and transparently subject to policy and obligations. Policy determines what actions are permitted. Obligations direct adaptation for the system in terms of launching or terminating services, or requiring changes to the application overlay networks. Our approach distributes policy enforcement and allows for centralized, distributed, or hybrid policy decision, information, and administration points. We have implemented GROUP as reusable Java APIs and tested it in a controlled environment where we demonstrate its usability by implementing a privacy perceiving ride-sharing solution that leverages real-time data from IoT devices.

In Chapter 9, we introduce Beelet; a geographically distributed virtual testbed management API that simplifies large-scale virtual testbed creation for research experiments in network function virtualization, distributed systems, and large-scale application overlay networks. Beelet uses Layer-3 overlay networks to interconnect application services that Beelet deployed in virtual machines. Beelet can use Amazon EC2 or Google cloud platforms to host the virtual machines and automatically interconnect them according to a predefined topology given by Beelet users. We demonstrate the effectiveness of Beelet through a usage model in which we deploy FogMQ for functional experiments. We also demonstrate the usage of Beelet to implement and evaluate Bird-VNE and RADV.

We finally conclude this dissertation in Chapter 10.

Chapter 2: Disciplined Tiny Memory Replication for Massive IoT Devices in LTE Edge Cloud

2.1 Introduction

The LTE all-IP architecture, built-in security, and spectral efficiency nominate LTE to become the dominant connectivity technology for the Internet of Things (IoT), while IoT services and applications create unprecedented traffic growth for 3GPP LTE/LTE-A networks [4]. For IoT applications, it is becoming a global consensus that cloud computing technology is an essential driver for IoT computation speedup, energy consumption, and service realizations [9, 144]. IoT applications include, for example, connected vehicles, smart grids and cities, and wireless sensors and actuators networks [35]. In such applications, IoT devices offload its computations by replicating small-sized (tiny) memory objects and transferring these memory replicas through LTE networks to a back-end cloud computing infrastructure that enables the IoT applications to scale its computing resources on elastic infrastructure instead of resource limited devices besides many other benefits. Despite the potential of LTE to efficiently transport these memory replicas to the scalable cloud infrastructure, the massive number of devices per cell, which is projected to reach 50,000 devices by 2020 (see [5]), renders an LTE network as the major communication bottleneck for cloud offloading that can significantly limits offloading performance gains.

Memory replication is a disciplined process in which consistency must be ensured such that a write operation is followed by a memory update operation with the cloud for each device [54]. On the other hand, the current LTE network access and uplink scheduling procedures introduce a significant latency and energy inefficiency to update a large number of tiny memory replicas. An LTE cell can become easily blocked, if only 10% of the devices it covers became active simultaneously to update their memory replicas with the cloud. Unsurprisingly, this bottleneck is *not* a result of bandwidth limitation; as an IoT device memory replica is typically a few Kilobytes and the LTE network is optimized for high throughput and low latency applications [85, 86, 80]. Nevertheless, the LTE standard is not optimized to support a large simultaneous access from devices while remaining delay and energy efficient; as this requires allocating a large number of control channels and wastes radio resources, which are primarily intended for transporting

conventional mobile users data. To remain a disciplined process, we design an *LTE-optimized* memory replication architecture and protocol to harvest the benefits of both LTE and cloud computing technologies.

In this chapter, we propose REPLISOM [1], a memory replication architecture and protocol based on: the emerging mobile edge computing paradigm [68], the Device-to-Device (D2D) communication technology [23], and the compressed sampling theory [62]. We summarize our contribution as follows:

- Improve the cloud responsiveness for IoT services and applications by distributing cloud resources geographically close to the IoT devices. Unlike Cloudlets, MAUI, CloneCloud, and COSMOS [144, 56, 54, 150], the proposed architecture enables the design of LTE-aware cloud procedures in general and memory replication protocols optimized for tiny-sized memory replication from a massive number of IoT devices in LTE in specific.
- Reduce the memory replication delay by diminishing the need of initiating the LTE random access procedure for each replica transfer, which introduces an undesirable delay, increased energy consumption, and risk of instability given the large number of simultaneously active devices. Unlike the application of compressed sampling in sensor network which relies on spatial and temporal correlation of sensor data [87, 171], the proposed memory replication protocol relies on two level of sparsity structures at the network and memory levels.
- Enhance the LTE signaling overhead and resource usage for memory replication by avoiding the allocation of unnecessary dedicated control channels per device, which wastes the scarce radio resources and risks the blocking of human communications. Unlike other protocols used in general purpose machine type communications in LTE [19, 173], our work relies on the disciplined nature of memory replication to design a pull based memory replication protocol which uses a significantly less number of control channels compared to direct memory replication using the conventional LTE access and data transfer procedures.

2.1.1 Solution Outline

The REPLISOM *architecture* is a mobile edge cloud architecture (see Figure 2.1) in which we augment the evolved NodeB (eNB) with cloud computing resources and refer to this augmented

[1] The name, REPLISOM , is inspired by *replisomes* that carries out replication of DNA.

architecture by the LTE edge cloud. The LTE edge cloud is a new *integrated* radio access network element that provides virtualizable computing, storage, and networking resources to clone device specific IoT applications and services. The architecture is a highly responsive system that neutralizes the back-hauling and routing bottlenecks which exist in current conventional cloud architecture. Deploying cloud computing resources in the proximity of the IoT devices allows *developing an LTE radio interface which is optimized for memory replication utilizing already in place technologies*. For example, an LTE capable IoT devices already incorporate D2D technologies that support efficient proximal devices discovery and direct communication. By utilizing the capabilities of the D2D technology and the existing LTE control and data channels, we show the possibility to improve the memory replication performance through an LTE-optimized protocol.

The REPLISOM *protocol* is an LTE-optimized memory replication protocol(s) that relies on pulling the memory replicas from the IoT devices instead of pushing the replicas from the devices to the LTE edge cloud. We observe two sources of sparsity in memory replication. The first source is at the network level, where the ratio of the active devices to the total number of devices is small even if the number of simultaneously active devices is large under the traffic models defined by 3GPP for machine communication. This source of sparsity is *independent* on any assumption about the devices memory contents (e.g. spatial or temporal correlation). The second source of sparsity is at the memory replica level, where the deltas of memory replicas typically exhibit few non-zero memory blocks. In REPLISOM , a device sends its updated memory replica to some other neighbor devices using D2D communication, while a receiving device compresses all the received memory replicas into a single compressed replica. The edge cloud then selects a number of devices, which is much less than the total number of devices in the LTE cell, and pulls the compressed replicas from these devices. By compressed sampling reconstruction algorithms, the cloud can recover the original replicas exactly utilizing the *sparsity at the network level*. Moreover, we show possible further improvements to the devices energy consumption and replication delay by utilizing the *sparsity at the memory replica level*.

Since the cloud pulls compressed replicas from a number of devices that is proportional to the number of updated replicas, there is no need for initiating the random access procedure. As an LTE device is already synchronized with its serving LTE cell to decode the cell's control channels, with REPLISOM a device just wakes up in predefined sub-frames to verify its pulling occasions and transmit its compressed replica while remaining in a deep sleep state if it is not pulled. Unlike directly pulling the original replicas from each device in the cell, which also does not require initiating the random access procedure, the number of control channels allocated for

the proposed protocol is significantly less than the number of control channels allocated to pull replicas from each device in the cell.

The remaining of this paper is organized as follows. We first discuss the related work in Section 2.2. Then, we present the proposed architecture in Section 2.3.1 where we discuss the LTE specific challenges and the architectural rule of the D2D technology. Section 2.4 delves into the proposed memory replication protocol and its relation to the compressed sampling theory and shows how we use the two sources of sparsity, at the network level and at the memory replica level, to design an efficient pull based memory replication protocol. In Section 2.5, we describe our performance benchmarks and provide numerical evaluations of REPLISOM in comparison to replica transfer using the conventional LTE procedures.

2.2 Background

Creating computing infrastructure back-ends for devices such as cloud platforms has been in the heart of the IoT research since its inception in 1991 [169]. The vision of cloud computing for IoT has evolved through the years to what we know today as Edge computing [68, 144] or Fog Computing [35]. These evolved platforms extend the cloud computing paradigm with new characteristics such as: location awareness, low latency networking, geographically distributed infrastructure, support of mobility, wireless access awareness, and cloud interoperability [9]. Our work focuses on the efficient design of memory replication protocol for the purpose of computation offloading in the LTE edge cloud with support of massive number of IoT devices.

Cloud offloading near the edge The idea of augmenting resource constrained devices with a resource-rich cloud infrastructure accompanied the evolution of mobile computing more than a decade ago [74, 105]. Computation offloading with a fine-grain memory replication has been the focus of research since then [54, 56, 134, 150]. New forms of cloud platforms (e.g. cloudlets) emerged to provide computing resources for proximate devices with a minimal communication delay. The success of computation offloading to improve the computational capacity and energy consumption in the Internet of things era is conditioned by the limits of the underlying networking technologies that support memory replication from massive number of devices (see results in [102, 177]). Our work investigates these limitations, architectural evolution, and protocol design to support cloud-centric IoT services and applications at LTE eNB.

Massive IoT devices in LTE The energy consumption and delay performance characteristics of Internet of Things (Machine Type Communication) in LTE has been one main focus of the cellular networks research and standardization efforts [86, 162]. Particularly, the delay and energy characteristics of the LTE random access and uplink transmission procedures resemble the major bottlenecks under network overload from massive number of IoT devices [80]. The existing approaches to improve the LTE performance in such overload situation focus on finding improvements for existing uplink transmission mechanisms [19, 86]. Our work is related to these research efforts as we anticipate the impact of the LTE bottlenecks on the memory replication performance, hence cloud offloading. Our proposed protocol is *specific to the memory replication traffic*, and not to any uplink traffic type, and its validity is conditioned by the evolution of Device-to-Device Communication technologies [23, 114, 75]. The LTE-aware design of the proposed memory replication protocols reduces the dependency on the LTE random access procedure and requires significantly less number of control channels.

Compressed Sampling in networking Our work is an application of the theory of sparse recovery in compressed sampling [30], which has several applications in networking. Approaches to a decentralized compression of networked data has gained a lot of attention in the last decade [87] and had applications in: network coding [127, 100], sensor measurements collection [166, 111, 171], network tomography [180], and medium access [69]. The application of compressed sensing in such applications utilizes the sparse properties of the data in different forms. In sensor networks, for example, spatial and temporal correlations of sensor measurements are the main sources of sparsity [166] which requires finding a network transformation to sparsify the network data (e.g. using graph wavelets, or diffusion wavelets) [87]. Our work relies on the sparsity of having a limited number of simultaneously active devices out of a large number of devices at the network level besides the sparsity of having a limited number of non-zero memory blocks in memory deltas at the memory level. These sources of sparsity do not require any particular transformation to sparsify the memory replicas.

2.3 LTE Architecture Evolution for Edge Computing and Massive IoT Devices

The current LTE architecture performs specialized processing that supports radio communication with LTE devices and traffic back-hauling. When an IoT device, in an LTE cell, offloads its

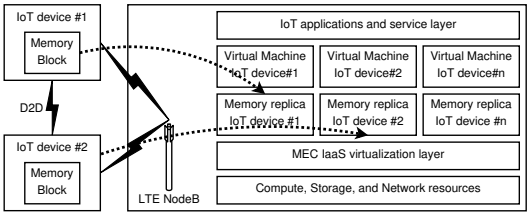


Figure 2.1: Proposed Memory replication architecture in LTE/LTE-A with Mobile Edge Computing and Device to Device communication.

computation to a cloud computing platform, it carries out LTE network access and uplink data transfer procedures including: initiating a random access, sending uplink scheduling requests, receiving uplink grants, and transmitting its uplink data (memory replica) using radio resource blocks (see Figure 2.6). Then, a memory replica (received packets at the LTE cell) passes through multiple stages of packet forwarding from the LTE cell through the serving gateway, to the packet data network gateway, to the Internet routers, to the data center routers until it finally reaches a cloud computing node that hosts a virtual machine - corresponding to the IoT device - that updates the replicated memory block. The current LTE architecture is well optimized for voice and data packet communication. However, given the service requirements of IoT including the support of: massive number of devices, reduced complexity, and power efficiency, the LTE architecture is not optimized for cloud computing offloading for IoT services and applications which requires tighter delay bounds on packets transmission.

2.3.1 Proposed Architecture

We propose to address the LTE architectural bottlenecks by deploying local cloud computing resources within the radio access network based on the mobile cloud computing paradigm [68]. Once the device memory replica reaches the LTE eNB, it becomes available to the IoT services and applications deployed at these local cloud resources. This architectural change improves the cloud responsiveness by distributing the cloud resources geographically close to the IoT devices (Figure 2.1) and paves the road to: $i)$ an improved IoT services and applications resiliency by splitting the cloud resources to local resources (to devices) and global resources (conventional cloud), $ii)$ simplified analytics and big data by capturing key information from devices with possible direct device access, $iii)$ reduced latency as applications react faster to devices and context changes away from possible congestion in other parts of the LTE network other than the

radio network, and iv) optimized cloud protocols that are aware of network information (e.g. radio conditions, performance statistics, and technology limitations).

2.3.1.1 Technical Challenges

Several technical challenges pertain to this LTE edge cloud architecture such as: the design of highly distributed applications, support of optimized applications and virtual machines portability, integration cloud security with current 3GPP-security requirements and practices, improving applications and cloud hardware resilience to match 3GPP availability and service continuity requirements, and design of LTE radio interface aware cloud protocols.

The LTE radio interface, in specific, resembles the major bottleneck for efficient memory replication in LTE edge computing. Three benchmarks capture the system wide efficiency of a memory replication protocol: the total delay, the total consumed energy, and the total number of control channels required to transfer *all the updated memory replicas from all the active devices* to the edge cloud (see Section 2.5 for a detailed description and evaluation). For example, the memory replication efficiency implies minimizing the time a single device waits between two successive replica updates. During that time the eNB is busy transporting memory replicas from other devices (besides conventional human communication). Several characteristics and observations render an LTE radio interface as the major bottleneck and motivate the design of an LTE-optimized memory replication protocol.

Large simultaneous replica updates Although the majority of IoT devices exhibit a memory change every few minutes, the massive number of devices per cell results in a large simultaneous replica updates per minute. Recent 3GPP studies on enhancements of LTE for IoT suggest new traffic models of IoT devices that can cause memory changes every 30 minutes down to 10 seconds in case of major failures which require the design of rapid network access procedures [6]. Let n denote the number of devices in an eNB, and $k = \rho n$ denote the number of devices with an updated replica at time t (active devices) where ρ is the ratio of the active devices to the total number of devices in one minute. Under the suggested 3GPP models, the parameter ρ typically ranges from 0.1 to 0.3 (i.e. 1000 to 15,000 simultaneous active devices per minute). The LTE physical layer, besides other system aspects, restricts a large number of simultaneous replica transfers due to several physical layer design aspects. For example, the finite sounding reference signal periodicity restricts the number of simultaneous active devices so that the eNB

is able to estimate the uplink channel quality with a finite accuracy (see [1] for more details on physical layer scalability limitations).

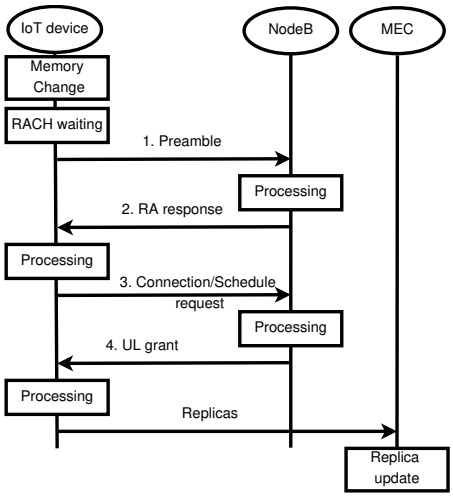


Figure 2.2: The *idle to active* scenario: LTE signaling for a replica transfer from the idle mode to the active state.

Access latency and control channels As the devices are not engaged in frequent packet transmission and reception, devices will typically remain in idle mode (not connected to the cell) for a long time to save their energy and reduce the cell interference. Unfortunately transitioning back from the idle mode to the connected mode results in an excessive access latency as every memory update involves an initiation of the random access procedure (see Section 5.1 in [2] for details on the random access procedure). We refer to this scenario as the *idle to active* scenario. Every device accessing the network from the idle mode transmits and receives at least four control messages (illustrated in Figure 2.2) : the random access request (device initiates the procedure), random access response (cell acknowledges the request and assigns initial resources), uplink connection/scheduling request (device requests uplink resources), and uplink grant (cell allocates uplink resources for data transmission). Random access requests can also collide. Let L denote the random access opportunities per second and γ denote the random access requests per second, the probability of collisions during the random access procedure is given by the $Pr_{collision} = 1 - e^{-\gamma/L}$ [5]. Even if an operator was able to increase L such that the random access initiation is collision free ($Pr_{collision} \approx 0$), recent studies suggest that in such hypothetical case the average random access latency *per device* ranges from 47 ms to 55 ms (measured from

the initiation till the first uplink transfer) in such hypothetical collision free scenario [5, 80].

Over-allocated scheduling opportunities Preventing devices from transitioning to the idle mode can improve the access latency significantly. In such scenario, a device stays in a dormant state for monitoring the control channels in predefined occasions, and does not need to initiate random access except if, for example, it lost frame synchronization, or there were no uplink resources available to send scheduling requests (see [2]). Optimized discontinues reception/transmission achieves energy saving for always connected devices (in dormant state) where devices go into deep sleep and wake up only in predefined occasions to maintain, for example, frame synchronization and decode other control channels. Figure 2.3 illustrates replica transfer from the dormant state where the device first identify a scheduling opportunity and sends an uplink scheduling requests. Once, the LTE cell allocates uplink radio resources for the device, the cell sends an uplink grant message to the device to start its transfer. We will refer to this scenario as the *dormant to active* scenario. This procedure exhibits the least possible latency, but ideally requires allocating scheduling opportunities for n devices, which is not necessarily feasible for a large number of devices (see [1] and [2]) and is inefficient due to the unnecessary allocation of control channels as we will detail in Section 2.5.

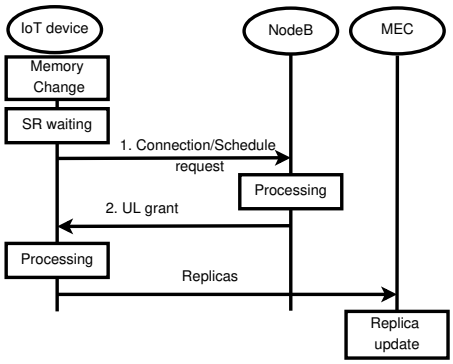


Figure 2.3: The *dormant to active* scenario: LTE signaling for a replica from the dormant state to the active state.

Untraceable memory changes A memory replication protocol can be pull based, where the edge cloud initiates the replicas transfer through paging the IoT devices, hence the random access procedure is not initiated and the scheduling requests are allocated only for pulled devices. With the current LTE specification, the cell can page a device to initiate an uplink transfer. There are

two challenges accompanying this process. First, the paging process is not ideal and involves latency, collisions, and capacity challenges that are as difficult as the random access procedure. Second, it is not trivial for the edge cloud to determine which devices are active to pull replicas from, without initiating unnecessary paging or pulling replicas from all the devices in the worst case.

2.3.1.2 Architectural Rule of D2D Communication

We address the previously discussed challenges by designing a pull based memory replication protocol using D2D communication (in specification starting 3GPP Release 12) and compressed sampling (Section 2.4). *Architecturally*, IoT devices can communicate directly with each other using the licensed cellular spectrum (in-band), or the unlicensed spectrum (out-band). In-band D2D can use the same operating band of the LTE cell in an underlay mode or a different band in an overlay mode. D2D communication requires: interference management, resource allocation, and device discovery services that devices can perform autonomously or by the LTE infrastructure assistance (i.e. small and home cells other than the macro cell in Figure 2.1) [23].

In the proposed architecture, device pairs can communicate autonomously in parallel with low power radio. Typically, the D2D transmission power, P_{d2d}, is a fraction of mW (e.g. 1 mW [110]). A device is also capable of communicating with a group of devices in multicast, and the total time required in transferring a replica from one device to the other is comparable to the idle to active scenario (Figure 2.2). Let r denote the maximum communication range of the LTE D2D technology[2]. A device is connected to a $Neigh(i) = \{j : Dist(i,j) \leq r\}$ neighbor set, where $Dist(i,j)$ denote the Euclidean distance between any two devices i and j. With an LTE cell that covers an area a and serves n devices [3] , a device i is directly connected to $\mathcal{E}\{Neigh(i)\} = \eta\pi r^2$ on average, where $\eta = \frac{n}{a}$ denote the network density.

2.4 Memory Replication Protocols

The main intuition behind the proposed REPLISOM memory replication protocol is to recognize that during a short time interval, the memory replicas of all the n devices in an eNB resemble a

[2]Early commercial solutions show LTE D2D communication range up to 500 meters and we assume $r < 200$ meters.

[3]3GPP suggests $a \approx 1$ square Kilometer as detailed in [6].

sparse vector, x, of length n that has k non-zero entries which represent replicas from k active devices. We refer to this observation as *the sparsity at the network level*. Hence. it is possible to recover all the replicas (the vector x) from few memory replica samples $m < n$ by the use of compressed sampling reconstruction algorithms [62, 87]. Efficient replica recovery is possible with compressed sampling: if we designed a *low complexity* protocol that samples the replicas *incoherently* with *few control channels*; and if we treated the memory replicas as *blocks of finite precision floating numbers* instead of low level binary bit streams. These insights enable the development of the proposed pull based memory replication protocol that does *not* have to learn which devices are active with an updated memory contents, while it pulls only memory replicas from a number of devices that is *far less* than n. The protocol works as follows (see Figure 2.4 for the messages flow).

2.4.1 Proposed Protocol

Suppose that k devices are active and updated their memory. Let p denote the memory page of a device i which is split into l blocks that are represented as finite precision floating numbers.

1. *An active device* i, upon updating p at time t, performs the following:
   1.1. creates the i-th memory replica, $x_{\mathrm{i}} \in \mathbb{R}$, as: $x_{\mathrm{i}} = \mathrm{float}(device : i,\ time : t,\ memory : p)$, where float is a function that casts the replica bits to a fixed point floating number.
   1.2. pushes x_{i} to randomly chosen neighbors in a *multicast* D2D communication.
2. *A receiving neighbor device* j performs the following:
   2.1. solicits memory replicas periodically from neighbor devices until it receives at least $d = O(log(n/k)/\epsilon$ updated replicas ($\epsilon \in (0, 1)$) from $N \subset Neigh(j)$ neighbor devices,
   2.2. aggregates all the received replicas into one compressed replica $y_{\mathrm{j}} = \phi_{\mathrm{jj}} x_{\mathrm{j}} + \sum_{\forall i \in N} \phi_{\mathrm{ji}} x_{\mathrm{i}}$ and stores only y_{j}, where $\phi_{\mathrm{ji}} \in \mathbb{R}$ is a predefined signature that the LTE edge cloud *initially* generates and assigns to the j-th device upon declaring another device i as a direct neighbor,
3. *The LTE edge cloud* performs the following:

3.1. randomly selects $m = O(k \log(n/k))$ out of n devices,

3.2. sends pulling requests to the selected devices using *pre-scheduled uplink grants* such that: the eNB can pull a device j only at $(j \mod m)$ occasions in the LTE frame, and the uplink grants include initial radio block allocation information for uplink transfer.

4. *A device* j remains in the dormant state and decodes possible uplink grants only at $(j \mod m)$ occasions; otherwise it remains in deep sleep to save energy. If j is pulled, it transfers y_{j} using the assigned radio blocks in its uplink grant (further dedicated control channels ensure that a radio conditions optimized radio blocks allocation after the initial assignment).

5. *The LTE edge cloud*, upon receiving the replica samples, finally recovers the k updated replicas by solving the l_1-minimization problem:

$$\underset{x}{\text{minimize}} \quad \| x \|_1 \quad \text{subject to} \quad \Phi x = y,$$

where x is the $n \times 1$ column vector such that its i-th element corresponds to the original replica x_{i}, y is the $m \times 1$ column vector such that its j-th element corresponds to the compressed replica y_{j}, and Φ is the $m \times n$ matrix such that its element in the j-th row and the i-th column corresponds to the signature ϕ_{ji} or equals zero if such signature was not used in computing y_{j} (i.e. j did not receive x_{i}) or the cloud never defined it (i.e. i is not a direct neighbor to j).

2.4.1.1 Protocol Correctness

The REPLISOM memory replication protocol is an application of the theory of compressed sampling with sparse measurement matrices [30]. The correctness of REPLISOM depends on the properties of the Φ matrix (step 5 in the protocol) from the compressed sampling theory [62], and the minimum number of direct neighbors $|Neigh(i)|$ of any IoT device i. On the other hand, the computation and communication overheads of the D2D communication steps of REPLISOM (steps 1.2 and 2.1) depend on the minimum value of d (step 2.1) for an accurate recovery by the theory of sparse compressed sampling recovery [30, 82].

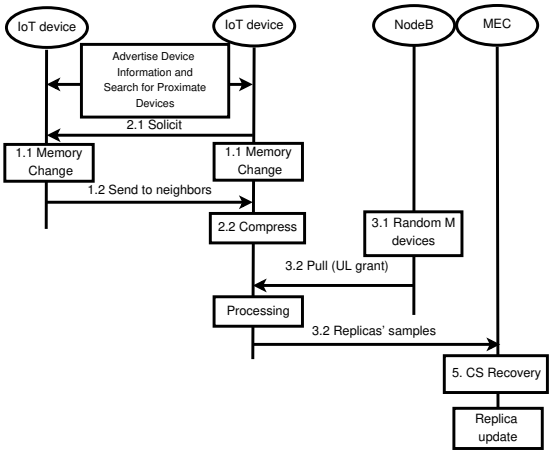


Figure 2.4: Proposed memory replication protocol.

The matrix Φ shall satisfy the *Restricted Isometry Property* (RIP), to ensure the correctness of REPLISOM and an accurate replicas recovery.

Definition 2.4.1 *An $m \times n$ matrix Φ is said to satisfy the $RIP(q)$ if, for any vector x that is k sparse, there exists a constant δ such that*

$$(1-\delta)\|x\|_q \leq \|\Phi x\|_q \leq \|x\|_q$$

An accurate recovery is possible if the matrix Φ is sparse and satisfies the $RIP(1)$ property (see Theorem 4 in [30] for formalism).

Theorem 2.4.1 *The $m \times n$ signature matrix Φ, with $m = O(k \log(n/k))$ and $d = O(log(n/k)/\epsilon)$, satisfies the $RIP(1)$ property for $\delta = 2\epsilon$.*

Proof see Appendix 2.6.

Numerically, a correct recovery depends on the exact number of neighbors, d, to which a device sends its memory replica in step 1.2 (i.e. the value of ϵ) and the number of active devices k. Intuitively, as k decreases, a device needs to send its replica to more neighbors to ensure information incoherence and a correct recovery. Figure 2.5 shows the probability to recover a single replica (out of k) with at most one-bit error and suggests that it is sufficient to design $d = 2\log(n/k)$ for an accurate replicas recovery.

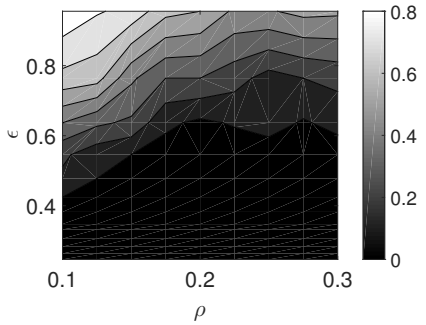


Figure 2.5: The probability to recover a memory replica with one-bit error for $n = 2000$, $\rho = k/n$, and $r \geq \sqrt{d/n/\pi}$ (i.e. $r \geq 27$ meters).

2.4.2 REPLISOM Improvement Through Utilizing Memory Sparsity

It is possible to further reduce the communication overhead of REPLISOM by utilizing the sparsity of memory pages deltas. Consider two consecutive memory pages of an IoT device, p_t and p_{t+1}. Typically, the memory page delta $p_{\text{delta}} = p_{t+1} - p_t$ represents an l vector of memory blocks where there are only s blocks that are non-zero. We refer to this as the *sparsity at the memory replica level*.

The straight forward approach to exploit such sparse structure of memory page deltas is to scan through p_{delta} and represent it using an $O(s \log(l))$ space-efficient sparse vector which contains only the non-zero blocks associated with their relative memory addresses. The memory replica of a device i is then constructed as $x_{\text{i}} = \text{float}(device : i,\ time : t,\ memory : p_{\text{delta}})$. As the device initially sends p_0 in full to the edge cloud, the cloud simply constructs subsequent pages from p_{delta} (e.g. $p_1 = p_{\text{delta}} + p_0$). This approach is not efficient for a large enough l as one must associates every non-zero block with its relative memory page address (i.e. requires $\log(l)$ bits) for sparse representation of p_{delta}.

We propose to use compressed sampling to exploit the sparsity of memory page deltas. Compressed sampling requires $w = O(s \log(l/s))$ bits to represent the s-sparse memory deltas. This approach does not require scanning through p_{delta} and works as follows:

1. the cloud generates a random Gaussian and *dense* $w \times l$ matrix, Υ_i, for each device i and sends Υ_i initially to the i-th device (a random matrix Υ_i is sufficient for exact recovery, see [62]),

2. a device i initially includes its p_0 in its replica x_{i},

3. as the device updates its memory, it constructs $x_{\rm i}$ as $x_{\rm i} = \text{float}(device : i,\ time : t,\ memory : p^c)$, where $p^c = \Upsilon_i p_{\rm delta}$,

4. upon recovering $x_{\rm i}$, as discussed in the memory replication protocol, the cloud recovers $p_{\rm delta}$ by solving the l_1-minimization problem:

$$\underset{p_{\rm delta}}{\text{minimize}} \quad \| p_{\rm delta} \|_1 \quad \text{subject to} \quad \Upsilon_i p_{\rm delta} = p^c,$$

5. finally, the cloud determines the full memory replica of the device $p_{t+1} = p_{\rm delta} + p_t$.

2.5 Benchmarks and Numerical Evaluation

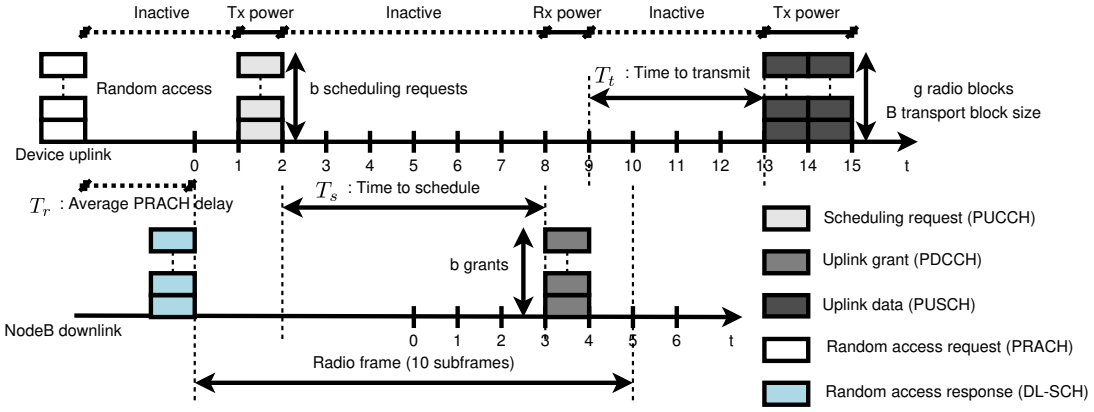


Figure 2.6: LTE time division duplex frame timing and main channels.

Before defining and evaluating the performance metrics for REPLISOM protocol compared to the *idle to active* and the *dormant to active* scenarios (Figure 2.2 and Figure 2.3), we first review the LTE radio frame structure, timing, control channels, and data channels.

2.5.1 The LTE Frame and Channels

The LTE time division duplex frame has an overall duration of 10 ms and consists of two half frames (downlink and uplink) each of 5 ms duration and a half frame consists of five subframes each of 1 ms duration. Each subframe carries physical control and data channels that carries logical channels information (see Section 4.5 from [2] for detailed mapping of logical channels to

transport and physical channels). The capacity and timing of these channels specifies the latency and energy consumption in replica transfers to the mobile edge cloud. Figure 2.6 illustrates the LTE frame along with the typical timing of these control and data information.

The Physical Uplink Shared Channel (PUSCH) is the traffic channel used for an uplink data transmission. The PUSCH contains g radio blocks in each subframe for data transmissions from at most g simultaneous devices. According to its measured radio condition at time t, the i-th device can transmit at most B_t^i bits defined as the transport block size. The device determines B_t^i according to its radio condition (translated into a modulation coding scheme) and the total number of allocated radio blocks (refer to the Table 7.1.7.1-1 and Table 7.1.7.2.1-1 from [1] for details). It is sufficient for our scope to assume that the transport block size is the same for all devices and is time independent. We refer to the transport block size as B.

Separate dedicated and common control channels are responsible for the transport of the radio interface control messages. The uplink half frame contains the Physical Uplink Control Channel (PUCCH) which carries the uplink scheduling requests, which a device uses to request for the PUSCH resources. Generally, each subframe can contain up to b simultaneous scheduling requests from different b devices. Moreover, the uplink half frame contains occasions of the Physical Random Access Channel (PRACH) which carries the random access request information for the initiation of the random access procedure (see Section 5.7 in [3] for random access timing).

The downlink half frame carries two main control information that are necessary for uplink transmission. First, the random access response in the Downlink Shared Control Channel (DL-SCH) which is addressed to a specific device that previously sent a random access request. The time between sending the random access request and receiving the random access response is the average random access delay, T_{r} (Average PRACH delay in Figure 2.6). The collisions during random access, the contention resolution procedure, and the propagation delay determine the actual value of T_{r}. Assuming a collision free random access, recent studies suggests that T_{r} is between 47 ms and 55 ms [5, 80]. The second control information is the uplink grants which is sent on the Dedicated Downlink Control Channel (PDCCH) that is designated to a specific device which previously sent an uplink scheduling request. Upon receiving a scheduling request it takes $T_{\mathrm{s}} \approx 10$ ms (Time to schedule in Figure 2.6) to process the request at the eNB and send the uplink grant to the requesting device. The eNB schedules an uplink transmission for any device after $T_{\mathrm{t}} \geq 4$ ms (Time to transmit in Figure 2.6) from receiving its uplink grant.

Symbol	Definition	Default Value / Range
n	total devices per *small* cell	1000 / Up to 50000
a	coverage area of eNB	60 meter^2 / 1 Km^2, $n = 50000$
k	active devices per cell	300 / 30 to 300
m	REPLISOM pulled devices	$m = 2k\log(n/k)$
d	D2D neighbors	$d = 2\log(n/k)$
b	scheduling requests per subframe	18 / 1 to 18
g	radio blocks per subframe	32 / $\{4, 8, 16, 32\}$
B	transport block size per radio block	408 bits / 16 to 584 (see 3GPP)
l	memory replica size	512 bytes / 16 to 2048 bytes
s	memory replica sparsity	$0.1l$ / $0.1l$ to $0.3l$
w	compressed replica size	$w = 2s\log(l/s)$
P_{rx}	consumed power during receive	100 mW
P_{tx}	consumed power during transmit	316 mW
P_{inact}	consumed power during inactivity	10 mW
P_{d2d}	consumed power during D2D	0.09 mW
T_{r}	average PRACH delay (collision free)	50 ms / 47 to 55
T_{s}	time to schedule	10 ms
T_{t}	time to transmit	4 ms

Table 2.1: Parameters summary and values for numerical evaluations.

2.5.2 Memory Replication Performance

We use system level simulations to evaluate the performance of the implemented protocol. Appendix B shows the detailed results of the system level simulations used in this evaluation.

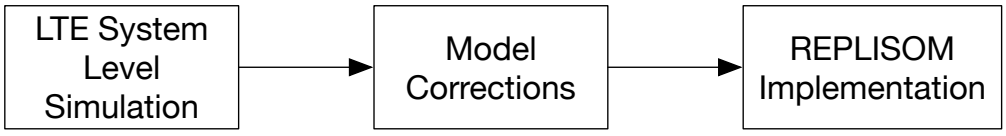


Figure 2.7: Systen level simulations.

Three performance metrics determine the efficiency of the memory replication from k simultaneous active devices in mobile edge computing. First, *the total allocated control channels*, which defines the total number of control channels, C, that the eNB allocates for devices to access the network and initiate an uplink data transmission. Second, *the total replication delay*,

which measures the total time $T = \sum_{i=1}^{k} T_i^{\text{acc}} + T_i^{\text{data}}$ that is required to update the k replicas; where for a device i, T_i^{acc} denote the access latency and T_i^{data} denote its replica transfer time (and already includes encryption overhead). Third, *the total consumed energy*, which measures the total energy $E = \sum_{i=1}^{k} P_{\text{rx}} \times T_i^{\text{rx}} + P_{\text{tx}} \times T_i^{\text{tx}} + P_{\text{inact}} \times T_i^{\text{inact}}$ to update the k replicas; where P_{rx} and T_i^{rx} denote the consumed power during a reception and the receive time of device i; P_{tx} and T_i^{tx} denote the consumed power during a transmission and the transmit time; and P_{inact} and T_i^{inact} denote the consumed power during an inactivity and the inactive time at which i only monitors the control channels in an optimized Discontinuous Reception (DRX) mode. Table 2.1 summarizes all the used parameters, and the notation definitions with their numerical values that are used in our following evaluation. Table 2.2 summarizes the three benchmarks, based on the timing and the channels definitions in Figure 2.6, for our proposed memory replication protocol, REPLISOM , with and without applying compressed sampling on the memory deltas and compared to the *idle to active* and *dormant to active* scenarios (Figure 2.2 and Figure 2.3).

Memory replication	C	T [ms]	E [μJ]
REPLISOM	m	$2\frac{d\times l}{B} + \frac{m\times(T_{\text{t}}+1)}{b} + \frac{m\times l}{(g\times B)}$	$P_{\text{d2d}}\frac{k\times l}{B} + m\left(P_{\text{inact}} \times T_{\text{t}} + P_{\text{tx}}\frac{l}{B} + P_{\text{rx}}\right)$
REPLISOM (with compressed replicas)	m	$2\frac{d\times w}{B} + \frac{m\times(T_{\text{t}}+1)}{b} + \frac{m\times w}{(g\times B)}$	$P_{\text{d2d}}\frac{k\times w}{B} + m\left(P_{\text{inact}} \times T_{\text{t}} + P_{\text{tx}}\frac{w}{B} + P_{\text{rx}}\right)$
Idle to Active (collision free random access)	$4k$	$\frac{k\times(T_{\text{r}}+T_{\text{s}}+T_{\text{t}}+2)}{b} + \frac{k\times l}{g\times B}$	$k\left(P_{\text{inact}}\left(T_{\text{r}}+T_{\text{s}}+T_{\text{t}}-1\right) + P_{\text{tx}}\left(\frac{l}{B}+2\right) + 2P_{\text{rx}}\right)$
Dormant to Active (ideal with no random access)	$n+k$	$\frac{n}{k\times b} + \frac{k\times(T_{\text{s}}+T_{\text{t}}+2)}{b} + \frac{k\times l}{g\times B}$	$k\left(P_{\text{inact}}\left(\frac{n}{k\times b}+T_{\text{s}}+T_{\text{t}}\right) + P_{\text{tx}}\left(1+\frac{l}{B}\right) + P_{\text{rx}}\right)$

Table 2.2: Benchmarks of proposed memory replication protocols comparing current LTE generic uplink transmission procedures.

2.5.2.1 The total allocated control channels

Since, REPLISOM requires only m device to be pulled, while each device consumes one uplink grant message, the eNB allocates a less number of control channels (m) compared to using conventional LTE procedures for the uplink data transfer (see Figure 2.8). Unlike the conventional LTE procedures which require a number of control channels that scales linearly in n, the number of channels in REPLISOM scales logarithmically in n. In the *dormant to active* scenario, the eNB allocates scheduling occasions to all n devices whether these devices will use them or not in addition to k uplink grants to the active devices. Although this behavior, minimizes the delay and power consumption, it significantly wastes the network resources as it requires the allocation of

$n+k$ control channels. In the *idle to active* scenario, the eNB allocates four control channels per device (a random access request, random access response, scheduling request, and uplink grant) requiring a total $4k$ control channels. Although, this scales linearly with k, as k increases this scenario requires a greater number of channels than those required by REPLISOM . Moreover, if the random access occasions, L, are not sufficient compared to the random access intensity γ, the number of control channels used in the *idle to active* scenario increases significantly due to collisions.

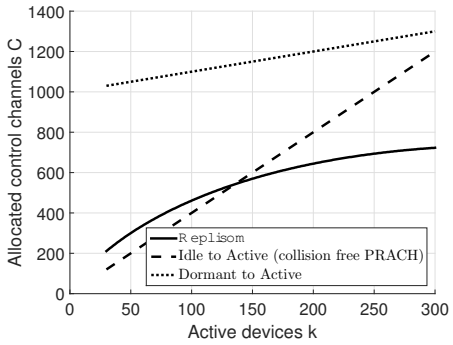


Figure 2.8: Allocated control channels for the proposed protocol comparing conventional LTE scenarios.

2.5.2.2 The total replication delay

Given the LTE current procedures, the *dormant to active* scenario achieves the lowest possible uplink replication delay that one can hope for (on the expense of wasted control channels). Similarly, the *idle to active* (under the collision free assumption) scenario resembles our assumed LTE worst case performance. In both cases the total replica transfer time is given by $T^{\mathrm{data}} = \frac{k \times l}{g \times B}$ for all the k active devices since the number of radio blocks needed to transmit an l-bits replica is l/B, and the network can transmit at most g blocks simultaneously. The total access delay in the *idle to active* scenario is given by $T^{\mathrm{acc}} = \frac{k \times (T_{\mathrm{r}} + T_{\mathrm{s}} + T_{\mathrm{t}} + 2)}{b}$, which is dominated by the average random access delay, T_{r}, per device. While in the *dormant to active* scenario the access delay is significantly reduced as it only takes: two subframes for sending a scheduling request and receiving an uplink grant, T_{s} subframes to schedule the uplink grant, T_{t} subframes to start the uplink transmission, and $\frac{n}{k}$ subframes between successive scheduling occasions for up to b devices to access the network simultaneously (i.e. $T^{\mathrm{acc}} = \frac{n}{k \times b} + \frac{k \times (T_{\mathrm{s}} + T_{\mathrm{t}} + 2)}{b}$).

REPLISOM reduces the total delay required to start replica transmissions in two ways. First,

as active devices send replicas to their neighbors in parallel it takes no more than $2\frac{d\times l}{B}$ subframes for the devices to construct replica samples regardless the number of active devices k (step 2.2). Second, as it only takes one uplink grant message to pull the replica samples from the devices, it requires $T_{\mathrm{t}}+1$ subframes before b devices start to transmit their replica samples (step 4). On the other hand, the total time that is required to transmit all replicas is governed by the values of m and l as $T^{\mathrm{data}}=\frac{m\times l}{g\times B}$. As the replica size increases, the total delay of the proposed protocol becomes strongly dependent on the uplink data transmission phase and is observably greater than the total delay in the *dormant to active* scenario (see Figure 2.9). Fortunately, the sparse structure of memory page deltas improves this undesired behavior where the total size of data transmission improves by a w/l factor. In general, the smaller the replica-size, the improved total delay we observe compared to the conventional LTE scenarios. This restricts the applicability of REPLISOM to replication of tiny sized memory pages (see Figure 2.10). Figure 2.9 shows that for the same replica size l, the proposed protocol (with compressed replicas) exhibits a total replication delay as the *ideal* dormant to active scenario and is slightly better as k approaches $0.3n$.

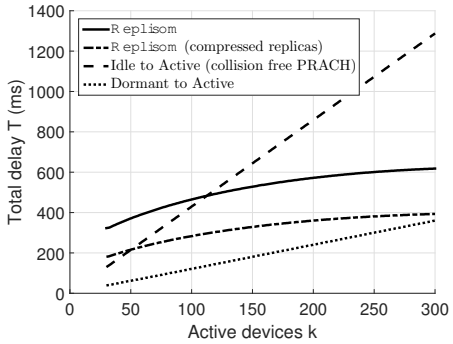


Figure 2.9: Total replication delay.

2.5.2.3 The total consumed energy

The total consumed energy of REPLISOM is generally worse than the conventional LTE scenarios because it requires m devices to become active compared to k devices in the conventional LTE scenarios (see Figure 2.11) although the energy consumed per device during a single replica transmission is significantly less. To see this, consider the inactive, transmit, and receive duration of a single device in the total consumed energy of Table 2.2. In REPLISOM , a device first

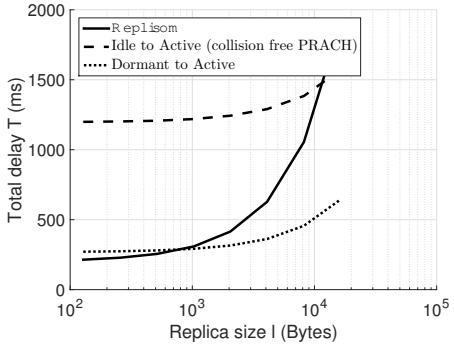


Figure 2.10: Replica size impact on delay.

transmits its replica to its neighbors with a low power for a duration of l/B subframes. Then, the device consumes P_{inact} power for a duration of T_{t} compared to $T_{\text{r}} + T_{\text{s}} + T_{\text{t}} - 1$ subframes in the *idle to active* scenario and to $\frac{n}{k \times b} + T_{\text{s}} + T_{\text{t}}$ subframes in the *dormant to active* scenario. For a replica transmission, a device consumes P_{tx} power for a duration of l/B subframes, that is reduced to w/B if the sparsity at the memory level is utilized, compared to $l/B + 2$ and $l/b + 1$ in the *idle to active* and the *dormant to active* scenarios respectively. A pulled device, in REPLISOM , consumes less energy at each pulling occasion, but since a device becomes active more often than in the conventional LTE scenarios it consumes more energy on a longer term.

Fortunately, the energy consumption disadvantage of REPLISOM does not hold true for small enough memory replicas (tiny replicas). This is illustrated in Figure 2.12 where energy consumption improves by reducing the replica size, which we also attain by utilizing the sparsity at the memory level. As the memory replicas become smaller, the less energy consumption per a single device activity becomes the dominant energy factor.

