ENHANCING SCALABILITY AND RELIABILITY IN MOBILE CORE NETWORKS

by

Binh Quang Nguyen

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

School of Computing
The University of Utah
May 2018

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Binh Quang Nguyen**

has been approved by the following supervisory committee members:

Jacobus Van der Merwe ,	Chair(s)	**29-Aug-2017** Date Approved
Sneha Kumar Kasera ,	Member	**29-Aug-2017** Date Approved
Robert Preston Riekenberg Ricci ,	Member	**29-Aug-2017** Date Approved
Ryan Stutsman ,	Member	**29-Aug-2017** Date Approved
Vijay Gopalakrishnan ,	Member	**20-Sep-2017** Date Approved

by **Ross T. Whitaker** , Chair/Dean of

the Department/College/School of **Computing**

and by **David B. Kieda** , Dean of The Graduate School.

ABSTRACT

The next generation mobile network (i.e., 5G network) is expected to host emerging use cases that have a wide range of requirements; from Internet of Things (IoT) devices that prefer low-overhead and scalable network to remote machine operation or remote healthcare services that require reliable end-to-end communications. Improving scalability and reliability is among the most important challenges of designing the next generation mobile architecture.

The current (4G) mobile core network heavily relies on hardware-based proprietary components. The core networks are expensive and therefore are available in limited locations in the country. This leads to a high end-to-end latency due to the long latency between base stations and the mobile core, and limitations in having innovations and an evolvable network. Moreover, at the protocol level the current mobile network architecture was designed for a limited number of smart-phones streaming a large amount of high quality traffic but not a massive number of low-capability devices sending small and sporadic traffic. This results in high-overhead control and data planes in the mobile core network that are not suitable for a massive number of future Internet-of-Things (IoT) devices.

In terms of reliability, network operators already deployed multiple monitoring systems to detect service disruptions and fix problems when they occur. However, detecting all service disruptions is challenging. First, there is a complex relationship between the network status and user-perceived service experience. Second, service disruptions could happen because of reasons that are beyond the network itself.

With technology advancements in Software-defined Network (SDN) and Network Function Virtualization (NFV), the next generation mobile network is expected to be NFV-based and deployed on NFV platforms. However, in contrast to telecom-grade hardware with built-in redundancy, commodity off-the-shell (COTS) hardware in NFV platforms often can't be comparable in term of reliability. Availability of Telecom-grade mobile core network hardwares is typically 99.999% (i.e., “five-9s” availability) while most NFV platforms only guarantee “three-9s” availability – orders of magnitude less reliable. Therefore, an NFV-based mobile core network needs extra mechanisms to guarantee its availability.

This Ph.D. dissertation focuses on using SDN/NFV, data analytics and distributed system techniques to enhance scalability and reliability of the next generation mobile core network. The dissertation makes the following contributions. First, it presents SMORE, a practical offloading architecture that reduces end-to-end latency and enables new functionalities in mobile networks. It then presents SIMECA, a light-weight and scalable mobile core network designed for a massive number of future IoT devices. Second, it presents ABSENCE, a passive service monitoring system using customer usage and data analytics to detect silent failures in an operational mobile network. Lastly, it presents ECHO, a distributed mobile core network architecture to improve availability of NFV-based mobile core network in public clouds.

CONTENTS

ABSTRACT . . . iii

LIST OF TABLES . . . viii

ACKNOWLEDGEMENTS . . . ix

CHAPTERS

1. INTRODUCTION . . . 1

1.1 Introduce new functionalities and enhance scalability in the Mobile Core Network . . . 2
1.2 Enhance reliability in mobile networks with data analytics and novel failure detection mechanism . . . 3
1.3 Enhance reliability of the mobile network in public clouds . . . 5
1.4 Contributions . . . 6
1.5 Organization . . . 8

2. BACKGROUND . . . 9

2.1 Overview of network architecture . . . 9
2.2 Data service in LTE/EPC . . . 11
2.3 Connectivity abstractions in LTE/EPC . . . 11
2.3.1 PDN connections . . . 12
2.3.2 EPS bearers . . . 12
2.4 Mobility . . . 13
2.4.1 Idle-mode mobility . . . 14
2.4.2 Active-mode mobility . . . 15
2.5 Components in LTE/EPC . . . 15
2.5.1 Control plane . . . 15
2.5.2 Data plane . . . 16
2.6 Data plane protocol stack . . . 16
2.6.1 Data plane protocol stack . . . 16
2.6.2 EPS bearer and GTP tunnels . . . 17

3. INTRODUCE NEW FUNCTIONALITIES AND ENHANCE SCALABILITY IN THE MOBILE CORE NETWORK . . . 19

3.1 SMORE: Software-defined networking mobile offloading architecture . . . 19
3.1.1 Introduction . . . 19
3.1.2 Motivation . . . 21
3.1.3 Typical deployment of a LTE/EPC network . . . 22
3.1.4 SMORE architecture . . . 22
3.1.5 Implementation . . . 26

3.1.6 Evaluation 30

3.1.6.1 Smore SDN 30

3.1.6.2 SMORE monitor and controller 31

3.1.6.3 Rtt improvement 32

3.1.7 Related work 33

3.1.8 Conclusion 34

3.2 SIMECA: SDN-based IoT mobile edge cloud architecture 34

3.2.1 Introduction 34

3.2.2 Motivation and overview 37

3.2.3 SIMECA architecture 40

3.2.3.1 SDN-based mobile edge network and cloud infrastructure 40

3.2.3.2 IoT service abstraction (ISA) 42

3.2.3.3 Lightweight IoT data plane 44

3.2.3.4 Lightweight IoT control plane 48

3.2.3.5 Multiple-region communications 54

3.2.4 Prototype implementation 56

3.2.5 Evaluation 58

3.2.5.1 Micro benchmark 60

3.2.5.2 System evaluations 62

3.2.6 Discussion 66

3.2.7 Related work 67

3.2.8 Repeatable experiment with SIMECA 69

3.2.9 Conclusion 69

4. ENHANCE RELIABILITY IN MOBILE NETWORKS USING DATA ANALYTICS 70

4.1 Introduction 70

4.2 Motivation 73

4.3 Approach 76

4.3.1 Customer usage data 76

4.3.2 Usage prediction and aggregation size 77

4.3.3 Practical user aggregation 80

4.3.4 Temporal usage aggregation 81

4.3.5 Usage aggregation 83

4.4 System overview 86

4.5 Implementation 90

4.6 Evaluation setup 91

4.7 Evaluation results 95

4.7.1 Variable-scale decomposition 95

4.7.2 Synthetic failure evaluation 96

4.8 Operational validation 101

4.8.1 Comparison with customer complaints 102

4.8.2 Alarm rate and true positive rate 102

4.8.3 Use cases 103

4.9 Discussion 105

4.10 Related work 106

4.11 Conclusion 107

5. ENHANCE RELIABILITY IN MOBILE NETWORKS USING DISTRIBUTED SYSTEM TECHNIQUES 108

5.1 Introduction 108
5.2 Background 111
5.2.1 Mobile core network architecture 111
5.2.2 Mobile core: a distributed state machine 112
5.3 Reliability in cloud-based EPC 114
5.3.1 Mobile network reliability requirements 114
5.3.2 Reliable EPC: state of the art 117
5.3.3 What does public cloud provide? 118
5.4 ECHO design 120
5.4.1 Problem space 120
5.4.2 ECHO architecture overview 121
5.4.3 Necessarily reliable entry point 122
5.4.4 Nonblocking cloud components 124
5.4.5 Correctness 127
5.5 Implementation 130
5.5.1 ECHO agents 130
5.5.2 Stateless EPC components 131
5.5.3 Cloud deployment 132
5.6 Evaluation 132
5.7 Related work 140
5.8 Conclusions 141