Generally, REPLISOM has delay and energy advantages over the conventional LTE scenarios if: the replica size is sufficiently small, or an LTE operator is limited in the number of resource blocks for control channels. Both conditions are of significant practical importance. Although there can be a large number of IoT devices, an individual device generally replicates a small sized data objects (see example applications in [5]). Additionally, the number of radio blocks that are allocated for control channels is limited by the maximum LTE bandwidth (20 MHz) and it is generally in an operator interest to allocate most of the radio blocks as data blocks for conventional network users that have tough quality of service requirements.

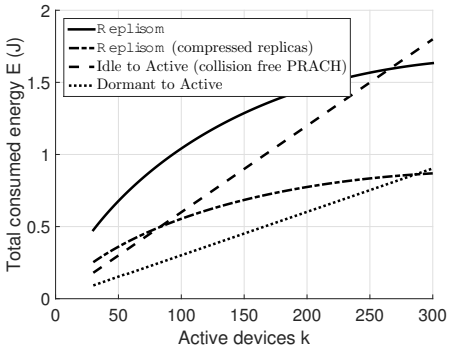


Figure 2.11: Total consumed energy.

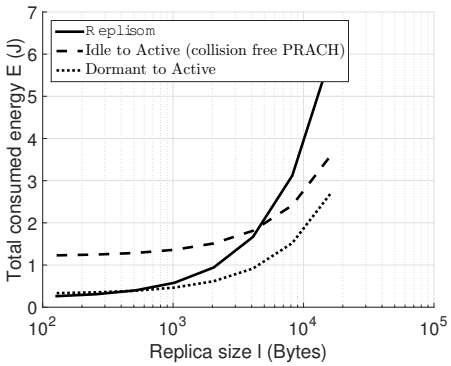


Figure 2.12: Replica size impact on power.

2.5.2.4 Impact of Different Parameters

The earlier discussion influenced the impact of the transport block size, B on the delay and energy consumption. It is obvious that as the radio conditions improve and B increases the total delay shall decrease and the device shall consume less energy (as it transmits for a shorter duration); but how the radio condition, hence the transport block size, does influence the delay of REPLISOM compared to the conventional LTE procedures? The poor radio conditions significantly reduce the transport block size, B, and render the proposed protocol to exhibit greater delay than the *idle to active* scenario. However, for moderate and good radio conditions and for the same memory replica size, l, the delay improves rapidly, so as the energy consumption, to approach the *dormant to active* performance as shown in Figure 2.13.

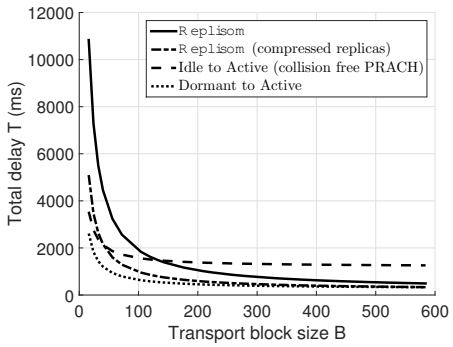


Figure 2.13: Transport block size impact on delay.

The total number of radio blocks, g, that are available per subframe also improves the total delay of REPLISOM , but has no impact on the energy consumption (see Figure 2.14). As the radio resources available to the eNB increase (e.g. increase bandwidth), the delay decreases rapidly. However, if the radio resources are limited as in the scenarios where human communication consumes most of the available radio resources, the total delay of REPLISOM becomes worse than the *idle to active* scenario.

2.6 Proof of Theorem 2.4.1

In this appendix we prove that Φ satisifies the $RIP(1)$ property, hence recovers all memory replicas x accurately (Theorem2.4.1). If one shows that Φ relates to the adjacency matrix of an expander graph, then it satisfies the $RIP(1)$ property. Let $G = (U, V, E)$ be a left-d-regular

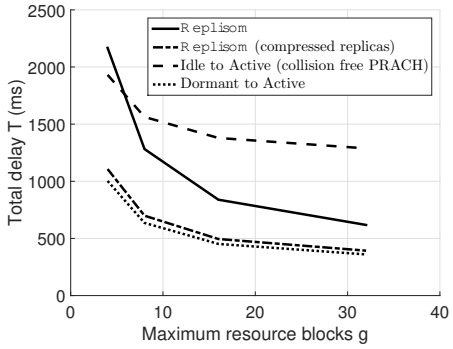


Figure 2.14: Radio blocks per subframe impact on delay.

bipartite graph, where U is its set of left vertices, V is its set of right vertices, and $E \subseteq U \times V$ is its set of edges, such that every left vertex in U has exactly d neighbors in V.

Definition 2.6.1 *A left-d-regular bipartite graph $G = (U, V, E)$ is an (k, d, ϵ)-expander if any set $S \subseteq U$ of at most k vertices has at least $(1 - \epsilon)d|S|$ neighbors.*

ASSUME:

1. $G = (U, V, E)$ is the left-d-regular bipartite graph such that: U represents all the n IoT devices and V represents all the m selected devices by the edge cloud, a left node i is connected to a right node j if the later received the replica x_{i} (in step 1.2 or 2.1), and A is the adjacency matrix such that $A_{ij} = 1$ *iff* $(i, j) \in E$

2. Ψ is a random i.i.d $m \times n$ matrix such that $\Psi_{ij} \propto 1/d$ and $\Phi = A \circ \Psi$ ($\circ$ denote matrix element-wise multiplication)

PROVE: G is an (k, d, ϵ)-unbalanced expander

Proof 1. The probability that a left vertex i has at least d neighbors for a network density $\eta = \frac{n}{a}$ is given by

$$Pr(|Neigh(i)| \geq d) = \left(1 - \sum_{i=0}^{d} \frac{(\eta\pi r^2)^i}{i!} e^{-\rho\pi r^2}\right)^n$$

(see Theorem 2 in [32] for details)

2. for a dense network (e.g. $n = 50000$, $a = 1$, $r = 0.2$), i has at least d neighbors almost surely

3. for $S \subset U$ such that $|S| \leq k$ and $M \subseteq V$ such that $|M| \leq m$, the neighborhood of S is completely contained in M with probability

$$Pr(Neigh(S) \subseteq M) \leq \left(\frac{|M|}{m}\right)^{d|S|}$$

4. G is not an expander if $|M| \leq (1-\epsilon)d|S|$

5. Let Pr' denote the probability that G is not an expander and is bounded by

$$\begin{aligned} Pr' &\leq \sum_{i=1}^{k} \binom{n}{i} \binom{m}{(1-\epsilon)di} \left(\frac{1-\epsilon)di}{m}\right)^{di} \\ &\leq \sum_{i=1}^{k} \left[\underbrace{\left(\frac{ne}{i}\right) \left(\frac{me}{(1-\epsilon)di}\right)^{(1-\epsilon)d} \left(\frac{(1-\epsilon)di}{m}\right)^{d}}_{z} \right]^{i} \\ &\leq \sum_{i=1}^{\infty} z^i = \frac{z}{1-z} \end{aligned}$$

6. as $i \leq \rho n$ and $d = O(log(n/k)/\epsilon)$, then $z \leq \frac{1}{10}$ and $Pr' \leq \frac{1}{8}$

Chapter 3: Replisom7: Reliable and Secure Data-Centric Cloning for IoT Devices in Edge Computing

3.1 Introduction

Computation offloading of resource-constrained devices with a fine-grain memory replication has been the focus of research accompanying the evolution of mobile and pervasive computing [25, 54]. New forms of cloud platforms (e.g. cloudlets, edge computing) emerged to provide computing resources where Big Data analytics is performed near the data sources instead of cloud-centric data-lakes. The success of Edge computing paradigm requires efficient replication of devices' data onto clones that run in high volume compute nodes co-located with existing cellular sites [12]. Two major factors pertain to an efficient memory replication: 1) communication over heterogeneous transport networks (see Figure 3.1), and 2) support of memory replication from a massive number of devices in limited geographical areas.

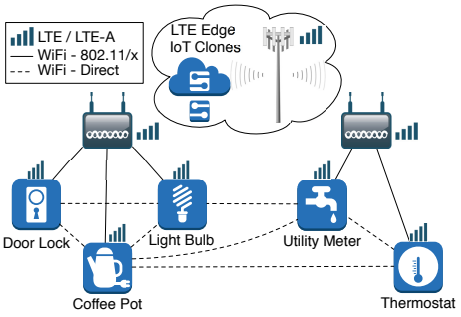


Figure 3.1: Heterogeneous Transport Networks For Devices and Edge Computing Communication. Devices can replicate their memory through direct uplink transfer, relaying data through a WiFi-LTE access point, or organize themselves in a spanning tree using RPL.

The REPLISOM architecture - we propose in [12] - is an edge computing architecture that augments the evolved NodeB (eNB) of LTE/LTE-A with integrated cloud elements. The cloud elements provide virtualizable compute, storage, and network resources for diverse IoT applications and services (see [109] for example appliactions). REPLISOM utilizes the capabilities of the D2D communication technologies and the existing LTE control and data channels to im-

prove the end-to-end memory replication delay and devices' energy consumption through an LTE-optimized protocol.

REPLISOM requires a cross-layer design between the memory replication protocol and the LTE/LTE-A stack which complicates REPLISOM implementation in practice. To achieve its promised performance gains, REPLISOM devices shall not perform the Random Access Procedure for it uplink replica transfers, which accounts for the major memory replication delay given a massive number of devices in an LTE site [12]. As REPLISOM relies on pulling memory replicas from devices, REPLISOM allows devices to remain idle and in low-power mode and only listen to memory pulling occasions at predefined time-slots. Such requirements require non-trivial changes to existing LTE/LTE-A implementations.

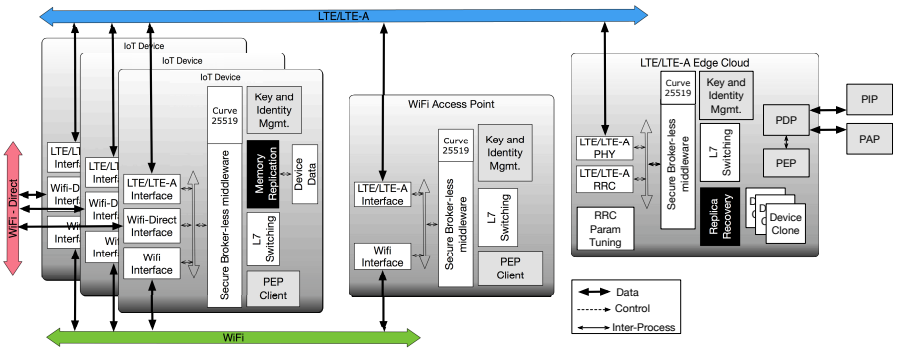


Figure 3.2: Replisom7 proposed architecture.

3.1.1 Background and Contribution

In this paper, we implement and evaluate REPLISOM as an application layer protocol instead of an LTE cross-layer protocol. We call this protocol version Replisom7. Like its original design - REPLISOM - Replisom7 uses compressive sensing to efficiently pull and reliably recover devices' data. Replisom7 uses ZeroMQ [89] as communication middleware over diverse co-existing technologies including LTE, WiFi, and WiFi-Direct (for device-to-device communication). We compare Replisom7 design approach to conventional device data transfer protocols such as MQTT and CoAP and demonstrate Replisom7's superiority in terms of native support of reliable memory replication.

Replisom7 also uses a federated OpenID [138] for devices identity management and authentication and adopts elliptic curve cryptography [101] of ZeroMQ for key management and en-

cryption. Finally, Replisom7 provides fine-grain access control and resource discovery through the XACML [71] standard. Through XACML, Replisom7 govenrs devices permissions to communicate over D2D links and to the Edge Cloud, as well as, defining entitlement of devices' services.

We also show that the lack of cross-layer awareness in Replisom7 prevents it from achieving the promised delay and energy consumptions gains of REPLISOM. To mitigate an undesired lack of performance due to cross-layer implementation absence, we show through emulations that through optimized LTE RRC parameters for paging and random access, we optimize Replisom7's operation to minimize the delay and energy consumption compared to existing techniques.We call this optimized protocol Replisom+.

Existing techniques include: LTE uplink data transfer procedure, data relaying in WiFi-LTE coexistence scenarios, and data routing through minimum spanning tree protocols such as the Routing Protocol for Low-Power and Lossy Networks (RPL). In LTE uplink data transfer, devices use their LTE modem to directly replicate their memory content using 3GPP uplink data transfer standards. The details of this procedure is described in [12]. Devices can also relay their data to the LTE Edge Cloud through a WiFi-Lte access point. A WiFi access point connects a handful number of devices (typically four devices), and routes their data to the LTE-Edge cloud through its LTE interface. In the last scenario, devices organize themselves in a spanning tree through the RPL routing protocol, in which the tree root (RPL DODAG root) acts as the gateway to the LTE Edge Cloud. Figure 3.1 summerizes the networking technologies and these existing approaches.

We describe Replisom7 architecture in Section 3.2 and its management-plane, data-plane, and control-plane functionality in Section 3.3. We demonestrate our experimental evaluation in Section 3.4 and conclude the paper in Section 3.5.

3.2 Communication Middleware Architecture from Devices to their Clones

Replisom7 runs on three types of nodes: devices, access points, and edge cloud servers co-located with a cellular site. Figure 3.2 illustrates Replisom7 architecture. A node can support one or more communication technology (LTE, WiFi, or WiFi direct). A communication technology interface optimises the node communication parameters according to the underlying technology. Any node minimally runs a secure communication middleware and a Layer-7 data switching function.

3.2.1 Switching and Memory Replication

We use ZeroMQ ([89]) as a lightweight communication middleware with a request/response servlet (server/client) in each node. A node x defines a list peer nodes (neighbors) that it directly communicate with, and a gateway node to forward messages to through a Layer-7 switching function if the message's destination node that is not included in x's list of peer-nodes. The Layer-7 Switching function in access points and cloud servers also organises the network topology on the device-to-device underlay. For example, we use the cloud's switching function to organise the devices as spanning trees instead of using the distributed control-plane of RPL as a potential scenario for evaluation.

A Memory Replication Function reliably and timely replicates devices' data to a Replica Recover function hosted on the edge cloud servers. The Replica Recovery Function orderly recovers the devices' data and pushs the recovered replicas to devices' clones that we implement as isolated processes in edge clouds. Clones act as surrogates of the devices and implement the actual IoT applications. Clones can also act on behalf of the devices for continous computaions in case the device become offline. In existing scenarios, the Memory Replication Function is simplly pushing, publishing, or sending the devices' data to the Memory Replica Function. We will discuss how we design the Replication and Recovery functions using compressive sensing in $\mathrm{Replisom7}$ design in Section 3.3.

3.2.2 Security and Discovery

$\mathrm{Replisom7}$ incorporates message encryption, authentication, and access control functions. We use elliptic curve cryptography to encrypt all data messages communicated through ZeroMQ on the data-plane according to Curve25519 specifications. On the control-plane, the Key and Identity Management function handles Crypto-keys management and node's authentication. The LTE Edge Cloud hosts an Identity server and uses the OpenID protocol to authenticate nodes [138]. OpenID allows seamless authentication of devices and access points using their users existing accounts.

We realise access-control via the XACML standards. Nodes' Policy Enforcement Point (PEP) Clients send access requests to a unique PEP running at the LTE Edge Cloud to obtain access control, entitlements, and obligation decisions. The edge cloud runs a Policy Decision Point that evaluates PEP requests according to XACML policies defined by the Edge Cloud

Property	Replisom7/+	MQTT	CoAP
Communication Model	Peer-to-Peer	Broker Publish/Subscribe	Client/Server or Peer-to-Peer
Transport	TCP or UDP	TCP	UDP
Reliability	Native (no explicit QoS)	QoS: delivered at least once	QoS: confirmable
Authentication	Federated OpenID	username/password	EAP
Security	CurveZMQ	SSL/TLS	DTLS
Resource Discovery	XACML Obligations	None	Server Side Query
Cross Layer Support	LTE-RRC and multi wireless Interface	None	None

Table 3.1: Replisom7 vs MQTT vs CoAP.

administrator through a unified Policy Administration Point (PAP). A Policy Information Point (PIP) stores policies attribute information that a Policy Decision Point (PDP) uses to form a decision or to add obligations to the decisions.

Access-control entitlement decision determines if two nodes are permitted to communicate directly with each other, if a node is permitted to use a particular access point as a gateway, or if a node is permitted to replicate its memory with the LTE Edge Cloud. Entitlement decision contains obligation information that a node can use for services and node parameter discovery. For example, if node x receives an entitlement decision to communicate with node y, the entitlement decision can contain the public key of y and the memory replication parameters of y as we will detail in Section 3.3.

3.3 Compressive Sensing Based Memory Replication

Replisom7's design splits into management and data planes. The management-plane carries out authentication, authorization, and peer-to-peer connections obligations. The data-plane carries out the memory replication from devices to their clones.

3.3.1 Management-plane

At initialization, devices authenticate with the LTE Edge cloud through the PIP client. When a device i detects another neighbor device j in D2D communication range, it sends an access control request to the LTE Edge Cloud to authorize that i can replicate its memory by direct communication with j. If permitted, the Edge cloud sends a random signatures ϕ_{ij} to i and ϕji to j that i and j shall use for memory replication over the data-plane. Since a ϕ_{ij} is a 32-bit floating point number, the overhead associated with this per node is at most $32 \times \log(n/k)/\epsilon$-bits

for the life time of the node, where n is the total number of devices and $k \leq \lceil 0.3n \rceil$ is a design parameter that indicates the maximum number of simultaneously active devices.

The Edge cloud also sends - as an XACML obligation - a list of data-plane services that i must expose for its neighbors. Among the obligations, the Edge cloud include a parameter $N = \log n/k$ to i. During the replication process - as we will detail in the data-plane description - a device i shall send its memory replicas for at least N neighbor devices such that the Edge cloud can correctly recover the replicas. Since we seperate the management-plane from the data-plane, no management overhead is associated with the Replication process.

3.3.2 Data-plane

Conventionally devices send its data to the Edge Cloud over an LTE uplink session, through a WiFi gateway that relays devices data to the Edge Cloud, or through the root device in a tree than spans the network of devices. In the later case devices can be organized as a spanning tree where devices route data until it reaches a root node that can relay data to the LTE Edge Cloud (see for the Routing for Low Power and Lossy Networks for example [170]). Unlike conventional methods, $\mathrm{Replisom7}$ relies on compressive sensing to recover replicas from devices that are organized in an arbitrary topology and can communicate over a D2D technology.

Let R_i denote a floating point number that device i shall replicate with the Edge cloud and SEQ is a cyclic sequence number of R_i. A device i sends R_i and SEQ to at least N randomly chosen neighbour devices over a low-power, low-latency D2D technology such as WiFi-Direct. If a device j receives R_i it adds it to a replica store that maintains a Replisom Replica which is an aggregate value of j's received replicas and $y_j = \phi_{jj} R_j + \sum_{\forall i \in N} \phi_{ji} R_i$. A device j also maintains a list of devices' IDs with replica values evaluated in a Replisom Replica and a list of sequence numbers of these replicas values.

The Edge cloud periodically sends a $PULL$ message to at least m devices that are randomly choosen such that $m = \lceil k \log(n/k \rceil$. At the end of a $PULL$ cycle, a device can recover the original devices replicas by solving the l_1-minimization problem:

$$\underset{x}{\text{minimize}} \quad \| x \|_1 \quad \text{subject to} \quad \Phi x = y,$$

where x is the $n \times 1$ column vector such that its i-th element corresponds to the original replica R_i, y is the $m \times 1$ column vector such that its j-th element corresponds to the Replisom replica

Mode	μ_t,	σ_t	μ_τ,	σ_τ	α	β	σ_p
WiFi Transmit	79.1	15.1	1.0	0.5	283.17	132.86	50.0
WiFi Receive	79.1	15.1	4.5	2.0	137.01	132.86	50.0
LTE Transmit Idle	260.19	15.8	5.0	2.5	438.39	1288.04	100.0
LTE Receive Idle	260.19	15.8	12.5	5.0	51.97	1288.04	50.0
LTE Transmit Idle (no RA)	14.0	1.0	5.0	2.5	438.39	1288.04	100.0
LTE Receive Idle (no RA)	14.0	1.0	12.5	5.0	51.97	1288.04	50.0
LTE Transmit Connected	43.2	1.5	5.0	2.5	438.39	1288.04	100.0
LTE Receive Connected	43.2	1.5	12.5	5.0	51.97	1288.04	50.0
WiFi-Direct Transmit	8.3	4.0	1.0	0.5	0.09	0.1	0.01
WiFi-Direct Receive	8.3	4.0	1.0	0.5	0.09	0.1	0.01

Table 3.2: Delay and power models parameters.

y_j, and Φ is the $m \times n$ matrix such that its element in the j-th row and the i-th column corresponds to the signature ϕ_{ji} or equals zero if such signature was not used in computing y_j. Since $\mathrm{Replisom7}$ recovers the same replicas in several pulling ocasions, it provides a native reliability for replica recovery as we will demonestrate in Section 3.4.

3.3.3 $\mathrm{Replisom+}$: Control-plane optimizations

The latency and energy gains of REPLISOM (see [12]) are based on a cross-layer optimization over the LTE protocol stack. In its original design, REPLISOM skips the random access procedure during the LTE uplink data transfer and timely listens to Replisom Replica pulling occasions using the LTE paging procedure. Since we implement $\mathrm{Replisom7}$ as an application layer protocol, we cannot directly incorporate fine-grain redesigns to the LTE procedures.

To remain delay and power effective, we introduce two control-plane optimizations to $\mathrm{Replisom7}$ and name this newly optimized protocol $\mathrm{Replisom+}$. $\mathrm{Replisom+}$incorporates LTE timers optimizations that are not possible with conventional uplink data transfer in LTE. With these optimizations, devices can avoid long contentions during the LTE Random Access Procedure and transition to an Idle Mode state as quickly as possible to minimize delay and save power.

As devices waits for Random Access contention resolutions, devices remains in an active state and consume more power. $\mathrm{Replisom+}$sets the mac-ContentionResolutionTimer to its minimum value to prevent unnecessary power consumption and prolonged wait time. This may introduce more collisions in the network, if conventional uplink transfers from devices is present.

However, as devices do not engage in a device-initiated data uplink transfers, devices do not frequently activate the Random Access procedure except to maintain synchronization with the LTE-site.

The periodic Replisom replica pulling approach allow devices to remain in the connected mode of minimal duration of time and listen to pulling occasions on predefined intervals. Although in Replisom+, we do not change devices' behavior in the MAC protocol to listen to certain paging occasions, we set the tail timer to its minimum value to prevent devices from remaining in connected mode longer than necessary. Typically the tail timer takes a value of 11 seconds which is considered too long and unnecessary given that devices shall not expect downlink traffic from the Edge Cloud after the pulling their data.

Before demonstrating the performance of Replisom7 and performance enhancements of Replisom+, we provide a design comparison to existing application layer protocols in the IoT landscape. Table 3.1 summarizes the architectural and functional aspects of Replisom7 and Replisom+.

3.4 Evaluation

We evaluate Replisom7 and Replisom+in a controlled testbed environments where any node in the network is emulated as a simple Linux process that shall communicate over ns-3 TapBridges. TapBrdiges in NS3 integrates real internet hosts into ns-3 simulations, which simplifies rapid protoyping of real world protocols and leverages the advantages of relying on existing ns-3 models for evaluations (see [78, 92, 125, 178]).

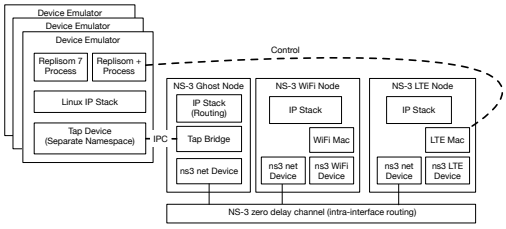


Figure 3.3: Device Emulators with NS3.

Figure 3.3 and Figure 3.4 show the interworking between the emulated nodes in the linux user space and the ns-3 nodes. For each nodes, a ghost ns-3 node communicate with its counter linux process over a tap bridge through virtual tunnels. Using network namespaces, we isolate the traffic over the user space nodes over a specific tap bridge. As device emulators require a

WiFi interface and an LTE interface, its corresponding ghost node route the traffic from device emulators to a corresponding ns-3 node (WiFi node or LTE node) according to the destination address of the packets as illustrated in Figure 3.3. Similarly, an Edge cloud's traffic is routed to a remote host node that's colocated with an eNodeb node inside ns-3 and communicate with it via a peer-to-peer link of 100 Gbps and 10 Micro Second delay. The ns-3 nodes representing the devices are organized on a grid with inter-node seperation distance of 50-meters. The devices' ns-3 nodes communicate over an $\mathrm{AdhocWifiMac}$ using the 802.11n standard at 5GHz with an $\mathrm{OfdmRate54Mbps}$ data mode.

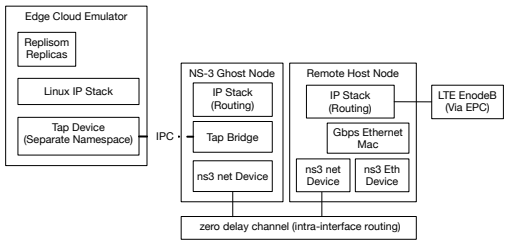


Figure 3.4: Edge Cloud Emulations with NS3.

A device generates a random 32-bit floating number as its data every 1 to 30 second chosen randomly according to a Poisson distribution. Power consumption during data transmission, reception and idle period are evaluated based on models proposed in [95, 22]. Unlike the complete numerical evaluation presented in [12] where we show REPLISOM's properties with a massive number of nodes, using ns-3 limits scaling the number of nodes beyond 40 nodes with ns-3 running using a 40 CPU/ 160GByte RAM server. ns-3 is also limiting the evaluation since the existing LTE module does not incorporate LTE paging procedure, support of more than 360 LTE interfaces, power consumption models, or LTE based device-to-device communication.

3.4.1 Delay and Power Models

Any node (device, access point, or LTE cloud) incroporates packet delays as real-time wall-clock seconds during send and receive according to the underlaying technlology. Delay values are modeled as $d = \frac{s}{\tau} + t$, where s is the packet size, t is network access delay, and τ is the transmission rate. The access delay is obtained as a random value $t \sim \mathcal{N}(\mu_t, \sigma_t^2)$, and the values of the transmission rate as $\tau \sim \mathcal{N}(\mu_\tau, \sigma_\tau^2)$. The exact values of μ_t, σ_t, μ_τ , σ_τ are evaluated based on experiments presented in [95, 22] and is given in Table 3.2. As replicas are 32-bit

floating point numbers, this make all packet payload less than 64 bytes and encryption time 2.8 ± 1.6 Micro Seconds, and decryption time 8.9 ± 5.57 Micro Seconds.

Power consumption during transmission and recepiton depends on the delay value d. We use a modefied version of the conjectured model in [95], to trace devices power and energy consumption as ns-3 does not incorporate such power models. A device power is given by $p = \alpha d + \beta + z$, where $z \sim \mathcal{N}(0,\, \sigma_p^2)$ and the parameters α, β, and σ_p are given in Table 3.2.

3.4.2 Results

We evaluation $\mathrm{Replisom7}$ and $\mathrm{Replisom+}$ compared to convetional uplink transfer in LTE (denoted LTE), routing replicas through a tree that spans devices (denoted Spanning Tree), and through using a WiFi gateways that connect four devices each (denoted WiFi-Relay). In each scenario, we compate the end-to-end delay per replica, the average energy consumption, and the loss rate for $n \in \{10, 20, 40\}$ and $k = 0.3n$ (maximum value of k for REPLISOM correct recovery).

Figure 3.5 shows the average delay per replica. As expected, $\mathrm{Replisom7}$ does not bring the promised delay improvement without control-plane changes to the LTE parameters and without adopting low power and low latency communication in WiFi-Direct [1].The limited scalability of ns-3 number of nodes stands still against demonestrating the delay improvement in REPLISOM for massive number of nodes as the LTE delay performance is effecient for the small number of nodes available to us in this experiment.

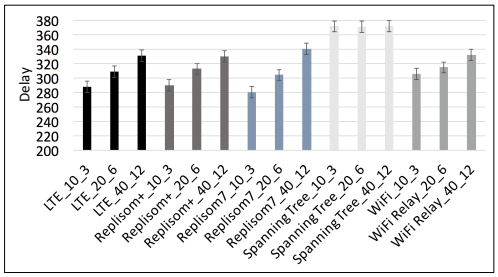


Figure 3.5: Average delay per replica in [ms] computed to a 0.1 error at 95% confidence.

Figure 3.6 shows the average energy consumption per replica and Figure 3.7 shows the fraction of time a node transitions into idle mode for minimal energy consumption. $\mathrm{Replisom7}$ and

[1] As NS-3 does not support WiFi-direct, we use WiFi-adhoc model instead with corrected parameters in Table 3.2

Replisom+ bring obvious energy savings to the devices as the number of devices increase. Although, the ns-3 experiments are limited to demonestrate the full potential, it's noticed that as the number of nodes n increases more devices transition to idle mode, where energy consumption per replica decreases. The reason behind this behavior is the Replisom7 and Replisom+ activate only $m = \lceil k \log(n/k) \rceil$ devices during a pulling occasions. As n increases, more nodes are allowed to remain in idle mode and save energy. WiFi relay shall always show the minimum energy consumption to devices, since devices do not communicate directly with LTE and offload intensive communication to the WiFi access points. This gain is achieved with investing a capital by deploying a large number of access points in a small and dense geographical region.

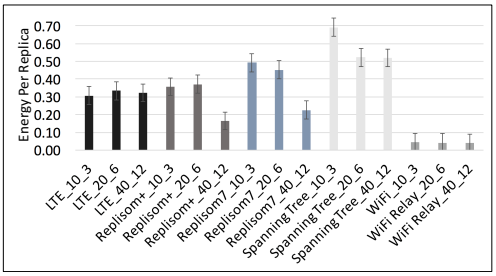


Figure 3.6: Average energy consumption per replica in [J] computed to a 0.1 error at 95% confidence..

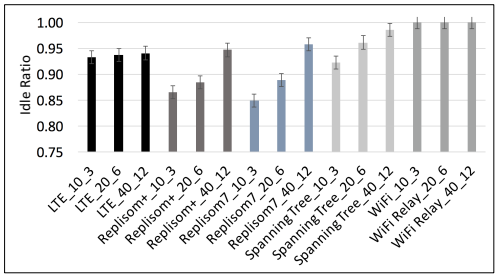


Figure 3.7: Idle mode rate computed as the number of Idle mode states to the total numner of states.

As Replisom7 and Replisom+ pull the same devices' replicas in several ocassions, they improve the overall reliability of the memory replication process. On the average, Replisom7 recovers the same replica in three replica pulling occasions, where the clone picks the majority value as the correct value. Such property is not natively attained in the design of other protocols. Figure 3.8 shows average loss rate measured for differnet scenarios. Replisom7 and Replisom+ remain resilient to packet loss. Both Replisom7 and Replisom+shows 90% improvement in Replica loss rate compared to Spanning Tree Protocols and 68% improvement

compared to LTE.

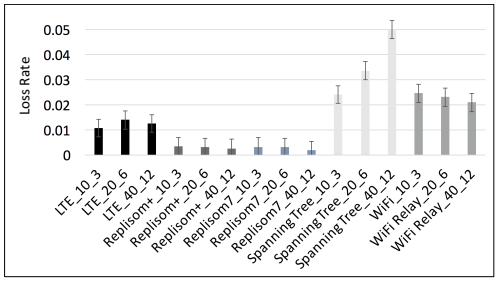


Figure 3.8: Replisom7 and Replisom+improved Loss Rate over other scenarios.

Our results suggest that minimum spanning tree protocols do not suit the heterogeneous network organization in Figure 3.1. The reason of this is the excessive routing overhead requirement in devices which make devices active almost all the time and engaged in transmission all the time, which increases energy consumption and introduces data transfer delays in intermediate nodes. Such minimum spanning tree protocols (e.g. RPL) better suits scenarios that lack network infrastructure.

3.5 Conclusion

We design and implement REPLSIOM as an application layer protocol (Replisom7) with authentication, authorization, access control and service discovery capabilities. We extend Replisom7 with a control plan in Replisom+ that derives its performance closer to the performance gains promised in REPLISOM. Using ns-3, we evaluate Replisom7 and Replisom+ and compare it to exiting methods that can be used for cloning devices' data in LTE Edge Cloud. Finally, we compare the design artefacts of Replisom7 and Replisom+ to MQTT and CoAP that are widely used protocols in IoT. Through emulated networks, we show that Replisom+ improves the energy consumption and end-to-end memory replication when compared to existing LTE protocols, minimum spanning tree protocols, and WiFi relaying.

Chapter 4: Efficient Virtual Network Embedding with Backtrack Avoidance for Mobile Networks of Clones

4.1 Introduction

Virtual network embedding in wireless networks can have a pivotal role in several areas including: sensor network virtualization [9], vehicular cloud [81], mobile edge computing [68, 112, 183], network based and geographically distributed cloud environment [17, 13], and cyber foraging [143]. By means of virtualization, it is possible to embed, with low cost, large-scale virtual sensor networks onto sensor-equipped physical devices (e.g. smart-phones, autonomous vehicles) so as to perform specific sensing tasks and autonomous, agile, and timely decisions in a distributed manner. Such virtual networks can support several applications such as: urban sensing, intelligent transportation, terrain exploration, disaster recovery, and surveillance. In addition, VNE can be used to enable virtual content delivery in wireless networks near the network edge. VNE algorithms can then deploy surrogates of services (e.g. networked virtual servers) in proximity to users to improve their perceived latency, where geographical locations and mobility patterns of users are crucial parameters to maintain a target content delivery quality. In a more general context, virtual network embedding in wireless networks can enable effective distributed processing of real-time content and allow agile decision making from data at its "actual sources".

The focus of this paper is on the design of virtual network embedding (VNE) techniques that enable on-demand mapping of virtual networks onto substrate mobile wireless networks. More specifically, the VNE problem consists of mapping the virtual nodes to substrate nodes and the virtual links to substrate paths in such a way that all resource (CPU, storage, and bandwidth) requirements of the virtual network are met. Here, a virtual network consists of a set of virtual nodes, each requiring CPU processing capability and storage capacity to process data in a predefined geographical area, and a set of virtual links connecting these virtual nodes, each requiring some bandwidth capacity. The substrate network, on the other hand, consists of a large set of mobile wireless nodes, each having sensing and Internet-access capabilities.

Unlike wired networks, mobile wireless networks' dynamics (e.g. node mobility, link instability) create new challenges that require new architectural and algorithmic considerations when

it comes to enabling VNE. Mobility of substrate nodes, in particular, may invalidate the operations of virtual networks as nodes move away from desired locations of some virtual nodes. Such a mobility can also change the connectivity of the substrate nodes–and so can the substrate paths–that are already used by virtual links, making them insufficient or invalid. In such cases, VNE solutions shall remap (migrate) invalid virtual networks to other substrate nodes and paths [93]. As migrations incur a significant overhead [117], we shall design architectural and algorithmic solutions that can effectively capture node mobility and topology changes, and minimize virtual network migrations due to nodes mobility while not compromising the effectiveness of VNE techniques.

The effectiveness of the VNE techniques can essentially be captured through three metrics: computation time (the time it takes to solve a VNE instance), embedding cost (the amount of overhead incurred and resources needed to solve a VNE instance), and acceptance rate (the ratio of successfully solved VNE instances to the total number of instances). Therefore, in addition to meeting the resource requirements, the aim of VNE techniques is to reduce the computation time, minimize the embedding cost, and increase the acceptance rate. The challenge, however, is that these three performance goals are often conflicting with one another. For instance, backtracking algorithms can, in general, find optimal solutions, but they do so in exponential time [174]. Other heuristic approaches, on the other hand, can find solutions in polynomial time, but these solutions are sub-optimal, thus leading to low acceptance rates [72].

In this paper, we develop VNE techniques that strike a good balance between these three performance goals by finding near optimal solutions in polynomial times (short execution times) while yielding high embedding profits (minimal embedding costs) and high acceptance rate. The proposed approach takes also into account potential virtual networks migrations due to substrate nodes mobility in its objective definition to minimize the anticipated overhead associated with migrating invalid virtual networks. Our proposed approach consists of designing algorithms that are based on backtracking techniques so as to ensure good solution optimality, while reducing the computational complexity and the embedding cost by exploiting the constraint propagation properties of the VNE problem. Essentially, they reduce the embedding complexity and cost by narrowing down the search space and avoiding backtracking as much as possible without compromising the solution quality so as to maintain high acceptance rates and minimize potential virtual network migrations. To recap, our contributions in this paper are twofold.

- Developing pruning techniques that reduce the embedding time and cost significantly by reducing the search space. These techniques eliminate the need for backtracking during

the embedding solution search, thereby enhancing the embedding time without compromising the optimality of the obtained VNE solutions.

- Developing techniques that account for the VNE embedding cost, expressed in terms of the amount of resources needed and the migration overhead incurred to successfully embed a virtual network, to devise VNE algorithms with minimal embedding costs and minimal potential virtual network migrations.

The rest of the paper is organized as follows. The next section surveys the existing techniques that are related to our proposed VNE approach. In Section 4.3, we state and formulate the VNE problem. We begin by modeling the virtual and substrate networks and the substrate node mobility, and by defining the node and link mapping steps to be performed during the VNE process. We then describe the overall design goals of the VNE technique. In Section 4.4, we present our pruning techniques proposed to reduce the embedding search space. We then, in Section 4.5, use these pruning techniques to develop a polynomial-time VNE algorithm, which leverages the benefits of our proposed pruning techniques to avoid backtracking while still maintaining the optimality of the obtained VNE solutions. In the same section, we also derive analytic bounds on the approximation ratio of the incurred objective value of the proposed algorithm. Finally, we present our experimental results and findings in Section 4.6.

4.2 Related Work

Virtual Network Embedding: There have recently been research efforts aiming to develop VNE algorithms, and the recent survey by Fischer et al. [72] presents a detailed classification of such algorithms. Broadly speaking, these algorithms can be classified into three categories: backtracking based algorithms (e.g. branch and bound), stochastic algorithms, and heuristics.

Backtracking based algorithms generally consist of formulating and solving the VNE problem using branch and bound or exact backtracking based techniques [116, 94, 37, 49, 38]. For example, Lischka et. al. [116] show that the VNE problem can be formulated as a graph isomorphism (which is known to be *NP-hard*) and then using a backtracking based algorithm to solve it. Backtracking can, in general, find optimal solutions. However, they do so in exponential time [57].

Stochastic algorithms like simulated annealing, particle swarm optimization, tabu search, or genetic algorithms, are other common approaches that can be used to search for VNE solutions. For example, [181] uses particle swarm optimization to find near optimal solutions in relatively

short execution times (as shown empirically). The major drawback of stochastic algorithms, besides their relatively long execution times, is their high likelihood of getting stuck in local minima.

Heuristic algorithms attracts the most attention of researchers given their less complexity when compared to exact backtracking algorithms. Heuristics on the other hand can only find inexact solutions and hardly provide tight approximation gaps [184, 53, 99, 159, 39, 84]. For example, Zhu and Ammar in [184] adopt one very basic greedy algorithm that greedily search for feasible nodes to serve a virtual network and then compute the shortest paths between these nodes. If the evaluated shortest paths can satisfy the demands of the virtual links, the virtual network is considered successfully embedded. This is the most simple but sub-optimal algorithm which brings no guarantee to solve the VNE problem. We refer to this algorithm throughout as baseline. The authors in [53] formulate the VNE problem as two stage, coordinated node and link mapping problems, that are both formulated as Mixed ILP (MIP), and then use a rounding relaxation to find near optimal solutions by an off-the-shelf solver. This algorithm can, however, be very slow especially when the size of the virtual network (number of nodes and links) is large, and is shown to have a worst case complexity of $O(n^{14}\, b^2\, \ln b\, \ln \ln b)$ where n is the number of substrate nodes, and b is the number of input bits to the linear program [53, 99]. Several other works adopted a similar approach to [53], formulating the VNE problem as MIP [179, 124]. Formulating the VNE problem as MIP allows a mechanical problem formulation that can address a wide range of objectives such as energy-awareness and fault-tolerance [153, 97, 17, 13]. Heuristic algorithms, though have better execution times than backtracking algorithms, do result in low acceptance rates, due to their sub-optimal embedding nature.

Our algorithm, Bird-VNE, follows a constraint processing design methodology and involves a simplified form of backtracking to bound the resulting approximation-ratio. Our algorithm is different from other backtracking based solutions in that it relies on the analysis of the constraint properties of the VNE problem. This analysis allows us to develop constraint processing algorithms specific to the VNE problem that effectively prune the search space. Unlike other heuristics, Bird-VNE allocates substrate paths directly to the requested virtual links, rather than separating node and link mapping or at most coordinating their allocations. This approach leads to a proved approximation-ratio that tightens the Bird-VNE performance which was first proposed in our work in [8].

Virtual Network Embedding and Migration in Wireless Networks: Designing VNE algorithms that account for network dynamics (e.g. wireless link quality instability, links failure,

node mobility, etc.) attracted little attention [176, 175, 167]. The authors in [167] discuss virtualization measures that can ensure network embedding feasibility in wireless networks under dynamic behaviors. Also in [175], the authors propose to use VNE over *static* wireless multihop networks. Unlike these papers, we design our VNE embedding considering wireless network dynamics due to substrate nodes that can invalidate already embedded virtual networks, hence mandating migrating these virtual networks to ensure service continuity.

Virtual network migration has also attracted the attention of some researchers to fix invalid virtual networks [93, 154, 21]. The work by Houidi et. al [93] is one example in which the authors propose to continuously monitor already embedded virtual networks and to detect possible events that may trigger migration, hence adaptively reembed these virtual networks. Unfortunately virtual network migration is accompanied with several challenges and overheads. A recent study demonstrates the potential migration challenges including: unavoidable packet loss, slow adaptability of switches to changes, and critical deadline time to switch packets to new paths. [117].

In this paper, we extend our work in [8] to take into account the potential virtual network migration overheads by minimizing the likelihood of migrating already embedded virtual networks which arises due to substrate node mobility. Our work also matches the recent recommendations in [117] where an awareness of the potential migrations during the Virtual network embedding phase is needed to avoid the migration drawbacks. Unlike existing virtual network migration algorithms, if we integrate Bird-VNE with a migration solution (e.g. as in [93]), that solution shall become activated less frequently.

4.3 System Model and Design Objective

We abstract and model the substrate (physical) network, consisting of a set S of n nodes, as an undirected graph $\Phi = (S, L)$ where L is the set of substrate links with each link $l \in L$ corresponding to a connected pair of nodes $s, s' \in S$. We assume that each node $s \in S$ offers a processing capacity $C(s)$, and each link $l \in L$ offers a bandwidth capacity $C(l)$.

In what follows, let $\mathbf{R}$ be the set of all possible paths between all substrate node pairs, where a path $P(s, s')$ between two substrate nodes s and s' is a sequence of connected links (or pairs of nodes) in L. Throughout the paper, $P(s, s')$ (or sometimes P) will also refer to the set of all the links constituting the path. The path length, $|P|$, and the bandwidth capacity, $C(P) = \min_{l \in P} C(l)$, characterize P.

We also consider that the substrate nodes are mobile, and adopt the modified Random Way Point (RWP) mobility model proposed in [115] to model the substrate node mobility. This model describes the mobility of any substrate node s by an infinite sequence of quadruples $\{(\mathbf{X}_{i-1}, \mathbf{X}_i, C_i, W_i)_s\}_{i\in\mathbb{N}}$, where i denotes the i-th movement sample of node s. For every movement sample i, s moves from the starting waypoint $\mathbf{X}_{i-1}$ to the target waypoint $\mathbf{X}_i$ with velocity C_i. Upon arrival to the target waypoint $\mathbf{X}_i$, s waits W_i time units.

Given the waypoint $\mathbf{X}_{i-1}$, the node chooses the target waypoint $\mathbf{X}_i$ randomly such that the included angle θ_i between the vector $\mathbf{X}_i - \mathbf{X}_{i-1}$ and the abscissa is uniformly distributed in $[0, 2\pi]$ and the transition length $Z_i = \|\mathbf{X}_i - \mathbf{X}_{i-1}\|$ is Rayleigh distributed. The angles $\{\theta_1, \theta_2, \ldots\}$ are i.i.d., and the transition lengths $\{Z_1, Z_2, \ldots\}$ of a substrate node s are also i.i.d. with parameter λ_s and a CDF $P(Z_i < z) = 1 - \exp(-\lambda_s \pi z^2),\ z > 0$.

Velocities C_i are generally i.i.d. random variables with arbitrary distributions. Even with randomly distributed velocities, it is sufficient for the purpose of this paper that $C_i \equiv C_s$, where C_s is a positive constant, equaling the average speed of substrate node s. Waiting times $\{W_1, W_2, \ldots\}$ of a substrate node s are also assumed to be i.i.d. exponential with parameter μ_s and a CDF $P(W_i < w) = 1 - \exp(-\mu_s w), w > 0$

The following are important stochastic properties of the modified RWP [115]:

1. *Transition time* T_r, defined as the time a substrate node spends between two successive waypoints. For a substrate node s moving with constant velocity C_s, the Probability Distribution Function (PDF) of T_r is $f_{T_r}(t) = 2\pi\lambda_s C_s^2 t \exp(-\lambda_s \pi C_s^2 t^2)$ and the (Cumulative Density Function) CDF is $P(T_r < t) = 1 - \exp(-\pi\lambda_s t^2 C^2),\ \lambda_s > 0$.

2. *Target waypoint distribution.* Given $\mathbf{X}_{i-1}$, the PDF of the target waypoint $\mathbf{X}_i$ in polar coordinates is given by
$$f_{\mathbf{X}_i}(r, \theta) = \lambda_s \exp(-\lambda_s \pi r^2). \tag{4.1}$$

We also assume that there exists a central node that is responsible for managing the substrate network and embedding the virtual network requests. That is, the central node will be receiving multiple different VNE requests in real time, and embedding them one at time. Each VNE request i is to be embedded for τ_i time units (i.e. τ_i is VNE i's service time).

4.3.1 Virtual network embedding

A VNE request can be represented as an undirected graph $\Upsilon = (V, E)$ where V is the set of the virtual nodes and E is the set of the virtual links (i.e. connected pairs of virtual nodes). In what follows, let $n_v = |V|$ and $m_v = |E|$. Each node $v \in V$ has a geographical location and a requested node stress $T(v)$ (e.g. processing capacity). Similarly, each virtual link $e \in E$ has a requested link stress $T(e)$ (e.g. link bandwidth). Table 4.1 summarizes the key notations.