6. CONCLUSION 142

6.1 Summary of the dissertation 142
6.2 Future research directions 145
6.2.1 Bridging the gap between applications and the mobile network 145
6.2.2 Architectural and protocol-level improvements for NFV-based mobile edge computing 146

REFERENCES 148

LIST OF TABLES

2.1 Acronyms and their full forms 10

2.2 Standardized QCIs and example applications. 13

3.1 Interface exposed by an *ISA* 42

3.2 Attachment point tracking table for P2P communications 46

3.3 OpenFlow rules installed at base stations to implement a P2P connection in SIMECA 49

3.4 Control plane message size, LTE/EPC vs. SIMECA 52

3.5 Workload of health monitoring sensors 60

4.1 Hit rate of ZIP code approximations 85

4.2 Aspects and evaluated values of generated failures 94

4.3 Overall results break down by different aggregations 96

5.1 Inter-DC availability in a major cloud provider 119

5.2 ZK configurations and cloud deployment options in ECHO evaluation with their latency and reliability profiles 134

ACKNOWLEDGEMENTS

I would like to thank my advisor, Jacobus Van der Merwe, for his mentoring, collaboration, and friendship throughout the years of my Ph.D. study. This dissertation would not have been possible without his guidance and support. I am grateful for his help in all of the projects in this dissertation. Looking back I have grown immensely, both technically and professionally, working with him. The skills I learned during the years with Kobus will surely be helpful throughout my career and my life.

Throughout the years of my Ph.D. study, I was fortunate to work with many great mentors in the industry. They provided great insights and supports when I worked on real-world challenging problems. I would like to thank Vijay Gapalakrishnan, Aman Shaik, Seungjoon Lee in AT&T Labs Research for their collaboration early in my Ph.D. study. I would like to thank Zihui Ge and He Yan in AT&T Labs Research for mentoring the work in Chapter 4 of this dissertation. My thanks go to Nakjung Choi and Marina Thottan in Nokia Bell Labs for the help on the SIMECA work in Chapter 3. I would like to thank Bozidar Radunovic and Thomas Karagiannis in Microsoft Research for their mentoring and help on the work in Chapter 5.

I was also fortunate to work with great people in the Flux Research Group at the University of Utah. They are both friends and colleagues who are always there when I need help. I would like to thank Kirk Webb, Gary Wong, Jonathon Duerig, Mike Hibler, and David Johnson for their help with setting up and resolving infrastructure related issues in my experiments. My thanks also go to Arijit Banerjee, Josh Kunz, Aisha Syed, Ren Quinn, Richard Li, David Hancock, Hyun-wook Baek, and Eric Edie for their valuable feedback, discussion, and reviewing my papers before submission.

I would like to thank Sneha Kasera, Robert Ricci, Ryan Stutsman and Vijay Gapalakrishnan for working on my Ph.D. committee. They provided valuable feedback and comments to improve this dissertation. Sneha Kasera, Robert Ricci and Vijay Gapalakrishnan are also my co-authors of my early work in my Ph.D. study. Ryan Stutsman also helped

with the design of the ECHO work in Chapter 5.

I would like to thank the student co-authors who contributed to the projects in this dissertation. Junguk Cho contributed to the implementation and evaluation of the Open vSwitch data plane extension in the SMORE work in Chapter 3.1. Candy Zhang worked on the implementation and evaluation of the Zookeeper client in the ECHO work in Chapter 5.

Finally, I would like to express my gratitude to my parents, Son Nguyen and Dung Thach, for always supporting and encouraging me throughout my life and my Ph.D. journey. I would like to thank my wife, Anhlinh, for her support through the long evenings when I worked on this dissertation and her valuable feedback and editing skill.

The works in this dissertation were supported by the National Science Foundation under grants 1305384, 1302688.

CHAPTER 1

INTRODUCTION

With the increasing popularity of smart phones and wireless connected devices, mobile networks (the current 4G-LTE and next generation 5G) have become a critical infrastructure; Smart phone traffic will exceed PC traffic by 2020; Global mobile data traffic will grow 3X as fast as fixed IP traffic from 2015 to 2020 and will be 16% of total IP traffic by 2020 [3]. Although details of the 5G mobile network architecture are still emerging, the 5G architecture is expected to enable numerous use cases with a wider range of characteristics than what is possible today, from broadband access to massive Internet of Things to ultrareliable communications (e.g., E-health services) and so on [58]. Given the criticality of the network and the emerging services it will be hosting, making the network more scalable and reliable are among the most important challenges of future mobile network architecture.

Following broader industry trends, including efforts related to 5G, this dissertation applied Network Function Virtualization (NFV) and Software-defined Networking (SDN) as well as big data processing and distributed system techniques to enhance scalability and reliability aspects of the current mobile core network architecture. It addressed limitations and problems with the current 4G mobile core network architecture in supporting a large number of Internet-of-Things (IoT) devices and low-latency services (Chapter 3). It enhanced mobile network reliability using a data analytics approach by developing a novel monitoring methodology and system to enable early detection of service failures in the network (Chapter 4). It designed a reliable control plane architecture to improve mobile network availability in public clouds (Chapter 5).

1.1 Introduce new functionalities and enhance scalability in the Mobile Core Network

Future mobile networks (i.e., 5G), will need to support a large number of Internet-of-Things (IoT) devices, around 26 billion by 2020 according to some estimates [67]. Although most existing IoT devices are using short-range radio access technologies, e.g., IEEE 802.15.4, emerging IoT devices are expected to also use cellular radio access technologies to take advantage of a wider coverage, support of mobility, well-managed and secured network [25]. The massive number of devices implies a fundamental question of how well the current cellular architecture would support IoT devices and what the network might need to change to favor them.

Despite the fact that 4G networks are completely packet based, the architecture was designed and optimized for human-to-human and human-to-machine communications. The result is the 4G architecture that works well for a *limited* number of connections that generate relatively high volumes of *high quality* traffic (e.g., smart phones streaming videos or voice calls) rather than a *massive* number of devices with a sporadic traffic pattern (e.g., room temperature sensors uploading a few Bytes of traffic every half an hour). For example, the Evolved Packet Core (EPC) architecture introduces Evolved Packet System (EPS) bearer notion in the EPC core network. While EPS bearers support high QoS data transmission (i.e., via GTP tunnels with 9 QoS Class Identifiers (QCIs) [23] representing different delay and throughput requirements), setting up or maintaining them incurs significant overhead [90]. For example, a standard LTE initial attach procedure incurs up to 6 control messages to set up a bearer, while a service request procedure incurs up to 4 control messages [24] every time a device has data to send.