Substrate Network		Random Way Point		Virtual Network	
Symbol	Definition	Symbol	Definition	Symbol	Definition
S	Set of substrate nodes	$\mathbf{X}$	A waypoint	τ_i	Service time
n	Number of nodes	C	Traveling velocity	Υ	Virtual network
Φ	Substrate network	W	Waiting time at $\mathbf{X}$	V	Set of virtual nodes
L	Set of substrate links	Z	Transition lenth	E	Set of virtual links
$C(s)$ and $C(l)$	Substrate capacities	λ	Rayleigh parameter	n_v and m_v	Number of virtual nodes/links
$\mathbf{R}$	Set of all paths	T_r	Transition time	$T(v)$	Processing demand
$P(s, s')$	A substrate path	$f_{\mathbf{X}_i}(r, \theta)$	Waypoint distribution	$T(e)$	Bandwidth demand

Table 4.1: Summary of notations.

Suppose that, at a given point in time, the central node has already received and successfully embedded a total of $k-1$ virtual network requests, $\Upsilon^{(1)}, \Upsilon^{(2)}, \ldots, \Upsilon^{(k-1)}$, and the k^{th} request, $\Upsilon^{(k)}$, has just arrived. The problem of embedding of the k^{th} virtual network $\Upsilon^{(k)} = (V^{(k)}, E^{(k)})$ into the substrate network Φ consists of the following two mappings.

Node mapping: maps each virtual node $v \in V^{(k)}$ to a distinct substrate node $s \in S$ subject to two constraints. One, s must be within Δ distance from v, where Δ is a parameter associated with the VNE request. Two, the sum of the requested processing capacities of all virtual nodes mapped to s (including those mapped from previous VNE requests) must not exceed the offered processing capacity of s. Formally, letting $Dist(u, v)$ denote the Euclidean distance between u and v, node mapping consists of finding a node mapping function, $\mathcal{M}\left(V^{(k)}\right) : v \in V^{(k)} \mapsto \mathcal{M}(v) \in S$, such that $\mathcal{M}(v_i) = \mathcal{M}(v_j)$ iff $v_i = v_j$, $Dist(\mathcal{M}(v), v) \leq \Delta$ for all $v \in V^{(k)}$, and $\sum_{v \in \cup_{i=1}^{k} V^{(i)}: \mathcal{M}(v) = s} T(v) \leq C(s)$ for all $s \in S$.

Link mapping: maps each virtual link $e \in E^{(k)}$ to a substrate path $P \in \mathbf{R}$ subject to two constraints. One, the end virtual nodes of e must correspond to the end substrate nodes of P. Two, for every $l \in L$, the sum of the requested bandwidth capacities of all virtual links (including those belonging to previous VNE requests) whose mapped paths go through the substrate link l must not exceed the offered bandwidth capacity of l. Formally, link mapping consists of finding a link mapping function, $\mathcal{M}\left(E^{(k)}\right) : e = (v, v') \in E^{(k)} \mapsto \mathcal{M}(e) = P(s, s') \in \mathbf{R}$, such that

$\mathcal{M}(v) = s$, $\mathcal{M}(v') = s'$, and $\sum_{e \in \cup_{i=1}^{k} V^{(i)}: l \in \mathcal{M}(e)} T(e) \leq C(l)$ for all $l \in L$.

Definition 4.3.1 *The embedding of $\Upsilon^{(k)}$ is said to be feasible when both the node mapping and link mapping tasks defined above are successful.*

Upon successfully embedding the k^{th} VNE request, the central node updates the locations of the substrate nodes, as well as the amounts of the available/remaining substrate resources. These are the remaining processing capacity of substrate node s, denoted by $R^{(k)}(s) = C(s) - \sum_{v \in \cup_{i=1}^{k} V^{(i)}: \mathcal{M}(v) = s} T(v)$, the remaining bandwidth capacity of substrate link l, denoted by $R^{(k)}(l) = C(l) - \sum_{e \in \cup_{i=1}^{k} V^{(i)}: l \in \mathcal{M}(e)} T(e)$, and the remaining path capacity of substrate path P, denoted by $R^{(k)}(P) = \min_{l \in P} R^{(k)}(l)$. Also, upon receiving a new VNE request, the central node constructs the mapping domains of the virtual nodes and links, which are defined as follows.

Definition 4.3.2 *The mapping domain D_v of a virtual node $v \in V^{(k)}$ is defined to be the set of all substrate nodes whose Euclidean distances to v are each less than Δ and whose remaining processing capacities are each greater than $T(v)$; i.e., $D_v = \{s \in S : Dist(s, v) \leq \Delta, R^{(k)}(s) \geq T(v)\}$.*

Definition 4.3.3 *The mapping domain D_e of a virtual link $e = (v, v') \in E^{(k)}$ is defined to be the set of all substrate paths whose end nodes (s, s') are in $D_v \times D_{v'}$ and whose remaining capacities are each greater than $T(e)$; i.e., $D_e = \{P(s, s') \in \mathbf{R} : (s, s') \in D_v \times D_{v'}, R^{(k)}(P(s, s')) \geq T(e)\}$.*

Figure 4.1 shows a VNE example, where the graph on the left side is the virtual network and that on the right side is the substrate network. In this example, the node mapping domains are $D_a = \{A, C\}$, $D_b = \{G, H\}$, and $D_c = \{B, E, F\}$, the link mapping domains are shown in dashed lines (e.g. $D_{(a,c)} = \{\{(A, B)\}, \{(A, E)\}, \{(A, E), (E, B)\}, \{(A, C), (C, D), (D, E)\}, \{(A, C), (C, D), (D, E), (E, B)\}, \{(A, B), (B, E)\}\}$). The VNE solution is given by (i) the node mappings, $\mathcal{M}(a) = C$, $\mathcal{M}(b) = H$, and $\mathcal{M}(c) = B$, and (ii) the link mappings, $\mathcal{M}((a, b)) = \{(C, D), (D, H)\}$, $\mathcal{M}((a, c)) = \{(C, A), (A, B)\}$, and $\mathcal{M}((b, c)) = \{(H, E), (E, B)\}$.

4.3.2 Probability of VNE migration due to node mobility

If a virtual node v is mapped to a substrate node s, a migration is triggered when the distance $d = Dist(v, s)$ becomes greater than Δ. More specifically, a migration will not be triggered

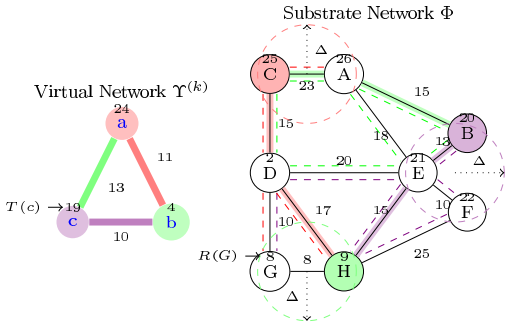


Figure 4.1: Virtual Network Embedding: node mapping domains are shown in dashed circles (radius=Δ) and link mapping domains are shown in dashed lines parallel to substrate paths.

due to s's mobility if s stays within the circle $A(v, \Delta)$ of diameter Δ centered at v for a period longer than τ, the service time of the virtual network request incorporating node v. From (4.1), the probability that the target waypoint of the substrate node is within $A(v, \Delta)$ is, for $0 \leq d \leq \Delta$,

$$P(A(v,\Delta)) = \int_{\Delta-d}^{\Delta+d} \int_{0}^{2\pi} f_{\mathbf{X}_i}(r,\theta)\, r \mathrm{d}r\mathrm{d}\theta,$$
$$= \exp(-\pi\lambda_s(d-\Delta)^2) - \exp(-\pi\lambda_s(d+\Delta)^2).$$

Let $H(s)$ be the probability that a migration is triggered due to the mobility of substrate node s. $H(s)$ can be approximated as the probability that neither the target waypoint is within $A(v, \Delta)$ and the total time spent in $A(v, \Delta)$ is $\geq \tau$ nor the target waypoint is outside $A(v, \Delta)$ and the transition time to the boarder of $A(v, \Delta)$ is $\geq \tau$. Computing the PDF of the total time spent in $A(v, \Delta)$ ($W + T_r$) requires convolution of the PDFs of W and T_r, and strong assumptions on relative values of λ_s, C_s, and τ, which are outside the control of the embedding algorithm. To simplify the analysis and the VNE objective design, we assume that: i) the time spent within $A(v, \Delta)$ is dominated by the waiting time at the target waypoint $\mathbf{X}_i$, ii) if the target waypoint is outside $A(v, \Delta)$, the whole transition time is spent within $A(v, \Delta)$, and iii) the waiting time of the starting waypoint has elapsed at the time of the virtual network embedding. Since we are mainly interested in evaluating the migration probability of a substrate node relative to other substrate nodes, the impact of these assumptions is minimal. With this, $H(s)$ can be expressed

as

$$H(s) = 1 - P(W \geq \tau)P(A(v, \Delta)) \\ - P(T_r \geq \tau)(1 - P(A(v, \Delta))) \tag{4.2}$$

To minimize the migration overhead, the VNE algorithm shall map virtual nodes to substrate nodes with the least migration probability, $H(s)$. Unlike traditional virtual network embedding and migration algorithms, this requires the estimation of the transition length and waiting time distribution parameters and the use of the estimated parameters to evaluate the migration probability associated with mapping a virtual node v to a substrate node s. The maximum likelihood estimation of the transition length parameter is $\hat{\lambda_s} = \frac{1}{4}\left(\overline{Z^2}\right)^{-2}$, where the $\overline{Z^2}$ denotes the second sample moment of Z, and that of the waiting time parameter is $\hat{\mu_s} = \frac{1}{\overline{W}}$, where $\overline{W}$ denotes the sample moment of W.

4.3.3 VNE design objective

Our objective is to develop an algorithm that finds feasible VNEs while maximizing the embedding profit and minimizing the migration overhead. We say that a feasible embedding is optimal when its profit is maximum[1]. Given a virtual network Υ, the profit is defined as the difference between the revenue generated from embedding Υ and its embedding cost, i.e. $\text{Profit}(\Upsilon) = \text{Revenue}\,(\Upsilon) - \text{Cost}\,(\Upsilon)$.

To achieve the VNE design objective, we model the embedding cost to capture the cost of node mapping, the cost of link mapping, and the potential cost of migration that may arise as a result of mobility. It is defined as

$$\text{Cost}\,(\Upsilon) = \sum_{v \in V} \alpha T\,(v) + \sum_{e \in E} \beta T\,(e) \times |\mathcal{M}\,(e)| \\ + \sum_{v \in V} \gamma(v) H(\mathcal{M}\,(v)), \tag{4.3}$$

where α and β denote the cost of processing and bandwidth resource units, respectively. The third term captures the cost of migration due to substrate nodes mobility, where $\gamma(v)$ is the cost of migrating the virtual node v. Intuitively, $\gamma(v)$ depends on the amount of resources allocated to v, as well as on v's connectivity to other virtual nodes.

[1]Modeling the objective as a maximization problem allows us to analytically bound the objective value, as shown later in Section 4.5.

We also define the revenue to be generated from successfully embedding Υ as

$$\text{Revenue}\left(\Upsilon\right) = \sum_{v \in V} \alpha' T\left(v\right) + \sum_{e \in E} \beta' T\left(e\right), \tag{4.4}$$

where α' and β' denote the price to be charged for each processing and bandwidth unit, respectively.

Observe that the embedding revenue in (4.4) depends only on the virtual network's requested resources and not on the VNE solution. Also recall that the function $H(\mathcal{M}\left(v\right))$ given in (4.3) represents the probability that a migration of v is triggered due to the mobility of the substrate node, $\mathcal{M}\left(v\right)$. It follows that maximizing the profit implies minimizing the embedding cost in (4.3), which implicitly minimizes the virtual network migration overhead due to mobility. Note that even though, in this paper, the function $H(\mathcal{M}\left(v\right))$ captures the likelihood of migration that is due to mobility, it can be used to represent/capture the migration due to any other network dynamics, like link failure.

4.4 Enforcing Domain Consistency

The node and link mapping domains, defined in Definitions 4.3.2 and 4.3.3, involve coupled constraints. A mapping of a virtual node v to a substrate node $s \in D_v$ impacts other nodes and links mapping domains in several ways. First, no other virtual nodes can be mapped to s. Second, we can only map virtual links that have v as an end node to substrate paths that have s as an end node. Moreover, a mapping of a virtual link e to a substrate path $P \in D_e$ restricts other virtual links from being mapped to the substrate paths that share one or more substrate links with P. The shared links become capacity bottlenecks as their bandwidth capacity must be greater than the required bandwidth of not only e but also other virtual links mapped to paths sharing these links. A backtracking algorithm resolves such constraint couplings by mapping virtual nodes and virtual links one at a time, and backtracking to previous steps when the algorithm encounters an unfeasible mapping.

A VNE algorithm can avoid backtracking (backtrack-free search) if the mapping domains of all virtual nodes and links are consistent. Enforcing domain consistency involves pruning the node and link mapping domains to avoid mappings that lead to an unfeasible embedding. Unfortunately, the use of the standard consistency propagation algorithms are exponential in time. This is because the constraint network of the VNE problem has a maximum degree that

is a function of n, while the running time of the standard consistency propagation algorithm, to ensure backtrack-free search, is exponential in the maximum degree of the constraint network (see [57] for details).

Fortunately, constraint propagation algorithms can take advantage of certain properties specific to VNE to prune the mapping domains in polynomial time through mapping domains consistency enforcement. In this section, we develop techniques that exploit these properties to avoid backtracking during the VNE search process, and use these techniques to design a polynomial time, *almost* backtrack-free VNE algorithm. There are two types of mapping domains consistency, virtual network topological consistency and substrate paths capacity consistency, which are presented next.

4.4.1 Virtual network topological consistency

We first enforce domain consistency to ensure that the topology of the resulting solution (node and link mappings) matches exactly the topology of the virtual network, i.e. topological consistent. This requires enforcing the following: (i) substrate nodes mapped to the virtual nodes must be *all different*, (ii) end nodes of the substrate paths in link mapping domains must have corresponding substrate nodes in the node mapping domains and vice versa, and (iii) substrate nodes in the node mapping domains must maintain similar virtual node degrees.

Alldifferent virtual node mapping constraint: The constraint to map virtual nodes to distinct substrate nodes is known as the alldifferent constraint in the constraint programming context, and we next state a useful corollary following from Régin's theorem [139] on the alldifferent constraint.

Corollary 4.4.1 *A virtual node mapping $v \in V \mapsto s \in D_v$ leads to an unfeasible embedding if the edge (v, s) does not belong to a maximum matching that covers all the virtual nodes in the bipartite graph $B = (V \cup S, \{(v, s) : \mathcal{M}(v) = s\})$.*

The above corollary can then be exploited to prune away nodes and links from the node and link mapping domains, and for completeness, we provide in Procedure 1 a brief description of such a pruning technique, which we term ALLDIFFERENT [139].

In Procedure 1, a residual graph, B', is defined as $B' = (V \cup S \cup \{t\}, M \cup E_2 \cup E_3 \cup E_4)$ where M is the set of edges in the matching directed from virtual nodes to substrate nodes, E_2 is the set of edges that are not in the matching M and are directed from substrate nodes to

Procedure 1: ALLDIFFERENT

Input: $V,\ D_{v \in V}$
Ensure: Distinct virtual node to substrate node mappings in $O\left(n_v^{1.5} n\right)$ [139].
1: Construct bipartite graph $B = (V \cup S, \{(v, s) : \mathcal{M}(v) = s\})$
2: Find a maximum matching M in B using Hopcroft-Karp algorithm [91]
3: **if** $|M| < n_v$ **then**
4: Return no feasible embedding for the given mapping domains
5: **end if**
6: Construct the residual graph B'
7: Compute the strongly connected components in B'
8: Prune the node mapping domains by deleting any edges connecting two different strongly connected components in B'.
9: **return** Narrowed virtual node mapping domains

virtual nodes, E_3 is the set of all directed edges from substrate nodes in the matching M to a dummy node t, and E_4 is the set of all directed edges from t to substrate nodes that are not in the matching M.

Step 8 in Procedure 1 prunes substrate nodes from the node mapping domains that can never lead to distinct node mappings. Any edge connecting two different strongly connected components in B' corresponds to a mapping from a virtual node v to a substrate node s and does not belong to any maximum cardinality matching, hence it is not possible to find a feasible embedding with distinct node mapping if v was mapped to s. Thus, s must be removed from D_v. The time complexity of Procedure 1 is bounded by the time required to find the maximum matching using the Hopcroft-carp algorithm in step 2. Since a virtual node can have at most n substrate nodes in its node mapping domain, the number of edges in the bipartite graph B cannot exceed $n_v \times n$ edges. In the worst case, the Hopcroft-carp algotihm requires $O(\sqrt{n_v} n_v n)$ steps, hence the ALLDIFFERENT time complexity is $O\left(n_v^{1.5} n\right)$.

Relational consistency of node and link mapping domains: In the example of Figure4.1, although mapping the virtual node c to F is feasible, doing so prevents us from finding a mapping to the virtual link (a, c), as there is no substrate path between F and any substrate node in the node mapping domain D_a.

From the definition of mapping domains, we can easily observe that if two virtual nodes v, v' are connected by a virtual link e, then the end points of the substrate paths in the virtual link mapping domain D_e is a subset of the cross product of the virtual node mapping domains $D_v \times D_{v'}$. We can now rely on this simple observation and the definition of the virtual link mapping domains to conclude the following:

Figure 4.2: Procedure 1 illustration. (a):A Maximum cardinality matching (thick edges), v is connected to s if $s \in D_v$, (b):Alternating graph with two strongly connected components. Edges crossing the strongly connected components cannot be in a maximum matching therefore Procedure 1 prunes them.

Lemma 4.4.2 *The node mapping $v \in V \mapsto s \in D_v$ leads to an unfeasible embedding if there exists a link $e = (v, v') \in E$ whose link mapping domain D_e does not contain a path ending at s. Similarly, a virtual link mapping $e = (v, v') \mapsto P(s, s')$ leads to an unfeasible embedding* if $s \notin D_v$ *or* $s' \notin D_{v'}$.

Proof Assume $v \mapsto s$ and a subsequent mapping of $e = (v, v')$ such that there is no path $P \in D_e$ ending at s. A mapping of e to any substrate path in D_e results in mapping multiple virtual nodes to the same substrate node. Also, $e = (v, v') \mapsto P(s, s')$ violates the link mapping Definition 4.3.3 if either $s \notin D_v$ or $s' \notin D_{v'}$.

Using Lemma 4.4.2, we propose two procedures to narrow down the node and link mapping domains: Procedures 2 and 3. The functions $u\,(D_e)$ and $v\,(D_e)$ return the sets respectively of the first and the second end nodes of all the paths in D_e. When applied to a path P, $u\,(P)$ and $v\,(P)$ return the path's first and second end nodes. In each iteration, Procedure 2 prunes the substrate nodes from the node mapping domains of the end nodes of the virtual links, if there is not any substrate path in their link mapping domains that also ends at those substrate nodes. Since for each virtual link the intersection operator (step 2 and 3) requires at most $O(n)$ steps as $\mid D_v \mid \leq n$, then Procedure 2 has a worst case time complexity of $O(m_v n)$.

Procedure 3 complements Procedure 2 by pruning a substrate path from the link domain of a virtual link if the substrate nodes ending that path cannot be found in the node mapping domains of the virtual nodes ending the virtual link. Since there are at most $O(n^2)$ paths in the substrate

Procedure 2: NODE-CONSISTENCY

Input: $E,\ D_{e \in E},\ D_{v \in V}$
Ensure: Virtual node mapping domains are consistent with virtual link mapping domains in $O(m_v\, n)$
1: **for all** virtual link $e = (v, v') \in E$ **do**
2: $\quad D_v \leftarrow D_v \cap u\,(D_e)$
3: $\quad D_{v'} \leftarrow D_{v'} \cap v\,(D_e)$
4: **end for**
5: **return** Narrowed virtual node mapping domains

network, the inner loop (step 2 to 6) of Procedure 3 requires at most $O(n^2)$ steps. Hence, the worst case time complexity of Procedure 3 is $O(m_v n^2)$.

Procedure 3: LINK-CONSISTENCY

Input: $E,\ D_{e \in E},\ D_{v \in V}$
Ensure: Virtual link mapping domains are consistent with virtual node mapping domains in $O(m_v\, n^2)$
1: **for all** virtual link $e = (v, v') \in E$ **do**
2: $\quad$ **for all** substrate path $P \in D_e$ **do**
3: $\quad\quad$ **if** $u\,(P) \notin D_v \vee v\,(P) \notin D_{v'}$ **then**
4: $\quad\quad\quad D_e \leftarrow D_e \setminus \{P\}$
5: $\quad\quad$ **end if**
6: $\quad$ **end for**
7: **end for**
8: **return** Narrowed virtual link mapping domains

Consistency of virtual and substrate node connectivity: The relational consistency of node and link mapping domains does not ensure connectivity of the virtual network, nor does it imply that the mapping domains can satisfy the virtual network connectivity requirements, especially when the node mapping domains overlap. To illustrate this, consider a new induced network of substrate nodes that represents the connectivity of the virtual link domains. In this induced network, substrate nodes are connected by an edge if there exists a path belonging to any link mapping domain that connects them. Induced network is defined formally next.

Definition 4.4.1 *Given a virtual network Υ, we define the induced network I of Υ as the undirected graph $I = (S_I \subset S, L_I)$ where $S_I = \cup_{v \in V} D_v$ and $L_I = \{(s, s') \in S_I^2 : \exists\ P(s, s') \in D_e$ for some $e \in E\}$.*

Definition 4.4.2 *For every connected component CC_I of I, the set $N_v(CC_I) = CC_I \cap D_v$ corresponding to the virtual node v is called the supernode of v. Let $\zeta(CC_I)$ be the number*

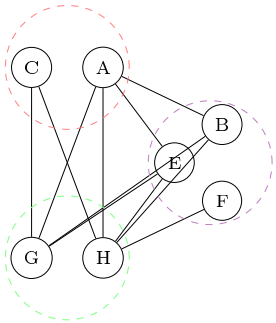


Figure 4.3: Induced network I from substrate network Φ in Figure4.1. I has one connected components CC_I and three supernodes (dashed circles). $\zeta(CC_I) = 3$, $\delta(F) = 1$ and equals 2 for all other nodes.

of distinct supernodes in CC_I. For every $s \in S_I$, we define $\delta(s)$ as the number of supernodes connected to s.

Figure 4.3 illustrates the induced network of the example given in Figure 4.1. This induced network is constructed by connecting a pair of substrate nodes in Figure 4.3 when there is at least one path connecting them in any link mapping domain. In general, if the mapping $v \mapsto s$ is feasible, the function $\delta(s)$ reflects the degree of the virtual node v, and if a connected component CC_Υ of Υ is mapped to a subset of substrate nodes in Φ, the function $\zeta(CC_I)$ reflects the number of virtual nodes in the connected component CC_Υ (size of CC_Υ).

Lemma 4.4.3 *Let $Deg_\Upsilon(v)$ denote the degree of virtual node v. A virtual node mapping $v \mapsto s$ leads to an unfeasible embedding* if *$Deg_\Upsilon(v) > \delta(s)$ or the size of the connected component of Υ (CC_Υ) that contains v is greater than the number of supernodes in CC_I that contains s.*

Proof Assume $v \mapsto s$ and $Deg_\Upsilon(v) > \delta(s)$, then there exist at least one virtual link e such that there is no substrate path P in D_e with one of its end substrate nodes equals s. Then, $v \mapsto s$ does not lead to a feasible embedding from Lemma 4.4.2. If $Deg_\Upsilon(v) \leq \delta(s)$ but $|CC_\Upsilon| > \zeta(CC_I)$, then there must exist an unmapped virtual node $v' \in CC_\Upsilon$, while all substrate nodes $s \in CC_I$ are already mapped to other virtual nodes in CC_Υ including v. Since v' must be mapped to one substrate node in CC_I to maintain connectivity, then mapping $v \mapsto s$ does not lead to a feasible embedding.

The DEGREE-CONSISTENCY procedure (Procedure 4), a direct application of Lemma 4.4.3, is a pruning technique that narrows down mapping domains through degree consistency enforce-

ment. Its complexity is bounded by computing $\delta(s)$ for all the substrate nodes in the virtual node mapping domains, which is $O(n^2)$.

Procedure 4: DEGREE-CONSISTENCY

Input: $E,\ D_{v \in V}$
Ensure: Degree Consistency in $O(n^2)$
1: **for all** virtual nodes $v' \in V$ **do**
2: **for all** substrate nodes $s' \in D_{v'}$ **do**
3: **if** $Deg_{\Upsilon}(v') > \delta(s')$ **then**
4: $D_{v'} \leftarrow D_{v'} \setminus \{s'\}$
5: **end if**
6: **end for**
7: **end for**
8: **for all** connected component $CC_{\Upsilon} \in \Upsilon$ **do**
9: **for all** connected component $CC_I \in I$ **do**
10: **if** $|CC_{\Upsilon}| > \zeta(CC_I)$ **then**
11: $D_{v'} \leftarrow D_{v'} \setminus CC_I,\ \forall v' \in CC_{\Upsilon}$
12: **end if**
13: **end for**
14: **end for**
15: **return** Narrowed virtual nodes domains

Running the ALLDIFFERENT, NODE-CONSISTENCY, LINK-CONSISTENCY, and DEGREE-CONSISTENCY procedures for one iteration removes some inconsistent mappings from the node and link mapping domains. To remove all the inconsistencies, these procedures must repeatedly be run sequentially until no further removal is possible from either the node or the link mapping domains. The process merging all these four procedures is captured in Algorithm 1, which essentially removes inconsistency, and hence avoids backtracking, by ensuring topological consistency of the node and link mapping domains. This algorithm is referred to as TOPOLOGY-CONSISTENCY.

The complexity of TOPOLOGY-CONSISTENCY is bounded by the number of times we run the procedure LINK-CONSISTENCY in step 4. This implies a complexity of $O(m_v\, n^2)$ in each iteration. In the worst-case scenario, TOPOLOGY-CONSISTENCY removes one substrate node from one node mapping domain and this corresponds to at least one removal of one substrate path from link mapping domains. Hence, it requires at most n iterations to remove all the substrate nodes from one node mapping domain, thus returning false[2]. Thus, the complexity of

[2]A more efficient implementation checks the condition in step 6 every time any procedure removes a substrate node/link from a mapping domain.

TOPOLOGY-CONSISTENCY is $O(m_v\, n^3)$. But since the maximum number of virtual nodes is the number of substrate nodes; i.e., $n_v \leq n$, then the complexity of TOPOLOGY-CONSISTENCY is $O(n^5)$.

Algorithm 1: TOPOLOGY-CONSISTENCY

Input: $E,\ D_{e\in E},\ D_{v\in V}$
Ensure: Topology Consistency in $O(m_v\, n^3)$
1: **repeat**
2: NODE-CONSISTENCY$(E,\ D_{e\in E},\ D_{v\in V})$
3: ALLDIFFERENT$(V,\ D_{v\in V})$
4: LINK-CONSISTENCY$(E,\ D_{e\in E},\ D_{v\in V})$
5: DEGREE-CONSISTENCY$(E,\ D_{v\in V})$
6: **if** $\exists\ D_{v'} = \emptyset,\ \forall v' \in V\ \vee\ D_e = \emptyset,\ \forall e \in E$ **then**
7: **return false**{T}here exist a virtual node or a virtual link with an empty mapping domain.
8: **end if**
9: **until** No node or link domain is changed
10: **return true**

4.4.2 Capacity disjoint paths consistency

We now study the second mapping domain consistency type, substrate paths capacity consistency. Let us refer again to the example given in Figure 4.1 and consider the link mapping sequence $(a,b) \mapsto P(C,H) = \{(C,D),(D,H)\}$ and $(a,c) \mapsto P(C,E) = \{(C,D),(D,E)\}$. The remaining bandwidth of the substrate link (C,D), $R\left((C,D)\right) = 15$, is less than the sum of the links' requested bandwidth capacities, which is $T\left((a,b)\right) + T\left((a,c)\right) = 24$. Hence, this mapping sequence is unfeasible. Clearly, a VNE algorithm will not backtrack if all substrate paths in the link mapping domains are disjoint (if topological consistency is enforced). However, constructing the link mapping domains from disjoint paths results in a degradation of the VNE acceptance rate (such a rate reflects the number of virtual networks that can be embedded into the substrate network), as well as in an increase in the embedding cost. Our proposed embedding algorithm does not force paths to be disjoint so as to increase the acceptance rate and decrease the embedding cost. Instead, our technique relies on the concept of *capacity disjoint* which we formally define next.

Definition 4.4.3 *For every substrate link l, let $\bar{D}_e(l) = \{P \in D_e : P\ \ni l\}$ and $\bar{E}(l) = \{e \in E : \bar{D}_e(l) \neq \emptyset\}$. We say that the paths in $\mathbf{R}' = \bigcup_{e\in\bar{E}(l)} \bar{D}_e(l)$ are* capacity disjoint iff *the remaining bandwidth capacity of l is greater than the sum of the requested bandwidth capacities*

of all the virtual links in $\bar{E}(l)$. *Formally, the paths in* $\mathbf{R}'$ *are said to be capacity disjoint* iff $R^{(k)}(l) \geq \sum_{e \in \bar{E}(l)} T(e)$.

Lemma 4.4.4 *A virtual link mapping* $e \mapsto P$ *leads to an unfeasible embedding* if *all the substrate paths in every unmapped virtual link's mapping domain are not capacity disjoint with* P.

Proof If a virtual link $e_i \mapsto P_i$ and in a next mapping step of virtual link e_j, all paths in D_{e_j} are not capacity disjoint with P_i, then any mapping $e_j \mapsto P_j \in D_{e_j}$ will result in at least one substrate link with negative remaining bandwidth.

Theorem 4.4.5 *The proposed* TOPOLOGY-CONSISTENCY *algorithm ensures a backtrack-free search if all substrate paths in all link mapping domains are capacity disjoint.*

Proof It follows from Lemmas 4.4.2, 4.4.3, and 4.4.4 and from Corollary 4.4.1.

An algorithm that aims to ensure a backtrack-free search may remove substrate paths that are not capacity disjoint from the virtual links mapping domains. Although such an algorithm will have a complexity advantage because it is backtrack-free, it degrades the acceptance rate and the cost as it will remove substrate paths that can actually lead to feasible or minimum cost embedding. Apparently, capacity disjoint paths condition is required only for substrate paths that are actually in an incurred embedding. In order to overcome the complexity problem while still minimizing the cost and maximizing the acceptance rate, we propose Algorithm 2 (CAPACITY-DISJOINT), which ensures that substrate paths in link mapping domains are capacity disjoint if they are likely to coexist in an incurred embedding.

The key idea of the CAPACITY-DISJOINT algorithm is to determine the worst case scenario in which the intersecting substrate paths in $\mathbf{R}'$ can become simultaneous mappings of virtual links in $\bar{E}(l)$. These paths are found by applying topological consistency procedures, discussed earlier, on the subsets of link and node mapping domains $\bar{D}_e \in \bar{E}(l),\ \bar{D}_v \in \bar{V}(l)$ (Steps 1 to 6), where $\bar{V}(l) \subset V$ is the set of end virtual nodes of virtual edges in $\bar{E}(l)$ and $\bar{D}_v(l) \subset D_v$ is the set of substrate node mappings deduced from $\mathbf{R}'$.

The CAPACITY-DISJOINT algorithm checks if all paths that are common to every substrate link l are capacity disjoint. If not, the algorithm removes first the substrate paths $\bar{D}_e \in \bar{E}(l)$ from the domain of the virtual link(s) e that has the largest link mapping domain size $|D_e|$ (Step 7 to 16). This is to minimize the chances of ending up with an empty link mapping domain,

thus maximizing the acceptance rate. Although it is clear that CAPACITY-DISJOINT algorithm does not eliminate backtracking entirely, it substantially reduces its likelihood of occurrence. We evaluate the likelihood of backtracking empirically in Section 4.6.

Algorithm 2: CAPACITY-DISJOINT

Input: $E,\ L,\ D_{e \in E},\ D_{v \in V}$
Ensure: Substrate paths are capacity disjoint if they are likely to coexist in an incurred embedding in $O(m\, m_v\, n^3)$.

1: **for all** $l \in L :\ \exists\, ps \in D_{e \in E},\ l \in P$ **do**
2: **repeat**
3: NODE-CONSISTENCY($\bar{E}(l),\ \bar{D}_e \in \bar{E}(l),\ \bar{D}_v \in \bar{V}(l)$)
4: ALLDIFFERENT($\bar{V}(l),\ \bar{D}_v \in \bar{V}(l)$)
5: LINK-CONSISTENCY($\bar{E}(l),\ \bar{D}_e \in \bar{E}(l),\ \bar{D}_v \in \bar{V}(l)$)
6: **until** No node or link sub-domain is changed
7: $R'(l) \leftarrow R(l)$
8: **for all** $e \in \bar{E}(l)$ ordered ascendingly by $|D_e|$ **do**
9: **if** $\bar{D}_e(l) \neq \emptyset$ **then**
10: $R'(l) \leftarrow R'(l) - T(e)$
11: **if** $R'(l) < 0$ **then**
12: $D_e \leftarrow D_e \setminus \bar{D}_e(l)$
13: **end if**
14: **end if**
15: **end for**
16: **end for**
17: **if** $\exists\ D_e = \emptyset,\ \forall e \in E$ **then**
18: **return false**
19: **end if**
20: **return true**

The CAPACITY-DISJOINT algorithm uses similar steps to determine possible simultaneous intersecting paths (Steps 2 to 6) for each substrate link l that intersects with some paths. Although these steps are performed on a subset of the mapping domains and it is unlikely to encounter the situation that every substrate link is a common link for all paths (as the substrate network will almost look like a path), the complexity of CAPACITY-DISJOINT is bounded by $O(m\, m_v\, n^3)$. This can be expressed as $O(n^7)$ if both the substrate and virtual networks are complete graphs and have the same number of nodes n.

4.5 Approximate Profit Maximization

TOPOLOGY-CONSISTENCY and CAPACITY-DISJOINT algorithms, discussed in the previous section, reduce the search space and improve the running time of backtracking search. However, even in the case of backtrack-free search, an optimal optimization algorithm, like branch and bound, may still traverse the whole search space through brute-force [57]. If we assume that the VNE problem is backtrack-free, it can be viewed as the maximum weight matching problem in a bipartite graph. The bipartite graph in this case is the set of virtual links on one side of the bipartite graph connected by weighted edges to the set of substrate paths on the other side and the edge weight is the profit attained by mapping a virtual edge to a substrate path. From (4.3) and (4.4), the profit of mapping a virtual edge $e = (v, v')$ to a substrate path $P = (s, s')$ is given by

$$\begin{aligned}\text{Profit}(e, P) =&(\alpha' - \alpha)(T(v) + T(v')) \\ &+ (\beta' - \beta \times |P|)T(e) \\ &- \gamma(v)H(s) - \gamma(v')H(s').\end{aligned}$$

However, a direct application of conventional maximum weight matching algorithms (e.g. Hungarian methods or Edmond's methods) is non-trivial. Fortunately, greedy approximations to the maximum weight matching are applicable, but with some needed modifications to enforce domain consistency and to verify solution feasibility in every step. We use this observation to propose Algorithm 3, which finds a VNE such that the incurred embedding profit is at most as half as the optimal profit in an attainable special case and at least $\frac{1}{n_v}$ in general.

Our proposed VNE algorithm, termed BIRD-VNE, starts by enforcing the mapping domains consistency using TOPOLOGY-CONSISTENCY and CAPACITY-DISJOINT. It then searches for an embedding by mapping the virtual links with the smallest link mapping domain sizes and greatest demands (Line 9) first to the substrate paths in their domains with the highest profit (Line 10). After mapping each virtual link (mapping step), the algorithm ensures feasible embedding according to Definition 4.3.1 (Line 20). If any mapping step results in unfeasible embedding, the algorithm starts over the mapping process from the first virtual link by assigning it to an unattempted mapping in its domain until a feasible embedding is found or all mappings of the first link are tried.

This algorithm still involves a simplified form of backtracking. The algorithm always backtracks to the first virtual link mapping step. In this case, the total number of backtracks is bounded in the worst case by the size of the smallest link mapping domain. Typically, the con-

sistency enforcing algorithms reduce the number of backtracks significantly as we will show empirically in Section 4.6. The following theorem bounds the worst case performance of BIRD-VNE.

Theorem 4.5.1 *In the worst case,* BIRD-VNE *is an $O(\frac{1}{n_v})$-approximation to the optimal embedding profit, and only a $\frac{1}{2}$-approximation if there are, on average, n_v paths of the same length between any two substrate nodes.*

Proof Let x be the profit of mapping the first virtual link e to the highest profitable path P in its link mapping domain in a single iteration (step 7). The following potential mappings become invalid and will never be attempted by the algorithm until a backtracking to step 7 is decided: (i) mapping e to other substrate paths in its link mapping domain except P, (ii) mapping any other virtual link to P, (iii) mapping another virtual link e' that shares one of its end virtual nodes with e with any other substrate paths except those that also share the same end substrate node with P. Let d be the maximum degree of the virtual network, the worst case will occur if we have exactly d paths of shortest length (highest profit) and the algorithm invalidates at most d mappings of the optimal mappings (at most in the first mapping). In this case, the sum of profits of the invalidated mapping cannot exceed dx. Since the profit is non-negative, the approximation ratio is $O(\frac{1}{d})$ or more conservatively $O(\frac{1}{n_v})$. However, if there are n_v redundant substrate paths of the same length between any two substrate nodes, BIRD-VNE invalidates at most two mappings that may be optimal. This can be repeated for at most $\frac{1}{2}m_v$ of the steps (9 to 19) and the sum of the profits of the invalidated mappings cannot exceed $2x$. In this later case, the approximation ratio is $\frac{1}{2}$, which proves the theorem.

4.5.0.1 Scalability and implementation consideration

The complexity of BIRD-VNE is analyzed as follows. The main loop (Step 10 to 25) has m_v iterations. In the worst-case scenario, for every virtual link, it checks the feasible mappings of n^2 paths. Then, the complexity of BIRD-VNE is bounded by the CAPACITY-DISJOINT complexity $O(m\, m_v\, n^3)$ and can be written as $O(n^7)$. Although BIRD-VNE is polynomial in time and scales much better than the state-of-the art algorithms, its $O(n^7)$ complexity may prevent applying it to very large scale networks. Fortunately, this complexity bound can be improved through simple but effective implementation improvements.

Algorithm 3: BIRD-VNE

Input: $\Upsilon = (V, E)$, $\Phi = (S, L)$
Require: $D_{\forall e \in E}$, $D_{\forall v \in V}$
Ensure: Embedding $\Upsilon \mapsto \Phi$ in $O(m\, m_v\, n^3)$

SolutionExist $\leftarrow$ TOPOLOGY-CONSISTENCY
SolutionExist $\leftarrow$ *SolutionExist* **and** CAPACITY-DISJOINT
SolutionExist $\leftarrow$ *SolutionExist* **and** TOPOLOGY-CONSISTENCY
if not *SolutionExist* **then**
    return "Reject virtual network."
end if
repeat
    $\mathcal{M}(e) \leftarrow \emptyset,\ \forall e \in E$
    for all $e = (v, v') \in E$ ordered ascendingly by $|D_e|$, and by $T(e)$ **do**
        for all $P = (s, s') \in D_e$ ordered descendingly by Profit(e, P) **do**
            if e is the first virtual link in the order of E **then**
                $D_e \leftarrow D_e \setminus P$
            end if
            if $e \mapsto P$ result in a feasible embedding **then**
                $\mathcal{M}(e) \leftarrow P$, $\mathcal{M}(v) \leftarrow u(P)$, $\mathcal{M}(v') \leftarrow v(P)$
                break
            end if
        end for
    end for
until Feasible embedding is found or all first virtual link mapping domain are attempted.
if No feasible embedding is found **then**
    return "Reject virtual network."
end if
return $\mathcal{M}(e),\ \forall e \in E$ and $\mathcal{M}(v),\ \forall v \in V$

The two procedures, NODE-CONSISTENCY and LINK-CONSISTENCY, can be easily implemented in parallel by implementing these algorithms on exactly m_v processing agents. In this case, the NODE-CONSISTENCY complexity is reduced to $O(n)$ while the LINK-CONSISTENCY complexity is reduced to $O(n^2)$. It then follows that the complexity of TOPOLOGY-CONSISTENCY is bounded by running ALLDIFFERENT at most n times, hence it is $O\left(n_v^{2.5}n\right)$. Similarly, the CAPACITY-DISJOINT complexity is also bounded by running ALLDIFFERENT at most m times, hence it is $O\left(n_v^{2.5}m\right)$. The complexity of CAPACITY-DISJOINT bounds the overall complexity of BIRD-VNE to $O\left(n_v^{2.5}m\right)$.

We can also improve the actual approximation ratio in practice by repeating step 7 to 19 until all virtual link mappings of the first virtual link are attempted (i.e. remove steps 14 to 17) while maintaining all the feasible solutions. We then pick the solution with the maximum total profit as our solution and the other solutions as backup solutions in case a migration is needed. This trick reduces the gap between the evaluated total profit and the optimal solution when compared to the worst case scenario, and preserves the same worst case complexity at the expense of the actual execution time.

4.5.0.2 Virtual network migration consideration

The proposed algorithm, BIRD-VNE, can still be used, with simple modification, if virtual network migration is needed. If the previously discussed implementation in Section 4.5.0.1 is adopted, we end up with multiple solutions to the same virtual network that can be quickly evaluated for feasibility, so as to choose one of these VNE solutions for migrating the virtual network instead of evaluating BIRD-VNE again from the beginning. Moreover, the following simple procedures can be carried out to perform migrations to individual nodes and links instead of migrating the whole virtual network.

Consider the case that a substrate path P is not capable of meeting the required demand of a virtual link e. This situation can happen, for example, in case of a link failure or congestion along the path, or failure of one or both end substrate nodes of P. In this case, we can immediately find another backup path (and substrate nodes if necessary), P', in $D_e\ \{P\}$ that has the largest profit and is also feasible with the current embedding $\mathcal{M}\left(e'\right),\ \forall e' \in E\ \{e\}$. This algorithm is as simple as running the steps from 9 to 19 in BIRD-VNE, while replacing D_e with $D_e\ \{P\}$ for only the virtual links that are impacted by the failure of P. If this fails, the whole embedding needs to be performed again by running BIRD-VNE.

4.6 Numerical Results

The effectiveness of the proposed algorithm, BIRD-VNE, is assessed in terms of the metrics suggested in [73]:

1. *Acceptance rate,* defined as the ratio of the total accepted virtual networks to the total requested virtual networks.
2. *Revenue to Cost ratio (R/C),* defined as $R/C = \sum_{\Upsilon} \text{Revenue}\,(\Upsilon) / \sum_{\Upsilon} \text{Cost}\,(\Upsilon)$.
3. *Average node and link utilization,* defined as $\sum_{s \in S} \frac{R(s)-C(s)}{nC(s)}$ and $\sum_{l \in L} \frac{R(l)-C(l)}{mC(l)}$, respectively.

In addition, we use the following metrics to assess the effectiveness of BIRD-VNE vis-a-vis of its ability to avoid backtracking, limit network migration, and achieve optimal VNE by comparing it to the optimal Brand and Bound technique.

1. *Average/Maximum Approximation ratio,* defined as the average/maximum ratio of the cost achieved by BIRD-VNE to that achieved by Branch and Bound.
2. *Backtrack-free ratio,* defined as the ratio of the total number of times in which BIRD-VNE finds a feasible embedding at the first attempt of the first virtual link mapping to the total number of accepted requests.
3. *Migration ratio,* defined as the ratio of the total number of virtual network migrations to the total number of accepted requests.

4.6.1 Simulation setup

We compare the performance of BIRD-VNE with two existing algorithms, Randomized Virtual Network Embedding with shortest path link mapping (RVINE-SP) and with multicommodity flow link mapping (RVINE-MCF) [53], which are integrated to an event-driven simulator that we developed [3]. We also compare the performance achievable under BIRD-VNE to that achievable under the basic Greedy algorithm, referred to as BASELINE and proposed in [184].

The simulator generates Φ and Υ according to Erdös−Rènyi model. Similar to [53], Φ has 0.5 probability of connecting any two substrate nodes, $n = 50$, $C\,(s) \sim U(0, 50), \forall s \in S$ and

[3] Implementations of RVICE-SP and RVINE-MCF are online available at http://www.mosharaf.com/ViNE-Yard.tar.gz

$C(l) \sim U(0,50), \forall l \in L$. Substrate nodes are placed randomly on a (25×25) grid. The mean inter-arrival time of virtual networks ranges from 5 to 25 networks per time unit, and the average service time is set to $\tau = 1000$ time units. Each pair of virtual nodes in Υ is connected with 0.5 probability, $n_v \sim U(1,10)$, $\Delta = 15$, $T(v) \sim U(0,20), \forall v \in V$ and $T(e) \sim U(1,50), \forall e \in E$. The routing of the substrate network $\mathbf{R}$ is computed once in the prepossessing initialization step using the all shortest path algorithm. All the cost parameters α, β, γ are set to unity in the simulations.

We simulate the mobility of substrate nodes by setting $\tau = 50$, and the average waiting time at each waypoint to $\mu_s^{-1} = 100$ time units for all the substrate nodes. All substrate nodes travel with the same constant speed $C_i = 5$ speed units, and the average transition length of all the nodes is 5 length units (i.e. $\frac{\lambda_s^2}{2} = 5$). We consider a wireless network infrastructure in which the connectivity between the substrate nodes are not impacted by their mobility since fixed clones of the mobile nodes actually execute the virtual network requests in a geographically distributed cloud infrastructure as discussed in Section 4.7 and as illustrated in Figure 4.11.

4.6.2 Performance evaluation

4.6.2.1 BIRD-VNE improves the acceptance rate

Figure4.8 shows that BIRD-VNE has a 15% better acceptance rate when compared to the other algorithms. The improvement in the acceptance rate is a direct result of Theorem 4.4.5 and is consistent for different loads. BIRD-VNE is likely to find a feasible embedding once it passes the consistency enforcement steps 1 to 6.

On the other hand, RVINE-SP and RVINE-MCF first rely on LP relaxations to solve the non-convex MIP problem, and then round the solution of the relaxation to the nearest integer. This way, RVINE-SP and RVINE-MCF may unnecessarily reject a VNE request by falsely concluding that it cannot be embedded. Moreover, when there is no solution, RVINE-SP and RVINE-MCF tend to spend a significant amount of time searching for solutions before eventually rejecting unfeasible requests as shown in Figure 4.4-b.

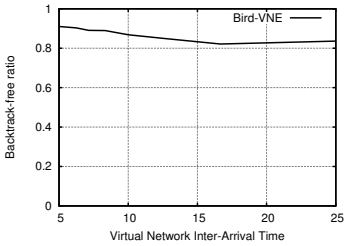


Figure 4.4: Backtrack-free ratio of BIRD-VNE shows the effectiveness of search space pruning.

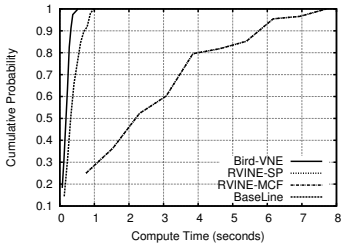


Figure 4.5: Experimental computation time CDF of BIRD-VNE shows its computation effectiveness.

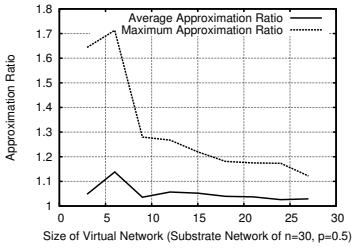


Figure 4.6: BIRD-VNE Maximum and average approximation Ratio (optimal is branch and bound).

4.6.2.2 BIRD-VNE avoids backtracking

Figure4.4-a shows the backtrack-free ratio of BIRD-VNE. BIRD-VNE is unlikely to encounter a backtracking, and finds a feasible solution from the first attempt. In this simulation setup, the backtrack-free ratio is greater than 80% regardless of the arrival rate. This demonstrates the effectiveness of TOPOLOGY-CONSISTENCY and CAPACITY-DISJOINT in pruning the search space by removing the virtual links and nodes that can cause backtracking. Moreover, in large-scale networks where link bandwidth is not a bottleneck, it is possible to ensure a 100% backtrack-free search by ensuring that all link mapping domains are capacity consistent according to Theorem 4.4.5.

4.6.2.3 BIRD-VNE minimizes and bounds the average cost

The approximation ratio is assessed by comparing the cost achieved by BIRD-VNE to the optimal cost achieved by branch and bound for a substrate network with 30 nodes. As shown in Figure4.4 - c, the cost achieved by BIRD-VNE is, on average, only about 5% higher than the optimal cost (i.e., average ratio = about 1.05). But the maximum cost can reach up to 70% higher than the optimal cost (maximum ratio = about 1.7).

4.6.2.4 BIRD-VNE results in the best revenue to cost ratio

The revenue to cost ratio reflects the average profit of BIRD-VNE, and is 20% better than RVINE-MCF as shown in Figure4.7. This is expected for two reasons. First, BIRD-VNE is a

$\frac{1}{2}$-approximation of the optimal cost, which contributes to the R/C ratio by minimizing the cost. Second, we have shown numerically that BIRD-VNE has the highest acceptance rate, which directly reflects on the total generated revenue by accepting as many virtual network requests as possible.

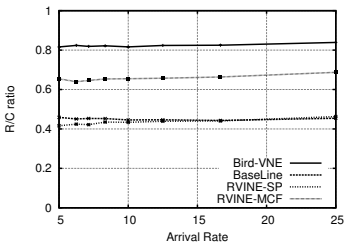


Figure 4.7: Revenue to Cost ratio for $\alpha = \beta = \gamma = \alpha' = \beta' = 1$.

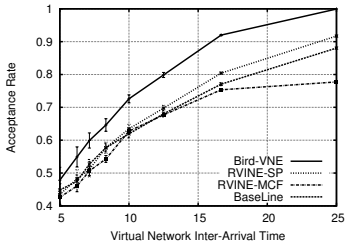


Figure 4.8: Acceptance rates of BIRD-VNE under different arrival rates.

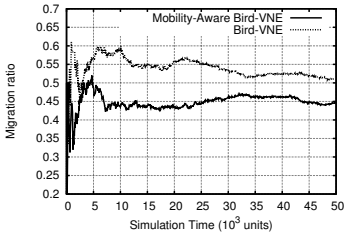


Figure 4.9: Mobility-aware Bird-VNE improves the migration ratio.

4.6.2.5 BIRD-VNE link utilization is better

The average link utilization achievable under BIRD-VNE is comparable to that achievable under RVINE-MCF when considering various inter-arrival rates, as shown in Figure4.10. However, for higher loads, the average link utilization of BIRD-VNE is less that that of RVINE-MCF, which confirms our earlier argument stating that BIRD-VNE tends to allocate shorter substrate paths to the virtual links with higher demands. On the other hand, the average node utilization achieved by BIRD-VNE is generally greater than that achieved by RVINE-MCF due to the better acceptance rates.

4.6.2.6 BIRD-VNE reduces the migration ratio

Figure4.9 shows the effectiveness of BIRD-VNE in minimizing the migration ratio. In this figure, Mobility-Aware Bird-VNE corresponds to $\gamma(v) = 1$ and Bird-VNE corresponds to $\gamma(v) = 0$. Even when the migration cost is low (i.e., $\gamma(v) = 1$), BIRD-VNE can reduce the migration ratio by at least 10%. This gain can be increased by increasing the migration cost (γ), which is a design trade-off. Observe that because of mobility, about 50% of the accepted virtual networks face migrations.

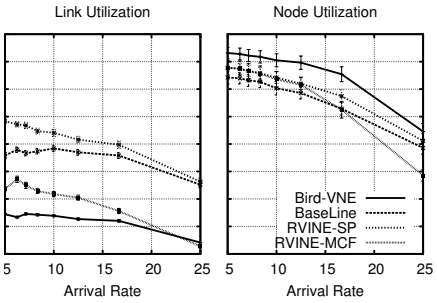


Figure 4.10: Average substrate node and link utilization.

4.7 Discussion and practical considerations

Several architectural and practical considerations pertain to the discussed virtual network network embedding solution. We discuss some possible approaches to address these challenges.

Topology changes: Substrate nodes are generally resources limited (e.g. smart-phones) and mobile which results in network topology changes that require updating all the substrate paths computations following any topology change. Updating all paths, $\mathbf{R}$, can be addressed architecturally or algorihmically. Figure 4.11 shows a possible architecture utilizing the emerging mobile edge computing to address this challenge by augmenting a wireless network infrastructure with distributed cloud resources. Each mobile node replicates its data and states (e.g sensors measurements, locations) to a corresponding clone that is proximate to the node (i.e. at the access point or cellular site). Clones are the actual entities that shall execute the virtual network requests. Cloning the mobile nodes provides several advantages over executing the virtual networks directly on the mobile nodes including: (i) providing manageable and salable processing and link capacity according to virtual networks demands, (ii) facilitating energy conservation of the actual mobile nodes which may be power limited (e.g. sensor nodes), (iii) preventing excessive latency compared to replicating nodes' data in *distant data-centers*, and (iv) preventing substrate network topology changes due to mobility. Unfortunately, the architecture shown in Figure 4.11 is not sufficient to prevent updating $\mathbf{R}$ in some cases such as back-hauling links or node failures or changes in clones deployment. Fortunately updating the set of all paths $\mathbf{R}$ is not as expensive as computing it from scratch and has remarkable long research history. The authors in [58], for example, study the combinatorial properties of graphs that can be used to update all shortest paths in dynamic networks in $O(n^2 \log^3 n)$ which is *not* a dominant factor in the complexity analysis of our proposed techniques as discussed in Section 4.4 and Section 4.5.

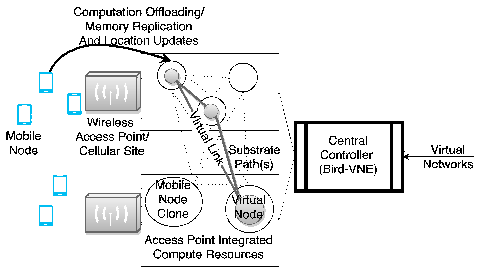


Figure 4.11: Architecture: Mobile nodes are connected through wireless infrastructure with integrated compute resources that host clones of mobile nodes and actually implement the requested virtual networks.

Mobility Model: the general RWP model cannot capture exact mobility patterns especially in walking scenarios. However, the recent modifications of the RWP model in [115] captures the mobility patterns almost exactly at the accuracy of cell level in 3GPP cellular networks which is suitable for several applications such as virtual sensor networks, and virtual content delivery networks. If a finer grain location resolution (e.g. locations of pedestrians at few meters error) were needed by some applications, the RWP model may fail to capture the exact mobility trajectory. In such cases, one can employ other mobility models that charachterize smooth movements of mobile nodes (see for e.g. the Semi-Markov Smooth model [182]), or employ model independant trajectory tracking methods (e.g. Kalman filtering) to track nodes locations. Such methods are outside the scope of this paper.

Multipath adoption: If multipath were allowed for mapping virtual links, we conjecture improvements particularly in the acceptance rate [174]. First, multipaths shall allow online path optimization, and traffic splitting for highly demanding virtual links. Second, multipaths shall increase link utilization making the most benefits of the network. Third, multipaths shall facilitate a better sharing of mobile wireless nodes. Finally, multipaths shall allow balancing the substrate network traffic used by the virtual networks and already existing services.

Chapter 5: Cloud of Things for Sensing-as-a-Service: Architecture, Algorithms, and Use Case

5.1 Introduction

Remote sensing applications will evolve through on-demand sensing services provided by the global network of sensor equipped devices in our homes, factories, cities, and bodies known as the Internet of Things (IoT). Today in smatphones 'only', there are seven sensors on average per device including: magnetometer, barometer, light, heart, humidity, and temperature sensors that one can use as participatory sensors to carry out applications like short-term weather forecasting [108, 120]. The density of smartphones' sensors in London today exceeds 14,000 sensor per square kilometer[1]. By 2020, the global number of sensor-equipped and location-aware devices (e.g. wearable, smart home, and fleet management devices) will reach tens of Billions, potentially creating dense, dynamic, location-aware, and onerous to manage networks of devices that can realize the vision of providing a versatile remote sensing services, known as 'Sensing as a Service' [9, 132, 8].

We conjecture that employing IoT devices' sensing resources in a *cloud computing like platform* to support remote sensing applications may be an effective approach to realize the Sensing as a Service vision [9]. The idea is to dynamically augment and scale up existing cloud resources (compute, storage, and network) by exploiting sensing capabilities of devices through cloud agents near the network edge to form a *global* system named the *Cloud of Things* (see Figure 5.1). The Cloud of Things is a geographically distributed infrastructure with cloud agent elements that continuously discover and pool sensing resources of IoT devices to be used by cloud users on-demand. This infrastructure provides elastic sensing resources that scale up and down according to remote sensing applications' demands, providing an optimized and controllable sensing resource utilization and pricing based on measurable usage.

Cloud of Things shifts the current, conventional remote sensing use of cloud platforms from a 'collect sensor data now and analyze it later' scenario to a usage scenario that *directly provides*

[1]The population density in London exceeds 4,000 inhabitants per square kilometer and the UK smartphone penetration reaches 55%.

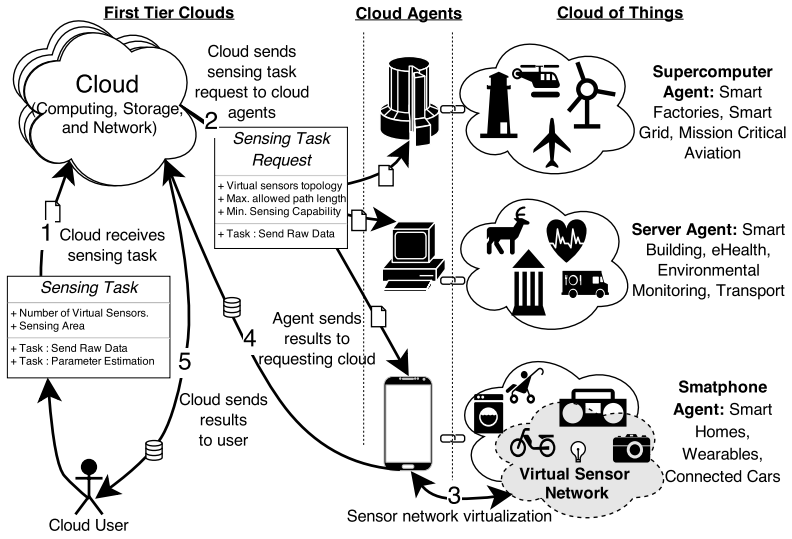
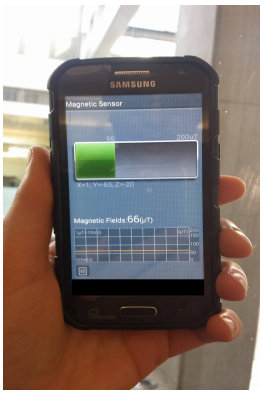


Figure 5.1: Sensor network virtualization in Sensing as a Service by different cloud agents near the edge. First tier clouds are conventional cloud computing platforms, and cloud agents are edge computing platform with evolved rule for Sensing as a Service. Arrows and numbers illustrate messages flow and sequence of the proposed usage scenario.

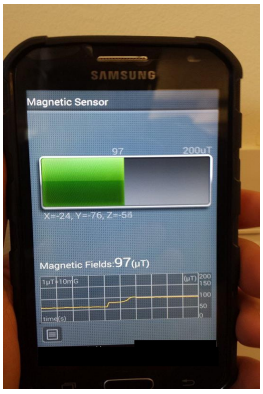
meaningful information from in-network processing of sensing data by IoT devices. Without such a conjectured infrastructure, remote sensing users can still gain access to sensing resources through conventional cloud back-end systems (see [149, 132]), with less opportunities to scale out sensing applications over the globally available sensing resources and with intolerable performance to applications that require responsive exploitation and fusion of sensing data and agile in-network decisions (e.g. localization [135] and estimation [10]).

5.1.1 Cloud of Things Infrastructure

Cloud platforms near the network edge already exist in different forms such as smartphones, personal computers, gateways, and servers to offer computation offloading to nearby devices in real-time (e.g. cloudlets [144, 104] and edge computing platforms [35, 150]). *We envision a new role of edge platforms as cloud agents that incorporate IoT devices as sensing resources (Figure 5.1) to scale up the conventional cloud with global and location specific sensing resources. We propose algorithmic solutions that provide*: (1) fast discovery of devices' dynamic sensing resources in specific geographical areas, (2) optimized device virtualization to serve as virtual



(a) Magnetic field sensor reading close to a window (low energy environment).



(b) Same sensor reading close to a power source (high energy environment).

Figure 5.2: Example of energy profiling from cheap magnetic field sensor in smartphones.

sensor networks by exploiting the discovered sensing resources, and (3) efficient in-network processing of sensing data from unreliable but dense sensors in IoT devices.

Cloud agents implement remote sensing applications as virtual sensor networks to be deployed on virtualizable IoT devices in a geographical area. A virtual sensor network performs distributed in-network processing of sensing data such as: aggregation, feature extraction, belief propagation, and consensus estimation to serve applications such as: distributed computer vision, data analytics, or on-demand context awareness. These virtual sensor networks may employ devices' sensing resources that are discovered by the various multiple cloud agents. Conventional cloud platforms provide a unified interface to cloud users to seamlessly use such global sensing resources from anywhere and at anytime while hiding complexities and supporting interaction between cloud agents. In Cloud of Things, IoT devices become surrogates of federated sensor networks (i.e administrated by a single organization) that can potentially reduce the total cost of ownership for remote sensing applications.

However, the IoT devices usually incorporate cheap and unreliable sensors to serve specific task that is not intended for remote sensing applications. For example, augmented reality applications in smartphones make use of the measurements from magnetic field sensors. The same magnetic field sensors can be used to profile energy levels in different environment (see Figure 5.2 for an example). The main problem with remote sensing applications based on IoT devices sensors is that individual measurements from the sensors of a single IoT device (e.g.

magnetic field) are insufficient for a reliable sensing task. In Figure 5.2, it is hard to distinguish the real context of a magnetic sensor reading changes; whether it is a result of user proximity to the device, changes in the device's orientation, or presence in a high energy environment. Similar unreliability problems appear in traffic congestion estimation in navigation applications (e.g. Google maps), weather prediction from smartphones barometers, or indoor localization.

A virtual sensor network of a group of independent IoT devices can solve this problem through distributed consensus estimation. Instead of analyzing measurements from an individual sensor, the virtual network use independent sensor measurements from several devices and executes an efficient *in-network* distributed consensus estimation algorithm to be able to efficiently achieve its goal (e.g. estimating energy level in an environment surrounding a user). In this paper, we focus our analysis and evaluation on distributed consensus estimation as the sensing task under study.

5.1.2 Contribution and Organization

In this paper, we propose a system to perform in-network analytics, such as distributed parameter estimation, based on commodity IoT devices that act as surrogates of wireless sensor networks (i.e. virtual sensor networks). We design a virtualization algorithm that suits the use case of describing analytics as on-demand virtual sensor networks and the challenges of the conjectured architecture in Figure 5.1. We also propose a distributed consensus parameter estimation algorithm to be executed by the optimized virtual sensor network. The distributed consensus algorithm provides a reliable, high quality parameter estimates from the low-quality and unreliable sensors in commodity IoT devices.

We discuss the technical challenges and our envisioned use case of the proposed architecture in Section 5.2. In Section 5.3, we first propose a sensing resource discovery algorithm that uses a gossip policy for propagating a sensing task requirements to devices (or their virtual instance at the edge cloud) and selects feasible devices to execute the task while responding to the dynamic changes of devices as fast as possible. Then, we propose RADV, an efficient virtualization algorithm, that deploys a virtual sensor network corresponding to the sensing task on top of a subset of the selected devices with minimal physical resources. In Section 5.4, we propose RADE; an efficient estimation algorithm that relies on the virtual sensor network, formed by our proposed virtualization algorithm, to estimate a set of unknown parameters in a distributed way and without requiring synchronization among the IoT devices. We discuss several related

work in Section 5.6. Finally, we numerically evaluate our proposed algorithms in Section 5.5 and conclude this paper in Section 5.7.

5.2 Architecture Usability and Challenges

The proposed Cloud of Things architecture allows cloud users to run remote sensing tasks, with certain specifications, virtually on any sensor-equipped IoT devices (see Figure 5.1). For example, a cloud user can profile pollution changes in cities from real-time temperature and CO2 concentration measurements collected by sensors in vehicles with defined precision and accuracy. The architecture consists of three main elements: IoT devices, first tier clouds, and cloud agents. **IoT devices** are sensor-equipped devices that can serve both specific and general purpose remote sensing applications. **First tier clouds** are conventional cloud platforms that provide unified interfaces to users to access the system and hide complexities underlying the realization of sensing services. Throughout, we refer to a first tier cloud as simply 'cloud'. Finally, **cloud agents** are trusted and resource-rich elements near the network edge that are well-connected to the Internet and to conventional cloud platforms. Cloud agents can be as powerful as supercomputers, or as flexible as smartphones according to the types of devices they serve and the computing resources these devices demand. Throughout, we refer to a cloud agent simply as 'agent'.

This architecture offers new sensing features and service level guarantees with several benefits. Deploying agents (cloud agents) close to devices improves responsiveness to sensing task requests and enables access to a globally available sensing resources. From the devices' viewpoint, cloud resources can be split into local resources (agents' resources) and global resources (clouds' resources) that can improve resiliency by migrating sensing tasks as the states of the devices - which carry out the sensing task - change. A cloud (first tier cloud) also acts as liaison to support coordination between distributed agents, while these agents can rapidly capture dynamics of the devices (e.g. utilization, connectivity, and availability). This approach simplifies analytics and big data with possible direct device access for agile in-network data processing and decision making. The proposed architecture finally allows the design of network-aware and performance-optimized cloud procedures.

5.2.1 Use case and System Model

Figure 5.1 summarizes message sequence and flow between the different architectural elements. These are detailed as next.

5.2.1.1 First tier clouds

A cloud (first tier cloud) handles *sensing tasks* initiated by a user with a unified interface (**step 1** in Figure 5.1). A *sensing task* defines physical parameters (e.g pollution changes) that the user wishes to estimate in a defined geographical area during a predefined time with certain sensing capabilities of the IoT devices carrying out the task. The sensing task objective can be: information retrieval of raw sensed data, or execution of distributed algorithms on a virtual sensor network deployed on multiple interconnected devices. We represent a *sensing task* by the triple, $\langle g, c, \delta \rangle$, where g denote the number of virtual sensors requested to perform the sensing task and the two parameters c and δ define the center and the radius of a geographical area of interest to the user's remote sensing application.

5.2.1.2 Sensing task requests

The cloud translates a sensing task to a corresponding *sensing task request* that it sends to its agents. A *sensing task request* defines the virtual sensors set, V, to be deployed on g connected devices, which are all located within distance δ from the area center c. For each virtual sensor $j \in V$, the cloud defines a *minimum sensing capability*, $R(j)$. The minimum sensing capability represents the minimum storage capacity, the minimum CPU computing power, and/or the minimum amount of time that devices (to carry out the sensing task) must fulfill. The cloud may also choose a suitable virtual topology that interconnects the virtual sensors so that they can execute distributed algorithms for in-network processing of devices' sensed measurements.

For example, for aggregation and belief propagation algorithms, the cloud organizes the virtual sensor network as a spanning tree. A star topology can also be adopted for distributed algorithms that are implemented using the map-reduce or graph-processing paradigms. Although consensus algorithms, which are our main focus, can run with any arbitrary topology, we show that a complete topology results in faster convergence. For our evaluation, we focus on three common virtual topologies: complete, cyclic, and star. For a given topology, let E denote the set of virtual links connecting the virtual sensors and $\Upsilon = (V, E)$ be the graph data structure that

represents the *virtual sensor network* of the virtual sensors (connected according to the given virtual topology). After translating a sensing task to its corresponding *sensing task request*, the cloud sends this request (i.e. the graph data structure Υ) to its agents (**step 2** in Figure 5.1).

5.2.1.3 The IoT devices capabilities

Agents manage a large number of interconnected *IoT devices*. A *device* i, at time t, maintains its geographical location denoted by $loc(i)$ and its current sensing capability denoted by $C(i)$. $C(i)$ defines the currently allowed sensing time, available processing capacity, and/or available memory capacity that the i-th device can allocate (at time t) to fulfill the minimum sensing capability demanded by a virtual sensor j (i.e. $R(j)$) to be deployed on i. *The sensing capability of a device can correspond to local device's resources (i.e. CPU, memory, storage, and sensors) or to resources at the edge cloud (agent) that the device may opportunistically use through computation offloading mechanisms.* We also assume that two devices can directly communicate with each other if they are within a transmission radius r. We model the network of all n devices, connected to a single agent, as the Euclidean geometric random graph, $\mathcal{G} = (S, L)$, where S denote the set of n devices, and L denote the set of all links connecting the devices. We assume that each sensor $i \in S$ is capable of estimating a vector of unknown parameters, $\theta \in \mathbf{R}^N$, through noisy sensors measurements, $x_i \in \mathbf{R}^M$. That is,

$$x_i = H_i\theta + u_i,\ i = 1, \ldots, n$$

where $H_i \in \mathbf{R}^{M \times N}$ is sensor i's sensing model (typically known to i only) relating x_i to θ, and u_i is an additive Gaussian noise with zero mean and variance σ_i^2. We assume that u_i and u_j are independent from one another for all $i, j \in S$. Because different sensors may have different sensing models and/or different measurement methods, it is very likely that different sensors have different estimates of θ. Also, we do not assume/require that the sensors are synchronized; that is, the consensus algorithms we develop in this paper to estimate θ are asynchronous.

5.2.1.4 Service Level Agreement (SLA) implications on agents configurations

Agents handle sensing task requests under agreed SLAs with users through the cloud. An SLA generally consists of: $i)$ a maximum time within which the sensing task must be completed, $ii)$ a feasible selection of IoT devices to carry out the sensing task under certain tolerances of the results accuracy, and $iii)$ a maximum task rejection rate defined as the ratio of the number of failures to handle sensing task requests to the total number of requests. The cloud translates an SLA to parameters that agents can use in their algorithmic solutions to discover sensing resources and virtualize devices efficiently. Defining all possible parameters that reflect any SLA is beyond the scope of this work and we consider only four parameters.

The first parameter is the absolute error of the estimated parameter θ, denoted by ϵ_{abs}. The second parameter is the relative error, ϵ_{rel}, of the parameter θ estimated by the different virtual sensors such that all sensors estimate θ within ϵ_{rel}. The third parameter, defined earlier, is the *minimum sensing capability* $R(j)$ of the j-th virtual sensor. The fourth parameter is the maximum allowed path length, $\bar{h}$, between any pair of virtual sensors. $\bar{h}$ limits the number of devices/hops a message, exchanged between virtual sensors, can go through.

A virtual link between two virtual sensors may map to devices that do not necessarily deploy a virtual sensor and the virtual sensor network just use these devices for message forwarding. We use $\bar{h}$ to impose an upper limit on these intermediate devices for two reasons. First, restricting the number of intermediate devices shall bound the sensing task performance by an SLA. Second, the sensor discovery and virtualization algorithms shall use the least number of hops and the least possible physical resources when mapping the virtual network, so as to maximize the Sensing-as-a-Service benefits (**step 3** in Figure 5.1). The implication of $\bar{h}$ on the virtualization design will be discussed later in this section.

5.2.1.5 Sensing task execution (consensus)

Consensus estimation resembles the most commonly used sensing task relying on a collection of measurements from unreliable sensors. In consensus estimation, the sensing task is to estimate a set of parameters, θ, based on the measurements sensed by the IoT devices so that the estimated parameters are at most ϵ_{abs} away from their actual value, and so that all the sensors consent to the same estimate value of θ with a tolerance of ϵ_{rel} according to the SLA.

Without loss of generality, consider indexing the selected g sensors in the virtual sensor network as $1 \dots g$ and let $x = \left[x_1^\mathsf{T}, \dots, x_g^\mathsf{T}\right]^\mathsf{T}$, $H = \left[H_1^\mathsf{T}, \dots, H_g^\mathsf{T}\right]^\mathsf{T}$, and $u = \left[u_1^\mathsf{T}, \dots, u_g^\mathsf{T}\right]^\mathsf{T}$. The combined measurements can then be written as $x = H\theta + u$. One simple approach of estimating θ is to have the cloud agent first collect from each virtual sensor i its measurement vector, x_i, and its sensing model, H_i, and then solve the following LS problem

$$\text{minimize} \quad \tfrac{1}{2}\|x - H\hat{\theta}\|^2 \tag{5.1}$$

where $\hat{\theta}$ is here the optimization variable. The unbiased Maximum-Likelihood (ML) estimate of θ is simply $\hat{\theta}_{\text{LS}} = \left(H^\mathsf{T}H\right)^{-1} H^\mathsf{T}x$.

5.2.2 Technical Challenges and Solutions Objectives

The proposed Cloud of Things architecture and use case envision designing algorithmic solutions with specific objectives, given the following set of challenges:

5.2.2.1 Sensing resource discovery

In the sensor network virtualization (**step 3** in Figure 5.1), an agent searches for devices with sensing capabilities that meet the sensing task requirements specified by the virtual sensor network data structure Υ. For a given Υ, the agent discovers devices' sensing capabilities and searches for a subset of devices, $S' \subset S$, such that a device $i \in S'$, if it is geographically located within δ distance from the center c, and the discovered sensing capability $C(i)$ satisfies the minimum sensing capability $R(j)$ demanded by at least one virtual sensor $j \in V$. We define the virtual domain, $\mathcal{D}(i)$, of a device i as

$$\mathcal{D}(i) = \{j \in V : C(i) \geq R(j), \|loc(i) - c\| \leq \delta\} \tag{5.2}$$

hence

$$S' = \{i \in S : |\mathcal{D}(i)| > 0\}. \tag{5.3}$$

The design objective of a sensing resource discovery algorithm is to construct the virtual domains, $\mathcal{D}(i)$ for all $i \in S$, as fast as possible and with minimal communication overhead between the agent and the devices and between the devices themselves.

The challenges related to sensing resource discovery arise from the large number of devices and their onerous to maintain dynamics. The large number of devices connected to an agent requires a scalable solution to discover devices' sensing capabilities and to decide if a device's current state (e.g. connectivity to other devices) allows it to deploy a particular virtual sensor. Moreover, the dynamics and rapid changes in the whole network, $\mathcal{G}$, including device availability, mobility, connectivity, and resource utilization, make it too difficult to maintain devices' states in a centralized manner. To address these challenges, we propose a distributed algorithm that propagates the graph data structure Υ to devices in $\mathcal{G}$ using a gossip policy as detailed in Section 5.3.1.

5.2.2.2 Virtualization

After performing the sensing resource discovery, an agent deploys the virtual sensor network, Υ, by means of devices virtualization. The virtualization task consists of finding: i) a set $A \subset S'$ of exactly g connected devices according to the virtual topology chosen by the cloud, and ii) a set $\mathcal{M}_A \subset \{(i,j) \in A \times V : j \in \mathcal{D}(i)\}$ of (device,virtual sensor) mapping pairs such that one virtual sensor maps to exactly one device and a device maps to one and only one virtual sensor in g. Also, the length $h(i,i')$ of any simple path connecting two distinct devices $i, i' \in A$ that maps a virtual link $(j,j') \in E$ must be less than or equal to $\bar{h}$. We refer to a $\{A, \mathcal{M}_A\}$ pair that satisfies the previous conditions as a *feasible virtualization* of the requested virtual sensor network Υ. Note that for any possible set A, there can exist multiple mappings, $\mathcal{M}_A$, and each can form a feasible virtulization. *The design objective of a virtualization algorithm is then to find the 'optimal' feasible virtualization, $\{A, \mathcal{M}_A\}^*$, that uses the least possible physical network resources.*

We now define and introduce what an 'optimal' feasible virtualization means. We consider that the number of virtual sensors and the number of virtual links of a given $\Upsilon = (V, E)$ determine the cloud cost of providing the sensing service, which is given by $\text{Cost}(\Upsilon) = \alpha|V| + \beta|E|$. The scalar α denote an incentive paid to each device that maps a virtual sensor, and the scalar β denote an incentive divided and paid to each device on a physical path that maps to a virtual link. An incentive could be monetary or could be in any other form (e.g., credit, service, etc.). On the other hand, the total devices' benefit from mapping the requested virtual network, Υ, can

be expressed as

$$\text{Benefit} = \sum_{(i,j)\in\mathcal{M}_A} \alpha\frac{C(i)-R(j)}{C(i)} + \sum_{(i,i')\in P} \beta\frac{\bar{h}-h(i,i')}{\bar{h}}, \tag{5.4}$$

where $h(i,i')$ is again the length (in number of hops) of the path connecting the device pair, (i,i'), mapping the virtual link between j and j', and $P = \{(i,i') \in A \times A : (i,j),(i',j') \in \mathcal{M}_A\,, (j,j') \in E\}$.

The total devices' benefit in (5.4) implies that the lesser the used physical resources, the greater the benefit to the devices. The first term of (5.4) captures the benefit loss of the i-th device from allocating resources to map a virtual sensor j. As the minimum demanded sensing capability $R(j)$ becomes negligible (w.r.t. the sensing capability $C(i)$), i gets higher benefit as it invests lesser fraction of its resources (e.g. energy, CPU, or memory) to map j for the same incentive α. Similarly, the second term captures the benefit loss of devices i and i', which map the virtual sensors j and j' respectively. Such benefit loss results from mapping the virtual link between j and j' with more intermediate devices, as the same incentive β for the virtual link (j,j') is divided on a greater number of devices (i.e. number of hops $h(i,i')$) compared to $\bar{h}$. Theoretically, $\bar{h}$ can take a value up to the diameter of $\mathcal{G}$. However, this shall not work in practice as the diameter of $\mathcal{G}$ is assumed to be much greater than a user desired diameter Υ. The virtualization algorithm that we propose in Section 5.3.2 consists of finding an 'optimal' feasible virtualization that maximizes the total benefit given in (5.4). We refer to the optimal solution as $\{A,\mathcal{M}_A\}^*$. Clearly, finding $\{A,\mathcal{M}_A\}^*$ is hard due to the factorial size of the solution space in n and due to the challenges, discussed earlier, associated with the sensing resources discovery task.

5.2.2.3 Distributed consensus estimation

The virtual sensor network determined during the virtualization phase executes the distributed sensing algorithms. The simple solution to the LS problem, proposed in (5.1), requires that each virtual sensor exchanges its measurement vector and its sensing model with the cloud agent, which can create significant communication overhead. Therefore, we instead propose a decentralized approach that relies on the virtual sensor network to provide an estimation of the parameter vector θ. We rely on the recent results presented in [168] to develop our distributed

estimation algorithm, which reduces communication overhead significantly when compared to the conventional ADMM approach [131] in addition to not requiring synchronization among sensors. The proposed algorithm is presented in Section 5.4.

5.3 Proposed Solutions for Sensing Resource Discovery and Virtualization

5.3.1 Sensing Resource Discovery

Although devices are directly accessible by cloud agents, contacting the devices at fine-grained time slots to discover their current sensing capabilities creates significant communication and computation inefficiency for large n. Such a centralized approach requires exchanging $O(n)$ messages, in each time slot, while constructing the virtual domains, given by (5.2), requires $O(n)$ time. Moreover, activating devices periodically to update their current sensing capabilities to their cloud agents is power inefficient, especially if the devices are battery operated.

We propose to perform sensing resource discovery through a gossip based algorithm that requires a time complexity of $O(r^{-1}\log n)$ and an average $\Theta(1)$ messages per device. In this algorithm, an agent propagates information about a received sensing task request, Υ, using the following 'gossip policy'.

The agent sends Υ to a randomly chosen device starting at $t = 0$. Then, any device that receives Υ continues sending Υ to a random device of its direct neighbors until one neighbor acknowledges that it has already double received the same version of Υ in a previous step; by then the device stops sending Υ. The agent does not need to send Υ to each device as the utilized gossip policy allows devices to disseminate Υ autonomously, and the network of devices is guaranteed to be connected with high probability if each device is connected to k neighbors and $k \geq 0.5139 \log n$ [27]. Since $\mathcal{G}$ is a connected network, this simple gossip policy guarantees that Υ reaches all the devices in $O(r^{-1}\log n)$ time (see [148] for time complexity analysis of general gossip protocols in Euclidean geometric random graphs). Hence a device i can construct $\mathcal{D}(i)$ according to (5.2) once it receives Υ and the agent can discover sensing resources of devices that are capable of satisfying the requirements of Υ as fast as possible with minimal communication overhead.

The agent and all its connected devices implement the active and passive threads shown in Figure 5.3. At the k-th time slot, let the device i be active and contact a random neighbor device

while True **do**
  wait Δt
  $s \longleftarrow$ random neighbor
  if Υ is $\emptyset$ **then**
    solicit Υ from s
  else
    send Υ to s
  end if
  receive Υ' from s
  if $\Upsilon' = \Upsilon$ **then**
    stop sending Υ
  else
    Υ' is newer than Υ
    $\Upsilon \longleftarrow \Upsilon'$
    evaluate $\mathcal{D}(i)$
  end if
end while

i) active thread at device i

while True **do**
  receive Υ' or
  solicit request from s
  if Υ is not $\emptyset$ **then**
    send Υ to s
  end if
  if Υ' is new **then**
    $\Upsilon \longleftarrow \Upsilon'$
    evaluate $\mathcal{D}(i)$
  end if
end while

ii) passive thread at i

Figure 5.3: proposed sensing resources discovery gossip based threads at device i.

i' (i.e., $(i, i') \in L$) with probability $T_{i,i'} > 0$. $T_{i,i}$ denote the probability that i does not contact any other device. Let the $n \times n$ matrix $T = [T_{i,i'}]$ be a doubly stochastic transition matrix of non-negative entries [40]. A natural choice of $T_{i,i'}$ is

$$T_{i,i'} = \begin{cases} \dfrac{1}{d_i + 1}, & \text{if } i = i' \text{ or } (i, i') \in L, \\ 0, & \text{otherwise,} \end{cases} \tag{5.5}$$

where $d_i = |\{i' \in S : (i, i') \in L\}|$ is the degree of i.

When i contacts i', they exchange information as follows (see Figure 5.3). i pushes Υ to i' only if i' does not have Υ, or pulls Υ from i' only if i does not have Υ. If i contacts i' and both devices have received Υ before, i stops contacting any other device. If $\mathcal{G}$ is connected, the proposed protocol guarantees that Υ is delivered to all IoT devices.

The actual running time of the proposed algorithm depends on the choice of the transition matrix T and the communication range of the used device-to-device communication technology. The running time is related to the mixing time of any random walk on $\mathcal{G}$ [40], which suggests

that there is an optimal value of $T_{i,i'}$ to minimize the mixing time and it is related to the second eigenvalue of the transition matrix. Moreover, in case of small r, the proposed algorithm is generally slow. Practically, this algorithm is suitable for device-to-device communication technologies that support communication ranges of few hundreds of meters, as in WiFi direct and LTE D2D and when $\mathcal{G}$ is sufficiently dense.

5.3.2 Virtualization

We present our proposed RADV algorithm. RADV consists of three phases: (I) pruning of virtual domains $\mathcal{D}(i)$ for all $i \in S$, (II) construction of benefit matrices in a distributed manner, and (III) solving assignment problems at virtual sensors. RADV results in multiple solutions each evaluated by a different sensor (device), and the cloud agent selects the solution with the maximum benefit.

Phase I—Virtual Domain Pruning

During this phase, we ensure that all virtualized sensors maintain the topology E by allowing a senor to receive the virtual domains of other sensors and delete a virtual sensor j from its domain if there exists a virtual link (j, j') such that j' is not included in any other received domains. Let $D_s \subset \{\mathcal{D}(i) : i \in S\}$ denote the virtual domains set that sensor s has at time k. Initially $D_s = \{\mathcal{D}(s)\}$ and $h(i, s) = 0$ for all $i \in S$[2]. Using the same transition matrix, T, defined in Eq. (5.5), s contacts only one of its neighbors s' at time k. Then, for all $\mathcal{D}(i) \in D_s \ : \ i \neq s'$, s pushes $\mathcal{D}(i)$ to s' only if s' did not receive $\mathcal{D}(i)$ before and $h(i, s) < \bar{h}$. Also, for all $\mathcal{D}(i) \in D_{s'} \ : \ i \neq s$, s pulls $\mathcal{D}(i)$ from s' only if s did not receive $\mathcal{D}(i)$ before and $h(i, s') < \bar{h}$. If no information is exchanged between s and s' at time k, s stops contacting any of its neighbors. However, s may restart contacting its neighbors again if it updated D_s after time $k + 1$.

When s constructs its D_s, it starts by pruning $\mathcal{D}(s)$. The pruning is performed by deleting a virtual sensor $j \in \mathcal{D}(s)$ (i.e., $\mathcal{D}(s) \leftarrow \mathcal{D}(s) \backslash \{j\}$) if none of the virtual sensors that are connected to j, $\{j' \in V : (j, j') \in E\}$, is not included in any received $\mathcal{D}(i)$, i.e. $j \notin \mathcal{D}(i) : \mathcal{D}(i) \in D_s$. This pruning rule ensures that the virtualized sensors maintain the required topology E and the constructed benefit matrices shall result in a feasible virtualization.

[2]Knowledge about other sensors existence is not needed, and h is typically evaluated dynamically.

Phase II—Construction of Benefit Matrices

As mentioned earlier, finding a feasible virtualization, $\{A, \mathcal{M}_A\}^*$, that maximizes the total benefit given in Eq. (5.4) is a hard problem due to the large size of the solution space. Therefore, this phase proposes an efficient way of solving this virtualization problem. Specifically, we propose a method that solves this problem in a distributed manner and without requiring any synchronization among sensors, as described next.

During this phase, each sensor s locally constructs its own set, $A^{(s)}$, of g sensors that s chooses as virtualized sensors to assign to virtual sensors in V. Each sensor s also maintains g row vectors, $B_i^{(s)} \in \mathbf{R}^{1\times g}$ and $i \in A^{(s)}$, that we define as the benefit vector of sensor i seen by s, where the j-th element, $B_{i,j}^{(s)}$, denotes the benefit of assigning participatory sensor $i \in A^{(s)}$ to the virtual senor $j \in V$ as seen by s, and is given by

$$B_{i,j}^{(s)} = \begin{cases} \alpha\dfrac{C(i) - R(j)}{C(i)} + \beta\dfrac{\bar{h} - h(j,s)}{\bar{h}} & \text{if } j \in \mathcal{D}(i), \\ 0 & \text{otherwise.} \end{cases}$$

Our objective is then to construct, for each $s \in S$, the benefit matrix $B^{(s)} = [B_{i\in A^{(s)}}^{(s)}]$ as fast as possible, and find a feasible virtualization, $\{A, \mathcal{M}_A\}$, that maximizes the total benefit,

$$\sum_{(i,j)\in\mathcal{M}_A} B_{i,j}^{(s)},$$

among all $s \in S$ without knowing the $\mathcal{G}$ structure. Moreover, the path length between a sensor s and any other sensor i that s includes in its benefit matrix must not exceed $\bar{h}$. Finally, a sensor s shall include only the benefit vectors of the g sensors with the largest possible benefit.

Each sensor s initially sets $A^{(s)} = A^{(s)} \cup \{s\}$ if $\mathcal{D}(s) \notin \emptyset$, sets $h(i,s) = 0$ for all $i \in S$, and sets

$$B_{s,j}^{(s)} = \begin{cases} \alpha\dfrac{C(s) - R(j)}{C(s)} + \beta, & j \in \mathcal{D}(s), \\ 0, & \text{otherwise.} \end{cases}$$

Also, s maintains a scalar, $b_s^{\min}$, defined as the minimum total benefit it has received from any other sensor and written as

$$b_s^{\min} = \min_i \sum_{j\in V} B_{i,j}^{(s)}.$$

s also maintains the corresponding sensor,

$$i_s^{\min} = \underset{i}{\operatorname{argmin}} \sum_{j \in V} B_{i,j}^{(s)}.$$

Initially, $b_s^{\min} = 0$ and remains so until $|A^{(s)}| = g$.

Using the same transition matrix, T, defined in Eq. (5.5), s contacts its neighbor s' only once at each time k. Then, for all $i \in A^{(s)} \ : \ i \neq s'$, s pushes the benefit vector $B_i^{(s)}$ to s' only if $h(i,s) < \bar{h}$ and

$$\sum_{j \in V} \left(B_{i,j}^{(s)} - \frac{\beta}{\bar{h}} \right) > b_{s'}^{\min}.$$

Also, for all $i \in A^{(s')} \ : \ i \neq s$, s pulls the benefit vector $B_i^{(s')}$ from s' only if $h(i,s') < \bar{h}$ and

$$\sum_{j \in V} \left(B_{i,j}^{(s')} - \frac{\beta}{\bar{h}} \right) > b_s^{\min}.$$

If no information is exchanged between s and s' at time k, s stops contacting its neighbors at time $k+1$. However, s may restart contacting its neighbors again if $B^{(s)}$ is updated after time $k+1$.

When s receives $B_i^{(s')}$, s updates $B_{i,j}^{(s)}$ as

$$B_{i,j}^{(s)} = \begin{cases} B_{i,j}^{(s)} - \dfrac{\beta}{\bar{h}} & \text{if } \ j \in \mathcal{D}(i), \\ 0 & \text{otherwise.} \end{cases}$$

If $i \notin A^{(s)}$, then we have two scenarios. In the first scenario, s still has not received g benefit vectors, so $b_s^{\min} = 0$ and $|A^{(s)}| < g$, then s updates its set of candidate sensors as $A^{(s)} = A^{(s)} \cup \{i\}$. In the other scenario in which $|A^{(s)}| = g$, s replaces the sensor corresponding to the minimum total benefit, $i_s^{\min}$, with i so that $A^{(s)} = A^{(s)} \setminus \{i_s^{\min}\} \cup \{i\}$. On the other hand, if $i \in A^{(s)}$, then s updates $B_{i,j}^{(s)}$ if $\sum_{j \in V} B_{i,j}^{(s')} > \sum_{j \in V} B_{i,j}^{(s)}$. Finally, s updates $b_s^{\min}$, $i_s^{\min}$, and $h(i,s)$ as $h(i,s) = h(i,s') + 1$.

Finding a feasible virtualization that maximizes the benefit $B^{(s)} = [\ B_{i \in A^{(s)}}^{(s)}\ ]$ instead of the benefit given in Eq. (5.4) makes the problem easier because every sensor has a different value for the benefit $B_{i,j}$ that depends only on the length of the physical path between i and s instead

of the path lengths of all possible combinations of sensor pairs (i, i') that can virtualize a virtual link. Intuitively, this relaxation still leads to an optimal or near optimal virtualization, because if $\mathcal{G}$ is very large and connected, the number of sensors that are directly connected by a single physical link (clique) grows logarithmically in n and hence this number is larger than g almost surely as $g \ll n$. In such a case, it is sufficient to ensure that the length of the paths between i and s and between i' and s are the shortest possible ones to ensure that the length of the path between i and i' is also the shortest, as in this case, s, i, and i' reside in the same clique with high probability. We evaluate the effectiveness of this relaxation in Section 5.5 and show that our virtualization algorithm performs well even when the condition $g \ll n$ does not hold.

Phase III—Solving Local Assignment Problem

After reception of the g benefit vectors, s proceeds to this phase of the algorithm only if it stops communicating and $|A^{(s)}| = g$. Each sensor $s \in S$ with $|A^{(s)}| = g$ solves locally the following assignment problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in A^{(s)}} \sum_{j \in \mathcal{D}(i)} B_{i,j}^{(s)} m_{ij} \\ \text{subject to} \quad & \sum_{j \in \mathcal{D}(i)} m_{ij} = 1, \ i \in A^{(s)}, \\ & \sum_{\{i : j \in \mathcal{D}(i)\}} m_{ij} = 1, \ j \in V, \\ & m_{ij} \in \{0, 1\}, \end{aligned} \tag{5.6}$$

where m_{ij} are binary optimization variables indicating whether the participatory sensor i is assigned to the virtual sensor j. The problem formulated in (5.6) is equivalent to the perfect maximum weight matching problem in a bipartite graph, and hence, we propose to use the classical Hungarian method to solve it (the worst case time complexity is $O(g^3)$ [107, 126]).