Moreover, in the current mobile network, data-path elements (i.e., SGW and PGW) that forward traffic to the Internet , as well as control plane elements dealing with mobility (i.e., MME) and policy (i.e., PCRF), are specialized hardware-based equipment or virtualized VNFs deployed in a limited number of physical locations (i.e., central offices). This leads to a high end-to-end latency for mobile services; traffic between endpoints need to travel to these centralized data centers, adding extra latency to the end-to-end path. On the other hand, the vision for the 5G network [25, 58] calls for services that will require significantly lower latencies (e.g., <5ms for tactile Internet), which the current 4G architecture will not be able to support.

The above limitations suggest improvements in the 4G architecture to support emerging devices and services. We argue that end services and the mobile core network's mobility functions should be placed closer to users in a distributed manner to improve network latency. Moreover, in order to support a massive number of IoT devices with sporadic traffic, the network must have lower overhead. We argue that the mobile architecture should be changed fundamentally both in control and data planes to reduce network overhead for a massive number of IoT devices. Specifically, we proposed a Software Defined Network offloading architecture (SMORE - section 3.1) to *seamlessly* offload a subset of mobile data traffic (e.g., IoT traffic) to distributed infrastructure hosting services at the network edge. We proposed a novel Software Defined mobile edge cloud architecture (SIMECA - section 3.2) that uses best-effort traffic delivery to reduce control and data plane overhead for a massive number of IoT devices.

1.2 Enhance reliability in mobile networks with data analytics and novel failure detection mechanism

Many critical services such as first response and public-safety are already running on current mobile networks. The 5G networks are expected to be even more reliable in order to host emerging services such as remote control (e.g., cloud robotics), E-health (e.g., remote surgery), etc. [25, 58]. Network operators have spent significant efforts on monitoring and detecting failures in a timely manner to maintain high service availability. However, despite the existing monitoring systems, detecting all service disruptions in an operational network is surprisingly challenging.

Somewhat counter-intuitively, network elements that support service functions are not always able to alarm on conditions which are in fact service impacting. This may be the result of, for example, software bugs in the network elements' firmware, or in the EMS (Element Management System) for the network elements, or due to configuration errors. It is possible that, even though all network metrics indicate a healthy network, customers might be experiencing degraded service or a complete service disruption. For example, deployment of a new service feature or a software upgrade to address a bug might trigger an unintended side effect (or indeed a new bug) that the monitoring system is not equipped to detect. We define such service disruptions that are not captured by network monitoring as *silent failures*. Furthermore, it is difficult to infer service quality perceived by customers using the status of the network. There is a complex relationship between the status of the network and the service quality the users experience. Because of redundancy mechanisms within the network, a particular network failure does not necessarily imply customer impact. For example, users associated with a failed cell tower could be picked up by neighboring towers as long as they are in the coverage range of the neighboring towers and the neighbors still have enough resources to handle the users' traffic [155].

Moreover, end-to-end user experience could be affected by problems happening beyond the network itself. In fact, if the problem happens on the devices themselves there will be little evidence on the network to detect this kind of disruption. For example, a bug in a firmware update could cause incompatibility of the device's Radio Stack with the Radio Network Controller (RNC). This could result in a large number of devices (e.g., all iPhone 6 devices in the U.S.) not being able to use the network. This kind of service disruptions is usually revealed by the customer-care database (e.g., customer calls) long after the failure occurred and typically has large customer impact before being noticed and fixed by the network operator.

This raises the question of *what* to monitor to efficiently capture all possible service disruptions (i.e., *silent failures*) with *high fidelity*. We propose ABSENCE, a *passive service monitoring* mechanism to detect service disruptions based on customers' usage. ABSENCE argues that users' usage, or lack thereof, is a reliable indicator of service outages or severe performance degradations in mobile networks. Although the approach sounds simple, ABSENCE needs to solve the following challenges: How should we aggregate customer usage in a way that increases the fidelity of a detection (e.g., ABSENCE must be able to distinguish a normal drop in customer usage and a service disruption)? How can we monitor services across multiple dimensions (e.g., device model, geographical location, etc)? How can we design a system that scales to an operational network with hundreds of millions of users in real time? We implemented and deployed ABSENCE in an operational network. ABSENCE was able to process Tera-Bytes of call record data per day and detected real failures in the network operator (Chapter 4).

1.3 Enhance reliability of the mobile network in public clouds

While detecting failures and fixing them in real time help increase service availability, it is more ideal to design and build a highly available network in the first place. In fact, a mobile network that is not available for a short amount of time could cause many users not to have services for tens of minutes; a crash in the mobile network could cause up to an hour of service outage on the users.

A mobile network is a set of distributed state machines. There are per-user states maintained on the core network's components and on the user device. It is required that the mobile core network should be "always" available and the per-user states must be consistent across the mobile core components and the device. Conventional mobile networks rely on hardware reliability to ensure availability and state consistency. Mechanisms such as active - standby or persistent UE context storage require either redundant or highly reliable hardware [98]. Typical cellular grade network components are built with "five-9s" reliability (i.e., availability of 99.999%) [59, 115].

On the other hand, future mobile core networks expect to use Virtualized Network Functions (VNFs) running on commodity servers or VMs in data centers or public clouds. In contrast to the highly-reliable hardware blades, public clouds often are not comparable in terms of reliability; public clouds are subject to planned or unplanned maintenance and VM migration. Some surveys show that a typical cloud service experiences unplanned outage for hours in a month [73] (i.e., "two-9s" or 99% of availability for 1 hour of outage per month).

This suggests a software-based solution to build a reliable mobile core network component on top of an unreliable infrastructure in public clouds. We enhance the reliability of the mobile network by introducing redundancy and distributed system techniques into the current mobile core architecture. Our proposal, called ECHO, provides the same properties that mobile core networks guarantee today. However, ECHO is built on top of different assumptions about the infrastructure hosting the mobile core network. ECHO introduces a reliable agent at eNodeBs to guarantee eventual completeness of a request. To enhance availability, ECHO separates state from the EPC state machines and turns each EPC component into stateless redundant instances sharing a reliable storage. However, redundancy in ECHO creates problems of maintaining state consistency. First, given that multiple instances of a component can process a request in the same time, how to make the shared state consistent. Second, given multiple instances of a component could create multiple duplicated state change requests to other components, how to make sure the requests create a single consistent state change across the EPC components.

ECHO deals with the first problem by making the component instances process requests *monotonically* and each instance updates the shared state *atomically*. This ensures only one component instance is able to update the shared state and no stale requests could reverse the updated state. ECHO deals with the second problem by making the component idempotent; duplicated requests will cause the same effect on a component. This, together with the first solution, ensures that a side effect will always be processed *linearizably*, which is equivalent to the processing in a conventional mobile core network. Moreover, the solutions in ECHO allow component instances to process requests in a nonblocking manner to improve availability.

We implemented ECHO using commercial smart-phones, LTE small cells and prereleased EPC core software. We evaluated ECHO in Microsoft Azure. ECHO showed a significant improvement in terms of reliability while introducing small latency overhead that is not perceivable by users.