We can also tolerate an error $\epsilon > 0$ of the resulting total benefit and relax the restriction of finding a perfect matching for large g. This relaxation is reasonable when there are enough sensors involved in solving these local optimization problems, as in this case we can pick the best solution and discard those without a perfect matching. In such a scenario, we can also use a linear time $(1 - \epsilon)$-approximation algorithm to solve (5.6) [63]. In this paper, we use the Hungarian method to solve our formulated optimization problems. Details of the algorithm are omitted due to space limitation; readers are referred to [107, 126, 63] for detailed information.

Each sensor solves locally the optimization problem given in (5.6) and sends its obtained

solution to the cloud agent. This is done asynchronously. The cloud agent then selects the solution that leads to the maximum total benefit, and keeps all other solutions for later use in the event that the network dynamics invalidate the selected solution before the virtual sensing task completes.

Complexity and message overhead. We assume that the topology of $\mathcal{G}$, devices mobility, and sensing capability are not changed during the execution of the virtualization phase. The time required to spread Υ across the network is $O(r^{-1}\log n)$ [148]. It takes $O(g)$ worst case time to evaluate $\mathcal{D}(i)$ locally at sensor i. Also, the time required to spread information in the pruning and benefit construction phases is $O(r^{-1}n\log n)$. The pruning of the virtual domain $\mathcal{D}(i)$ requires node i to examine g received virtual domains, each having at most g entries. The worst case local running time of pruning is then $O(g^2)$. Finally, the local running time of the Hungarian method is $O(g^3)$. Hence, the overall complexity is $O(\max\{r^{-1}n\log n, g^3\})$.

The average number of messages communicated per sensor during the sensor search phase is $\Theta(1)$ and each message is $O(g)$ in size. During pruning of virtual domains, since every sensor exchanges a maximum of n domains each of size that is also $O(g)$, the average number of messages communicated per sensor is $O(n)$. However, because we restrict that messages to be communicated up to $\bar{h}$ hops for only a group of sensors that support the requirements of Υ, the average number of messages per sensor is typically small. Figure 5.4 shows the total time and the average number of messages per sensor required during both the domain pruning and the benefit construction phases. The total time growth is linearithmic in n when Υ is sent to exactly one sensor and when $\mathcal{G}$ is connected. This time can, in practice, be decreased significantly if Υ is initially sent to multiple sensors. Additionally, the average number of messages per sensor is shown to scale linearly with n, and is typically a very small fraction of n.

5.4 Proposed Solutions for Distributed Estimation

After completing the sensor virtualization task, using RADV, the virtual sensors run an in-network parameter estimation algorithm to compute $\hat{\theta}$ distributedly. In this section, we present our proposed RADE algorithm. We first follow the standard ADMM approach to derive primal, dual and Lagrangian variable update equations, then we describe the proposed RADE algorithm. For clarity of notation, in what follows, we refer to the set of g selected devices, determined by RADV, simply as A.

The centralized estimation approach given in (5.1) is first decomposed into g local estimates

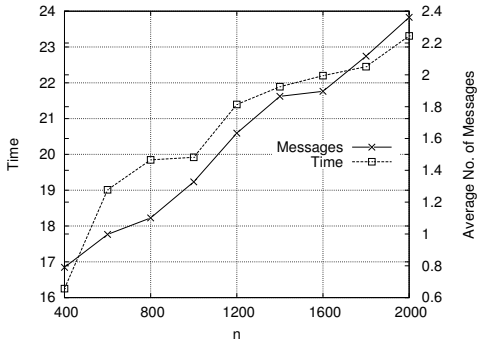


Figure 5.4: Time (in number of iterations) and message overhead (in average number of communicated messages) resulting from constructing the benefit matrices, $g = 10$.

of θ (one $\hat{\theta}_i$ for each $i \in A$) while constraining the local estimates with the coupling constraints $\theta_i = \theta_j$ for all $(i, j) \in P$. This results in the following optimization problem:

$$\begin{aligned} &\text{minimize} \quad \tfrac{1}{2} \sum_{i \in A} \|x_i - H_i \theta_i\|^2 \\ &\text{subject to} \quad \theta_i - \theta_j = 0 \text{ for all } (i, j) \in P, \end{aligned} \tag{5.7}$$

where $\{\theta_i, i \in A\}$ are the optimization variables.

By introducing an auxiliary variable, z, we decouple the constraints in (5.7), so that $\theta_i - z = 0$ for all $i \in A$ [129]. However, this requires that z be shared among all the g virtual sensors. Instead, we introduce g auxiliary variables, z_i, and equivalently write the optimization problem as

$$\begin{aligned} &\text{minimize} \quad \tfrac{1}{2} \sum_{i \in A} \|x_i - H_i \theta_i\|^2 \\ &\text{subject to} \quad \theta_j - z_i = 0 \text{ for all } (i, j) \in P. \end{aligned} \tag{5.8}$$

Let $\lambda = \{\lambda_{i,j} \in \mathbf{R}^{N \times 1} : (i, j) \in P\}$ and $\rho = \{\rho_{i,j} \in \mathbf{R} : (i, j) \in P\}$ denote respectively the set of Lagrangian multipliers and the set of penalty parameters. The augmented Lagrangian is given by

$$\begin{aligned} \mathcal{L}_\rho(\theta, z, \lambda) = \sum_{i \in A} \Bigg[& \tfrac{1}{2} \|x_i - H_i \theta_i\|^2 \\ & - \sum_{j \in A : (i,j) \in P} \lambda_{i,j}^{\mathsf{T}} (\theta_i - z_j) \\ & + \sum_{j \in A : (i,j) \in P} \tfrac{\rho_{i,j}}{2} \|\theta_i - z_j\|^2 \Bigg]. \end{aligned} \tag{5.9}$$

By setting the gradient w.r.t θ_i of Eq. (5.9) to zero and solving for θ_i, we get

$$\begin{aligned}\theta_i &= \left(H_i^{\mathsf{T}} H_i + \sum_{j \in A:(i,j)\in P} \rho_{i,j} I\right)^{-1} \\ &\quad . \left(H_i^{\mathsf{T}} x_i + \sum_{j \in A:(i,j)\in P} (\lambda_{i,j} + \rho_{i,j} z_j)\right).\end{aligned}$$

Similarly, we solve for z_i by setting the gradient w.r.t to z_i to zero and rearranging the indices of the Lagrangian multipliers and the penalty parameters. It follows that

$$z_i = \frac{1}{g} \sum_{j \in A:(i,j)\in P} \left(\theta_j - \frac{1}{\rho_{j,i}} \lambda_{j,i}\right).$$

The former analysis leads to the conventional ADMM-based distributed consensus estimation algorithm given by

$$\begin{aligned}\theta_i^{(k+1)} &= \left(H_i^{\mathsf{T}} H_i + \sum_{j \in A:(i,j)\in P} \rho_{i,j} I\right)^{-1} \\ &\quad . \left(H_i^{\mathsf{T}} x_i + \sum_{j \in A:(i,j)\in P} \left(\lambda_{i,j}^{(k)} + \rho_{i,j} z_j^{(k)}\right)\right), \\ z_i^{(k+1)} &= \tfrac{1}{g} \sum_{j \in A:(i,j)\in P} \left(\theta_j^{(k)} - \tfrac{1}{\rho_{j,i}} \lambda_{j,i}^{(k)}\right), \\ \lambda_{j,i}^{(k+1)} &= \lambda_{j,i}^{(k)} - \rho_{j,i} \left(\theta_j^{(k+1)} - z_i^{(k+1)}\right),\end{aligned} \tag{5.10}$$

where the superscript k denotes the value of the variable at the k-th iteration. This conventional ADMM algorithm, given in (5.10), requires synchronization and variable update among the sensors [41, 172]. Moreover, at each iteration k, each sensor i must send $z_i^{(k)}$ and $\theta_i^{(k)}$ to all other sensors it is connected to, so as to evaluate their $k+1$ primal, dual, and Lagrangian multipliers. When M is small, this algorithm incurs communication overhead that can be shown to be worse than the communication overhead incurred by centralized estimation methods. However, when M is large, the conventional ADMM algorithm incurs lesser communication overhead than what centralized estimation methods incur, but it still remains practically unattractive due to other weaknesses, detailed later in Section 5.5.

Given the absolute and relative tolerances, ϵ_{abs} and ϵ_{rel}, specified by the SLAs, we define the

primal and dual tolerances, controlling the convergence of the algorithm at iteration k, as

$$\epsilon_i^{pri}(k) = \sqrt{g}\epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max(\|\theta_i^{(k)}\|, \| - z_i^{(k)}\|),$$

and

$$\epsilon_i^{dual}(k) = \sqrt{g}\epsilon_{\text{abs}} + \epsilon_{\text{rel}} \sum_{j \in A} \|\rho_{j,i}\lambda_{j,i}\|.$$

The tolerances, ϵ_i^{pri} and ϵ_i^{dual}, define the stopping criteria of sensor i; i.e., sensor i stops updating θ_i and z_i when

$$\|\theta_i^{(k+1)} - z_i^{(k+1)}\| < \epsilon_i^{pri}(k), \tag{5.11}$$

and

$$\|z_i^{(k+1)} - z_i^{(k)}\| < \epsilon_i^{dual}(k). \tag{5.12}$$

The stopping criteria of RADE are different from those of the conventional ADMM. Unlike the conventional ADMM where all sensors shall stop computations all at the same time using a common stopping criteria and common primal and dual tolerances, the stopping criteria (Eq. (5.11) and Eq. (5.12)) of RADE allow a sensor i to stop its computations asynchronously and independently from other sensors. However, these criteria are not enough to ensure asynchronous implementation, as synchronization is still required for dual and primal variable updates at iteration $k+1$ due to their dependencies on k.

To ensure full asynchronous implementation, we use the doubly stochastic transition matrix, $T \in \mathbf{R}^{g \times g}$, where $T_{i,j}$ is the probability that a sensor i contacts another sensor j at any iteration, for deciding the communications among sensors. We can have

$$T_{i,j} = \begin{cases} \dfrac{1}{d_i' + 1} & \text{if } i = j \text{ or } (i,j) \in P, \\ 0 & \text{otherwise,} \end{cases}$$

where $d_i' = |\{j \in A : (i,j) \in P\}|$ is the degree of the virtual sensor, in Υ, that i virtualizes. At iteration $k+1$, sensor i may need to contact only one sensor j, unless both of i's stopping criteria, Eq. (5.11) and Eq. (5.12), are already satisfied. Whereas sensor j can be contacted by more than one sensor if j is not contacting any other sensor, even when both of j's stopping criteria are satisfied.

Upon contacting j, sensor i pushes $\theta_i^{(k)}$ to j only if i's primal stopping condition is not

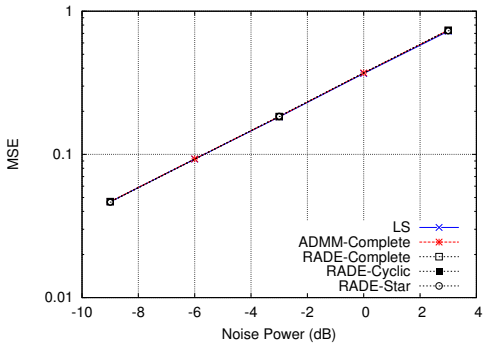


Figure 5.5: MSE of RADE compared to those achieved under ADMM and LS at different noise power and for different virtual sensor network topologies, $g = 10$.

satisfied and pushes $z_i^{(k)}$ to j only if i's dual stopping condition is not satisfied. Also, i pulls $\theta_j^{(k)}$ from j only if j's primal stopping condition is not satisfied and pulls $z_j^{(k)}$ only if j's dual stopping condition is not satisfied. Finally, both i and j update their $k+1$ variables using the most recent values they received from other sensors.

Mean square error and convergence. The asynchronousness and randomization design of RADE do not impact the MSE achieved by RADE when compared to ADMM. This is explained as follows. In both ADMM and RADE, the number of necessary dual and primal variables updates that are needed until convergence remains unchanged, so that convergence to the same estimate is guaranteed in both algorithms. Figure 5.5 shows the MSE achievable under both RADE and ADMM when compared to LS under each of the three studied sensor network topologies: complete, star, and cycle. These results show the optimality of RADE that we intuitively discussed. All approaches have the same accuracy. But of course each of them does so at a different performance cost, as will be discussed later.

On the other hand, RADE exhibits a linear convergence rate ($O(1/k)$), similar to what the conventional ADMM does. Figure 5.6 shows the number of time steps required for both RADE and ADMM to converge under different relative tolerance parameters, ϵ_{rel}. RADE convergence tends to be more restricted by the randomization nature of the algorithm for smaller values of ϵ_{rel}, which can be seen by the increasing number of steps as g increases if $\epsilon_{\text{rel}} = 10^{-2}$. ADMM generally requires a lesser number of steps to converge by relaxing the consensus constraint (through reducing ϵ_{rel}). However, as will be seen in the numerical results section later, this increase in the number of convergence steps is acceptable when considering the amount of communication overhead that the algorithm saves.

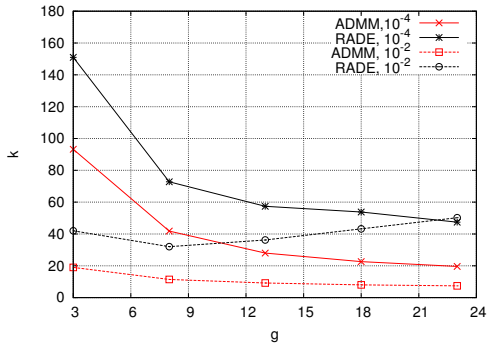


Figure 5.6: Number of time steps (k) needed until convergence of RADE when compared to ADMM for complete topology under different relative tolerance values ϵ_{rel}.

5.5 Numerical Results

In this section, we evaluate the performance of the proposed RADV and RADE algorithms through simulations. In our simulations, $\mathcal{G}$ and Υ, are generated using the parameters summarized in Table 5.1. We evaluate the performance for a complete, cyclic, or star virtual sensor network topology, with a randomly chosen central location, c. We consider receiving and servicing only one virtual sensing task request at a time. The $\mathcal{G}$ topology and connectivity can change rapidly. For a single Υ, we assume that the network change rate is slow enough for the completion of the sensor search and virtualization phases. The absolute and relative tolerances, ϵ_{abs} and ϵ_{rel}, are set to 10^{-4} unless specified otherwise.

Table 5.1: Simulation Parameters

Parameter	g	r	$C(i)$	$R(j)$	δ	$\bar{h}$
Value	10	0.1	$\sim U(50, 100)$	$\sim U(25, 50)$	0.2	20

Figure 5.7 shows the rejection rate encountered with different Υ topologies and n values. As we only consider one single request at a time, the results shown in this figure reflect mainly the impact of the virtual sensor network topology, the number of sensors n, and the simulations parameters given in Table 5.1 on the rejection rate. The denser the network of IoT devices is, the lower the rejection rate, implying that the cloud is capable of granting higher number of requests.

One way of assessing the effectiveness of the virtualization algorithm is by measuring the difference between the total virtualization benefit given in (5.4) and the cost associated with

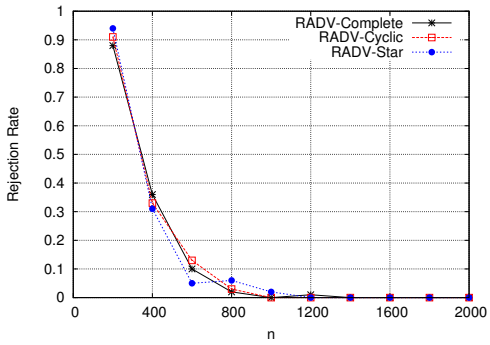


Figure 5.7: Rejection rate encountered at different n.

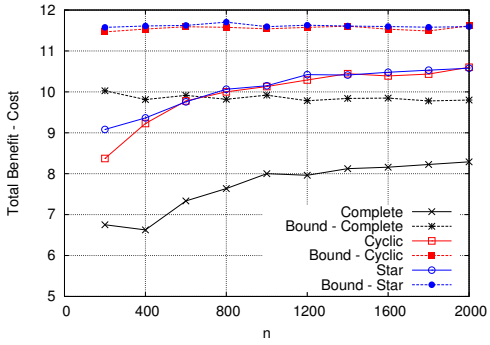


Figure 5.8: Virtualization cost of RADV when compared to the upper bound under different topologies.

the sensor virtualization introduced in Section 5.3.2. For a given number of virtual sensors, the cost is mainly determined by the choice of the topology (star topology has the lowest cost and complete topology has the highest one). For a given topology, the total benefit is maximized when each virtual sensor is assigned to the IoT devices with the maximum capacity and each virtual link is mapped to exactly one physical link. We refer to this maximized benefit as the upper bound.

In Figure 5.8, we evaluate the virtualization effectiveness achieved by RADV under different virtual topologies. As the network gets denser, RADV achieves a Total Benefit $-$ Cost that is very close to the upper bound. Since the lowest possible virtualization cost is with star or cyclic topologies, it is desired by the cloud to arrange each virtual sensing task in a star or a cyclic topology. This observation holds true for a more general topologies. On the other hand, convergence and communication overhead of the distributed estimation is also impacted by the cloud agent's choice of the virtual topology. This creates a design trade-off, as we will see in the

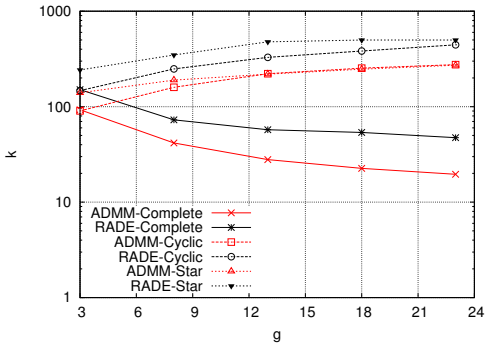


Figure 5.9: Number of time steps until convergence of RADE when compared to ADMM under different topologies.

next two paragraphs.

Figure 5.9 shows the impact of the virtual topology choice on the convergence performance of RADE when compared to ADMM. If g is small (three to eight), the impact of the virtual topology on convergence of RADE and ADMM is minimal. This is because the degree of parallelism (number of virtual sensors active at the same time) is restricted by the small number of virtual sensors g. In such a scenario, it is convenient for the cloud agent to always arrange the virtual sensors in a star topology. However, as g increases, the impact of choice of the virtual topology becomes significant as the degree of parallelism is higher in a complete topology, enabling RADE to converge much faster as g gets larger. This convergence becomes slower with star and cyclic topologies. This is because in star and cyclic topologies, only few sensors are active at a time, making RADE and ADMM converge in a number of steps comparable to that of the ADMM's sequential implementation. In this later scenario, the cloud agent shall arrange the virtual sensors as a complete topology unless the SLA permits slower convergence.

Moreover, RADE converges in a higher number of steps when compared to the conventional ADMM. This is because in ADMM, all sensors are active at each time, and a sensor exchanges its updated variables with all of its neighbors, whereas in RADE, only disjoint sensor pairs are active at a time and variables are updated only between pairs of sensors. Nevertheless, we argue that this loss in speed of convergence for RADE is marginal when compared to the significant savings in communication overhead.

Figure 5.10 shows the total number of $O(N)$ sized messages exchanged during estimation when comparing RADE, ADMM, and LS for $M = 100$. The number of messages exchanged by RADE is at least an order of magnitude less than the number of messages generated under

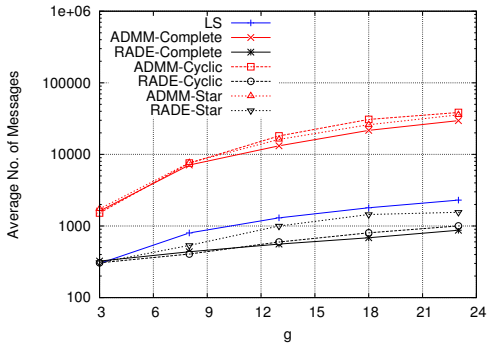


Figure 5.10: Communication overhead when comparing RADE to ADMM and LS under different g values.

ADMM. Also the communication overhead of RADE is less than the centralized LS especially as M becomes large. This savings in communication overhead is attributed to the asynchronous design of RADE in which messages among sensors are only exchanged if new values of a primal or dual variables are changed away from their specified tolerances.

5.6 Related Work

5.6.1 Network Virtualization

Network virtualization techniques proposed in the past decade consist mainly of virtual network embedding algorithms, which instantiate virtual networks on substrate infrastructures [72, 29]. Most of these virtual network embedding algorithms are centralized (e.g. [8, 49]) due to the ease of deployment of centralized approaches in cloud platforms where the cloud provider desires to have full control on the physical network resources. Distributed network virtualization techniques are suited for Cloud of Things, given the size of the network, and are also proposed for applications in resource allocation in distributed clouds, wireless sensor network virtualization, and cloud network as a service [64, 16].

Beck et al. propose a hierarchical partitioning of any substrate network [159] and solve the network virtualization (virtual network embedding) problem on the scale of smaller partitions by delegating the problem to delegate nodes. In our context, their algorithm can progress in four steps: (1) partitioning the network of IoT devices, (2) assigning delegation nodes among the IoT devices that actually perform the network virtualization, (3) setting distributed lock trees to avoid

inconsistent solutions among different delegation nodes, and (4) embedding the virtual sensor network within the scope of the delegation nodes. Several assumptions in this work prevent this method applicability in Cloud of Things. The authors require that a centralized node manage the IoT network topology to perform the partitioning. The centralized node partitions the IoT devices in groups that are highly interconnected. This requirement is very hard to achieve, if not impossible, in an Internet-scale network of IoT devices, not only because of the size of the network but also due to its highly dynamic nature that prevents tracking the states of the devices and their connectivity. Moreover, to apply the same method in cloud of things, the delegation nodes of Beck's method needs to learn a significant amount of information about the IoT devices in their neighborhood, which creates significant practical problems such as: timely information retrieval, privacy concern, and computation power.

Esposito and Ibrahim propose to model the network virtualization as a network utility maximization problem, where the utility is a general function that is measured on each hosting node (i.e. IoT device) [66]. They solve the problem distributivity using primal dual decomposition. To employ their algorithm in our Cloud of Things context, the non-convex constraints of the network virtualization problem need be relaxed. Such a relaxation is known to have a negative impact on the accuracy of the solution and may lead to false decisions [8]. Moreover, this method requires the definition of a single utility for the IoT device that shall not reflect the actual embedding cost due to network conditions, and only captures network conditions seen locally by the IoT device. Finally, solving the problem using primal-dual decomposition requires cooperation between *all* IoT devices in the network, which prevents this method from scaling on large-sized networks without adequate network partitioning as approached by Beck et al. in [159].

In [67, 65], the authors propose a distributed virtual node mapping algorithm using consensus-auction. To apply it in our context, a utility function needs to be defined for each IoT device that is based on the devices local capacity and link bandwidth (known only to the device). The IoT devices then start bidding for virtual sensors to maximize its utility. Although the node mapping is very close to optimal, the overall result of the algorithm is not necessarily optimal since the link mapping uses only first hop link information of the substrate nodes. The authors also assume that paths are computed using $3-$shortest paths, which overlook other path diversity that can be found in a large substrate network.

The earlier methods are designed for federated data centers, and cannot be applied to the Cloud of Things context. In our approach, each device with a non-empty virtual mapping domain first solves the problem within a radius of $\bar{h}$ of the subgraph centered at the IoT device.

Then, the agent selects the best solution by reducing different solutions found by the devices to a single best solution. Unlike the work of Beck et al. in [159], partitioning and delegation are an implicit process of the algorithm performed through the gossip policy used during the sensing resource discovery phase where only the IoT devices with a non-empty virtual mapping domain start to execute the next steps of the algorithm. This approach does not require updated knowledge about all the IoT devices and the topology of the network. The IoT devices perform virtualization by exchanging minimal information to construct their local benefit matrices and solving an assignment problem. Unlike [66, 67, 65], a benefit matrix constructed by a device captures all network information within a radius of $\bar{h}$ around that device, thereby improving the obtained solution closeness to optimal without overlooking important network characteristics. And unlike [66, 159], we do not restrict the link mapping to use $k-$shortest paths, thereby allowing the use of diverse paths in the network constructed during the randomized gossip policy used during the benefit matrices construction.

5.6.2 Distributed Estimation

Distributed parameter estimation approaches have been proposed in [131, 157, 79]. Estimation can for e.g. be carried out by first computing a local estimate at each virtual sensor and then perform a distributed weighted average of the local estimates [157]. This approach results in an ML estimate, but does not limit/bound the variation between mean square errors of local estimates. More recently, Paul et al. [131] propose a distributed estimation algorithm based on ADMM. Although this approach results in an optimal mean square error when compared to LS, it exhibits a significant in-network communication overhead that requires even more messages to be exchanged among sensors than that exchanged in the centralized LS. One approach also proposed in [131] to overcome this problem is to approximate the computation of primal and dual variables at each step of the algorithm by using predictions and earlier versions of these variables instead of sharing them at each iteration which marginally reduces the communication overhead. In addition to the increased communication overhead, conventional ADMM requires synchronous operation of the sensors. This is very challenging from a practical viewpoint, and does not scale well especially when applied in the IoT context. It has been shown recently that an asynchronous implementation of ADMM has $O(1/k)$ convergence [168]. Our proposed estimation algorithm is both asynchronous and distributed, and reduces communication overhead significantly when compared to the conventional ADMM approach [131].

5.7 Conclusion and Discussion

We have shown the potential of Cloud of Things to scale cloud computing vertically by exploiting sensing resources of IoT devices to provide Sensing as a Service. We have proposed a global architecture that scales Cloud of Things horizontally by employing edge computing platforms in a new role as cloud agents that discover and virtualize sensing resources of IoT devices. We have described cloud agents technical challenges and design objectives for sensing resources discovery and virtualization that can dispatch offering virtual sensor networks deployed on IoT devices to cloud users with in-network processing capabilities. We gave a taxonomy of the potential sensing tasks, their applications, and there challenges. We have proposed our sensing resource discovery solution based on a gossip policy to discover sensing resrouces as fast as possible and RADV: our virtualization solution. We have shown through analysis and simulations the potential of RADV to achieve reduced communication overhead, low complexity, and closeness to optimal such that RADV employs minimal physical resources in devices virtualization with maximal benefit. We also proposed RADE for distributed consensus estimation as we believe it is one major sensing task in Sensing as a Service. Using simulation, we show that RADE reduces the communication overhead significantly without compromising the estimation error when compared to the traditional ADMM algorithm. We also show that the convergence time of our proposed algorithms maintain linear convergence behavior, as in the case of conventional ADMM.

Chapter 6: Flocking Virtual Machines in Quest of Responsive IoT Cloud Services

6.1 Introduction

Live Virtual Machine (VM) migration is essential for improving the responsiveness of cloud services. VMs, hosted in Edge Cloud (EC) platforms, install resource-rich cloud services close to data sources to realize Internet of Things (IoT) applications. Such VMs provide, for example, real-time video analytics services from cameras to mobile applications. Hosting VMs near the edge can subdue the end-to-end services latency from hundreds to tens of milliseconds compared to hosting VMs in conventional cloud platforms [113, 145]. Existing resource provisioning algorithms migrate VMs from one EC to the other in response to devices mobility and services computational capacity requirements. VM migration ensures minimal average latency between mobile devices and the EC [163, 98]. Migration can achieve other goals such as load balancing, efficient service chaining and orchestration, infrastructure cost minimization, efficient content delivery, and energy efficiency, to name a few [48].

However, migrating VMs in response to the mobility of IoT devices is a limited decision given the diverse IoT use cases. Such a mobility-triggered migration assumes a constraining IoT use case in which a cloud service, installed in VMs, communicates with one - and only one - IoT device [145]. Consider EC services for smart glasses as an example. Migrating a VM, which hosts video processing cloud services of a Google glass, minimizes the video processing latency as the glass/user moves, a goal that is only reasonable for the Google Glass use case, in which the Google Glass is the singleton client that uses the EC video processing services [145, 163]. However, in general IoT use cases, several VMs, very likely to be deployed in different ECs, are needed to execute distributed algorithms (e.g. computer vision feature extraction, consensus, and aggregation [161]) and require intensive communication among the VMs and/or the devices. Distributed computer vision, for example, enables distributed context-aware applications such as autonomous vehicles and intelligent traffic systems [60]. Predominantly for modern IoT applications, developers implement analytics via large-scale distributed algorithms executed by VMs in geographically distributed and heterogeneous cloud platforms [90, 10, 8].

In this paper, we propose Flock; a simple protocol by which VMs autonomously migrate between heterogeneous cloud platforms to minimize their weighted latency measured with end-user applications, IoT devices, and other peer-VMs. In Flock, a VM uses only local latency measurements, processing latency of its hosting cloud, and local information provided by its hosting cloud about other clouds which the VM can migrate to. A VM greedily migrates to a cloud platform that reduces its regularized weighted latency. We prove, using game-theory, that Flock converges to a Nash Equilibrium (NE) and its Price of Anarchy (PoA) is $(1+\epsilon)$ given the properties of our proposed social value function. We discuss how Flock serves a generic goal that we can redesign using simple tweaks to achieve other objectives such as load balancing and energy minimization. Our design allows VMs to imitate a *flocking-like* behavior in birds comprising separation, alignment, and cohesion rules.

6.2 System Model and Objective

We consider a network of VMs modeled as a graph $G = (V, P)$, where V denotes the set of n VMs and P denotes the set of VM pairs such that $p = (i, j) \in P$ if the i-th and j-th VMs communicate with each other. Let $d_{ij} \in \mathbb{R}^+$ denote the traffic demand between VMs i and j and assume that $d_{ij} = d_{ji}$. We also consider a set, A, of m clouds (i.e. ECs, or conventional clouds) that communicate over the Internet. A VM i autonomously chooses its hosting cloud.

Let $x_i \in A$ denote the cloud that hosts i and let $l(x_i, x_j) > 0$ be the average latency between i and j if they are hosted at x_i and x_j respectively (Note: if i and j are hosted at the same cloud $x_i = x_j$). We assume that l is reciprocal and monotonic. Therefore, $l(x_i, x_j) = l(x_j, x_i)$ and there is an entirely nondecreasing order of $A \rightarrow A'$ such that for any consecutive $x_i, x_i' \in A'$, $l(x_i, x_j) \leq l(x_i', x_j)$. The reciprocity condition ensures that measured latencies are aligned with peer-VMs and imitates the alignment rule in bird flocking. We model $l(x_i, x_j) = \tau(x_i, x_j) + \rho(x_i) + \rho(x_j)$, where $\tau(x_i, x_j)$ is the average packet latency between x_i and x_j, and $\tau(x_i, x_j) = \tau(x_j, x_i)$. The quantity $\rho(x)$ is the average processing delay of x modeled as: $\rho(x) = \delta \sum_{i \in V : x_i = x} \sum_{j \in V} d_{ij} / (\gamma(x) - \sum_{i \in V : x_i = x} \sum_{j \in V} d_{ij})$, where δ is an arbitrary delay constant and $\gamma(x)$ denotes the capacity of x to handle all demanded traffic of its hosted VMs. An increased value of $\rho(x_i)$ signals the VM i that it is crowding with other VMs in the same cloud, imitating the separation rule in bird flocking.

A VM i evaluates its weighted latency with its peers if hosted at x as

$$u_i(x) = \sum_{j \in V} d_{ij} l(x, x_j) / \sum_{j \in V} d_{ij}. \tag{6.1}$$

Our objective is to design an autonomous VM migration protocol that converges to an outcome $\sigma = (x_1, x_2, \ldots, x_n)$ that minimizes the sum of weighted latency $\sum_{i \in V} u_i(x_i)$. That is to say, σ maximizes the responsiveness of all VMs given their peer-to-peer communication pattern (i.e. connectivity and demands).

6.3 Flock: Autonomous VM Migration Protocol

We design a simple protocol that allows a VM to autonomously decide its hosting cloud among a set $A_i \subseteq A$ of available clouds by relying on only local regularized latency measurements. We call A_i the strategy set of i. Every cloud x evaluates its current weight $w_x = \sum_{i:x_i=x} u_i(x)$ and advertises a monotonic non-negative regularization function $f(w_x) : \mathbb{R}^+ \to \mathbb{R}^+$, such that $\alpha < f(w_x) < 1$ for $\alpha > 0$, to all the VMs that are hosted at x.

Since each VM is hosted by a single cloud and all VMs have access to the same strategy set, the VM migration problem is modelled as a singleton symmetric weighted congestion game with the objective of minimizing the social cost $C(\sigma) = \sum_{x \in A} w_x f(w_x)$. If $f(w_x)$ is approximately 1, this game model is approximately equivalent to minimizing $\sum_{i \in V} u_i(x_i)$. Throughout, we use $x \xrightarrow{i} y$ to mean that a VM i migrates from cloud x to cloud y. Letting $\eta \leq 1$ be a design threshold, we propose the following migration protocol:

Flock: Autonomous VM migration protocol.

Initialization: Each VM $i \in V$ runs at a cloud $x \in A$.

Ensure: A Nash equilibrium outcome σ.

During round k, do in parallel $\forall i \in V$

Greedy migration process imitating cohesion in flocking:

i solicits its current strategy A_i from x.

i randomly selects a target cloud $y \in A_i$.

if $u_i(y) f(w_y + u_i(y)) \leq \eta u_i(x) f(w_x - u_i(x))$ **then**

  $x \xrightarrow{i} y$.

end if

We begin by proving that Flock converges to a Nash equilibrium, where each VM chooses a single cloud (strategy) and no VM has an incentive (i.e. less average latency) to migrate from its current cloud. Then, we derive an upper bound on Flock's PoA for general regularization functions, f, following a similar approach to [33]. Finally, we propose a regularization function $f(w_x) \approx 1$ that achieves a tight PoA of at most $1 + \epsilon$.

6.3.1 Equilibrium

Convergence to a Nash equilibrium in Flock is non-obvious given the inter-dependency of a VM's average weight with its peers.

Theorem 6.3.1 Flock *converges to a Nash equilibrium outcome.*

Proof *Any step of* Flock *corresponds to choosing a random outcome from a finite number of outcomes. We show that any step of* Flock *reduces the social value $C(\sigma)$ and $C(\sigma)$ is bounded below. We first show that if a VM i migrates from cloud x to cloud y, the increase in y's weight is less than the decrease in x's weight. We then use contraction to show that if a subset of i's peers have a total latency that increased after i migrates, the remaining peers must have a total latency that decreased by a greater value. Let σ^k and w_x^k denote the game outcome and the weight of x at round k respectively, and let $\Delta w_x = w_x^{k+1} - w_x^k$. If $x \xrightarrow{i} y$, then*

$$u_i(y) f(w_y + u_i(y)) \leq \eta u_i(x) f(w_x - u_i(x)).$$

As $\alpha < f < 1$, we have two extreme cases: $u_i(y)\alpha \leq \eta u_i(x)$, or $u_i(y) \leq \eta\alpha u_i(x)$, which implies that:

$$u_i(y) \leq \frac{\eta}{\alpha} u_i(x). \tag{6.2}$$

Because of the migration: $\Delta w_x < 0$ and $\Delta w_y > 0$, but $|\Delta w_x| \geq u_i(x)$, and $|\Delta w_y| \leq u_i(y)$, otherwise i would not migrate, then

$$|\Delta w_y| \leq |\Delta w_x| . \tag{6.3}$$

Let $z_i = u_i(x_i) f(w_{x_i})$ denote the regularized weighted latency and let z_i^t denote its value at round k. After the migration, i's peers split into two subsets: $V_{inc} = \{j \in V : z_j^{k+1} > z_j^k\}$, and $V_{dec} = \{j \in V : z_j^{k+1} < z_j^k\}$. Assume after the migration step that $\sum_{j \in V_{inc}} z_j > \sum_{k \in V_{dec}} z_k$. For

$f \approx 1$, $\sum_{j \in V_{inc}} u_j(x_j) > \sum_{k \in V_{dec}} u_k(x_k)$. *Substitute with the weighted latency value from* (6.1) *and since the demands (i.e.* d_{ij}*) remain unchanged,* $\sum_{j \in V_{inc}} l(x_j, y) > \sum_{k \in V_{dec}} l(x_k, y)$. *By the reciprocity of* l, $\sum_{j \in V_{inc}} l(y, x_j) > \sum_{k \in V_{dec}} l(y, x_k)$, *and* $u_i(y) \geq u_i(x)$, *which contradicts* (6.2). *By this contradiction and from* (6.3), *the social value* $C(\sigma^{k+1}) \leq C(\sigma^k)$. *Since* n *and* m *are finite, the number of all possible outcomes is finite. An outcome after a VM migration,* σ^{k+1}, *corresponds to randomly choosing an outcome from the finite outcome space which reduces the social value. Since* $C(\sigma) > 0$, *then* Flock *must converge to a Nash equilibrium.* ■

6.3.2 Price of Anarchy

We first give a generic upper bound of the PoA, then we propose our regularization function that tightens the PoA.

Lemma 6.3.2 *The social value of* Flock *has a perfect PoA at most* $\lambda/(1-\varepsilon)$ *if for* $\varepsilon < 1$ *and* $\lambda > 1 - \varepsilon$ *the regularization function satisfies* $w^* f(w + w^*) \leq \lambda w^* f(w^*) + \varepsilon w f(w)$, *where* $w \geq 0$ *and* $w^* > 0$.

Proof *Let* σ *denote a Nash outcome and* σ^* *denote any alternative outcome. Also let* w_x *and* w_x^* *denote the weight on* x *in outcomes* σ *and* σ^* *respectively. Similarly, let* x_i *and* x_i^* *denote* i*'s strategy (hosting cloud) in* σ *and* σ^* *respectively. By definition of a Nash outcome,* $\forall i,\ u_i(x_i) f(w_{x_i}) \leq u_i(x_i^*) f(w_{x_i^*})$. *Summing over all VMs we get,* $C(\sigma) = \sum_{i \in V} u_i(x_i) f(w_{x_i}) \leq \sum_{i \in V} u_i(x_i^*) f(w_{x_i^*})$ *However,*

$$\begin{aligned}
\sum_{i \in V} u_i(x_i^*) f(w_{x_i^*}) &\leq \sum_{x \in A} \sum_{i : x_i = x} u_i(x) f(w_x + u_i(x)) \\
&\leq \sum_{x \in A} w_x^* f(w_x + w_x^*) \\
&\leq \sum_{x \in A} \lambda w_x^* f(w_x^*) + \varepsilon w_x f(w_x) \\
&= \lambda C(\sigma^*) + \varepsilon C(\sigma).
\end{aligned}$$

Then, $C(\sigma) \leq \lambda C(\sigma^*) + \varepsilon C(\sigma)$. *Rearranging we get, POA* $= C(\sigma)/C(\sigma^*) \leq \lambda/(1-\varepsilon)$. ■

Theorem 6.3.3 *The regularization function* $f(w) = exp(-1/(w+a))$ *tightens the PoA to* $1+\epsilon$ *for a sufficiently large constant* a *and reduces the game to the original VM migration problem, i.e. minimizing* $\sum_i u_i(x_i)$.

Proof *For*

$$\lambda \leq \frac{f(w_{max}+w_{min})}{f(w_{min})}\left(1-\varepsilon\frac{w_{max}f(w_{max})}{w_{min}f(w_{max}+w_{min})}\right),$$

$f(w)=\exp(-1/(w+a))$ *satisfies the condition* $w^*f(w+w^*)\leq\lambda w^*f(w^*)+\varepsilon wf(w)$ *(verify by inspection). For an infinitesimally small value of* ε*, we can choose* a *such that* $\lambda=1+\epsilon$*, hence the POA* $\leq 1+\epsilon$*. If* a *is sufficiently large* $f(w_x)\approx 1$*, hence* $C(\sigma)\approx\sum_i u_i(x_i)$*.* ■

6.4 Experimental Results and Discussion

We simulate Flock using SimPy (see [121] with several simulation rounds. In each round we describe the clouds as a complete graph data structure with inter-cloud latency modeled as $\tau \sim \text{Uniform}(10,100)$ and cloud capacity as $\gamma \sim \text{Uniform}(50,100)$. We simulate the peer-to-peer relations of a VMs as a binomial graph with $d \sim \text{Uniform}(1,10)$. Each simulated VM asynchronously runs Flock.

Figure 6.1 shows the average Flock rounds, k, required to converge to a Nash equilibrium at 95%-confidence interval with 0.1 error in simulations rounds. Although in the worst case $k=O(n\log(nf_{max}))$, where f_{max} is the maximum value of the regularization function f [51], Figure 6.1 suggests that Flock scales better than $O(n)$ on average. The proof of the average convergence time is complex and depends on properties of the social value $C(\sigma)$ and other parameters. We leave this proof for future work. Figure 6.2 shows the PoA statistics of Flock with

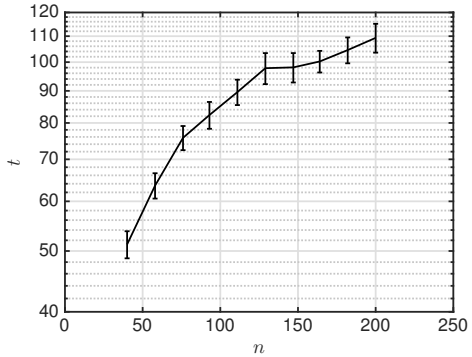


Figure 6.1: Convergence of Flock: $m=37$, $\tau\sim\text{Uniform}(10,100)$, $d\sim\text{Uniform}(1,10)$, $a=9$, $\gamma\sim\text{Uniform}(50,100)$, and $\eta=0.9$.

different η. We implemented the optimal solution as brute-force in simulations that evaluates

the minimum social value among all possible VM to cloud assignments. As $\eta \approx 1$, the maximum PoA achieved matched the theoretical value of 1.21. In practice, it is desirable to keep $\eta < 1$ (e.g. $\eta = 0.7$) to avoid migrations with insignificant improvements and alternating migrations between clouds. Although the worst case PoA for $\eta = 0.7$ approaches 4.5, the average PoA is acceptable in practice.

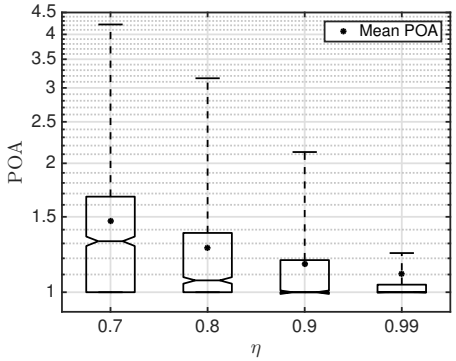


Figure 6.2: Price of Anarchy statistics for $m = 5$, $n = 8$, $\tau \sim$ Uniform$(10, 100)$, $d \sim$ Uniform$(1, 10)$, $\gamma \sim$ Uniform$(50, 100)$, and $a = 9$.

Figure 6.3 and Figure 6.4 show the fairness of Flock as it maintains a homogenous PoA for individual VM. In Figure(a), we evaluate the minimum utility value of each VM and compare it to the utility value achieved by Flock. As shown, Flock maintains a homogenous PoA accross individual VMs. In Figure(b), we show the variance of weighted latency accross VMs compared to the mean weighted latency (Coefficient of Variance). As the Coefficient of Variance is very low and the PoA is homogenous for individual VMs, it is unlikely that a single VM starves with a less than close-to-optimal latency.

6.4.1 Special Cases: Load-balancing and Energy Efficiency

Flock can be redesigned with simple tweaks to suit other common cloud resource provisioning problems. We consider load-balancing and energy efficiency as special cases.

In load balancing, we seek an outcome σ such that the total load is distributed proportionally to the clouds' capacities. We first force the latency $\tau = 0$ between any cloud pairs $x, y \in A$. This is equivalent to marginalizing the effect of latency on the social value. We also force the VMs to ignore their peer relationships (i.e. $P = \emptyset$). The problem transforms immediately to

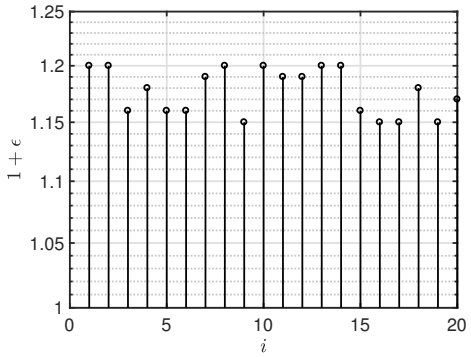


Figure 6.3: Faireness measures - approximation ratio of individual virtual machine.

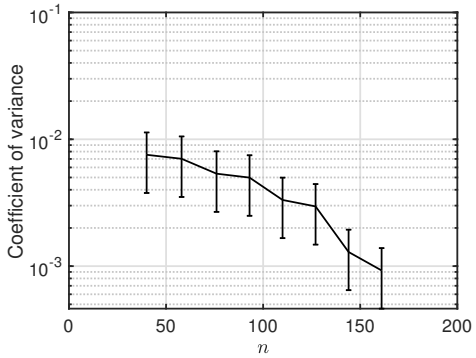


Figure 6.4: Faireness measures - Cofficent of Variance. The shown values are at ± 0.02 error with 95% confidence interval..

minimizing the sum of cloud utilization. A VM in this setup greedily migrates to the least loaded cloud.

Figure 6.5 shows the ideal mean utilization for a load-balanced system ('+') and the mean utilization using Flock ('*'). The box-plot also gives a summary statistics of the extreme value and the 75%-percentile samples. Flock performs very well as a load balancing protocol.

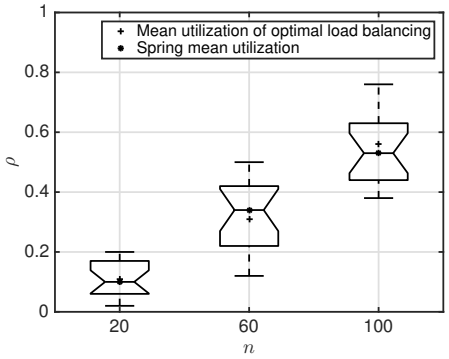


Figure 6.5: Flock usage for load balancing ($P = \emptyset$, $\tau = 0$) for $m = 20$, $d \sim$ Uniform$(1, 10)$, and $\gamma \sim$ Uniform$(50, 100)$.

The energy-efficiency goal seeks an outcome σ in which the maximum number of clouds are idle (i.e. with $\rho = 0$). We (virtually) force each VM to be connected to all other VMs (i.e. $P = V \times V$). We also force τ to be a very large value (i.e. $\tau \to \infty$, and $d = 1$). With this tweak, a VM favors a cloud that hosts the largest number of VMs and with enough capacity ρ. Figure 6.6 shows the ideal number of idle clouds that minimizes the energy consumption under a certain load (upper bound) and the number of idle clouds using Flock under the same load.