1.4 Contributions

This dissertation makes the following contributions:

• Novel offloading and mobile edge cloud architecture and novel control/data planes to support a massive number of IoT devices

We present SMORE, an architecture to realize offloading in an LTE/EPC mobile network. Specifically, we show how offloading for selected traffic of subscribed users can be realized without modifying the standard LTE/EPC interactions, even when devices are mobile (Chapter 3.) We propose SIMECA, a novel mobile edge cloud architecture that leverages NFV, SDN to enable a new service abstraction that is more suitable for IoT devices and services. The new service abstraction in SIMECA is realized by novel control and data planes which scale better and have lower overhead to better support IoT devices, compared to the current LTE/EPC control plane. We evaluate SMORE and SIMECA using a prereleased

EPC mobile core software, Software-defined Radio base stations, OpenVswitch and SDN controller (Chapter 3.)

- **Passive service monitoring system to detect failures in mobile networks**

We propose a novel service disruption detection technique and a system called ABSENCE that infers service disruptions by monitoring aggregated customer usage. Our design is informed by a data driven exploration of the problem domain using real operational data from a major mobile operator. ABSENCE uses a scalable Hadoop-based implementation of our approach which is capable of performing service disruption detection by processing huge volumes of anonymized CDR data (e.g., hundreds of millions of records every hour for mobile data service) in a streaming manner, for all the mobile services associated with an operational mobile network. Using data from the same operational mobile network, we perform a systematic data-driven evaluation of our approach by introducing a comprehensive synthetic set of both network and mobile device failure scenarios. We also compare our results with ground truth from actual service disruption events and present a number of case studies showing the effectiveness of our approach (Chapter 4.)

- **Novel architecture design to enhance reliability of mobile network components in public clouds**

We propose ECHO, a reliable EPC architecture for public clouds. ECHO is based on the observation that EPC runs user-specific distributed state machines in parallel. We proposed the main building blocks for a reliable implementation of the distributed state machine, and we then mapped these blocks to the original EPC system components. We proposed a mechanism to ensure consistent state change across replicated components in a mobile network. We implemented the required EPC modifications into OpenEPC [55] and deployed ECHO on Microsoft Azure cloud. We performed an extensive evaluation of the system using real mobile phones as well as synthetic workloads. We showed that ECHO is able to cope with host and network failures, including several data-center component failures, without end-client impact. ECHO showed performance comparable to commercial cellular networks today.

1.5 Organization

This dissertation is organized as follows. The next chapter (Chapter 2) will provide basic background knowledge in order to understand motivations and contributions in the following chapters. The chapter describes the mobile network's architecture, its data service, connection abstractions, mobility, and network stacks. The three chapters after that describe the three contributions of the dissertation. Specifically, Chapter 3 describes enhancing scalability and introducing new functionalities in the Mobile Core Network using Software Defined Networking, novel network architecture and network protocols. Chapter 4 describes enhancing reliability in Mobile Networks with data analytics and a novel failure detection mechanism. Chapter 5 describes enhancing reliability in Mobile Networks with distributed system techniques and a distributed mobile control plane. The last chapter (Chapter 6) concludes the contributions and discusses future research directions related to the topics of this dissertation.

CHAPTER 2

BACKGROUND

This chapter provides background knowledge of the LTE/EPC architecture: its network architecture overview, data service, connectivity abstractions, data and control planes and network stack. This information is required to understand the contributions in the following chapters.

2.1 Overview of network architecture

As shown in Figure 2.1, a LTE/EPC network consists of a wireless radio access network (Radio Access Network or RAN) and a wired mobile core network (EPC core network). The RAN consists of multiple base stations (eNodeBs) that communicate with end devices (User Equipment - UE) over a radio interface. The EPC core network consists of a data plane and a control plane. The control plane interacts with the UE to authenticate and set up or modify connections in the data plane to enable end-to-end communications between the UE and other IP networks. The control plane also keeps track of the UE when it moves across eNodeBs. The data plane consists of gateways that forward IP packets from the eNodeBs to another IP network or Internet.

A minimum EPC core network that provides data service will have Mobility Management Entity (MME), Home Subscriber Server (HSS), Policy and Charging Rules Function (PCRF) in the control plane, Serving Gateway (SGW) and Packet Data Network Gateway (PGW) in the data plane. Table 2.1 shows the abbreviations and their full names that are frequently used in this dissertation.

An overview of interactions between components in a LTE/EPC network is as follows. After power up, a UE synchronizes with a nearby eNodeB and sets up a control plane channel with the network's control plane (i.e., MME). The UE then authenticates itself and the network by exchanging a shared secret it has on its preinstalled SIM card with the MME. During authentication, the MME queries the HSS to obtain the UE's subscriber

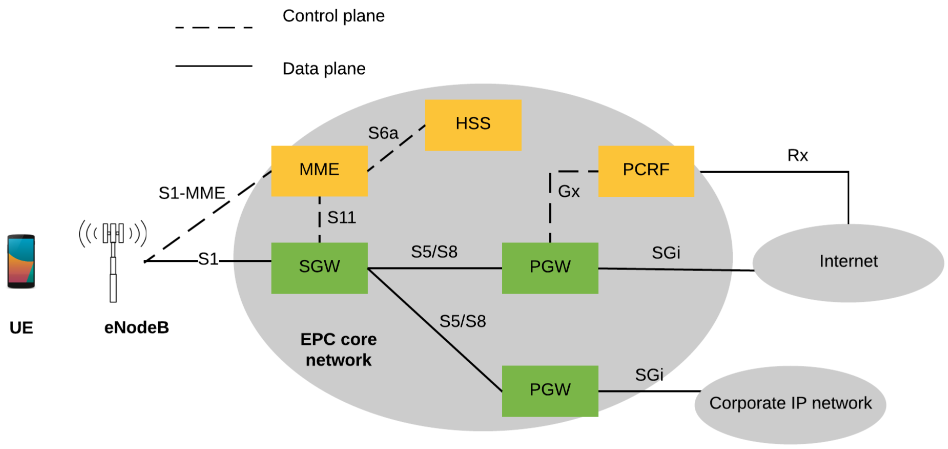


Figure 2.1. LTE/EPC architecture

Table 2.1. Acronyms and their full forms

Acronym	In Full
LTE	Long Term Evolution
EPC	Evolved Packet Core
EPS	Evolve Packet System
PDN	Packet Data Network
APN	Access Point Name
GTP	GPRS Tunneling Protocol
TEID	Tunnel Endpoint Identification
UE	User Equipment
eNodeB	Evolved NodeB
MME	Mobility Management Entity
HSS	Home Subscriber Server
PCRF	Policy and Charging Function
SGW	Serving Gateway
PGW	Packet Gateway
QCI	QoS Class Identifier
GBR	Guaranteed Bit Rate
PDCP	Packet Data Convergence Protocol
RLC	Radio Link Control

information. If the authentication is successful, the UE then requests to connect to an IP network, e.g., Internet. When it receives a connection request, the MME messages the SGW which in turn messages the PGW to set up an Evolved Packet System (EPS) bearer in the data plane. After setting up the EPS bearer, MME replies to the UE with the bearer information and the UE is able to communicate with the IP network via the eNodeB. Similar interactions happen when the UE moves across eNodeBs. The UE requests a handover procedure with the MME which in turn updates the data plane to the target eNodeB.

2.2 Data service in LTE/EPC

The LTE/EPC network is designed for IP services; the role of the RAN and the EPC core network is to provide IP communications between two end-points, the UE and an IP host that the UE is communicating with. All applications running on the UE can make use of the IP connection to exchange IP packets with the other end-point. The applications could either be provided by the mobile operator, by a company in a corporate IP network or by a normal service provider in the Internet.

Figure 2.2 shows an application running on a data service in LTE/EPC. After successfully attaching to the LTE/EPC network, the UE is assigned an IP address. From the UE's perspective, the gateway node (i.e., PGW) is the first IP hop for the assigned IP of the UE. There is a point-to-point link between the UE and the PGW through an eNodeB and a SGW (not shown in the figure). All traffic generated by the applications on the UE must go through the PGW via the point-to-point link. This way, the PGW is an IP anchor for the UE upon mobility. When the UE moves across base stations, the point-to-point link changes. However, its associated PGW never changes and therefore the IP connection survives the mobility.

2.3 Connectivity abstractions in LTE/EPC

The LTE/EPC network provides different connectivity abstractions for the UE. A UE could have one or multiple Packet Data Network (PDN) connections to one or multiple IP networks. A PDN connection is the most coarse-grain connectivity abstraction a UE can have. Under each PDN connection, the UE can set up multiple EPS bearers each with a

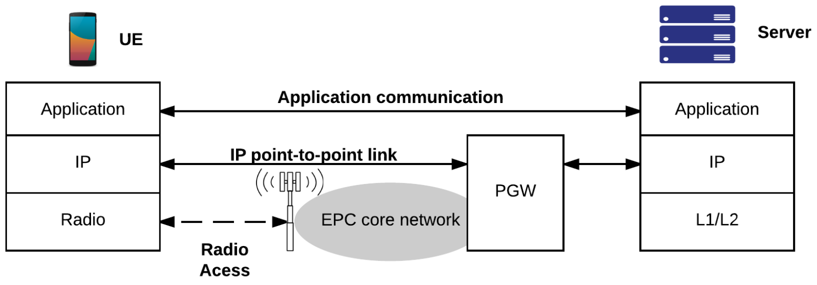


Figure 2.2. Data service in LTE/EPC

different QoS class that is needed by applications. More details about PDN connections and EPS bearers are as follows:

2.3.1 PDN connections

When a UE powers up it will request at least one PDN connection to an IP network. The PDN connection is identified by its Access Point Name (APN). For example, as shown in Figure 2.3, a UE can have two PDN connections, one to the Internet (APN name is Internet) and another one to a Corporate "ABC" at the same time. The UE could also request an additional PDN connection to an IP Multimedia Subsystem (IMS) for services such as VoIP if needed. With each PDN connection, the UE is assigned an IP address associated with that IP network.

2.3.2 EPS bearers

A UE can request multiple EPS bearers under a PDN connection. All of the EPS bearers under a PDN connection will have the same end-point IP address (i.e., the IP address assigned to the UE by that IP network). Each bearer, however, has a different QoS specification that is used for traffic differentiation. An EPS bearer is a point-to-point connection from the UE to the PGW in Figure 2.2.

The EPS bearer represents a fine-grained level of QoS control in the LTE/EPC. By default, a PDN connection must have a *default bearer* that has the same life time as the PDN connection. The default bearer often does not provide strict QoS guarantees. Additional *dedicated bearers* could be requested by specific applications on the UE. Dedicated bearers often provide specific QoS guarantees for applications. For example, low-latency

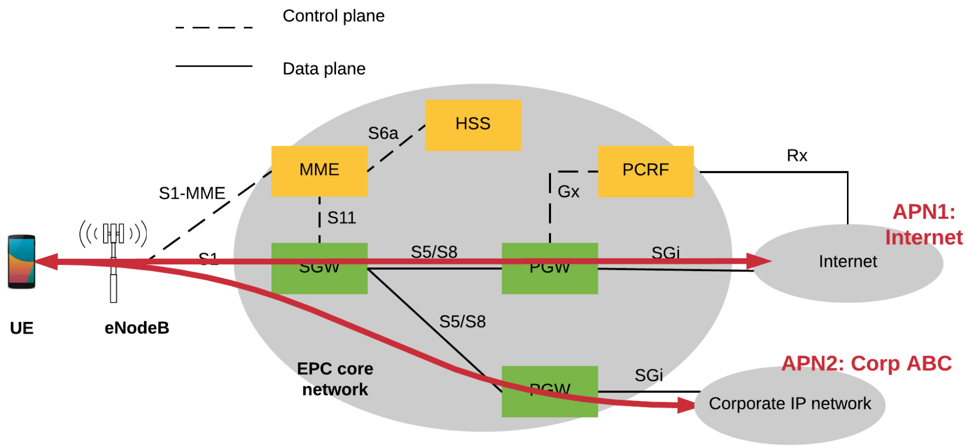


Figure 2.3. PDN connections in LTE/EPC network

applications might request a low-latency dedicated bearer to send data. The dedicated bearers are ephemeral and will be removed if the applications no longer need them.

A bearer associates with a QoS class identifier (QCI) that specifies a delay budget and a loss rate budget of that bearer. A bearer could be Guaranteed-Bit Rate (GBR) if it guarantees a minimum bit rate or Non-BGR if it does not. Table 2.2 shows the standardized QCIs, the delay and loss budgets and their example applications.

2.4 Mobility

Mobility is the core feature of a mobile network. Since an eNodeB can only cover a certain geographical area, a moving UE could break its association with an eNodeB if it moves out of the coverage of that eNodeB and initiates another association with a nearby eNodeB. The primary goal of the LTE/EPC network is to provide functionality of mobility

Table 2.2. Standardized QCIs and example applications.

QCI	Type	Priority	Delay budget	Loss rate	Applications
1	GBR	2	100ms	10^{-2}	Conversational Voice
2		4	150ms	10^{-3}	Conversational Video
3		3	50ms	10^{-3}	Real-time gaming
4		5	300ms	10^{-6}	Buffered Video
5	None-GBR	1	100ms	10^{-6}	IMS signaling
6,8,9		6	300ms	10^{-6}	TCP-based Www, ftp, etc.
7		7	100ms	10^{-3}	Live voice streaming

which ensures the following:

• The network or traffic from other IP networks can reach the UE given the association point (i.e., eNodeB) of the UE is changing due to mobility. This UE-eNodeB association could change either when the UE is radio-active (i.e., sending/receiving data) or radio-idle (i.e., not sending/receiving data).

• Ongoing sessions from/to the UE must be maintained as the UE moves. This means the end-point IP address on the UE does not change and the connections are *seamlessly* adjusted during mobility.

The following describes two different modes of mobility in LTE/EPC. The mobility management described here only applies for *intra-LTE* mobility, i.e., mobility between eNodeBs in a LTE/EPC network.

2.4.1 Idle-mode mobility

In idle-mode mobility, a UE is radio-idle and does not have any ongoing IP sessions with PDN network(s). The network, however, needs to keep track of the location of the UE in case it wants to reach the UE when traffic to that UE is generated. The UE reports its location to the network even when it is radio-idle.