6.4.2 Impact of system dynamics

We now study the impact of various system dynamics on Flock's convergence. If we considered any two Flock steps at discrete times $(k-1)$ and k, the values of the VM's utilities, u_i, evolve randomly. Both external factors and VM migrations can influence this evolution and introduce difficulty in the analysis of Flock convergence. The simplest approach to deal with such dynamics is to assume that changes in latencies, EC capacities, and any other state are much slower than Flock convergence. This means that changes in u_i due to external factors such as link bandwidths or EC failures varies slowly. It also means that Flock steps are very rapid and

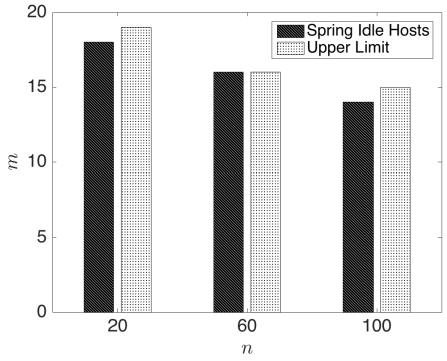


Figure 6.6: Flock usage for energy efficiency ($P = V \times V$, $\tau \to \infty$, $d = 1$) for $m = 20$, and $\gamma \sim$ Uniform$(50, 100)$.

Flock measures only average values of l or any other system state. Fortunately, we have shown that Flock converges in a finite number of steps. However, we cannot guarantee that this number of steps are taken much faster than changes in any system state. We will follow the approach in [106] to show that despite that Flock may take slow migration decisions, its convergence still holds if we modeled the network dynamics (e.g. l) as a continuous-time Markov process that has values that are generated by the migration steps of Flock.

Consider modeling the utility of a VM as a general discrete-time stochastic process such that: $\forall k \in \mathbb{Z}, u_{i,k+1} = u_{i,k} + b_k \omega_k$, where $u_{i,k+1}$ is the utility value of VM i at the discrete-time k, b_k is a design parameter, and $\omega_{i,k}$ is a random variable that takes values according to both external factors that impact u and Flock migrations prior to time k. For all $k \in \mathbb{Z}$, the values of ω are determined by $\omega_{i,k} = h(u_{i,k}, Y_{i,k})$ and $Y_{i,k} = \int_k^{k+1} f(m_i(t))dt$, where $m_i(t)$ is a continuous-time Markov process of generator $G^{u_{i,k}}$ and with values in a finite set $\mathcal{M}$, $f : \mathcal{M} \leftarrow \mathbb{R}^K$ is an arbitrary mapping, and $h : \mathbb{R}^L \times \mathbb{R}^K \leftarrow \mathbb{R}^L$ is a bounded continuous Lipschitz function in u and is uniformly distributed in Y. The Markov process $m_i(t)$ is irreducible and ergodic where $m_i(k) = \lim_{t \to k, t<k} m_i(t)$. We previously assumed that $u_{i,k}$ is bounded, which is true given the bounded latencies, VM demands, and EC capacities. We also can enforce such bounded values of u by projecting $u_{i,k}$ to a finite subset of $\mathbb{R}^L$. Finally, we assume that b_k is positive and decreasing in k, such that b_k is constant as $k \to \infty$, so that $\sum_k b_k = \infty$ and $\sum_k b_k^2 < \infty$.

By adopting the discrete time factor into the values of $u_{i,k}$, we transfer Flock into a class of stochastic approximation algorithms with controlled continuous-time Markov noise [36]. As b_k is a decreasing step size, the speed of variations in $u_{i,k}$ decreases and vanishes as k increases.

This is equivalent to modeling $m_i(t)$ as a Markov process with a fixed generator (e.g. when τ values are frozen), and converges to an ergodic behavior. Hence, as k increases, Flock uses averaged values of u_i and represents a stable dynamical system (see [36] for formal proofs of the convergence of stochastic approximation algorithms). We call the modified migration protocol, controlled-Flock and is given as:

Controlled-Flock: Autonomous VM migration protocol.

Initialization: Each VM $i \in V$ runs at a cloud $x \in A$.

Ensure: A Nash equilibrium outcome σ.

1: During round k, do in parallel $\forall i \in V$

Greedy migration process imitating cohesion in flocking:

2: i solicits its current strategy A_i from x.

3: i randomly selects a target cloud $y \in A_i$.

4: **if** $u_{i,k}(y)f(w_y + u_{i,k}(y)) \leq \eta u_{i,k}(x)f(w_x - u_{i,k}(x))$ **then**

5: $\quad x \xrightarrow{i} y$.

6: $\quad x = y$

7: **end if**

8: Update $u_{i,k+1}(x) = u_{i,k}(x) + b_k f(w_x)$

In the above algorithm, b_k is a decreasing step size and is left as a design parameter, where step 8 is a standard first-order auto-regressive model. For example $b_k = 1/k$ satisfies the decreasing condition of b_k and that $\sum_k b_k = \infty$ and $\sum_k a_k^2 < \infty$. If b_k is constant, we would obtain weak convergence only. The function $f(w_x)$ serves as the random variable $\omega_{i,k}$ for a VM i. In such case $f(w_x) \equiv h(u_{i,k}, Y_{i,k})$, where $Y_{i,k} = \int_k^{k+1} \sum_{j:x_j=x, j\neq i} u_{j,t}(x)dt$ and $u_{j,t}(x)$ is the continuous-time value of u_j. The Markov process $m_i(t)$ in this case represents the current outcome σ^k and that $f(m_i(t)) = \sum_{j:x_j=x, j\neq i} u_{j,t}(x)$ is an arbitrary mapping that is bounded and continuous Lipschitz function. We can easily verify that the properties of stochastic approximation algorithms with controlled continuous-time Markov noise as described earlier are satisfied and convergence of Controlled-Flock is ensured given the various dynamics.

Figure 6.7 shows the convergence of Controlled-Flock under dynamics of link latencies and cloud capacity that are faster than Flock decisions. We use the same simulation setup in Section 6.4, where Flock decisions are taken every other simulation step t instead of every simulation step. At each simulation step we sample the packet latency for a link $\tau(t)$ from a uniform distribution of an average $\mu_\tau \sim \text{Uniform}(10, 100)$ and a minimum value of zero. Sim-

ilarly we sample the cloud capacity of a cloud $\gamma(t)$ from a uniform distribution of an average $\mu_\gamma \sim \text{Uniform}(50, 100)$ with a minimum value of zero. The zero values of the latency and capacity simulate momentary instability of link and cloud resources. As shown in Figure 6.7, Controlled-Flock maitains the same convergence and PoA properties of Flock. However, the actual number of migrations required to converge to a nash equilibrium is greater. This is expected as Controlled-Flock does not reach steady state until b_k approaches a constant value as $k \to \infty$, such that rapid variations in measurements become indifferent to migration decisions.

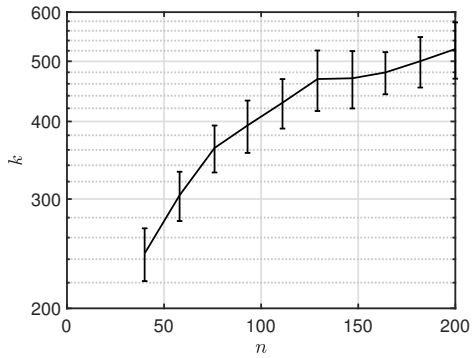


Figure 6.7: Convergence of Controlled-Flock: $m = 37$, $\tau \sim \text{Uniform}(10, 100)$, $d \sim \text{Uniform}(1, 10)$, $a = 9$, $\gamma \sim \text{Uniform}(50, 100)$, and $\eta = 0.9$.

6.4.3 Migration cost

Migration of a VM can be too costly to perform very frequently [164]. It can also degrade performance by introducing migration noise [103]. Depending on the implementation details of a VM and the workload it serves, migration can involve transferring a large volume of data between ECs, which introduces a significant network overhead. For some cloud services, frequent migration can cause intolerable services interruption, as the "down-time" due to a migration can range from few milliseconds to seconds [55]. Accounting for the migration cost in Flock can be desired for several applications. We incorporate the migration cost in Flock utility values by two methods.

First, we include the migration cost as an external dynamics in the VMs' utilities and use Controlled-Flock to minimize the utility, which also includes the average migration cost. Let $g_i(k) \in \{0, 1\}$ indicate whether i migrated at time k or not. Also let $R_i(k+1) = \beta_i R_i(k) + (1 -$

$\beta_i)g_i(k)$ denote the average forgetting value of the migrations of VM i, where $0 \leq \beta_i \leq 1$ is a VM specific parameter that reflects the impact of frequent migrations on service disruptions. The migration cost, C_i, is an increasing function of $R_i(k)$, $C_i(R_i(k))$. To incorporate the migration cost, we redefine u_i for the target cloud in Flock as

$$u_i(y) = \sum_{j \in V} d_{ij} l'(y, x_j) / \sum_{j \in V} d_{ij},$$

where $l'(y, x_j) = l(y, x_j) + C_i(R_i(k)) + C_j(R_j(k))$ and $u_i(x)$ (i.e. utility for current cloud) is memorized. This is equivalent to penalizing the latencies values measured with an additional dynamic that accounts for the estimated average number of migrations. As R_i is a function in prior migration, the C_i value increases as i migrates more frequently and vanishes over time as i pauses migrations. This tweak perceives the numerical properties of both l and u, hence maintains the convergence and PoA results of Controlled-Flock.

Although the first approach minimizes the average migration cost as $k \to \infty$, it requires exchanging the estimated migration cost between each $i, j \in E$ to maintain the reciprocity condition of l. This rapid message exchange introduces a significant communication overhead. Moreover, incorporating the average migration cost in u_i only has a long-term benefit on the system as k grows and it can causes undesirable short-term service disruption for some applications. For such interruption-sensitive applications, it may be beneficial to delay the migration decisions using the η parameter instead of incorporating the migration cost into the utility function. In this second approach each VM evaluates its own η_i accounting for the estimated number of migration as: $\eta_i(k) = \frac{1}{\exp(R(k))}$. As the VM, i, performs less migrations over time, $\eta_i(k) \to 1$ and i migrates for any slight improvement in its utility. Whereas if i performs frequent migrations, $\eta_i(k) \to 0.36$, where i only migrates if the migration decision brings a significant improvement to u_i. The drawback of this approach can be seen in Figure 6.2, where we sacrifice the tightness of the PoA as $\eta < 1$.

Migration cost of an individual VM can also be reduced in practice if the VM workload does not require persistent state maintenance. For example, if a VM is running device's cloning function as in [54], it can be sufficient to only migrate device's meta-data (few kilobytes) and use it to start a new cloning VM at the target cloud. We apply this trick in developing a message brokering system (see [7]) to minimize the messaging latency for IoT devices. For advanced workload types, systems such as SonicMigration and adaptive pre-paging ([103, 88]) can be

used to minimize the VM migration overhead by examining the memory pages in a fine-grain manner and transfer only memory pages that are necessary for an application.

6.5 Conclusion

We propose Flock; a simple autonomous VM migration protocol. Flock considers the peer-to-peer interaction of VMs in heterogeneous edge and conventional cloud platforms. We show that Flock converges to a Nash equilibrium with $(1 + \epsilon)$ PoA. Flock minimizes the average latency of VMs as a generic goal with diverse use cases and can be easily redesigned to serve other purposes such as load-balancing and energy efficiency.

Chapter 7: When Clones Flock Near the Fog

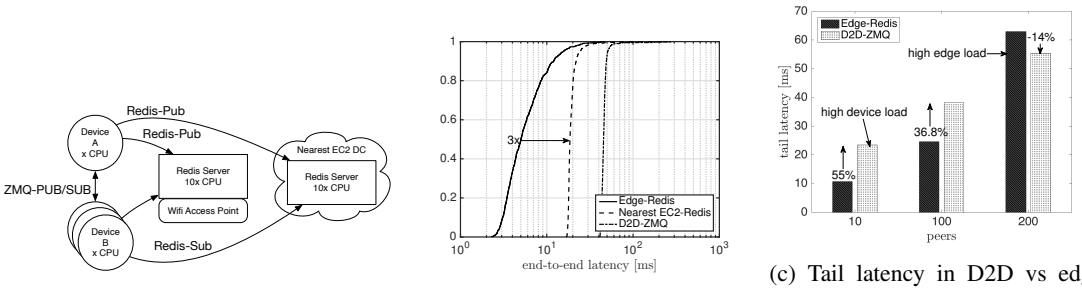
7.1 Introduction

Many large-scale applications are sensitive to latency as they rely on messaging sub-systems between geographically distributed devices and cloud services. Even if 10% of messages were delayed for longer than 150-300 ms, applications like remote-assisted surgery and real-time situation awareness may not be feasible [20, 136]. A bounded tail end-to-end latency is a cornerstone for the realization of large-scale IoT applications near the network edge [141, 34].

When devices communicate through a middle message broker, successive packets queuing in multi-hop paths becomes a major source of latency. For example, the average end-to-end latency of messages exchanged using a Redis broker in a close amazon data-center is three times longer than deploying the same broker one-hop away from devices. Broker-less messaging using device-to-device communication does not necessarily solve the successive packets queuing problem. In IoT applications, a device communicates a large number of messages with many devices. Devices limited processing and memory capacity become another major source of latency for large-scale distributed applications. Experiments show that direct device-to-device messages can experience double the end-to-end latency compared to brokering the messages through a one-hop away broker (see Section 7.2).

If devices are cloned in a one-hop away cloudlet [144], a device's clone can provide message brokering service so that interacting devices can communicate with minimal latency and allow devices to offload intensive computation in very large memory and processing nodes that host the clones. Of course, communicating through a one-hop away clone may still cause long *tail end-to-end latency* when the broker service relays messages to distant devices. If a clone can measure: $1)$ messaging demand with other devices/clones, $2)$ the tail latency experienced by messages, and $3)$ the potential latency of other cloudlets/cloud platforms, clones can self-migrate between cloud platforms to always ensure a bounded weighted tail end-to-end latency. We show how autonomous clone migration can mimic birds flocking and we prove that it is stable and it achieves a tight minimal latency that is $(1+\epsilon)-$far from optimal.

The use of cloudlets and dynamic service migration to solve latency problems are not new:



(a) Experiments setup. (b) CDF of latencies. (c) Tail latency in D2D vs edge brokers.

Figure 7.1: Motivating experiments: the sources of latency in devices publish/subscribe communication.

Cloudlets [145] reduce the single-hop latency from 0.5-1 seconds to tens of milliseconds, and technologies like MobiScud and FollowMe [165, 156] migrate clones to sustain an average single-hop Round Trip Time (RTT) at nearly 10 ms. Such schemes struggle to make optimal migration decisions despite using central control units as they: adopt too constraining migration metric (average single-hop latency) and trigger migration only if devices locations change [163].

However, applications in fog computing [34] necessitate the deployment of inter-networking clones in heterogeneous platforms (cloudlet/clouds) without centralized administration. In this fog environment, clones communicate with several geo-distributed devices and other clones where the *tail weighted end-to-end latency* - that considers the 99-th percentile of computation latency plus the communication latency between clones/devices - becomes the primary latency measure instead of the average RTT of a single-hop.

We design self-deploying brokering clones that discover cloud hosting platforms and autonomously migrate between them according to self-measured weighted tail end-to-end latency, ultimately allowing us to stabilize clones deployment and achieve a near minimum latency given an existing infrastructure limits.

We implement FogMQ as a simple Linux service that provides four key features. It:

1. reduces the successive queuing problem by ensuring a bounded *tail weighted end-to-end latency* and a rapid adaptation to shared hosting nodes and network variations and *without sacrificing computation offloading gain* in cloud platforms;

2. autonomously discover and migrate to heterogeneous cloud/edge platform in an Internet-scale system *without the need of a central monitoring and control unit*;

3. simple and *requires no change* in existing cloud platforms controllers.

7.2 Motivation and Challenges

We first show that multi-hop queuing along Internet paths is a major source of end-to-end latency for IoT applications. We then show that devices' limited compute and memory resources standstill against latency reduction by direct device-to-device communication.

7.2.1 Sources of latency

When we host a message broker in a multi-tenant cloud platform, the end-to-end latency degrades as *network interference* delays the broker's messages. Network interference occurs when the broker messages share: ingress/egress network I/O of its host, and one or more queues in the data-center network switches [28]. Hosts and network resources become spontaneously congested by traffic-demanding applications. For a single-authority cloud, an operator can control network interference of latency-sensitive applications with switching, routing, and queue management policies besides controlling contention for hosts' compute, memory, and I/O resources [133, 50].

Network interference is harder to control for IoT applications. As devices communicate using cloud-hosted brokers, messages share network resources of multi-hop paths with diverse and unmonitored traffic. Multiple, unfederated authorities manage network resources along these paths, which make it hard to enforce *unified* traffic shaping or queuing management policies. Adding also variations in devices traffic demand, communication pattern with other devices, and mobility it becomes particularity hard to trace devices traffic, delays, and infrastructure conditions to find optimal policies with centralized solutions. Multi-hop queuing along Internet paths can account for a 3x degradation in end-to-end latency on average.

Figure 7.1a illustrates experiments to quantify this latency degradation. We install a Redis server in a VM-instance in the nearest amazon EC2 data-center (EC2-Redis) and install another Redis server in a same capacity VM in a host that is co-located with our WiFi access point (Edge-Redis). Our host runs other workloads. We emulate devices as simple processes running on another host that uses the same access point. We ensure that all VMs and hosts are time-synchronized with zero delays and jitter *during the experiment execution time*. A device emulator A publishes $10K$ messages to either Redis servers and another device emulator B subscribes to A's messages. Figure 7.1b shows the Cumulative Distribution Function (CDF) of the end-to-end latency measured as the time between receiving a message at B and publishing it from A. The tail end-to-end

latency for Edge-Redis is $15.6ms$, while it measured at $24.2ms$ for EC2-Redis accounting for 1.5x tail end-to-end latency improvement by avoiding the multi-hop path to the closest EC2 instance and 3x improvement on average.

7.2.2 Why broker-less is not always the answer?

Benchmark	50%	$99^{th}\%$
Redis, 1000 messages	523.2	$1,276.0\mu s$
ZeroMQ, 1000 messages	314.8	$647.6\mu s$
Redis, $10,000$ messages	620.1	$2,010.5\mu s$
ZeroMQ, $10,000$ messages	320.3	$652.9\mu s$

Table 7.1: Median and 99^{th} end-to-end latency of Redis and ZeroMQ measured under different loads (number of messages).

Direct device-to-device communication using broker-less message queues can be thought to be better than using message brokers. The obvious reasons for broker-less queues, such as ZeroMQ [89], superiority are their lightweight implementation, and usage of a minimal number of shared queues, switches, routers, and access points, between communicating devices. Table 7.1 shows the median and tail end-to-end latency of ZeroMQ and Redisunder different loads, where ZeroMQ can deliver $10,000$ messages three times faster than Redis.

Unfortunately, if the devices are resource limited, the latency superiority of direct device-to-device communication is not always maintained. We return to our motivating experiment in Figure 7.1a. We limit the resources used by the devices emulators using Linux cgroups such that a device emulator can use no more than 10% of the CPU time compared to EC2-Redis or Edge-Redis. Figure 7.1b shows that the average end-to-end latency of D2D-ZeroMQ is 7 times longer than Edge-Redis , and the tail end-to-end latency is 4 times longer. Several factors can contribute to this deteriorated performance including the wireless environment loading and implementation details of either Redis or ZeroMQ. However, the main factor that limits direct device-to-device latency is the limited compute resources of the devices emulators.

To emphasis this observation, we increase the number of the publishing device peers (i.e. number of subscribing device emulators) until the Edge-Redis server becomes loaded. Figure 7.1c shows the tail end-to-end latency for different number of peers. As we increase the number of peers, the latency superiority of the Edge-Redis starts to diminish, until we reach the 200 peers points at which our host becomes loaded at 90% utilization and the tail end-to-

end latency of broker-less D2D-ZeroMQ becomes better by 14%. Broker-less device-to-device messaging is only better if a device computational resources are sufficiently large, which is an unrealistic assumption for most IoT devices.

7.3 FogMQ System Design

Our motivating experiments show that multi-hop queuing along Internet paths is a major source of tail latency for cloud-based messaging systems and that the latency improvement promise from device-to-device communication cannot be always attained due to limited devices resources. FogMQ tackles multi-hop queuing by reducing the queuing of messages. Primarily, if message brokers can self-deploy and migrate across cloud platforms (from edge to cloud and including cloudlets) according to the communication pattern of the devices, then we will diminish the impact of multi-hop queuing delay. In the extreme case, if two resource-limited devices communicate through brokers in the same unloaded host and using the same access point, we can achieve a finite minimal bound on the latency.

In this section, we derive an intuitive design of FogMQ by which we bound the weighted end-to-end latency of devices' clones in FogMQ given an arbitrary network of heterogeneous cloud platforms. Although the stability and bounded performance of our design is intuitive, we solidify this intuition by relating the design to the theory of singleton weighted congestion games [76, 77, 33], where we show that self-deploying clones reach a NE and it tightens the PoA of the weighted end-to-end delay.

7.3.1 Social networks of clones

To begin, we assume that devices communicate with each other according to IoT applications requirements and form a social network of devices. Typically, the convergence of man-machine interactions in IoT will drive devices to form a social network [24]. This network can form according to existing social network structure of devices' users or according to the required communication between devices that is inherited from application design.

The idea of modeling IoT analytics applications as social networks is simple. Applications are modeled as graphs (e.g. [90, 14, 142]) of inter-networked microservices, nano-services, virtual machines, or containers. Devices contribute to the execution of an IoT application by *publishing* their data to a brokering clone of the device. Other devices that are participating

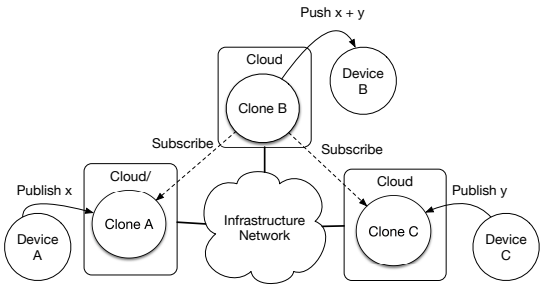


Figure 7.2: Example overlay aggregation tree formed by devices clones.

in the execution of the application also *publish* their information to their clones. On the other hand clones *subscribe* to each other according to the application modeled graph which forms an overlay network of clones that execute the application within resource-rich compute and network nodes. Upon completion of the executions clones may *push* the results back to devices and the application users. Figure 7.2 illustrates a simple tree aggregation application for data retrieved from three devices. Device-A and Device-B publish their data, x and y, to their clones. Clone-C subscribes to data from clone-A and clone-B, evaluates $x+y$, and pushes the result to device-C.

The pub/sub communication pattern provides an efficient messaging middle-ware for applications modeled as large-scale graph structures. We can rely on already in-place subscription and matching languages to effectively route information between devices and clones, and inter-clone. Existing pub/sub systems also simplify addressing and clone authentication, hence the design of large-scale applications as overlay network between the clones.

Generally, the overlay network design of the clones is either structured or unstructured and focus mainly on reducing clones fanout to minimize the communication overhead between the clones. For example, topic-connected overlays are designed such that devices interested in the same topic are organized in a dissemination overlay [47]. Overlay network design forms the foundation for distributed pub/sub and directly impact scalability and applications performance [26, 44, 155]. We assume that an overlay topology of clones is given and we model it as a social network of clones. We model the network of clones as a graph $G = (V, P)$, where V denotes the set of n clones and P denotes the set of clone pairs such that $p = (i, j) \in P$ if the i-th and j-th clone communicate with each other.

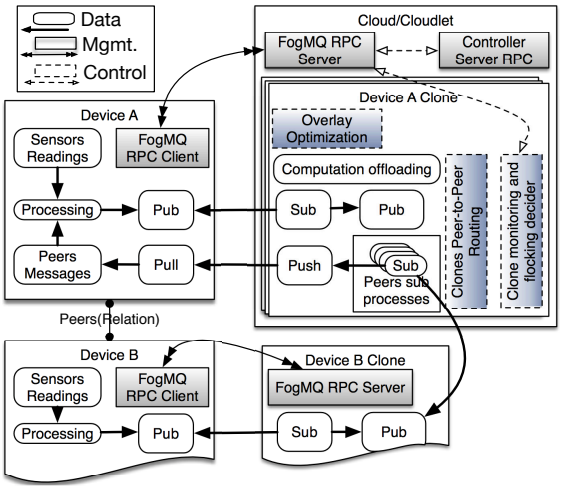


Figure 7.3: FogMQ Architecture

7.3.2 FogMQ Architecture

FogMQ consists of a management-plane, control-plane, and data-plane. The management-plane comprises nanoservices and clients for devices onboarding, clone creation, and cloud-network tomography. The control-plane consists of clone monitoring and migration, overlay optimization, and peer-to-peer routing functions. The data-plane uses the publish/subscribe communication pattern for messaging between devices and their clones as well as inter-clone communications. Figure 7.3 illustrates the high level architectural elements of FogMQ and its management, control, and data modules.

FogMQ initially clone a device at the closest cloud/cloudlet from a set A of m cloud/-cloudlets that are available to all devices and that can communicate over the Internet. Association with nearest brokering server is a standard approach in existing cloud-based brokers. An Remote Procedure Call (RPC) client in the device is responsible for the device registration, and peer relationship definition with other devices by which a device participates in the execution of a distributed application. A typical approach that clients can use to initiate clones is to query a global geo-aware Domain Name Service (DNS) load balancer to retrieve the Internet Protocol (IP) address of the nearest FogMQ RPC server. With the integration of cloud computing in cellular systems [12], devices can also use native cellular network procedures to initiate with clones in the nearest cellular site to the device.

The FogMQ RPC server realizes the device clone in a cloud platform as a virtual machine,

container, or native process, where processes are always a favorable design choice to avoid virtualization overhead. Recent $\mathrm{Linpack}$ benchmark [61] shows that containers and native processes can achieve a comparable number of floating point arithmetic per second that is at least $2.5\mathrm{x}$ greater than virtual machines. Despite that containers have a better advantage for privacy and security, as they provide a better administration, network, storage, and compute isolation, containers networking configuration can account for $30\mu s$ latency overhead compared to a minimal network configuration of native processes [70]. As we will discuss later implementing clones as processes has an advantage over both virtual machines and containers as it incur minimal migration overhead in our design.

Once FogMQ creates a clone for a device, the clone subscribes to the devices published messages that contain preprocessed sensors reading. Subscribing to the devices messages eases the clone migration processes as we will detail later. If a clone migrates from one cloud to the other, any changes to the clone's IP address or network configuration become transparent to the device. Upon migration, the clone resubscribes to the devices messages, allowing the device to continue publishing its messages without the need to notify the device of its clone migration.

A device's clone can process the devices message in its computation offloading module (see Figure 7.3) with high processing, memory, and storage capacities. The computation offloading module of the clone also executes distributed applications defined as overlay networks that interconnect several clones. To exchange messages between peer-clones of an overlay network, a clone creates a separate process to process its peer messages. Each process subscribes to the published messages from its corresponding peer-clone to make messages from peers available for the computation offloading module. If needed a clone pushes messages and/or computation results back to its device using a push/pull messaging pattern. Figure 7.3 illustrate the messages flow between different modules for a simple example in which Device-A is a peer to Device-B.

The overlay optimization module and the peer-to-peer routing module are responsible for optimizing the fan-out of overlay networks and the routing decisions. Although the design of these mechanisms is integral to the performance of the overall system, overlay design and routing optimization algorithms are orthogonal to the scope of this paper as we focus on autonomous migration decisions that minimize the tail end-to-end latency.

7.3.3 Cloud network tomography

An administrator runs FogMQ servers as application middle-wares that have network tomography functionalities. In addition to managing the clones, the network tomography functions assist the clones to measure and discover their local and potential hosting platforms, hence autonomously decide their optimal hosting clouds. The tomography function consists of measuring the average processing delay of a clone's host and the network latency between the hosting cloud and any other target cloud. It also comprises a function to discover potential cloud platforms.

7.3.3.1 Discovering cloud platforms

FogMQ servers in different clouds form a peer-to-peer network that can autonomously evolve and discover each other. A FogMQ server discovers other FogMQ servers explicitly in a bootstrapping phase and implicitly as it creates and manage clones. Once a FogMQ server starts, it register with bootstrapping nodes that also authenticate the server. If a bootstrapping node permits the server to join the network of FogMQ servers, the server becomes entitled to query any bootstrapping nodes to learn about other FogMQ servers that already exist.

Additionally, whenever a server can implicitly learn about other FogMQ servers from clones communication sessions. For two clones to communicate, each clone needs to learn the IP address of the hosting node of the other clone by querying a clones registry (e.g. distributed key-value store). The server proxies such control queries, and from the query responses it can learn about new IP addresses of other hosting platforms and query the bootstrapping nodes for more information if needed.

7.3.3.2 Processing delay and network latency

The FogMQ servers are also end-points that measure the average processing delay and network latency of the hosting cloud and of any other cloud. Processing delay of any cloud platform is a function of the utilization of the node that hosts the clone. A FogMQ server can programaticlly query the utilization of virtual machines using existing cloud platforms Application Programming Interfaces (APIs). Network latency can be actively measured by ICMP protocols, or passively measured by logging the RTT statistics of ongoing Transport Control Protocol (TCP) sessions.

Each device clone self-monitors and characterizes the demands with its peers and evalu-

ates latencies with the assistance of the FogMQ server. For clones, i and j, let $d_{ij} \in \mathbb{R}^+$ denote the traffic demand between i and j and assume that $d_{ij} = d_{ji}$. Let $x_i \in A$ denote the cloud that hosts i and let $l(x_i, x_j) > 0$ be the average latency between i and j if they are hosted at x_i and x_j respectively (Note: if i and j are hosted at the same cloud $x_i = x_j$). We assume that l is reciprocal and monotonic. Therefore, $l(x_i, x_j) = l(x_j, x_i)$ and there is an entirely nondecreasing order of $A \rightarrow A'$ such that for any consecutive $x_i, x_i' \in A'$, $l(x_i, x_j) \leq l(x_i', x_j)$. The reciprocity condition ensures that measured latencies are aligned with peer-clones and imitates the alignment rule in bird flocking. FogMQ servers align the measured latencies such that x_i and x_j use the same values of $l(x_i, x_j)$. We model $l(x_i, x_j) = \tau(x_i, x_j) + \rho(x_i) + \rho(x_j)$, where $\tau(x_i, x_j)$ is the average packet latency between x_i and x_j, and $\tau(x_i, x_j) = \tau(x_j, x_i)$. The quantity $\rho(x)$ is the average processing delay of x modeled as: $\rho(x) = \delta \sum_{i \in V : x_i = x} \sum_{j \in V} d_{ij} / (\gamma(x) - \sum_{i \in V : x_i = x} \sum_{j \in V} d_{ij})$, where δ is an arbitrary delay constant and $\gamma(x)$ denote the capacity of x to handle all demanded traffic of its hosted clones. An increased value of $\rho(x_i)$ signals the clone i that it is crowding with other clones in the same cloud which is imitating the separation rule in bird flocking.

7.3.4 Clones shall flock

To resolve the successive queuing problem, we design a simple protocol by which a clone autonomously decides its hosting cloud using only local network tomography provided by its hosting FogMQ server. Clones greedily minimize their perceived weighted latency. In this section, we analytically show that that this protocol is stable and close to optimal. In Section 7.4.4, we demonstrate these properties when we functionally evaluate FogMQ. A clone i evaluates its weighted latency with its peers if hosted at x as

$$u_i(x) = \sum_{j \in V} d_{ij} l(x, x_j) / \sum_{j \in V} d_{ij}. \tag{7.1}$$

Our objective is to design an autonomous clone migration protocol that converges to an outcome $\sigma = (x_1, x_2, \ldots, x_n)$ that minimizes the sum of weighted latency $\sum_{i \in V} u_i(x_i)$. That is to say, σ maximizes the responsiveness of all clones given their peer-to-peer communication pattern (i.e. connectivity and demands).

A clone i learns a set $A_i \subseteq A$ from its FogMQ server. The set A_i is called the strategies set of i. For a cloud x, FogMQ server evaluates its current weight $w_x = \sum_{i : x_i = x} u_i(x)$ and advertises

a monotonic non-negative regularization function $f(w_x) : \mathbb{R}^+ \to \mathbb{R}^+$, such that $\alpha < f(w_x) < 1$ for $\alpha > 0$, to all the clones that are hosted at x. As any clone is hosted by a single cloud and all clones have access to the same strategies set, we model the migration problem as a singleton symmetric weighted congestion game that minimizes the social cost $C(\sigma) = \sum_{x \in A} w_x f(w_x)$. If $f(w_x) \approx 1$, this game model approximates to minimizing $\sum_{i \in V} u_i(x_i)$. Let $x \xrightarrow{i} y$ denote that a clone i migrates from cloud x to cloud y and let $\eta \leq 1$ denote a design threshold. We propose the following migration protocol:

Flock: Autonomous clone migration protocol.

Initialization: Each clone $i \in V$ runs at a cloud $x \in A$.

Ensure: A Nash equilibrium outcome σ.

1: During round t, do in parallel $\forall i \in V$
   Greedy migration process imitating cohesion in flocking:
2: i solicits its current strategies A_i from x.
3: i randomly selects a target cloud $y \in A_i$.
4: **if** $u_i(y) f(w_y + u_i(y)) \leq \eta u_i(x) f(w_x - u_i(x))$ **then**
5: $\quad x \xrightarrow{i} y$.
6: **end if**

We prove that Flock converges to a Nash equilibrium, where each clone chooses a single cloud (strategy) and no clone has an incentive (i.e. less average latency) to migrate from its current cloud. Then, we derive an upper bound on Flock's PoA for general regularization functions, f, following a similar approach to [33]. Finally, we propose a regularization function $f(w_x) \approx 1$ that achieves a tight PoA of at most $1 + \epsilon$.

Theorem 7.3.1 Flock *converges to a Nash equilibrium outcome.*

Proof *We show that any step of* Flock *reduces the social value $C(\sigma)$ and $C(\sigma)$ is bounded below. We first show that if a clone i migrates from cloud x to cloud y, the increase in y's weight is less than the decease in x's weight. We then use contraction to show that if a subset of i's peers have a total latency that increased after i migrates, the remaining peers must have a total latency that decreased by a greater value. See Chapter 6.*

Lemma 7.3.2 *The social value of* Flock *has a perfect PoA at most $\lambda/(1-\varepsilon)$, if for $\varepsilon < 1$ and $\lambda > 1 - \varepsilon$ the regularization function satisfies $w^* f(w + w^*) \leq \lambda w^* f(w^*) + \varepsilon w f(w)$, where $w \geq 0$ and $w^* > 0$. See Chapter 6.*

Theorem 7.3.3 *The regularization function $f(w) = exp(-1/(w+a))$ tightens the PoA to $1+\epsilon$ for a sufficiently large constant a and reduces the game to the original clone migration problem, i.e. minimizing $\sum_i u_i(x_i)$.*

Proof *For*

$$\lambda \leq \frac{f(w_{max}+w_{min})}{f(w_{min})}\left(1-\varepsilon\frac{w_{max}f(w_{max})}{w_{min}f(w_{max}+w_{min})}\right),$$

$f(w) = \exp(-1/(w+a))$ *satisfies the condition $w^*f(w+w^*) \leq \lambda w^* f(w^*) + \varepsilon w f(w)$ (verify by inspection). For an infinitesimally small value of ε, we can choose a such that $\lambda = 1+\epsilon$, hence the POA $\leq 1+\epsilon$. If a is sufficiently large $f(w_x) \approx 1$, hence $C(\sigma) \approx \sum_i u_i(x_i)$.* ■

7.4 FogMQ Prototyping and Evaluation

We prototyped FogMQ and evaluated it in a virtual testbed in AWS cloud. We implemented all FogMQ components as nanoservices that communicate using ZeroMQ. We deployed FogMQ and the device emulators in several geographically distributed virtual machines in all EC2 regions and availability zones. Our objective from this prototyping is to have a reference implementation of FogMQ by which we test its functionality and verify the stability and near optimality properties in a shared cloud and Internet environment.

7.4.1 Devices on-boarding and overlay setup

In our prototype, we have emulated IoT devices as Linux processes that send messages according to a Poisson distribution with a random average inter-arrival time of 1 to 30 seconds. Each device emulator has a random unique ID. We used a Redis to register the devices emulators and to track the clones deployments across different clouds.

Figure 7.4 shows the device emulator initialization sequence. A device emulator registers itself with the nearest FogMQ server. The emulator uses its ID, host IP address, and ZeroMQpublish and pull ports for the registration. The FogMQ server adds the device information to the Redis key-value store. This step can also be extended for devices authentication. Once registered, the FogMQ server starts a clone for the device and acknowledge the device that it is ready to add peers. The clone subscribes to the devices ZeroMQpublish port.

Whenever a message becomes available to the device from one of its peers, the clone pushes the message to the device pull port. The devices peer-to-peer relationships - in our evaluation -

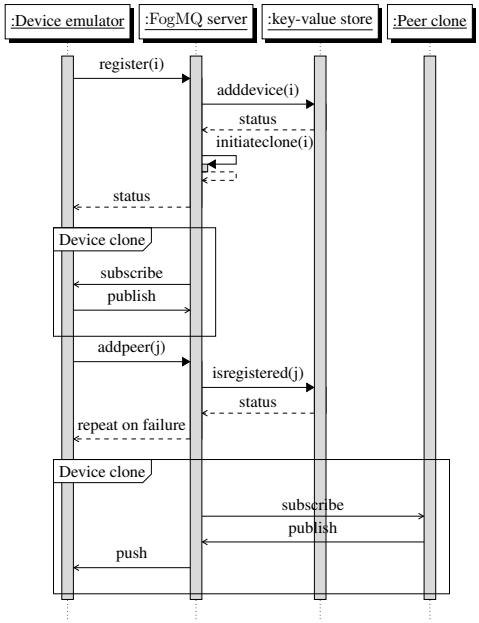


Figure 7.4: Device emulator onboarding, cloning, and adding peers.

are determined based on real social network dataset from [123]. Devices emulators adds their peer at the clone side using only the peer device ID. The device clone retrieves the peer clone host IP from the key-value store. If the peer was not registered in the key-value store, the device clone instructs the device to back-off for a randomly chosen timeout before retrying to add the peer. Otherwise, the peer sub processes subscribes to messages from the peer clone to establish the clone-to-clone overlay communication.

7.4.2 Flock and clones migration

We now detail how we implement Flock using nanoservices exposed by FogMQ servers and how clones autonomously and efficiently migrate between cloud platforms. Each clone periodically executes the sequence diagram that we illustrate in Figure 7.5. In our implementation we arbitrarily choose the execution period to be $30s$ and it is left as a design parameter that can be configured to tune how rapid clone migration shall be.

To execute Flock , a clone i retrieves a potential target cloud y, from its FogMQ server running at its hosting cloud x. Then i exeutes two RPCs with both FogMQ servers at x and y to

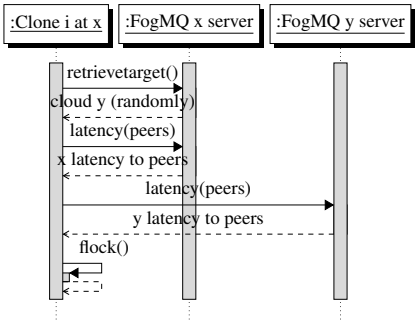


Figure 7.5: Flock sequence diagram.

determine the current and the potential latencies with its peers. When a server receives a $latency$ RPC for a peer j, it immidetly answers with previously performed latency measurements with the cloud hosting j, or perform active measurements to determine the estimated latency. Finally the clone executes Flock to decide if a migration shall be performed or not.

If the migration condition in Flock is satisfied, the clone i sends its metadata an RPC to x requesting to migrate to y as illustrated in Figure 7.6. The server x forwards i's meta-data to y to initiate a clone for device-i at y. The server y replaces the current hosting cloud of i to y and the new clone at y subscribes to i's messages. Finally server y instructs x to terminate its migrated clone.

Figure 7.6 illustrates the migration sequence diagram and Figure 7.7 shows the total time needed for clone migrations as Flock continues execution over time. By only transferring the meta-data, which is less than 140 bytes in size, we ensure that clone migration overhead is minimum. Alternatively, one can take a snapshot of the clone at x, transfer the entire clone memory, and restart the clone at y. This can resemble a significant overhead as we could transfer unnecessary information. As shown in Figure 7.7, the parameter η can have an impact on the migration overhead at initial execution where higher values of η eases the migration decisions. Morevoer, The use of the publish/subscribe communication here makes such migration transparent to device i, and prevents potential data loss.

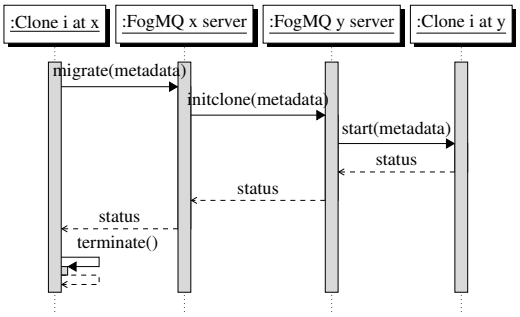


Figure 7.6: Clone migration sequence diagram.

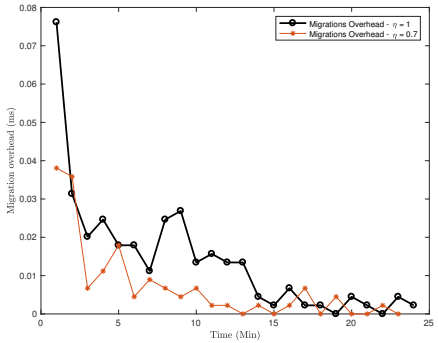


Figure 7.7: FogMQ migration overhead

Figure 7.8: Code snippets for autodisconnect.

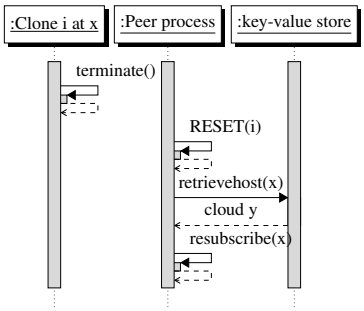


Figure 7.9: Clone peer termination detection.

7.4.3 Accounting for data loss

Although making new clone to device subscription before breaking the old one prevents data loss from a device to FogMQ, migration can still results in data loss between peers. First, the peer processes that originally subscribed to i's clone at x needs to detect its termination and resubscribe to the new clone at y. Second, the new clone needs to enforce a history Quality of Service (QoS), by which the new clone retains devices data until the peer processes subscribes to the new clone and retrieve all historical data. The solution to the last problem is straight forward. FogMQ implementation allows an administrator to configure FogMQ to either keep all devices data in the new clone until delivery to peer processes, or to keep a configurable number of messages (either by number or duration).

To solve the first problem, we need two mechanism to detect the termination of the original clone and discover the new hosting cloud of the new clone. We use in-place connection-monitoring in ZeroMQ to detect clone termination. The code snippet in Figure 7.8 shows the main thread for clone connection monitoring. If the monitoring thread sees an EVENT_DISCONNECT event, it sends a RESET signal to the peer process. As the peer process receives a RESET signal, it queries the current host from the key-value store, and attempts to resubscribe to the clone at the new hosting cloud.

EC2 Regions	Virtual Machines
ap-northeast-1	4
ap-southeast-1	8
eu-west-1	6
sa-east-1	4
us-east-1	8
us-west-1	4
us-west-2	7

Table 7.2: Testbed distribution in EC2 regions

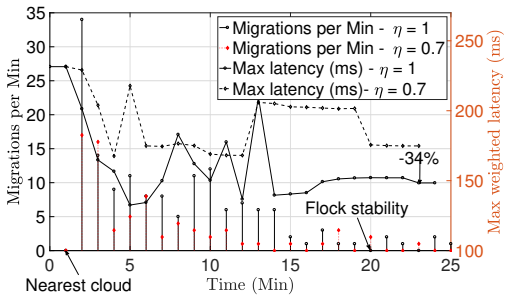


Figure 7.10: FogMQ stability for differnet values of η

7.4.4 Experimental results

We deplyed our prototype in a geographically distributed testbed comprising fourty one virtual machines running in EC2 regions as shown in Table 7.2. One virtual machine is running a Redis key-value store to maintain experiments logs and a bind DNS server. Twinty virtual machines are running FogMQ, and each of the remaining virtual machines runs five device emulators. We generated the peer-to-peer relationships of the hundred device emulators as a subset of the Facebook dataset from [123].

Our *functional evaluation* validates the stability of FogMQ given the flocking of clones and its capability to subdue to end-to-end weighted latency of the clones. The use of geographically distributed virtual testbed challenges FogMQ's stability and optimality given the real variability and uncertainty of the Internet and the hosting cloud conditions. We also show the advantage of using Flock in FogMQover the widely used architecture in which devices relay messages through the nearest broker.

Figure 7.10 shows a 30 minutes experiment. Initially all device emulators register with the nearest FogMQ server by querying the DNS server. The Flockis activated at minute 1, where

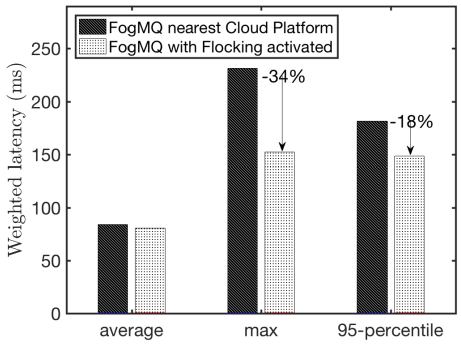


Figure 7.11: FogMQ Latency

all servers verify that all device emulators are registered from the Redis server. At this minute, clone starts to autonomously migrate between the server and the secondary access show the total migrations per minute. The number of migrations decrease rapidly and by minute 20, all clones reside in a FogMQ server where they cannot improve their measured latency anymore. This is a stability point of Flock. After minute 20, we notice some clones perform migrations in which they quickly respond to variations in the system.

Figure 7.11 shows the summary statistics of the weighted latency comparing the initial deployment and the stability point of FogMQ. The average latency is not significantly improved. On the other hand the tail end to end latency shown as the maximum latency and the 95-percentile latency has improve by 34% and 18% respectively. On the secondary axis of Figure 7.10, we show the maximum weighted latency measured by clones. As Flock progresses, also stabilizes at $150ms$ compared to $231ms$ initially. This stable behavior demonstrates the capability of Flock to maintain tail latency under tight control, which we theoretically proved that it is no far thatn $1 + \epsilon$ from an optimal.

7.5 Related Work

We propose autonomous brokering clones for designing large-scale distributed pub/sub systems as a major mechanism that complements existing techniques to minimize the tail end-to-end messaging latency. Researches focus on three main techniques for the development of simple, scalable, and resource economic pub/sub systems: 1) content-centric above layer-3 routing between brokers (e.g. [42, 43, 18, 45, 137]), 2) overlay brokers network topologies designs (e.g.

[152, 46, 158, 44, 128]), and 3) content-centric in-network caching (e.g.[59, 52, 122, 122]).

Distributed pub/sub systems organize brokers, devices, or routing functions as an overlays and sub-overlays at the application layer. Upon constructing an efficient overlay network, routing protocols above layer-3 build minimum-cost message dissemination paths to deliver messages to subscribers according to specific topic-interest. Caching policies replicate clones' contents closer to devices interested in a content for faster repetitive publishing. For a given routing, overlay topology, and caching mechanisms, FogMQ ensures that these mechanisms achieve their full potentials by self-reorganizing the deployment of brokers through migrations in heterogeneous, unmanaged, and dynamic cloud environments. Unlike widely adopted centralized systems (e.g. Redis), FogMQ suits the large-scale applications and use cases of IoT and avoids the limitations of broker-less systems (e.g. ZeroMQ).

7.6 Conclusion

There are many message brokers, cloud offloading, and communication middle-wares that support IoT analytics. We propose FogMQ, a message broker and cloning system for scalable and geographically distributed analytics near and at the network edge. FogMQ uses Flockto enable device clones to autonomously migrate across heterogeneous cloud/edge platforms. FogMQ servers expose tomography functionalities that enables devices clones to take migration decisions without complete knowledge about the hosting platform. This feature suites the deployability and scalability challenges across Wide Area Networks (WANs) and the Internet.

We discuss the prototyping of FogMQ and its evaluation of a virtual and geographically distributed testbed using public cloud resources from AWS-EC2. In our functional evaluation, we demonstrated the stability of Flock, and the ability of FogMQ to improve the tail latency by one third of that attained by a widely used architecture in which device communicate through the nearest message broker. In our design we not only take into account devices' location, but also it traffic patter and communication relationships with other devices and cloud services.

Chapter 8: Policies are not Barriers when Overlay Networks GROUP

8.1 Introduction

GROUP is a dynamic application overlay platform for controlled, trusted, and reliable communication among heterogeneous devices and services in cloud and edge computing environments. GROUP eases development and policy management for such complex environments. It has a control plane that creates application overlay networks. An application overlay network is data plane that is used to exchange messages between application services. Each data plane can also act as a control plane to create and manage additional application overlay networks. Control plane actions are uniformly and transparently subject to policy and obligations. Policy determines what actions are permitted. Obligations direct adaptation for the system in terms of launching or terminating services, or requiring changes to the application overlay networks. Our approach distributes policy enforcement and allows for centralized, distributed, or hybrid policy decision, information, and administration points. We have implemented GROUP as reusable Java APIs and tested it in a controlled environment where we demonstrate its usability by implementing a privacy perceiving ride-sharing solution that leverage real-time data from IoT devices.

We provide a secure group platform for dynamic, controlled, trusted, and reliable information exchanges among devices and application services from different vendors that may operate on shared resources from the edge through to the enterprise and cloud. The platform is guided by policies from participating vendors. The policies govern access to data, participation in distributed analytics, and access to control interfaces. Policies may be situationally dependent. The approach is designed to support a vast number of distinct but possibly interconnected instances of systems, efficiently enabling the diversity of configuration and dynamism that will be required at an ever expanding edge computing footprint. We use a ride sharing service use case we name tax-e-bay to demonstrate the platform.

The dynamic secure group platform is based on application overlay networks. The application overlay networks can have arbitrary topologies, as best suited to the problem at hand [10]. The topologies are realized by a middleware environment that hides networking complexities from developers and allows for scalable, real-time, reliable, and secure data exchanges within

and across administrative domains (see [147, 130]. This enables the implementation of systems where data remains close to where it was generated and problem-specific topologies of application services collaborate to perform analytic tasks such as similarity search or machine learning without transferring their data to a central data lake.

GROUP eases policy management for such complex environments. Control plane actions are uniformly and transparently subject to policy and obligations. Data plane communications are not subject to policy, as the communication links in realized topologies have already been deemed as permitted. Our approach distributes policy enforcement points (PEP) and allows for centralized, distributed, or hybrid policy decision point (PDP), policy information points (PIP), and policy administration points (PAP). This makes the approach more scalable than systems that enforce policy for each data exchange or that require a centralized policy enforcement, decision, or administration point. Further, existing middleware systems lack support for the dynamic creation of groups of application that we show are useful for facilitating application development.

We describe our architecture and design in Section 8.2, where we detail how GROUP creates application overlay networks through communication middlewares and uses eXtensible Access Control Markup Language Version 3.0 (XACML 3.0) policies [71] and obligations for authorization and application development. In Section 8.2, we validate our design through the tax-e-bay use case. We finally summarize the article and discuss future work in Section refconclusion.

8.2 Architecture and Design

GROUP can be deployed on many kinds of computing nodes including dedicated physical servers or IoT devices, virtual machines, or containers in locations from the edge, to business permesis, to cloud computing data centers (see Figure 8.1). Each node minimally comprises of an orchestrator and a middleware service that provides for intra and inter node application communications. The orchestrator is responsible for application service lifecycle related control functions for a node. These include starting, stopping an application service, and requesting the service perform obligated group management activities. A device management service is also illustrated. It shows IoT devices interacting with the management service using an IoT specific protocol. Typical device interactions include registering/unregistering, pushing and pulling data, and receiving control commands. Dynamic secure groups provide for all further communications among the device management service and other application services.

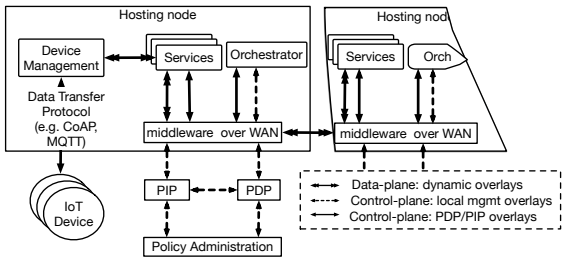


Figure 8.1: GROUP architecture.

8.2.1 Application Overlay Networks

An application overlay network is a directed graph, G with its vertex set, V, containing the members communicating over G's links. Members are application services, where an application service may belong to many graphs at the same time. A directed link $i \rightarrow j \in V$ restricts communication between i and j, such that i only sends messages to j but not vise versa. A graph instance has a name and purpose and context for its communications. For example, different graph and link instances may correspond to different programming interfaces for data access, analytic processes, or access to control interfaces. The applications must implement the corresponding logic.

A member i can own an overlay G, perform actions on G, and receive events from GROUP APIs as G changes. Member i can perform two actions: $1)$ invite another member j to join G, or $2)$ join an overlay G upon receiving an invitation. We refer to the invite and join actions by I and J respectively. With the middleware environment we employ, invitations are sent as point to point messages. An overlay is not considered created until at least two members (including its owner) join. As a member joins and leaves an overlay, GROUP triggers events to the overlay owner: $E(G, j)$ indicating that member j has joined G and $L(G, j)$ indicating that j has left G.

8.2.2 Policies and the Policy Decision Point

Application vendors or deployers use Policy Administration Point (PAP)s to define XACML 3.0 policies for GROUP's members. These policies are then pushed to Policy Decision Point (PDP)s for satisfying policy decision requests. Policies govern each members' entitlements to perform actions related to overlays and define obligations that guide system adaptation.

Each member i has an authorization policy P_i that a PDP evaluates as i performs an action

or receives an event. The PDP decision can either be permit, deny, or not applicable in case P_i is missing. Each policy P_i comprises of two authorization rules: an invite rule and a join rule. The invite rule comprises a list R_I of overlay names, i.e., graph names, that a member i is permitted to invite other members to. Similarly, the join rule comprises a list R_J of overlays that i is permitted to join. A policy also comprises two obligation rules: an on member join rule and an on member leave rule. The on member join rule comprises a list R_E of obligations that the PDP sends to i if j joined G. Similarly, the on member leave rule comprises a list R_L of obligations that the PDP sends to i if j left G. The obligations may cause the start or termination of an application service, or a request for a service to perform some group management actions.

According to the type of the action or event, an inviting or invited member assumes the role of a Policy Enforcement Point (PEP) that triggers an XACML 3.0 request to the PDP to make an authorization decision and/or retrieve obligations. First a member i assumes the PEP role and request the PDP to evaluate if i is permitted to create an overlay G and invites another member j. If j receives an invitation from i to join G, it assumes the PEP role and requests the PDP to evaluate: $1)$ the invite rule of P_i, and $2)$ the join rule of P_j. If $G \in R_I$ of P_i, then i is permitted to invite j to G and if $G \in R_J$, then j is permitted to join G otherwise j ignores the invitation. After j receives the authorization to join G, j is entitled to communicate to its linked member in G's topology without further policy decisions. After j join G, i receives an on member join event upon which it assumes a PEP role and triggers an XACML 3.0 request to the PDP to activate P_i and retrieves the obligations list R_E. Similarly, if j leaves G, i triggers an XACML 3.0 request and retrieves R_L. Our design ensures that the minimal number of policies are evaluated to authorize members to communicate over an overlay.

8.2.3 Bootstrapping and PDP Proxying

When an administrator boots an orchestrator, o, the orchestrator must invite at least one PDP, d, to join o's PDP-overlay, G_{od}. The orchestrator must also invite at least one Policy Information Point (PIP), i, to join o's PIP-overlay G_{oi}. The control overlays, G_{od} and G_{oi}, are generally graph data structure of a hub and spoke topology rooted at o and can include multiple PDPs or PIPs for scalability, load-balancing, or if desired to support separately host policies from different administrative domains. Upon receiving the invitations from o, d locally enforces the invite rules of P_o and the join rules of P_d to determine if o is permitted to send and invitation for G_{od} and if d is permitted to join it. Similarly, i enforces the invite rules of P_o and the join rules

of P_i.

Once i joins G_{oi}, o enforces the on member join rule of P_o to determine the services that o needs to launch in the hosting node. Then, the PDP sends R_E to the orchestrator on G_{od}. In this context, the administrator shall define R_E as a list of services that the orchestrator must launch. As it receives R_E, o launches all services $s \in R_E$. As s starts, it creates it local management overlay, G_{so}, and invites o to join it. Using the previously discussed pattern, o enforces the invite rule of P_s and the join rule of P_o through d communicating on G_{od}.

After launching all services in R_E, o assumes a PDP-proxy and PIP-proxy rules that it exposes to all services in its hosting node. When a service, s need to enforce a policy with the PDP, it sends its XACML 3.0 request to its orchestrator o. Currently, o forwards these local requests to d.

8.2.4 Service Discovery and PIP Proxying

As an orchestrator o, joins a local management overlay G_{so}, o registers s with the PIP, i. The orchestrator sends a remote register call on G_{oi} where i stores s's attributes, supported overlays, and supported functionalities in a distributed data-store. In its initial phase a PIP, i, also invites the PDP, d, to a control overlay G_{id}. This control overlay not only allows the PIP to enforce policies with the PDP, but also allows the PDP to use advanced information about the services from the PIP to make authorization and obligation decisions. For example, a policy may require situational environmental information that is maintained within the PIP.

A service can discover other services by querying the PIP through the service's orchestrator. A service, i, sends a query to its orchestrator, o, on G_{io} specifying the attributes, names, or functionalities of the services it need to invite to its application overlays. The orchestrator sends the query to the PIP where it searches for the required services and notify the orchestrator, hence the service. The middleware environment we employ handles the network-level service discovery through gossip-protocols to determine the hosting node of the service.

8.3 A Ride-sharing Use Case

We validate our architecture and design by implementing a distributed ride-sharing use case for smart-cities with n computing nodes deployed to telecom branch offices throughout the city. Figure 8.2 illustrates an example for New York city.Ride vendor services $\{r_{11}, r_{12}, \ldots, r_{nm}\}$ -

are launched at each hosting node when the vendor has cars operating near the computing node and terminated when it doesn't. Ride vendor services track car current locations, occupancy, and battery power levels, where r_{ij} denote the ride vendor service j hosted at node i. In this use case, customers requests rides using their smart-watches that interact wtih a city wide ride sharing service c. A ride request includes the watche's customer id, current location, and destination.

Figure 8.2: Tax-e-bay use case: an exemplary ride sharing system for New York City.

The city wide service anonymizes the ride requests to hide customers identities and send it to the closest city proxy service $p \in \{p_1, p_2, \ldots, p_n\}$ hosted in nodes $1 \ldots n$ on a static city-proxy overlay, G_{cp}. The overlay G_cp is a star topology, where c is the central node and $p_1 \ldots p_n$ are leaf nodes. As p_i receives the ride request, it queries all ride vendors in node i connected to p_i in a star overlay, G_{p_ir} (with p_i being G_{p_ir}'s central node). With this query, p_i determines which vendors can feasibly satisfy the ride request (e.g. can redirect vacant cars to the customer location within a bounded time) and forwards a list of feasible ride vendors, V to c. Consequently, c creates a dynamic ride auction group G_{ride} in which vendors run a distributed auction algorithm to select a car from a vendor with the minimum possible price.

We use GROUP to create static and dynamic overlays for the ride-sharing use case as follows. We first create the following XACML 3.0 policies: P_c, $P_{r_{ij}}$, P_{p_i}, and P_{o_i}, where o_i is the orchestrator of node i and o is the orchestrator in the node hosting c. P_c permits c to invite others to $\{G_{co}, G_{cp}, G_{\text{ride}}\}$ in its invite rule. $P_{r_{ij}}$ permits r_{ij} to join $\{G_{p_ir}, G_{\text{ride}}\}$ in its join rule and to invite o_i to $G_{r_{ij}o_i}$ in its invite rule. P_{p_i} permits p_i to join $\{G_{cp}\}$ and to invite o_i to $\{G_{p_io_i}\}$. An orchestrator policy P_{o_i} permits o_o to invite the PDP, d, and PIP, i, to $\{G_{o_ii}, G_{o_id}\}$. It also permits o_i to join $\{G_{r_{ij}o_i}, G_{p_io_i}\}$. The on member join rule of o_i lists the services that o_i needs to launch as obligations (i.e. $R_E = \{p_i, r_{i,j}\}$).

8.4 Acknowledgments

This work was done while Sherif Abdelwahab was a Research Associate intern with Hewlett-Packard labs under the mentor-ship of Jerome Rolia. The authors would like to thank Christine Kelley, the associate dean of the College of Engineering at Oregon State University, to make Sherif's internship happen. The statements and opinions expressed herein are solely those of the authors and do not constitute official statements or positions of Hewlett-Packard enterprise, Micro-focus, or any of their partners. The authors are grateful to Amip Shah, Kiara Corrigan, Geoff Lyon, Doron Shaked, and Sagi Schein who contributed to this work.

Chapter 9: Beelet: Large-Scale Overlay Network Experiments in Public Clouds

9.1 Introduction

Today, emerging applications of overlay networks requires experimentation in geographically distributed testbeds. Distributed analytics near the network edge is one example applications in which several hundreds of microservices and Internet of Things devices execute distributed computation on data from devices close to access points [144, 146, 10]. While existing cloud platforms can provide a controlled and reproducible environment for evaluating new cloud architectures for such applications, their assumption that they can (or should) always be used for experimenting large-scale and geographically distributed overlay networks is problematic.

For example, to experiment FogMQ [7] we need to easily deploy FogMQ services and several hundreds of device emulators in a large number of virtual machines that are geographically distributed and interconnect them as flat networks. To remain cost effective we shall be able to dynamically create, interconnect, and configure such virtual testbed, run experiments for a predefined time period, collect results, and then destroy virtual testbed as soon as the results are available for analysis. In some experiments we also need to measure and account for real Internet and shared cloud resources variations. Experiments in such variable uncontrolled and real environments allows credible prototyping of new architectures and systems that face todays real-world challenges such as scalability, multi-tenant support, and handling of resource shortage.

Figure9.1 Illustrates our vision for networks of geographically distributed virtual testbeds. Virtual machines realize the testbed and can be hosted on commodity hosts, telecommunication platforms, public clouds, private clouds, or a heterogeneous combination of them. The virtual machines host application services, processes, or containers from different research experiments. The hosted services communicate over predefined virtual topologies and can reuse IP address spaces for simultaneous isolated experiments.

We introduce Beelet; A geographically distributed virtual testbed management API that simplifies large-scale virtual testbed creation for research experiments in network function virtual-

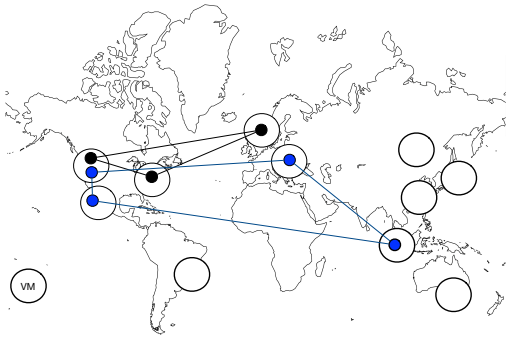


Figure 9.1: Beelet high-level vision.

ization, distributed systems, and large-scale application overlay networks. Beelet uses Layer-3 overlay networks to interconnect application services that Beelet deployed in virtual machines. Beelet can use Amazon EC2 or Google cloud platforms to host the virtual machines and automatically interconnect them according to a predefined topology given by Beelet users. We demonstrate the effectiveness of Beelet through a usage model in which we deploy FogMQ for functional experiments. We also demonestrate the usage of Beelet to implement and evaluate Bird-VNE and RADV.

9.2 Design and Implementation

Beelet consists of a management-plane, control-plane, and data-plane. Figure9.2 illustrates Beelets management and control planes. The management-plane is responsible for creating, configuring, and destroying virtual machines across heterogeneous cloud platforms. We refer to such virtual machines as the testbed workers. Experiments in Beelet are defined as overlay topologies of containers that run micro and nano services. The control-plane is responsible for the dynamic configuration of testbed workers connectivity, containers control, experiments execution, and results collection. Figure9.3 illustrates Beelets data-plane. The data-plane realizes overlay networks between containers or services using Virtual Extensible LAN (VXLAN) tunnels [119].

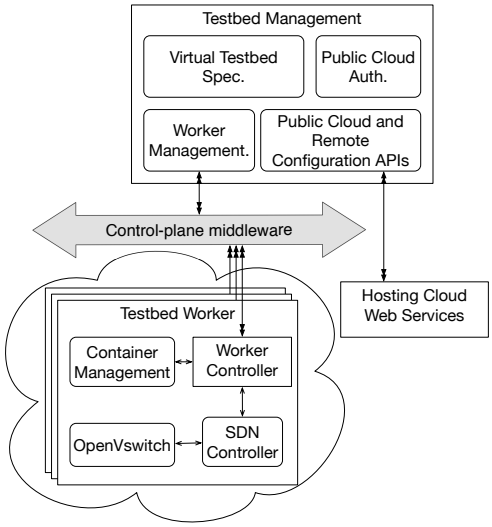


Figure 9.2: Beelet control and management planes.

9.2.1 Testbed Management

Beelet comprises programming APIs that creates the testbed worker in a public clouds. For each virtual testbed, Beelet creates a master worker that hosts management functions of the entire testbed workers. After this worker is created, all other testbed management functions are handled through remote procedure calls between the local Beelet software and Beelet worker management services in the master worker.

A virtual testbed is specified in JSON format as a virtual topology of testbed workers given as input to Beelet. Each testbed worker is defined to be hosted in a particular cloud provider, geographical location, and availability zone. Beelet translates the input specification into a set of remote procedure calls that are executed by the master worker. The master worker creates the virtual machines hosting the testbed workers and configure an OpenVSwitch service in each worker and an SDN controller to control the switch. It also starts a worker controller that allows workers to run distributed control functions, and a container management microservice that compartmentalizes and executes experiments artifacts.

9.2.2 Testbed Workers

Once a testbed worker controller starts, it bootstraps the testbed worker through the master workers controller. Workers controllers communicate over a control-plane middleware. In our implementation, we use ZeroMQ [89] as a lightweight broker-less middleware for inter and intra workers communication. The bootstrapping process involves periodic discovery of the containers and services needed to run a virtual testbed, the virtual topologies of the testbeds, and the security credentials to enable secure communication. This periodic bootstrapping allows dynamic changes of the overlay topology of the virtual testbeds, creating new virtual testbeds using already running testbed workers, or destroying existing ones.

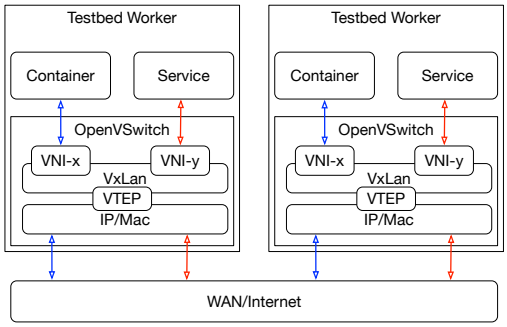


Figure 9.3: Beelet control and data-plane.

In Figure9.3, we illustrate the data-plane of two simple virtual testbeds. In the bootstrapping phase both workers retrieve a list of networked services. In our example these are one container and one daemon service in each worker. The worker controller instructs the container management module -through RPC calls - to start and configure the container and the service. The two services communicate over a virtual link x and resemble one distributed experiment. That services communicate using the same address spaces of the two containers which also run an isolated distributed experiment.

As a worker i learns about another worker j as having direct virtual links with it for a given experiment, it starts a distributed overlay links creation process. Worker i contacts worker j over the - control-plane middleware - to create a bi-directional tunnel for each virtual testbed as specified in the overlay link description in the testbed spec file. To create the tunnel for the virtual link x each worker controller sends an RPC call that is executed by the worker local Software Defined Networking (SDN) controller specifying the remote and local IP addresses of the worker and a unique VXLAN Network Identifier (VNI) for each virtual testbed.

9.3 Usage Model

To demonstrate Beelet in action, we used it to deploy FogMQ functional experiments in a geographically distributed testbed. Experiments in FogMQ comprises two main entities the FogMQ service and geographically distributed IoT device emulators. A device emulator associates with the nearest FogMQ service through geographical DNS lookup facilitated by a bind DNS service that is local to the testbed. Experiments results are collected in a centralized key-value store for post-experiment analysis.

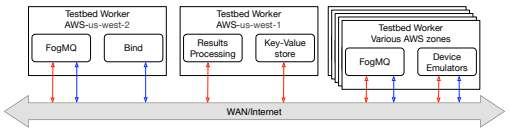


Figure 9.4: Beelet useage in FogMQ experiments.

Figure9.4 shows the virtual testbed used in this use case. The testbed comprises two virtual topologies. The blue virtual topology represents the data communication between the device emulators and FogMQ, and the red virtual topology represents results communications for stats collection. By isolating stats collection traffic from data communication, we can enforce communication policies that prevents traffic resulting from stats collection from interfering with data communication.

9.3.1 BirdVNE Proof Of Concept Implementation

Beelet can also be used to implement Bird-VNE. Figure 9.5 illustrates the software architecture. Substrate network topologies are created as VirtualTestbed Spec of the Testbed management module. The Worker Management module maintains a complete view of the testbed workers topologies which realize the substrate networks, maintain testbed workers utilization (e.g. using sysstat [83]) and network tomography information (e.g. using iperf [160]). Maintining a consistent view of the network entail a significant overhead as we inject traffic in the network to fill the bandwidth pipe and implies incorrect functionality due to errors in bandwith estimation in a shared cloud environment. Such mesurement must be performed on Layer-3. Figure 9.6 shows the measurement overhead evaluated as the injected measurement traffic in a reporting period (1 second) divided by the maximum theoritical bandwidth of the testbed worker virtual network interface.

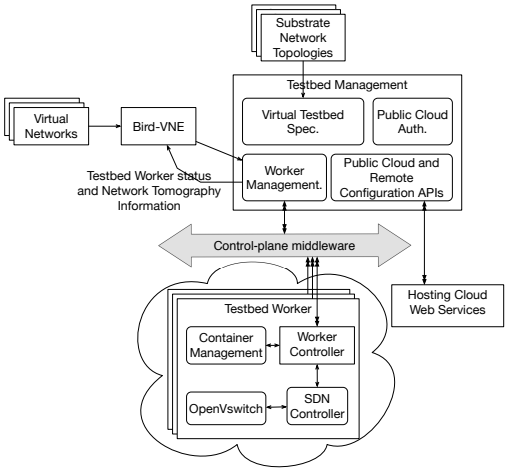


Figure 9.5: Proof-of-Concept of Bird-VNE Implementation.

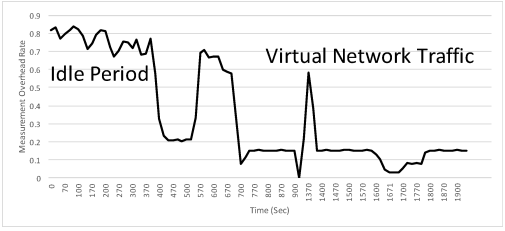


Figure 9.6: Overhead of Network Tomography measured between two testbed workers hosted at different availability zones in an AWS-region as micro instances.

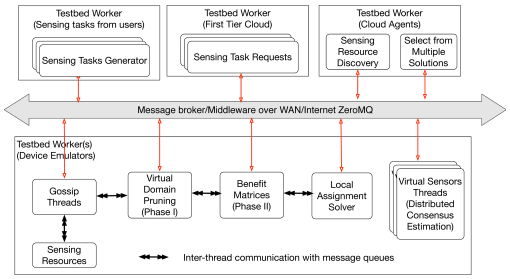


Figure 9.7: RADV Proof of concept of testbed implementation

The worker management module feeds this information to Bird-VNE. Virtual networks are given to Bird-VNE as NetworkX graph data structures. The Bird-VNE module then pass the resulting embedding solutions to the worker management module. The worker management module communicates the resulting embedding solution to the worker controllers. Finally, the worker controllers realize the virtual nodes as containers through the container management module and the virtual links through the SDN controller as VXLan tunnels.

9.3.2 RADV Proof Of Concept Implementation

Figure 9.7 illustrates the software architecture to implement RADV proof-of-concept using Beelet. One testbed worker comprises processes that generate sensing tasks as python objects. Sensing task generators seralize the sensing task objects and send it to another testbed worker which translates sensing tasks to sensing task requests as graph data structure using the networkX library. Another testbed worker hosts cloud agent functionalities including sensing resource discovery and final embedding solution selection. Other testbed workers host device emulators.

A device emulator consists of several threads. The sensing resource thread in the device emulator simulates the functionalities of sensors, or utilization of the device emulator. Sensing resource discovery functionality is split between the cloud-agent testbed worker (in sensing resource discovery process) and the device emulator testbed worker (in Gossip Threads). Figure 9.8 and Figure 9.9 show the scaling of the Service Disovery Time and number of messages with the number of used device emulators which scales linearithmicly as expected if fully connected topology. The actual value of this time is strongly dependant on the number of socket files a device emulator maintains, and how often we open and close the sockets in this proof-of-concept implementation. A device emulator also comprises threads that implements virtual

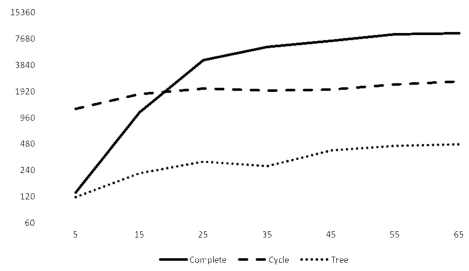


Figure 9.8: RADV Service Discovery Time in Seconds.

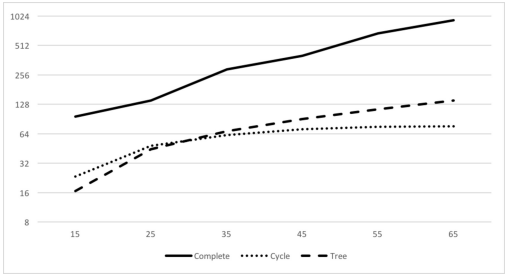


Figure 9.9: RADV Service Discovery Number of Virtual Network Messages

domain pruning, benefit matrix construction, and local assignmenent problem solvers. Virtual sensors are also python threads that communicate over wide area networks using a message broker system.

Table 9.1 shows the Link embedding time and the execution time of Bird-VNE and RADV proof-of-concept implementation. The Link embedding Time includes the time to create two containers in different testbed workers and establish a VxLan tunnel between them. As link embedding is performed in parallel, it is independant from the number of virtual links or the number of virtual nodes. This time significantly varies with the used template of the container image. For Bird-VNE, the execution time is the computation time required from receiving an input virtual network to the time of finding a solution. For RADV, the execution time is the computation time required after the service discovery phase to the time of finding a solution. Both execution times exclude the virtual embedding times[1]

[1]Profiling execution time in a multi-tenant cloud enviornment is an invalid metric analysis.

Metric	Value
Link Embedding Time	494.8 ± 126.4 **seconds**
Bird-VNE Execution Time	210.8 ± 0.12 **ms**
RADV Execution Time (Without Service Discovery)	314.8 ± 104.2 **ms**

Table 9.1: Embedding and Execution Time Statistics for Bird-VNE and RADV.

9.4 Related Work

CloudLab [140], GENI [31], and Emulab [151] all are academic federated testbeds for distributed systems, cloud computing and networking research. Each testbed can provide hundreds of simultaneous compute slices that suites distributed and overlay network experiments. Experiments are limited in particular geographical locations and generally are reliable and reproducable. CloudLab in particular operates on top of three data-centers in Utah, Wisconsin, and South Carolina and integrates with GENI and Emulab. This makes it attractive for experiments that requires geographically distributed resources.

The above testbeds do not meet the requirements we offer in Beelet. First they operate on a dedicated networking infrastructure for Internet2, which prevents it capturing real-world scenarios of a shared infrastructure for prototyping. Second, despite their geographical distribution, they operate in limited geographical locations which constraints evaluating geographically distributed and large-scale functional experiments. Third, the utilization of these infrastructures are reaching warning peaks that cross 86%. Finally, there is no notion of dynamisms , or predefined virtual topologies of the overlay networks that one can create on top of these platforms.

9.5 Conclusion

Beelet complements existing experimental environments to create large-scale overlay networks on top of major public cloud infrastructure. Beelet leverages the decreasing cost of public clouds and the emergent availability of academic research credits to use high-capacity data-centers operating AWS, Azure, and Google Cloud. Beelet introduces three important features that do not exist in current testbeds: dynamic testbed creation and extendability, deployability across heterogeneous compute resources, and large-scale geographical distribution.

Chapter 10: Conclusion

This chapter provides a summary of the thesis, discusses its contribution and limitations, and shed a light on directions of future work.

10.1 Summary

This thesis has introduced techniques for designing and implementing scalable and dynamic overlay networks to satisfy the requirements of Edge Computing. The theoretical techniques developed in Flock and REPLISOM has led to the development of the thesis's proof-of-concepts designated as Replisom7 and FogMQ.

In Chapter 2, we propose REPLISOM , a cloud resources augmented eNB architecture with an LTE-optimized memory replication protocol for the Internet of Things applications. REPLISOM works with the in-place LTE technologies and the emerging D2D technologies to efficiently replicate tiny-sized memory pages from a massive number of devices as fast as possible with the minimal control channel requirements via the sparse reconstruction in compressed sampling theory. REPLISOM also utilizes the sparsity at the memory level to further improve the delay and energy consumption. With extensive numerical evaluations of the delay and energy consumption benchmarks, we demonstrate the benefits of REPLISOM to overcome the LTE bottlenecks that arise from simultaneous access of devices for memory replication. REPLISOM may be redeveloped to suite existing 802.11a/b/g and LTE technologies.

Chapter 3 describes our proof-of-concept implementation of REPLISOM at layer7. In this implementation, we provide essential encryption, authentication, access control and discovery features required in practical Edge computing platforms. We also propose high-level parameter tuning that improves REPLISOM's performance by adopting a self-optimizing the Radio Resource Control parameters in LTE to minimize the end-to-end memory replication delay and devices' energy consumption such that we approach the promised gains of REPLISOM. We experimentally demonstrate the superiority of REPLISOM to conventional uplink data transfer in LTE and data aggregation protocols in existing heterogeneous wireless networks.

In Chapter 4, we show that the coupled constraints in the virtual network embedding prob-

lem make it intractable. Instead of over-provisioning the physical network and splitting virtual links across multiple paths, we propose VNE techniques that effectively prune the search space, thereby reducing the execution times by avoiding backtracking, while not compromising the quality of the obtained VNE solutions, expressed in terms of acceptance rates. Our simulations show that the likelihood of performing a backtrack-free search is greater than 80%, confirming the effectiveness of the proposed pruning techniques. These techniques are then exploited to design a polynomial-time, $\frac{1}{2}$-approximation VNE algorithm. We show analytically and empirically that the proposed algorithm outperforms MIP-based algorithms in terms of the revenue to cost ratio and the acceptance rate while minimizing the migration cost arising due to the mobility of physical nodes.

In Chapter 5, we have shown the potential of Cloud of Things to scale cloud computing vertically by exploiting sensing resources of IoT devices to provide Sensing as a Service. We have proposed a global architecture that scales Cloud of Things horizontally by employing edge computing platforms in a new role as cloud agents that discover and virtualize sensing resources of IoT devices. We have described cloud agents technical challenges and design objectives for sensing resources discovery and virtualization that can dispatch offering virtual sensor networks deployed on IoT devices to cloud users with in-network processing capabilities. We gave a taxonomy of the potential sensing tasks, their applications, and there challenges. We have proposed our sensing resource discovery solution based on a gossip policy to discover sensing resrouces as fast as possible and RADV: our virtualization solution. We have shown through analysis and simulations the potential of RADV to achieve reduced communication overhead, low complexity, and closeness to optimal such that RADV employs minimal physical resources in devices virtualization with maximal benefit. We also proposed RADE for distributed consensus estimation as we believe it is one major sensing task in Sensing as a Service. Using simulation, we show that RADE reduces the communication overhead significantly without compromising the estimation error when compared to the traditional ADMM algorithm. We also show that the convergence time of our proposed algorithms maintain linear convergence behavior, as in the case of conventional ADMM.

In Chapter 6, we propose Flock; a simple autonomous VMs migration protocol. Flock considers the peer-to-peer interaction of VMs in heterogeneous edge and conventional cloud platforms. We show that Flock converges to a Nash equilibrium with $(1+\epsilon)$ PoA. Flock minimizes the average latency of VMs as a generic goal with diverse use cases and can be easily redesigned to serve other purposes such as load-balancing and energy efficiency.

In Chapter 7, we propose FogMQ, a message broker and cloning system for scalable and geographically distributed analytics near and at the network edge. FogMQ uses Flockto enable device clones to automously migrate across hetrogenous cloud/edge platforms. FogMQ servers expose tomophgraphy functionalities that enables devices clones to take migration decisions without complete knowledge about the hosting platform. This feature suites the deployability and scalability challenges accorss WANs and the Internet. We discuss the prototying of FogMQ and its evaluation of a virtual and geographically distributed testebed using public cloud resources from AWS-EC2. In our functional evaluation, we demonestrated the stability of Flock, and the ability of FogMQ to improve the tail latency by one third of that attained by a widley used architecture in which device communicate through the nearest message broker. In our design we not only take into account devices' location, but also it traffic patter and communication relationships with other devices and cloud services.

In Chapter 8, we propose GROUP to enhance the flexibility of the middleware solutions for IoT use cases. GROUP allows for the dynamic creation and managed change of overlay networks of application services that are permitted to interact for certain purposes. We used the authorization and obligation infrastructure of the XACML 3.0 policies to securely and scalably orchestrate services interactions. In the future, orchestrators can act as a first layer PDP for scalability. The PIPs can also trigger asynchronous events to notify the services with updates to their queries. The orchestrator could also cache partitions of the PIP store to reduce services discovery time. With GROUP orchestration, flexible policy enforcement, devices and application services come/go or move around and GROUP automatically adapts its application overlay networks.

In Chapter 9, We show the advantages of experimenting new architectures and systems on existing public cloud is the inherent support of massive capacity, several geographical locations, and prototyping for challenges in real cloud platforms and the Internet. Configuring flat networks is relatively straightforward with VXLAN tunnels, however the challenge is to do it with distributed control that can dynamically learn about changes in the testbed. For this reason, we believe that Beelet is a very good starting point for virtual testbeds in heterogeneous cloud environments that can be administered by multiple authorities. We enhanced the flexibility of creating such virtual testbeds so that it can be organized in any virtual topology and demonstrated its usage model through creating virtual testbeds for FogMQ.

10.1.1 Thesis Contributions

This thesis develops prototypes for efficient device cloning and device-to-device communication in heterogeneous Edge and Cloud Platforms. The decision to pursue this prototype implementation is based on the studies and lessons learned from designing and developing on-demand virtual network embedding algorithms that the thesis presents as BirdVNE and RADV in Chapter 4 and Chapter 5 respectively.

The limitation of BirdVNE and RADV, particularly their suitability for realization in a shared cloud environment, motivated the development of Flocking Virtual Machines as a key algorithm for optimizing dynamic overlay networks for objectives such as minimizing latency, improving energy consumption, or balancing load in cloud and edge environments. The thesis also discovers hidden sparsity in networks of massive number of devices, and shows the performance and economic benefits of designing device cloning protocols that exploit this sparsity through the compressive sensing theory. Finally, the thesis conjecture the concept of Cloud of Things that scales up cloud platforms with sensing resources of the highly dynamic and growing Internet of Things (IoT) to enable on-demand remote sensing.

The following points enumerate the thesis contributions:

- BirdVNE: Efficiently embeds on-demand virtual networks onto substrate networks using the theory of constrained optimization. Bird-VNE has applications in: cloud resource allocation, sensor network virtualization, and data-parallel applications.
- RADV: Distributed resource discovery and network virtualization using gossip protocols and distributed optimization. RADV has applications in: sensing as a service, scheduling data-parallel applications in Edge computing, and resource provisioning in heterogeneous clouds.
- REPLISOM: Efficient memory replication protocol for massive number of IoT devices in LTE/LTE-A edge clouds using the theory of compressive sensing and machine type communication. Replisom introduces native cloud procedures design in wireless networks protocols.
- Flocking Virtual Machines: A simple and scalable protocol that enables live migration of Virtual Machines (VMs) across heterogeneous edge and conventional cloud platforms to improve the responsiveness of cloud services.
- FogMQ: A message broker system that minimizes the tail-latency of devices communicating in heterogeneous Edge and Cloud platforms.

10.2 Future Work

The research results presented in this thesis lead to several theoretical and experimental research questions.

Flock theoretical results and applications: Our simulation results - in Chapter 6 - suggest that Flock scales better than $O(n)$ on average. A formal proof of this result will fortify the understanding of the expected Flock complexity in addition to its worst case complexity. Flock algorithm shall be applied to applications other than overlay network optimization and virtual network embedding. For example, a centralized implementation of Flock shall optimize resource assignment in wireless networks with the objective to minimize total wireless interference. Fortunately, the numerical properties of inter-device interference aligns with the utility function properties in Flock (monotonic, and reciprocal). Moreover, Flock suits the design requirements of several emerging applications such as service chaining, cloud-based radio access networks, and content delivery networks.

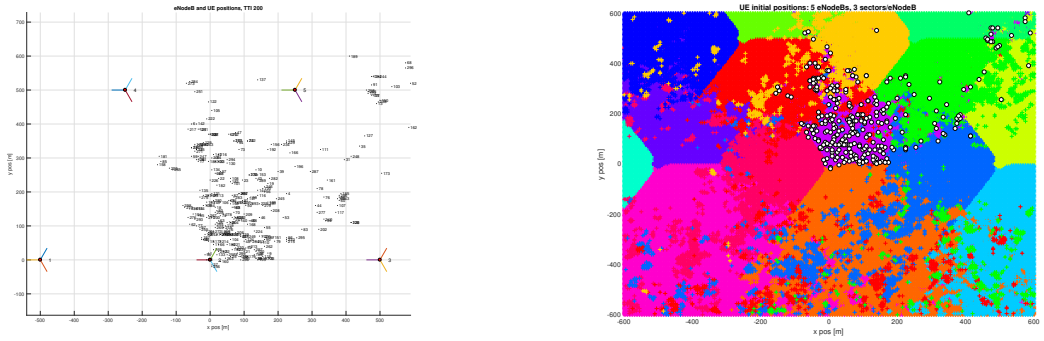
REPLISOM for many-to-many relations: The thesis provides an in-depth study of REPLISOM in a one-to-many architecture, where one edge cloud replicates the memory of many devices. Nevertheless, REPLISOM is extendable to a many-to-many architecture, where several replicating nodes pulls devices replica and reach consensus about devices memory contents. This suits applications where consensus is required besides the replication processes. It remains unclear though if REPLISOM brings performance, reliability, or security advantages compared to existing consensus algorithms.

REPLISOM and Flock performance over other transport networks: The thesis focus the design of REPLISOM for 3GPP and WiFi-direct networks, and the implementation of Flock for wired networks. Extending the applicability of both protocols requires extensive performance bench-marking of the developed prototypes for not only 3GPP networks but also for other technologies such as: low-power Bluetooth, WiFi, ZigBee, or IEEE 802.19 standard. Additionally, studying Flock performance when deployed over wireless networks shall bring new insights that particularly aid in the development of reliable network tomography (e.g. latency measurements) in wireless environments.

Network tomography in shared cloud environments: Effective and efficient bandwidth measurements are the main setbacks in prototyping Bird-VNE and RADV. Although that this limitation motivated the development of Flock, it remains important to develop efficient mechanisms for network tomography where the network and host resources are shared by tenants whose resource usage are unknown and cannot be inferred. The newly developed techniques shall be different from existing tools and systems, in the sense that the assumption that a single authority manages the entire infrastructure across all the network layers is not valid. These new methods shall also avoid approaches that attempt to infer resource usage by other tenants, as these approaches violate fundamental security principles in shared cloud environments and shall not be used in practice.

APPENDICES

Appendix A: LTE System Level Simulations for REPLISOM numerical models



(a) Initial devices positions

(b) Devices positions with respect to site coverage.

Figure A.1: Simulation Setup

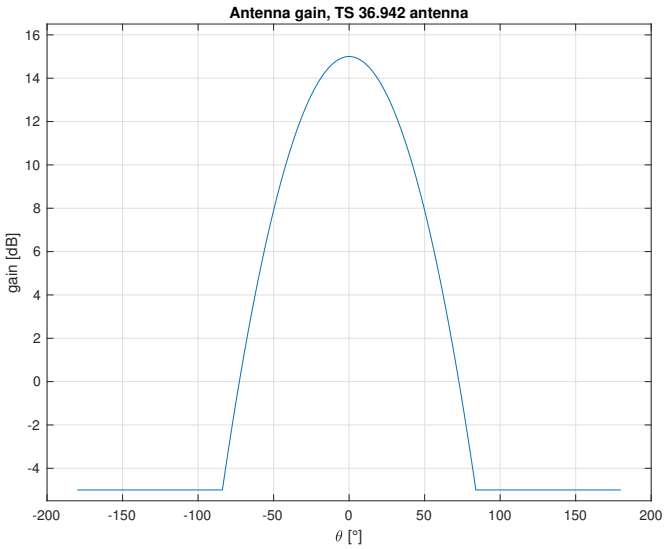


Figure A.2: Used antenna pattern per cell-sector.

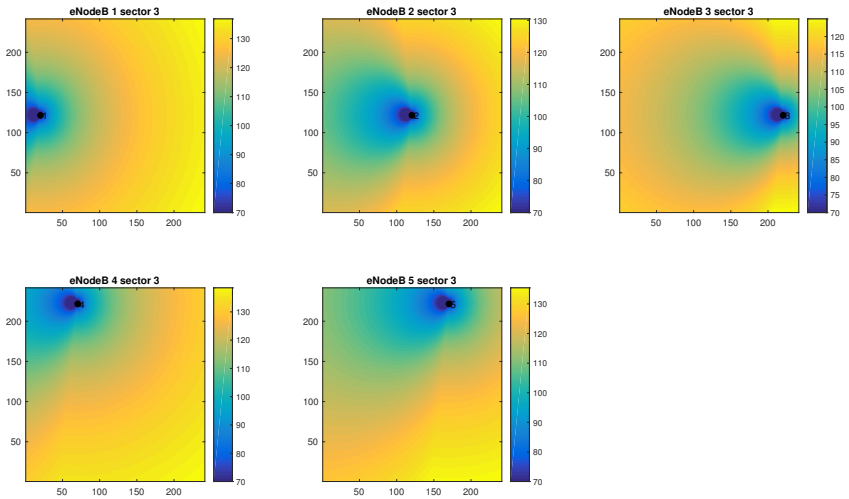


Figure A.3: Sectorized coverage map given the sites position and the used antenna pattern.

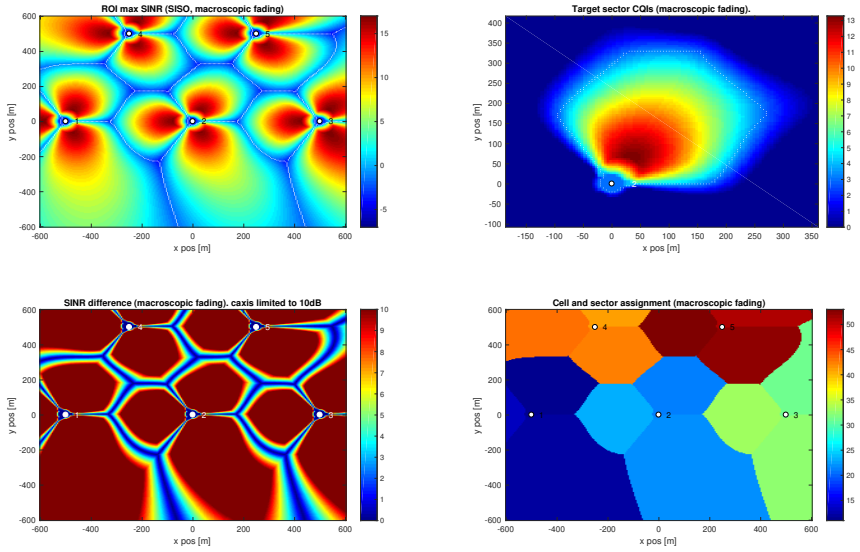


Figure A.4: Detailed coverage map showing CQI, SNR, and SNR difference in the region of interest (sector-2).