The eNodeBs are grouped into Tracking Areas (TAs). Each TA consists of a number of eNodeBs depending on the deployment as shown in Figure 2.4. A UE reports its location periodically after a fixed interval of time regardless of whether the UE is radio active or not. UE also reports its location immediately if it moves out of a TA. This way, the network knows "roughly" where the UE is at the granularity of a TA. Note that the smaller the TA, the more accurately the network can track the UE with higher signaling overhead due to more frequent location update.

When the UE is idle and the network wants to reach it (e.g., if there is traffic destined to that UE), the network *pages* the UE for its location before sending the traffic (i.e., *paging procedure*). The MME sends a probe signal to all SGWs which in turn notify the eNodeBs in the TA that the UE was last seen. The eNodeBs broadcast the signal to all UEs and the UE that is being looked at for replies. The UE then activates its radio interface and sets up a connection with the network for traffic delivery.

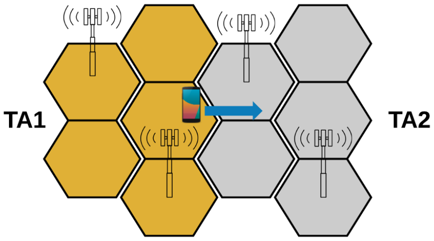


Figure 2.4. Tracking Areas in LTE

2.4.2 Active-mode mobility

In active-mode mobility, a UE has ongoing IP session(s) with PDN network(s). The network needs to maintain the IP sessions when the UE moves from an eNodeB to another eNodeB. When connected to an eNodeB, the UE periodically reports the signal quality of its neighbor eNodeBs. If a neighbor eNodeB has "better" signal quality, the current eNodeB triggers a *handover procedure* to handle the UE to the target eNodeB. After the handover, the point-to- point connection between the eNodeB and the PGW changes: the point-to-point connection switches from the old eNodeB to the new eNodeB.

2.5 Components in LTE/EPC

This section provides more details about components of LTE/EPC and their roles in the architecture. The components are divided into Control Plane and Data Plane.

2.5.1 Control plane

• **MME.** The main component of the control plane is the Mobility Management Entity (MME) which controls the LTE access network. It is responsible for: (1) selecting SGW during an initial attach and handover, (2) tracking the UE during idle-mobility and paging, (3) activating or deactivating the EPS bearers on the data plane per request from UE, (4) authenticating the UE if necessary based on the subscribing information from HSS.

• **HSS.** HSS is a database of the network that stores user data such as credentials for authentication, default APNs of users and users' access authorization information.

• **PCRF.** PCRF encompasses policy control decision and flow-based charging control functionalities. The PCRF receives service information from an Application Function (AF) server and decides how the data flow for a service is handled. The AF server typically lives in the Internet. The UE contacts the AF server via its default bearer.

2.5.2 Data plane

• **SGW.** The gateway serves as an aggregation point for multiple eNodeBs in a geographical area and an anchor point for intra and interarea mobility. When the UE moves within an area, traffic towards the UE always go to the SGW of that area. When the UE handovers to an eNodeB served by a different SGW, traffic towards the old SGW will be forwarded to the new SGW. SGW also serves as a buffer of downlink data traffic when a paging procedure happens.

• **PGW.** This gateway connects the mobile network with the Internet or other IP networks. The PGW allocates an IP address to the UE. It also performs deep packet inspection, packet filtering, or implements the QoS aspects of EPS bearers. PGW is also an IP anchor point for intertechnology handovers (e.g., LTE to WiMax handover).

2.6 Data plane protocol stack

This section provides detailed information on how data plane IP packets are processed at each component and how an EPS bearer delivers IP packets by GPRS Tunneling Protocol (GTP) tunnels.

2.6.1 Data plane protocol stack

Figure 2.5 shows the protocol stack of the data plane of LTE/EPC. The UE maintains an IP session with the PGW which in turn maintains another IP session (e.g., using Network Address Translation (NAT)) with the server. The application on the UE runs on top of these IP sessions.

At the RAN, the UE and eNodeB exchange IP packets using Packet Data Convergence Protocol (PDCP) and Radio Link Control (RLC) Protocol and LTE MAC/PHY. When they arrive at the eNodeB, IP packets from a UE will be delivered to the PGW through GTP tunnels running on top of an IP network (UDP/IP) between eNodeB, SGW, and PGW.

The following shows an example of how an uplink IP packet is processed at each component. When arriving at the eNodeB the IP packet popped out of the PDCP layer is then encapsulated using GTP on top of UDP/IP transport. Note that the relay at the

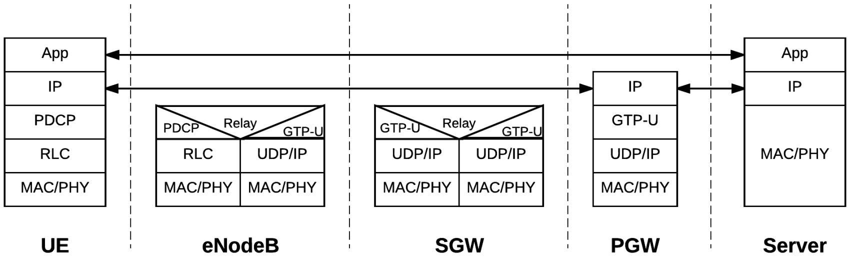


Figure 2.5. LTE/EPC data plane protocol stack

eNodeB needs to look up which GTP tunnel to use because there are multiple GTP tunnels per UE. The GTP-encapsulated packets will be sent to the SGW which decapsulates the packet and encapsulates it again with a different GTP/UDP header. The SGW then sends the GTP packets to PGW. The PGW then decapsulates the packets to get the inner IP packets. It will send the IP packets to the server or alternatively NATs the packets with its routable address and sends out to the server.

2.6.2 EPS bearer and GTP tunnels

Figure 2.6 shows an EPS bearer in LTE/EPC. An EPS bearer is a point-to-point link between the UE and the PGW. The IP packets from/to the UE are delivered to/from the PGW by the EPS bearer. A UE has at least one default EPS bearer and optionally multiple dedicated EPS bearers per PDN connection. The interfaces of eNodeB-SGW/SGW-PGW pairs are called S1-U and S5, respectively.

An EPS bearer consists of 6 GTP tunnels: 3 GTP tunnels for each uplink and downlink direction. At the RAN interface between the UE and eNodeB, a radio bearer is set up per EPS bearer. This radio bearer is identified by a Data Radio Bearer (DRB) Identifier. Between the eNodeB and SGW are 2 S1-U GTP tunnels that are identified at the eNodeB and SGW by UL-S1U Tunnel EndPoint Identifier (TEID) and DL-S1U TEID. Similarly, there are 2 S5 GTP tunnels between SGW and PGW that are identified by UL/DL-S5 TEIDs.