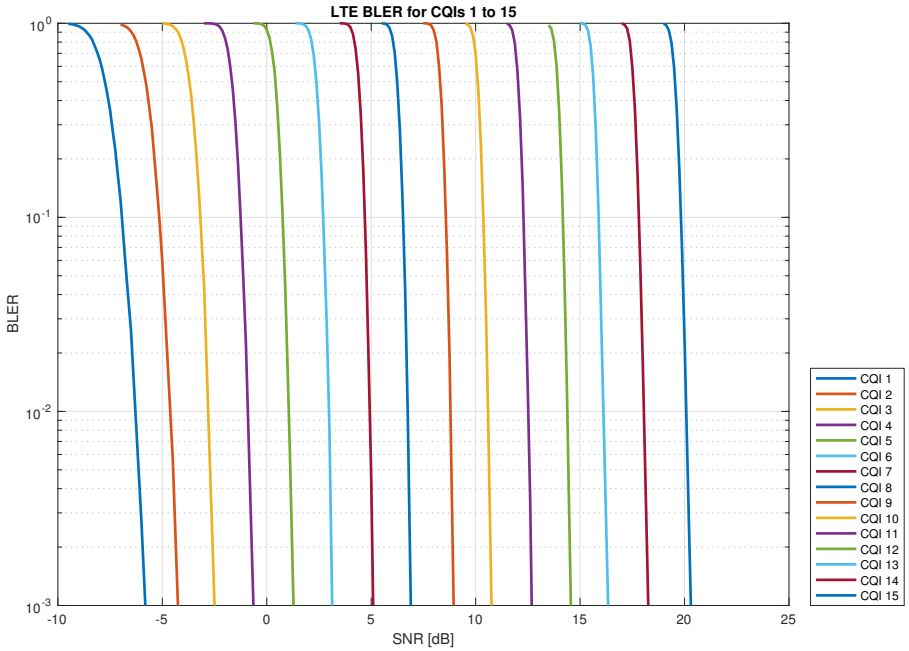


Figure A.5: BLER model.

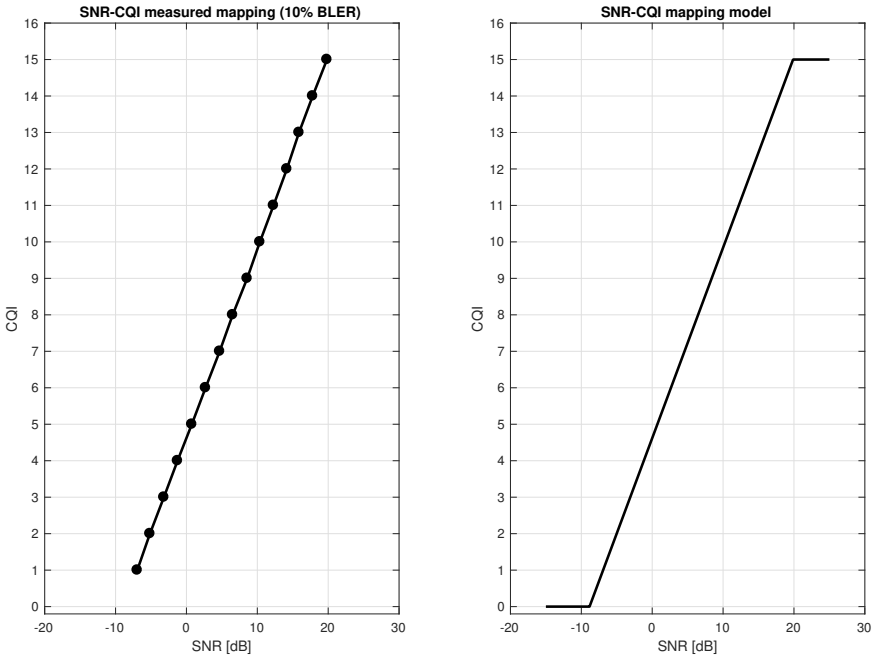


Figure A.6: CQI and SNR model and relationship.

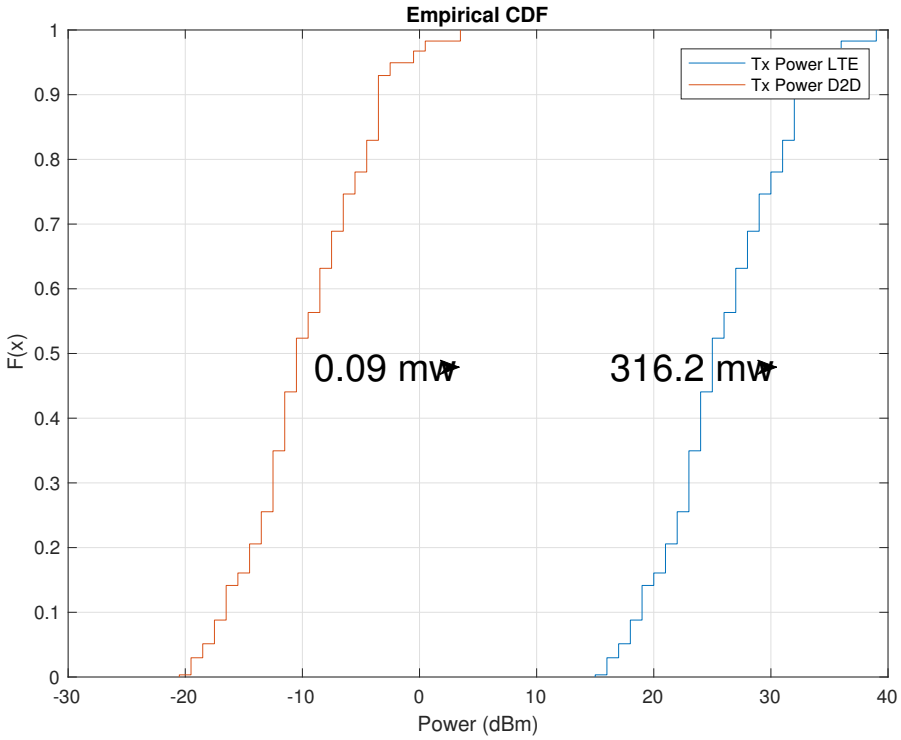


Figure A.7: Emperical CDF of transmit power in cell and device-to-device modes

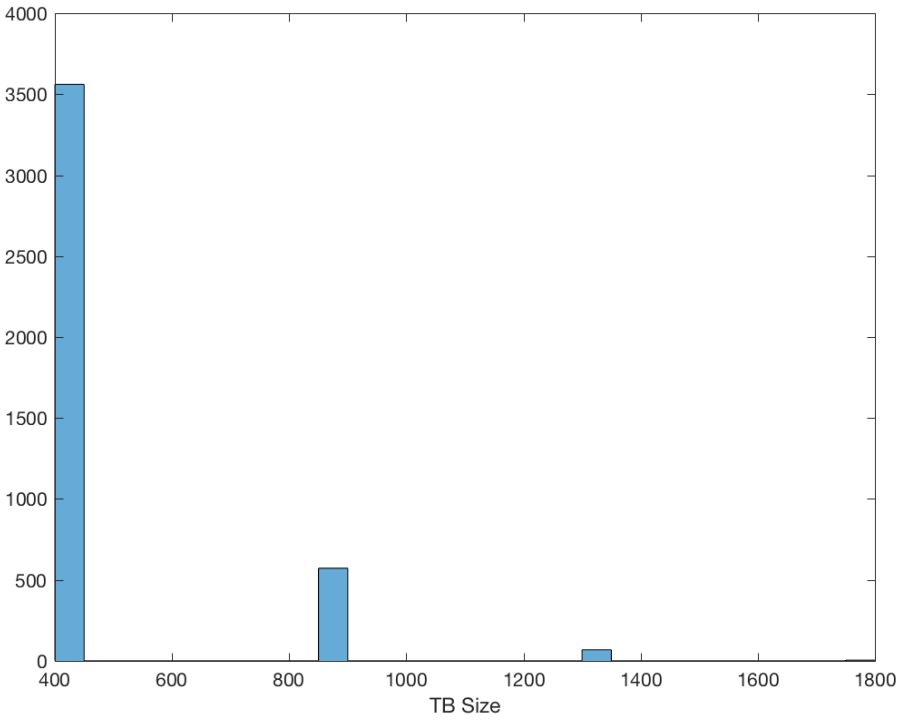


Figure A.8: Transport block size statistics through the entire simulations.

Appendix B: Blueprint: Extending Beelet capabilities to support CloudLab/GENI

The Beelet virtual testbed supports only AWS and GCE cloud resources. This makes our proposed techniques undeployable on GENI/CloudLab. In this appendix, we describe how we extend Beelet's capabilities to deploy experiments on GENI/CloudLab resources and use it to profile real inter-cloud delays and utilization.

A major barrier that constraints the usage of GENI/CloudLab is the substantial gap in design between Beelet and the implemented proof-of-concepts of our techniques compared to GENI Operations and Management APIs. GENI uses low level compartments to support a wide spectrum of basic components (e.g. an edge computer, customizable router, programmable access point, or optical link) in aggregates.

However, the main building block of Beelet is a cloud platform provisioning APIs that can be easily used to create virtual machines as testbed workers (Beelets basic testbed component) that executes the proof-of-concept implementations of our techniques. In order to use GENI operations and management tools (e.g. omni, geni-lib, and RSpec), we need to architect, design, develop, and test Beelet and the developed proof-of-concepts to use these basic GENI building blocks from scratch in a very lengthy development cycle. Fortunately, recent advancements in ExoGENI, and CloudLab allows the definition of in-place resource specification that we can use to deploy highly-configurable OpenStack (an open source cloud platform for Infrastructure as a Service) instances as GENI resources.

In this appendix, we give a blueprint on how to use OpenStack nova to augment the Public Cloud and Remote Configuration module of Beelet and extend its usability to support CloudLab. Figure B.1 shows a simple and effective modification that allows using Beelet not only on GENI/CloudLab bare-metal resources, but also on any other OpenStack-based public cloud provider (e.g. RackSpace, Mirantis, Redhat, or IBM), free OpenStack clouds, or even on a single laptop using devstack. The model-correction and validation is a placeholder for a module for network and host tomography (i.e. bandwidth, and processing metrics) of virtual machines in public cloud environments, where its development is a parallel research topic that's outside the scope of this thesis.

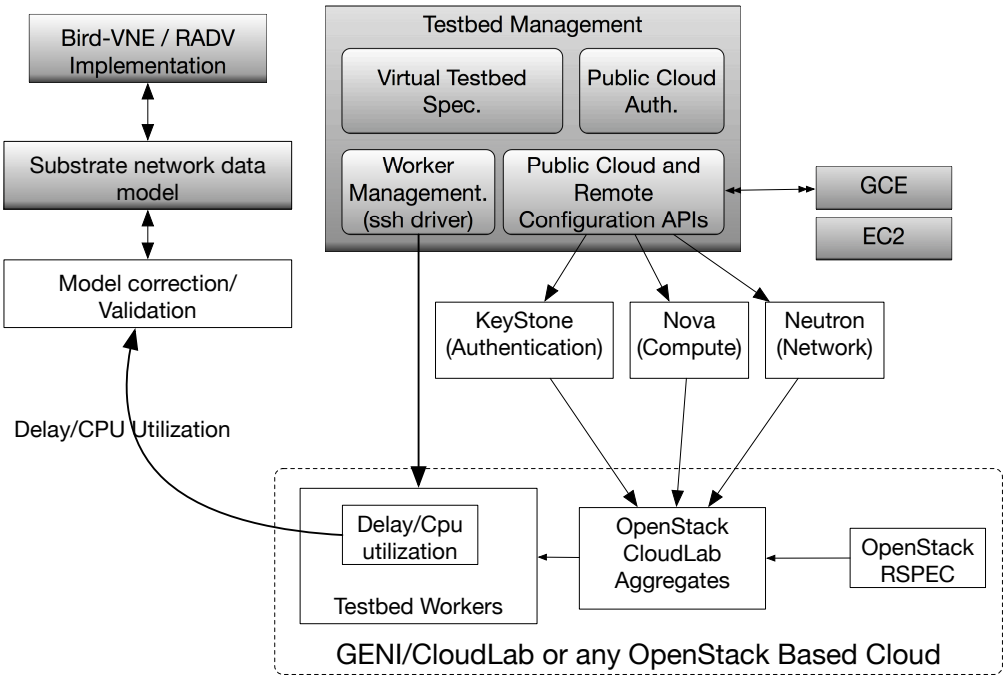


Figure B.1: Software architecture to extend Beelet functionality to support CloudLab via OpenStack APIs.

List of Acronyms

ADMM Alternating Direction Method of Multipliers

API Application Programming Interface

CDF Cumulative Distribution Function

DNS Domain Name Service

EC Edge Cloud

IoT Internet of Things

IP Internet Protocol

LS Least Squares

ML Maximum-Likelihood

MSE Mean Square Error

NE Nash Equilibrium

PAP Policy Administration Point

PDP Policy Decision Point

PEP Policy Enforcement Point

PIP Policy Information Point

PoA Price of Anarchy

QoS Quality of Service

RADE Randomized and Asynchronous Distributed Estimation

RADV Randomized and Asynchronous Distributed Virtualization

RPC Remote Procedure Call

RTT Round Trip Time

SDN Software Defined Networking

SLA Service Level Agreement

TCP Transport Control Protocol

VM Virtual Machine

VNI VXLAN Network Identifier

VXLAN Virtual Extensible LAN

WAN Wide Area Network

XACML 3.0 eXtensible Access Control Markup Language Version 3.0

Bibliography

[1] 3GPP. LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures. TS TS 36.213, 3rd Generation Partnership Project (3GPP), October 2010.

[2] 3GPP. EUTRA Medium Access Control (MAC) protocol specification. TS TS 36.321, 3rd Generation Partnership Project (3GPP), September 2011.

[3] 3GPP. LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Physical channels and modulation. TS TS 36.211, 3rd Generation Partnership Project (3GPP), January 2011.

[4] 3GPP. Service requirements for Machine-Type Communications (MTC); (Release 13). TS TS 22.368, 3rd Generation Partnership Project (3GPP), December 2014.

[5] 3GPP. Study on RAN Improvements for Machine-type Communications; (Release 11). TR TR 37.868, 3rd Generation Partnership Project (3GPP), February 2014.

[6] 3GPP. Cellular System Support for Ultra Low Complexity and Low Throughput Internet of Things; (Release 13). TR Draft 45.8xx, 3rd Generation Partnership Project (3GPP), March 2015.

[7] Sherif Abdelwahab and Bechir Hamdaoui. Fogmq: A message broker system for enabling distributed, internet-scale iot applications over heterogeneous cloud platforms. *arXiv preprint arXiv:1610.00620*, 2016.

[8] Sherif Abdelwahab, Bechir Hamdaoui, and Mohsen Guizani. Bird-vne: Backtrack-avoidance virtual network embedding in polynomial time. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 4983–4989. IEEE, 2014.

[9] Sherif Abdelwahab, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. Enabling smart cloud services through remote sensing: An internet of everything enabler. *Internet of Things Journal, IEEE*, 1(3):276–288, June 2014.

[10] Sherif Abdelwahab, Bechir Hamdaoui, Mohsen Guizani, and Taieb Znati. Cloud of things for sensing as a service: Sensing resource discovery and virtualization. In *Global Telecommunications Conference (GLOBECOM 2015), 2015 IEEE*. IEEE, 2015.

[11] Sherif Abdelwahab, Bechir Hamdaoui, Mohsen Guizani, and Taieb Znati. Efficient virtual network embedding with backtrack avoidance for dynamic wireless networks. *IEEE Transactions on Wireless Communications*, 15(4):2669–2683, 2016.

[12] Sherif Abdelwahab, Bechir Hamdaoui, Mohsen Guizani, and Taieb Znati. Replisom: Disciplined tiny memory replication for massive iot devices in lte edge cloud. *IEEE Internet of Things Journal*, 3(3):327–338, 2016.

[13] Mohamed Abu Sharkh, Manar Jammal, Abdallah Shami, and Abdelkader Ouda. Resource allocation in a network-based cloud computing environment: design challenges. *Communications Magazine, IEEE*, 51(11):46–52, 2013.

[14] Nesreen K Ahmed, Nick Duffield, Jennifer Neville, and Ramana Kompella. Graph sample and hold: A framework for big-graph analytics. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1446–1455. ACM, 2014.

[15] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *ACM SIGCOMM computer communication review*, volume 40, pages 63–74. ACM, 2010.

[16] Hamada Alshaer. An overview of network virtualization and cloud network as a service. *International Journal of Network Management*, 25(1):1–30, 2015.

[17] Ahmed Amokrane, Mohamed Faten Zhani, Rami Langar, Raouf Boutaba, and Guy Pujolle. Greenhead: Virtual data center embedding across distributed infrastructures. *Cloud Computing, IEEE Transactions on*, 1(1):36–49, 2013.

[18] Kyoungho An, Aniruddha Gokhale, Sumant Tambe, and Takayuki Kuroda. Wide area network-scale discovery and data dissemination in data-centric publish/subscribe systems. *Network*, 1(P4P3):P3, 2015.

[19] Sergey Andreev, Anna Larmo, Mikhail Gerasimenko, Vitaly Petrov, Olga Galinina, Tuomas Tirronen, Johan Torsner, and Yevgeni Koucheryavy. Efficient small data access for machine-type communications in lte. In *Communications (ICC), 2013 IEEE International Conference on*, pages 3569–3574. IEEE, 2013.

[20] Mehran Anvari, Tim Broderick, Harvey Stein, Trevor Chapman, Moji Ghodoussi, Daniel W Birch, Craig Mckinley, Patrick Trudeau, Sanjeev Dutta, and Charles H Goldsmith. The impact of latency on surgical precision and task completion during robotic-assisted remote telepresence surgery. *Computer Aided Surgery*, 10(2):93–99, 2005.

[21] Dushyant Arora, Anja Feldmann, Gregor Schaffrath, and Stefan Schmid. On the benefit of virtualization: Strategies for flexible server allocation. In *Proc. USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, 2011.

[22] Arash Asadi and Vincenzo Mancuso. Wifi direct and lte d2d in action. In *Wireless Days (WD), 2013 IFIP*, pages 1–8. IEEE, 2013.

[23] Arash Asadi, Qing Wang, and Vincenzo Mancuso. A survey on device-to-device communication in cellular networks. pages 1801–1819, 2014.

[24] Luigi Atzori, Antonio Iera, and Giacomo Morabito. Siot: Giving a social structure to the internet of things. *Communications Letters, IEEE*, 15(11):1193–1195, 2011.

[25] Rajesh Balan, Jason Flinn, Mahadev Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. The case for cyber foraging. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 87–92. ACM, 2002.

[26] Roberto Baldoni, Roberto Beraldi, Leonardo Querzoni, and Antonino Virgillito. Efficient publish/subscribe through a self-organizing broker overlay and its application to siena. *The Computer Journal*, 50(4):444–459, 2007.

[27] Paul Balister, Béla Bollobás, Amites Sarkar, and Mark Walters. Highly connected random geometric graphs. *Discrete Applied Mathematics*, 157(2):309–320, 2009.

[28] 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.

[29] Abdeltouab Belbekkouche, Md Hasan, Ahmed Karmouch, et al. Resource discovery and allocation in network virtualization. *Communications Surveys & Tutorials, IEEE*, 14(4):1114–1128, 2012.

[30] Radu Berinde, Anna C Gilbert, Piotr Indyk, Howard Karloff, and Martin J Strauss. Combining geometry and combinatorics: A unified approach to sparse signal recovery. In *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, pages 798–805. IEEE, 2008.

[31] Mark Berman, Jeffrey S Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61:5–23, 2014.

[32] Christian Bettstetter. On the minimum node degree and connectivity of a wireless multihop network. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 80–91. ACM, 2002.

[33] Kshipra Bhawalkar, Martin Gairing, and Tim Roughgarden. Weighted congestion games: The price of anarchy, universal worst-case examples, and tightness. *ACM Transactions on Economics and Computation*, 2(4):14, 2014.

[34] 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*, pages 169–186. Springer, 2014.

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

[36] Vivek S Borkar. Stochastic approximation with controlled markovnoise. *Systems & control letters*, 55(2):139–145, 2006.

[37] Juan Felipe Botero, Xavier Hesselbach, Michael Duelli, Daniel Schlosser, Andreas Fischer, and Hermann De Meer. Energy efficient virtual network embedding. *Communications Letters, IEEE*, 16(5):756–759, 2012.

[38] Juan Felipe Botero, Xavier Hesselbach, Andreas Fischer, and Hermann De Meer. Optimal mapping of virtual networks with hidden hops. *Telecommunication Systems*, 51(4):273–282, 2012.

[39] Juan Felipe Botero, Miguel Molina, Xavier Hesselbach-Serra, and José Roberto Amazonas. A novel paths algebra-based strategy to flexibly solve the link mapping stage of vne problems. *Journal of Network and Computer Applications*, 36(6):1735–1752, 2013.

[40] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *Information Theory, IEEE Trans. on*, 52(6), 2006.

[41] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[42] César Cañas, Eduardo Pacheco, Bettina Kemme, Jörg Kienzle, and Hans-Arno Jacobsen. Graps: A graph publish/subscribe middleware. In *Proceedings of the 16th Annual Middleware Conference*, pages 1–12. ACM, 2015.

[43] Nuno Carvalho, Filipe Araujo, and Luis Rodrigues. Scalable qos-based event routing in publish-subscribe systems. In *Network Computing and Applications, Fourth IEEE International Symposium on*, pages 101–108. IEEE, 2005.

[44] Chen Chen, Hans-Arno Jacobsen, and Roman Vitenberg. Algorithms based on divide and conquer for topic-based publish/subscribe overlay design. *IEEE/ACM Transactions on Networking*, 2015.

[45] Chen Chen and Yoav Tock. Design of routing protocols and overlay topologies for topic-based publish/subscribe on small-world networks. In *Proceedings of the Industrial Track of the 16th International Middleware Conference*, page 2. ACM, 2015.

[46] Chen Chen, Yoav Tock, Hans-Arno Jacobsen, and Roman Vitenberg. Weighted overlay design for topic-based publish/subscribe systems on geo-distributed data centers. In *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*, pages 474–485. IEEE, 2015.

[47] Chen Chen, Roman Vitenberg, and Hans-Arno Jacobsen. A generalized algorithm for publish/subscribe overlay design and its fast implementation. In *Distributed Computing*, pages 76–90. Springer, 2012.

[48] Liuhua Chen, Haiying Shen, and Karan Sapra. Distributed autonomous virtual resource management in datacenters using finite-markov decision process. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–13. ACM, 2014.

[49] Xiang Cheng, Sen Su, Zhongbao Zhang, Hanchi Wang, Fangchun Yang, Yan Luo, and Jie Wang. Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM Computer Communication Review*, 41(2):38–47, 2011.

[50] Ron C Chiang and H Howie Huang. Tracon: interference-aware scheduling for data-intensive applications in virtualized environments. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 47. ACM, 2011.

[51] Steve Chien and Alistair Sinclair. Convergence to approximate nash equilibria in congestion games. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 169–178. Society for Industrial and Applied Mathematics, 2007.

[52] Kideok Cho, Munyoung Lee, Kunwoo Park, Ted Taekyoung Kwon, Yanghee Choi, and Sangheon Pack. Wave: Popularity-based and collaborative in-network caching for content-oriented networks. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pages 316–321. IEEE, 2012.

[53] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Tran. on Networking*, 20(1), 2012.

[54] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[55] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.

[56] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[57] Rina Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

[58] Camil Demetrescu and Giuseppe F Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM (JACM)*, 51(6):968–992, 2004.

[59] Mohamed Diallo, Serge Fdida, Vasilis Sourlas, Paris Flegkas, and Leandros Tassiulas. Leveraging caching for internet-scale content-based publish/subscribe networks. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1–5. IEEE, 2011.

[60] Chong Ding, Bi Song, Akshay Morye, Jay Farrell, Amit K Roy-Chowdhury, et al. Collaborative sensing in a distributed ptz camera network. *Image Processing, IEEE Transactions on*, 21(7):3282–3295, 2012.

[61] Jack J Dongarra, Piotr Luszczek, and Antoine Petitet. The linpack benchmark: past, present and future. *Concurrency and Computation: practice and experience*, 15(9):803–820, 2003.

[62] David L Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4):1289–1306, 2006.

[63] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM (JACM)*, 61(1):1, 2014.

[64] Patricia Takako Endo, Andre Vitor de Almeida Palhares, Nadilma Nunes Pereira, Glauco Estacio Goncalves, Djamel Sadok, Judith Kelner, Bob Melander, and Jan-Erik Mångs. Resource allocation for distributed cloud: concepts and research challenges. *Network, IEEE*, 25(4):42–46, 2011.

[65] Flavio Esposito and Francesco Chiti. Distributed consensus-based auctions for wireless virtual network embedding. In *Communications (ICC), 2015 IEEE International Conference on*, pages 472–477. IEEE, 2015.

[66] Flavio Esposito and Ibrahim Matta. A decomposition-based architecture for distributed virtual network embedding. In *Proceedings of the 2014 ACM SIGCOMM workshop on Distributed cloud computing*, pages 53–58. ACM, 2014.

[67] Floriana Esposito, Donato Di Paola, and Ibrahim Matta. On distributed virtual network embedding with guarantees. *Transactions on Networking*, 2014.

[68] ETSI. Mobile-Edge Computing. Technical white paper, European Telecommunications Standards Institute (ETSI), September 2014.

[69] Fatemeh Fazel, Maryam Fazel, and Milica Stojanovic. Random access compressed sensing for energy-efficient underwater sensor networks. *Selected Areas in Communications, IEEE Journal on*, 29(8):1660–1670, 2011.

[70] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*, pages 171–172. IEEE, 2015.

[71] David Ferraiolo, Ramaswamy Chandramouli, Rick Kuhn, and Vincent Hu. Extensible access control markup language (xacml) and next generation access control (ngac). In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*, pages 13–24. ACM, 2016.

[72] Andreas Fischer, Juan Felipe Botero, M Till Beck, Hermann De Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *Communications Surveys & Tutorials, IEEE*, 15(4):1888–1906, 2013.

[73] Andreas Fischer, Juan Felipe Botero Vega, Michael Duelli, Daniel Schlosser, Xavier Hesselbach Serra, Hermann De Meer, et al. Alevin-a framework to develop, compare, and analyze virtual network embedding algorithms. 2011.

[74] Jason Flinn. Cyber foraging: Bridging mobile and cloud computing. *Synthesis Lectures on Mobile and Pervasive Computing*, 7(2):1–103, 2012.

[75] Gábor Fodor, Erik Dahlman, Gunnar Mildh, Stefan Parkvall, Norbert Reider, György Miklós, and Zoltán Turányi. Design aspects of network assisted device-to-device communications. *Communications Magazine, IEEE*, 50(3):170–177, 2012.

[76] Dimitris Fotakis, Spyros Kontogiannis, Elias Koutsoupias, Marios Mavronicolas, and Paul Spirakis. The structure and complexity of nash equilibria for a selfish routing game. In *Automata, Languages and Programming*, pages 123–134. Springer, 2002.

[77] Dimitris Fotakis, Spyros Kontogiannis, and Paul Spirakis. Selfish unsplittable flows. *Theoretical Computer Science*, 348(2):226–239, 2005.

[78] Abdurrahman Fouda, Ahmed N Ragab, Ali Esswie, Mohamed Marzban, Amr Naser, Mohamed Rehan, and Ahmed S Ibrahim. Real time video streaming over ns3 based emulated lte networks. *Int. J. Electr. Commun. Comput. Technol.(IJECCT)*, 4(3), 2014.

[79] Federica Garin and Luca Schenato. A survey on distributed estimation and control applications using linear consensus algorithms. In *Networked Control Systems*, pages 75–107. Springer, 2010.

[80] Mikhail Gerasimenko, Vitaly Petrov, Olga Galinina, Sergey Andreev, and Yevgeni Koucheryavy. Energy and delay analysis of lte-advanced rach performance under mtc overload. In *Globecom Workshops (GC Wkshps), 2012 IEEE*, pages 1632–1637. IEEE, 2012.

[81] 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.

[82] Anna Gilbert and Piotr Indyk. Sparse recovery using sparse matrices. Institute of Electrical and Electronics Engineers, 2010.

[83] Sébastien Godard. Sysstat: System performance tools for the linux os, 2004.

[84] Long Gong, Yonggang Wen, Zuqing Zhu, and Taewoo Lee. quanitsubstrate. In *INFOCOM, 2014 Proceedings IEEE*, pages 1–9. IEEE, 2014.

[85] Antonis G Gotsis, Athanasios S Lioumpas, and Angeliki Alexiou. M2m scheduling over lte: Challenges and new perspectives. *Vehicular Technology Magazine, IEEE*, 7(3):34–39, 2012.

[86] Monowar Hasan, Ekram Hossain, and Dusit Niyato. Random access for machine-to-machine communication in lte-advanced networks: issues and approaches. *Communications Magazine, IEEE*, 51(6):86–93, 2013.

[87] Jarvis Haupt, Waheed U Bajwa, Michael Rabbat, and Robert Nowak. Compressed sensing for networked data. *Signal Processing Magazine, IEEE*, 25(2):92–101, 2008.

[88] Michael R Hines and Kartik Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 51–60. ACM, 2009.

[89] Pieter Hintjens. *ZeroMQ: Messaging for Many Applications*. " O'Reilly Media, Inc.", 2013.

[90] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. Mobile fog: A programming model for large-scale applications on the internet of things. In *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, pages 15–20. ACM, 2013.

[91] John E Hopcroft and Richard M Karp. An nˆ5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

[92] Md Endadul Hoque, Hyojeong Lee, Rahul Potharaju, Charles E Killian, and Cristina Nita-Rotaru. Adversarial testing of wireless routing implementations. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, pages 143–148. ACM, 2013.

[93] Ines Houidi, Wajdi Louati, Djamal Zeghlache, Panagiotis Papadimitriou, and Laurent Mathy. Adaptive virtual network provisioning. In *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, pages 41–48. ACM, 2010.

[94] Ines Houidi and Djamal Zeghlache. Exact adaptive virtual network embedding in cloud environments. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2013 IEEE 22nd International Workshop on*, pages 319–323. IEEE, 2013.

[95] Junxian Huang, Feng Qian, Alexandre Gerber, Z Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A close examination of performance and power characteristics of 4g lte networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 225–238. ACM, 2012.

[96] Raj Jain and Subharthi Paul. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine*, 51(11):24–31, 2013.

[97] Abdallah Jarray and Ahmed Karmouch. Cost-efficient mapping for fault-tolerant virtual networks. *Computers, IEEE Transactions on*, 64(3):668–681, 2015.

[98] Mike Jia, Jiannong Cao, and Weifa Liang. Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. *Cloud Computing, IEEE Transactions on*, PP(99), 2015.

[99] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[100] Sachin Katti, Saurabh Shintre, Sidharth Jaggi, Dina Katabi, and Muriel Medard. Real network codes. In *Proc. Forty-Fifth Annual Allerton Conference*, pages 389–395, 2007.

[101] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[102] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953. IEEE, 2012.

[103] Akane Koto, Hiroshi Yamada, Kei Ohmura, and Kenji Kono. Towards unobtrusive vm live migration for cloud computing platforms. In *Proceedings of the Asia-Pacific Workshop on Systems*, page 7. ACM, 2012.

[104] Emmanouil Koukoumidis, Dimitrios Lymberopoulos, Karin Strauss, Jie Liu, and Doug Burger. Pocket cloudlets. *ACM SIGPLAN Notices*, 47(4), 2012.

[105] Mads Darø Kristensen. Execution plans for cyber foraging. In *Proceedings of the 1st workshop on Mobile middleware: embracing the personal communication device*, page 2. ACM, 2008.

[106] CS Kubrusly and J Gravier. Stochastic approximation algorithms and applications. In *Decision and Control including the 12th Symposium on Adaptive Processes, 1973 IEEE Conference on*, volume 12, pages 763–766. IEEE, 1973.

[107] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[108] Nicholas D Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T Campbell. A survey of mobile phone sensing. *Communications Magazine, IEEE*, 48(9):140–150, 2010.

[109] Jay Lee, Hung-An Kao, and Shanhu Yang. Service innovation and smart analytics for industry 4.0 and big data environment. *Procedia Cirp*, 16:3–8, 2014.

[110] Namyoon Lee, Xingqin Lin, J Andrews, and R Heath. Power control for d2d underlaid cellular networks: Modeling, algorithms and analysis. pages 1 – 13, 2013.

[111] Sungwon Lee, Sundeep Pattem, Maheswaran Sathiamoorthy, Bhaskar Krishnamachari, and Antonio Ortega. Spatially-localized compressed sensing and routing in multi-hop sensor networks. In *Geosensor Networks*, pages 11–20. Springer, 2009.

[112] Aris Leivadeas, Chrysa Papagianni, and Symeon Papavassiliou. Socio-aware virtual network embedding. *Network, IEEE*, 26(5):35–43, 2012.

[113] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 1–14. ACM, 2010.

[114] Xingqin Lin, Jeffrey G Andrews, Amitabha Ghosh, and Rapeepat Ratasuk. An overview of 3gpp device-to-device proximity services. *Communications Magazine, IEEE*, 52(4):40–48, 2014.

[115] Xingqin Lin, R Ganti, Philip Fleming, and J Andrews. Towards understanding the fundamentals of mobility in cellular networks. 2012.

[116] Jens Lischka and Holger Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proc. of ACM workshop on Virtualized infrastructure systems and architectures*. ACM, 2009.

[117] Shihmin Lo, Moataz Ammar, Ellen Zegura, and Marwan Fayed. Virtual network migration on real infrastructure: A planetlab case study. In *Networking Conference, 2014 IFIP*, pages 1–9. IEEE, 2014.

[118] Anil Madhavapeddy and Satnam Singh. Reconfigurable data processing for clouds. In *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, pages 141–145. IEEE, 2011.

[119] Mallik Mahalingam, D Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T Sridhar, Mike Bursell, and Chris Wright. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. Technical report, 2014.

[120] Clifford F Mass and Luke E Madaus. Surface pressure observations from smartphones: A potential revolution for high-resolution weather prediction? *Bulletin of the American Meteorological Society*, 95(9):1343–1349, 2014.

[121] 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.

[122] Miguel Matos, Ana Nunes, Rui Oliveira, and José Pereira. Stan: exploiting shared interests without disclosing them in gossip-based publish/subscribe. In *IPTPS*, page 9, 2010.

[123] Julian J McAuley and Jure Leskovec. Learning to discover social circles in ego networks. In *NIPS*, volume 2012, pages 548–56, 2012.

[124] Matheus Melo, Susana Sargento, Ulrich Killat, Andreas Timm-Giel, and Jorge Carapinha. Optimal virtual network embedding: Node-link formulation. *Network and Service Management, IEEE Transactions on*, 10(4):356–368, 2013.

[125] Tom Molloy, Zhenhui Yuan, and Gabriel-Miro Muntean. Real time emulation of an lte network using ns-3. In *Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). 25th IET*, pages 251–257. IET, 2013.

[126] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial & Applied Mathematics*, 5(1):32–38, 1957.

[127] Nam Nguyen, Douglas L Jones, and Sudha Krishnamurthy. Netcompress: Coupling network coding and compressed sensing for efficient data communication in wireless sensor networks. In *Signal Processing Systems (SIPS), 2010 IEEE Workshop on*, pages 356–361. IEEE, 2010.

[128] Melih Onus and Andréa W Richa. Minimum maximum-degree publish-subscribe overlay network design. *IEEE/ACM Transactions on Networking (TON)*, 19(5):1331–1343, 2011.

[129] Daniel Pérez Palomar and Mung Chiang. A tutorial on decomposition methods for network utility maximization. *Selected Areas in Communications, IEEE Journal on*, 24(8):1439–1451, 2006.

[130] Gerardo Pardo-Castellote. Omg data-distribution service: Architectural overview. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pages 200–206. IEEE, 2003.

[131] Henning Paul, Jörg Fliege, and Armin Dekorsy. In-network-processing: Distributed consensus-based linear estimation. *Communications Letters, IEEE*, 17(1):59–62, 2013.

[132] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Sensing as a service model for smart cities supported by internet of things. *Transactions on Emerging Telecommunications Technologies*, 25(1):81–93, 2014.

[133] Xing Pu, Ling Liu, Yiduo Mei, Sankaran Sivathanu, Younggyun Koh, Calton Pu, and Yuanda Cao. Who is your neighbor: Net i/o performance interference in virtualized clouds. *Services Computing, IEEE Transactions on*, 6(3):314–329, 2013.

[134] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. Odessa: enabling interactive perception applications on mobile devices. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 43–56. ACM, 2011.

[135] Anshul Rai, Krishna Kant Chintalapudi, Venkata N Padmanabhan, and Rijurekha Sen. Zee: zero-effort crowdsourcing for indoor localization. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 293–304. ACM, 2012.

[136] Umakishore Ramachandran, Kirak Hong, Liviu Iftode, Ramesh Jain, Rajnish Kumar, Kurt Rothermel, Junsuk Shin, and Raghupathy Sivakumar. Large-scale situation awareness with camera networks and multimodal sensing. *Proceedings of the IEEE*, 100(4):878–892, 2012.

[137] Weixiong Rao, Kai Zhao, Yan Zhang, Pan Hui, and Sasu Tarkoma. Towards maximizing timely content delivery in delay tolerant networks. *Mobile Computing, IEEE Transactions on*, 14(4):755–769, 2015.

[138] David Recordon and Drummond Reed. Openid 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, pages 11–16. ACM, 2006.

[139] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *AAAI*, volume 94, pages 362–367, 1994.

[140] Robert Ricci, Eric Eide, and CloudLab Team. Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications. *; login:: the magazine of USENIX & SAGE*, 39(6):36–38, 2014.

[141] Stephen M Rumble, Diego Ongaro, Ryan Stutsman, Mendel Rosenblum, and John K Ousterhout. It's time for low latency. In *HotOS*, volume 13, pages 11–11, 2011.

[142] Nadathur Satish, Narayanan Sundaram, Md Mostofa Ali Patwary, Jiwon Seo, Jongsoo Park, M Amber Hassaan, Shubho Sengupta, Zhaoming Yin, and Pradeep Dubey. Navigating the maze of graph analytics frameworks using massive graph datasets. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 979–990. ACM, 2014.

[143] Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, 2001.

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

[145] Mahadev Satyanarayanan, Zhuo Chen, Kiryong Ha, Wenlu Hu, Wolfgang Richter, and Padmanabhan Pillai. Cloudlets: at the leading edge of mobile-cloud convergence. In *Mobile Computing, Applications and Services (MobiCASE), 2014 6th International Conference on*, pages 1–9. IEEE, 2014.

[146] Mahadev Satyanarayanan, Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, Wenlu Hu, and Brandon Amos. Edge analytics in the internet of things. *IEEE Pervasive Computing*, (2):24–31, 2015.

[147] Douglas C Schmidt and Hans van't Hag. Addressing the challenges of mission-critical information management in next-generation net-centric pub/sub systems with opensplice dds. In *IPDPS*, pages 1–8. Citeseer, 2008.

[148] Devavrat Shah. *Gossip algorithms*. Now Publishers Inc, 2009.

[149] Xiang Sheng, Jian Tang, Xuejie Xiao, and Guoliang Xue. Sensing as a service: Challenges, solutions and future directions. *Sensors Journal, IEEE*, 13(10):3733–3741, 2013.

[150] Cong Shi, Karim Habak, Pranesh Pandurangan, Mostafa Ammar, Mayur Naik, and Ellen Zegura. Cosmos: computation offloading as a service for mobile devices. In *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*, pages 287–296. ACM, 2014.

[151] Christos Siaterlis, Andres Perez Garcia, and Béla Genge. On the use of emulab testbeds for scientifically rigorous experiments. *IEEE Communications Surveys & Tutorials*, 15(2):929–942, 2013.

[152] Gerry Siegemund, Volker Turau, and Kahled Maamra. A self-stabilizing publish/subscribe middleware for wireless sensor networks. In *Networked Systems (NetSys), 2015 International Conference and Workshops on*, pages 1–8. IEEE, 2015.

[153] Sen Su, Zhongbao Zhang, Alex X Liu, Xiang Cheng, Yiwen Wang, and Xinchao Zhao. Energy-aware virtual network embedding. *Networking, IEEE/ACM Transactions on*, 22(5):1607–1620, 2014.

[154] Gang Sun, Vishal Anand, Hong-Fang Yu, Dan Liao, Yanyang Cai, et al. Adaptive provisioning for evolving virtual network request in cloud-based datacenters. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 1617–1622. IEEE, 2012.

[155] Yunlei Sun, Xiuquan Qiao, Bo Cheng, and Junliang Chen. A low-delay, lightweight publish/subscribe architecture for delay-sensitive iot services. In *Web Services (ICWS), 2013 IEEE 20th International Conference on*, pages 179–186. IEEE, 2013.

[156] Tarik Taleb and Adlen Ksentini. Follow me cloud: interworking federated clouds and distributed mobile networks. *Network, IEEE*, 27(5):12–19, 2013.

[157] Paula Tarrío, Ana M Bernardos, and José R Casar. Weighted least squares techniques for improved received signal strength based localization. *Sensors*, 11(9):8569–8592, 2011.

[158] Yuuichi Teranishi, Ryohei Banno, and Toyokazu Akiyama. Scalable and locality-aware distributed topic-based pub/sub messaging for iot. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2015.

[159] Michael Till Beck, Anath Fischer, Hermann de Meer, Juan Felipe Botero, and Xavier Hesselbach. A distributed, parallel, and generic virtual network embedding framework. In *Communications (ICC), 2013 IEEE International Conference on*, pages 3471–3475. IEEE, 2013.

[160] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. Iperf: The tcp/udp bandwidth measurement tool. *htt p://dast. nlanr. net/Projects*, 2005.

[161] Roberto Tron and Rene Vidal. Distributed computer vision algorithms. *Signal Processing Magazine, IEEE*, 28(3):32–45, 2011.

[162] Dimitris Tsolkas, Eirini Liotou, Nikos Passas, and Lazaros Merakos. Lte-a access, core, and protocol architecture for d2d communication. In *Smart Device to Smart Device Communication*, pages 23–40. Springer, 2014.

[163] Rahul Urgaonkar, Shiqiang Wang, Ting He, Murtaza Zafer, Kevin Chan, and Kin K Leung. Dynamic service migration and workload scheduling in edge-clouds. *Performance Evaluation*, 91:205–228, 2015.

[164] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *IEEE International Conference on Cloud Computing*, pages 254–265. Springer, 2009.

[165] Kaiqiang Wang, Minwei Shen, Junguk Cho, Arijit Banerjee, Jacobus Van der Merwe, and Kirk Webb. Mobiscud: A fast moving personal cloud in the mobile network. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, pages 19–24. ACM, 2015.

[166] Wei Wang, Minos Garofalakis, and Kannan Ramchandran. Distributed sparse random projections for refinable approximation. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 331–339. ACM, 2007.

[167] Xin Wang, Prashant Krishnamurthy, and David Tipper. Wireless network virtualization. In *Computing, Networking and Communications (ICNC), 2013 International Conference on*, pages 818–822. IEEE, 2013.

[168] Ermin Wei and Asuman Ozdaglar. On the o (1/k) convergence of asynchronous distributed alternating direction method of multipliers. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pages 551–554. IEEE, 2013.

[169] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

[170] Tim Winter. Rpl: Ipv6 routing protocol for low-power and lossy networks. 2012.

[171] Xuangou Wu, Yan Xiong, Panlong Yang, Shouhong Wan, and Wenchao Huang. Sparsest random scheduling for compressive data gathering in wireless sensor networks. In *IEEE Transactions on Wireless Communications*, volume 13, pages 5867–5877. IEEE, 2014.

[172] Guang Xu, Henning Paul, Dirk Wuebben, and Armin Dekorsy. Fast distributed consensus-based estimation (fast-dice) for cooperative networks. In *Smart Antennas (WSA), 2014 18th International ITG Workshop on*, pages 1–8. VDE, 2014.

[173] Xiaobin Yang, A Fapojuwo, and E Egbogah. Performance analysis and parameter optimization of random access backoff algorithm in lte. In *Vehicular Technology Conference (VTC Fall), 2012 IEEE*, pages 1–5. IEEE, 2012.

[174] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Computer Comm. Review*, 38(2), 2008.

[175] Donggyu Yun, Jungseul Ok, Bongjhin Shin, Soobum Park, and Yung Yi. Embedding of virtual network requests over static wireless multihop networks. *Computer Networks*, 57(5):1139–1152, 2013.

[176] Donggyu Yun and Yung Yi. Virtual network embedding in wireless multihop networks. In *Proceedings of the 6th International Conference on Future Internet Technologies*, pages 30–33. ACM, 2011.

[177] Irene Zhang, Adriana Szekeres, Dana Van Aken, Isaac Ackerman, Steven D Gribble, Arvind Krishnamurthy, and Henry M Levy. Customizable and extensible deployment for mobile/cloud applications. In *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*, pages 97–112. USENIX Association, 2014.

[178] Jinxue Zhang and Zheng Qin. Taprouter: an emulating framework to run real applications on simulated mobile ad hoc network. In *Proceedings of the 44th Annual Simulation Symposium*, pages 39–46. Society for Computer Simulation International, 2011.

[179] Sheng Zhang, Zhuzhong Qian, Jie Wu, Sanglu Lu, and Leah Epstein. Virtual network embedding with opportunistic resource sharing. *Parallel and Distributed Systems, IEEE Transactions on*, 25(3):816–827, 2014.

[180] Yin Zhang, Matthew Roughan, Walter Willinger, and Lili Qiu. Spatio-temporal compressive sensing and internet traffic matrices. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 267–278. ACM, 2009.

[181] Zhongbao Zhang, Xiang Cheng, Sen Su, Yiwen Wang, Kai Shuang, and Yan Luo. A unified enhanced particle swarm optimization-based virtual network embedding algorithm. *Int'l J. of Comm. Systems*, 26(8), 2013.

[182] Ming Zhao and Wenye Wang. A unified mobility model for analysis and simulation of mobile wireless networks. *Wireless Networks*, 15(3):365–389, 2009.

[183] Kan Zheng, Tarik Taleb, Adlen Ksentini, Thomas Magedanz, Mehmet Ulema, et al. Research & standards: advanced cloud & virtualization techniques for 5g networks [guest editorial]. *Communications Magazine, IEEE*, 53(6):16–17, 2015.

[184] Yong Zhu and Mostafa H Ammar. Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM*, 2006.