The mobile network also uses GTP for supporting per user mobility and quality of service. Each tunnel can be distinguished by a unique Tunnel EndPoint Identifier (TEID). When a packet is encapsulated, the TEID field in the GTP-U packet header is set to the value expected by the tunnel destination. The eNodeB/SGW/PGW obtains the TEIDs

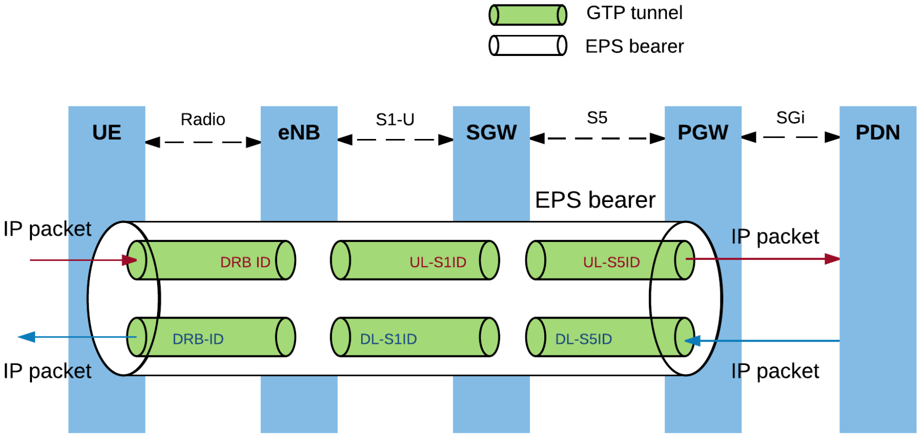


Figure 2.6. An EPS bearer and GTP tunnels in LTE/EPC data plane

from the control plane (MME) during the connection set-up or modification. The GTP tunnels between SGW and PGW stay persistent unless the UE moves across SGWs. The GTP tunnels between eNodeB and SGW are ephemeral; when the UE is radio-inactive the tunnels are removed. When the UE has data to send it signals the LTE/EPC control plane to set up these tunnels before sending data packets.

CHAPTER 3

INTRODUCE NEW FUNCTIONALITIES AND ENHANCE SCALABILITY IN THE MOBILE CORE NETWORK

This chapter focuses on enhancing scalability of the Mobile Core Network using Software Defined Networking and Network Function Virtualization. It first presents SMORE, a practical mobile offloading architecture using Software-defined Networking. SMORE's offloading mechanism also allows alternative mobile core networks to be deployed in parallel with the current Evolved Packet Core (EPC) network. It then presents SIMECA, a SDN-based mobile core network architecture designed specific for a massive number of Internet-of-Thing devices.

3.1 SMORE: Software-defined networking mobile offloading architecture

3.1.1 Introduction

Unprecedented growth in data traffic on mobile networks [89] is driven by the popularity of smartphones and tablets, advances in bandwidth available on mobile networks and the availability of a myriad of mobile applications and services. Despite these undeniable successes, the delay experienced across core mobile networks is still problematic for many applications that need near-realtime communication. Online gaming applications, for example, have stringent quality of service (QoS) demands and latency can negatively interfere with game play [50, 77].

A key architectural reason for delay remaining relatively high in the mobile core is the fact that the gateway nodes (i.e., Packet Data Network Gateways (PGW) in LTE/EPC nomenclature), are deployed in a highly centralized fashion [32]. This results in inefficient hierarchical routing and high delay—packets travel for a significant distance in the mobile core network before even reaching the Internet [57].

Approaches to reduce end-to-end delay involve various offloading strategies [75, 99], including offloading to data centers inside the mobile provider core network footprint [34]. Despite the potential benefits of these approaches, they are inherently difficult to deploy in real networks—because of the fundamental complexity of mobile architectures, approaches that require significant changes to those architectures are very hard to realize in practice.

In this work we propose an alternative approach to *realizing* offloading in a mobile network: our Software defined network Mobile Offloading aRchitecturE (SMORE). Like previous offloading approaches [34], we propose to deploy offloading data centers within the core of the mobile network and to selectively redirect traffic to these data centers. The defining characteristic of SMORE is that this offloading is accomplished *without* requiring modifications to the functionality of the core network proper, that is, without modifications to the functionality of network elements in the core mobile network. SMORE is realized by deploying an SDN infrastructure at aggregation points in the mobile core network. This SDN infrastructure allows two functions critical to the functioning of SMORE. First, it enables *monitoring* of the mobile network control plane through a lightweight monitoring component. The SMORE monitor continuously extracts information from the mobile network control plane and makes this information available to the SMORE controller. Second, based on this information and the service logic executed by the SMORE controller, the SDN infrastructure is used to transparently realize the offloading of selected traffic to offloading data centers. We argue that the ability to deploy these changes without modifying functional components of the mobile core is critical to making the deployment of new functionality such as offloading practical.

This work makes the following contributions:

- We develop the SMORE architecture to realize offloading in an LTE/EPC mobile network. Specifically, we show how offloading for selected traffic of subscribed users can be realized without modifying the standard LTE/EPC interactions, even when devices are mobile.
- We present a prototype realization of our architecture using an LTE/EPC testbed and an Open vSwitch (OVS) based SDN. Specifically, our OVS enhancements allow transparent encapsulation and decapsulation of offloaded traffic.

3.1.2 Motivation

Latency is an important factor that impacts mobile applications performance. An end-to-end latency of more than 30ms causes a normal video streaming application to drop its frame rate that is noticeable to human eyes [92, 132]. Moreover, 5G networks are expected to host new use cases that require even lower latency. For example, remote closed-loop control applications [7] require a low-latency network to deliver commands to the remote machine and receive feedback in real-time; virtual or augmented reality applications [58] also require lower latency to transfer camera's video frames to servers for processing. Even for web browsing, a low-latency network is preferable to increase service revenue [6].

On the other hand, the end-to-end delay of 4G LTE networks is high; a typical round-trip latency between a mobile device and a service is 70-75ms [78]. In the end-to-end latency picture, the latency between base stations and the core network's gateways (SGW,PGW) contributes the largest portion (i.e., mobile core network's latency). Given a typical LTE radio latency between a mobile device and a base station is around 15ms RTT [16] and the latency between the mobile gateway (PGW) and the service is small, the mobile core network's latency is around 45-55ms RTT. Even though the 3GPP standard has mechanisms to prioritize traffic for different applications [23], prioritizing won't help given the network latency.

The reason for such a high latency in the mobile core network is the physical distance between base stations and the mobile gateways (SGW, PGW); while base stations are distributed, the mobile gateways are only available in a limited number of locations in the country [32, 151]. As mobile network components are hardware-based, proprietary and highly-capable, it is expensive and wasteful to deploy them in a distributed manner. This results in high network latency between the base stations and the gateways. A possible solution is to offload traffic at the network edge to a local edge cloud to eliminate the mobile core's latency. However, unclear questions are (1) where should an operator deploy the offloading to reduce the latency while not increasing the cost of deployment too much? and (2) how to offload traffic in a transparent way? In other words, the system only offloads traffic that require low network latency and leaves other LTE traffic untouched. These are also the two main contributions of this work.

3.1.3 Typical deployment of a LTE/EPC network

Figure 3.1 also depicts typical deployment information that is relevant to our approach. EPC components (MME, SGW and PGW) are typically deployed in a small number of centralized locations (or central offices), e.g., on the order of 10 in the US [32, 151]. This means that each such centralized location serves a large geographic area, with thousands of eNodeBs. LTE/EPC is a packet-based architecture, which means that there exists a packet "transport network", i.e., a regular IP network, in between the eNodeBs and the centralized EPC locations. For efficiency, connectivity from a set of eNodeBs gets aggregated at a regional aggregation point. As shown in Figure 3.1 these aggregation points are called (or co-located with) Mobile Telephone Switching Offices (MTSOs) and there are typically an order of magnitude more MTSOs than centralized EPC locations in a typical deployment. As such, MTSOs are an attractive deployment location for offloading data centers [34].

3.1.4 SMORE architecture

The SMORE architecture is depicted in Figure 3.2 in the context of an LTE/EPC mobile network. The service goal of SMORE is to reduce end-to-end delay for selected traffic by offloading such traffic to *offloading cloud* infrastructures close to mobile devices. Regional aggregation points in the form of MTSOs represent an ideal target for deploying

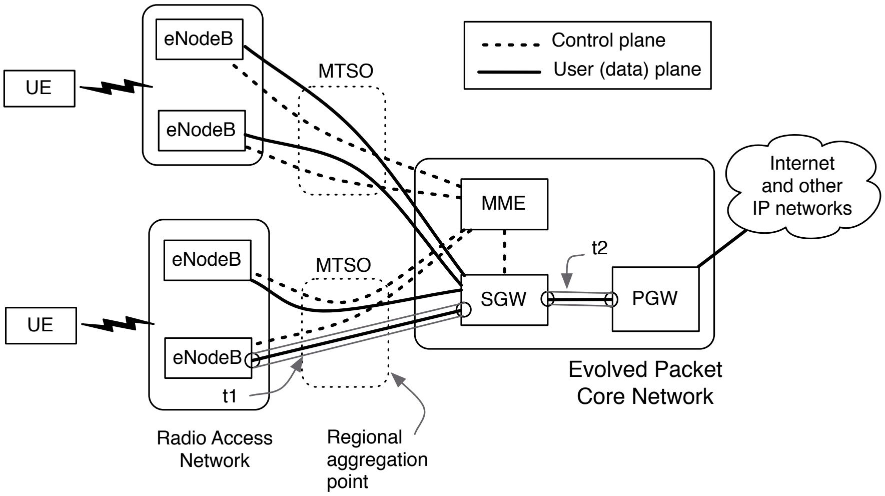


Figure 3.1. LTE/EPC architecture and its deployment.

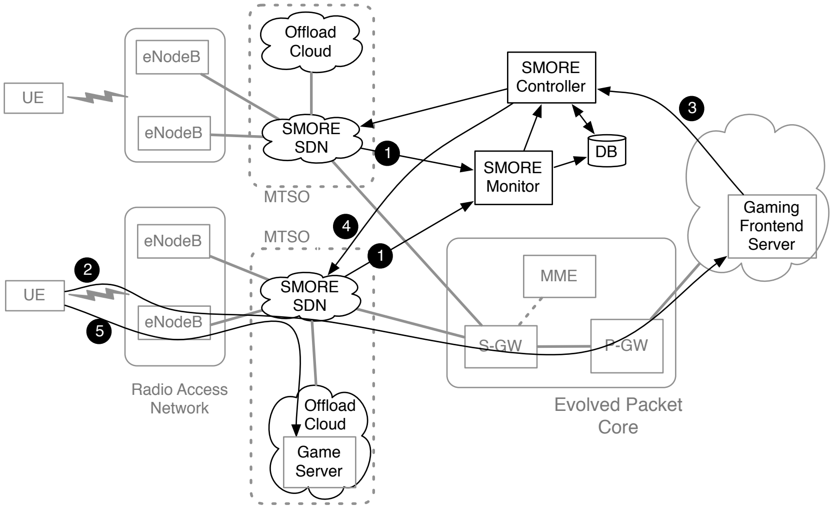


Figure 3.2. SMORE architecture

these offloading infrastructures for two reasons. First, based on current centralized mobile network deployments, all traffic passes through MTSOs en-route to the Internet. Further, the MTSO locations in a typical deployment are at a sweet spot, both in terms of geographical coverage (to reduce delay) and the number of locations (filling the space between approximately 10 central office locations and thousands of eNodeB locations). As shown in Figure 3.2, SMORE assumes that the offloading cloud platform and the SMORE *SDN substrate* are deployed at MTSO locations. The SDN substrate enables offloading to the offload cloud and also allows the SMORE *monitor* to snoop on the LTE/EPC control plane. The SMORE monitor continuously monitors the control plane, extracting relevant information. The monitor enters information into a database and also provides triggers to the SMORE *controller* about UE attach and mobility events. The SMORE controller interacts with the SMORE SDN substrate to effect offloading when needed and provides a front-end that Internet based service providers can interact with to register for offloading services. Note that in all cases service providers using the SMORE service will only be able to offload traffic associated with their own services.

• **On-demand use case.** The interactions between these components are best explained with a typical use case, using the numbered steps from Figure 3.2. The SMORE SDN and monitor allow the control plane to be continuously monitored (#1 in the figure). To simplify exposition, we assume an Internet-based service provider, e.g., a gaming provider, who wants to make use of the SMORE service. Note that this generalizes to any Internet service provider that wants to make use of the SMORE offloading service. End-users of the game service access the Internet-based gaming frontend server via the normal data path after attaching to the mobile network (#2). Realizing that it wants to offload the user in question to an offload instance, logic in the gaming frontend server requests offloading via the interface provided by the SMORE controller (#3). The SMORE controller consults the SMORE database to determine the current location of the UE in question, and proceeds to configure the SDN substrate in the appropriate MTSO location to set up offloading of the desired subset of traffic (#4). When the UE connects to the game server, this traffic will be transparently redirected to the offload server in the MTSO cloud (#5).

• **Subscription use case.** Service providers (or end users) can also opt to make use of a subscription model for offloading. Once subscribed, information about the subscriber UE is maintained in the SMORE database. In this case, every time a subscriber UE connects to the network, the offloading path is constructed without the need for step #3 in the previous scenario. When the UE attaches, the SMORE monitor detects this event and signals the SMORE controller, which immediately installs the appropriate rules in the SMORE SDN.

• **Interaction with standard LTE/EPC.** We now consider in more detail how SMORE can realize its functionality without requiring any modifications to LTE/EPC network elements. We specifically consider how the SMORE monitor extracts information from the LTE/EPC control plane during UE attach and handoff procedures. The information obtained from these procedures triggers the installation of rules in the SMORE SDN to realize offloading in a transparent manner.

• **Attach.** Figure 3.3 shows how the SMORE monitor snoops on the control plane interactions between the eNodeB and the MME during UE attach and subsequently helps trigger the pushing of required offloading policies in the SMORE SDN via the SMORE controller. Specifically, during the UE attach procedure, the SMORE monitor extracts the UE's IMSI (International Mobile Subscriber Identity) and TAI (Tracking Area Identifier) from the 'attach request message' sent to the MME via the eNodeB (#2). When the MME responds with the 'attach accept' message (#5), the monitor extracts the UE's IP address, SGW's IP address, SGW's TEID, and UE's GUTI (Globally Unique Temporary ID). Finally, the monitor obtains the eNodeB's IP address and eNodeB's TEID when the eNodeB re-

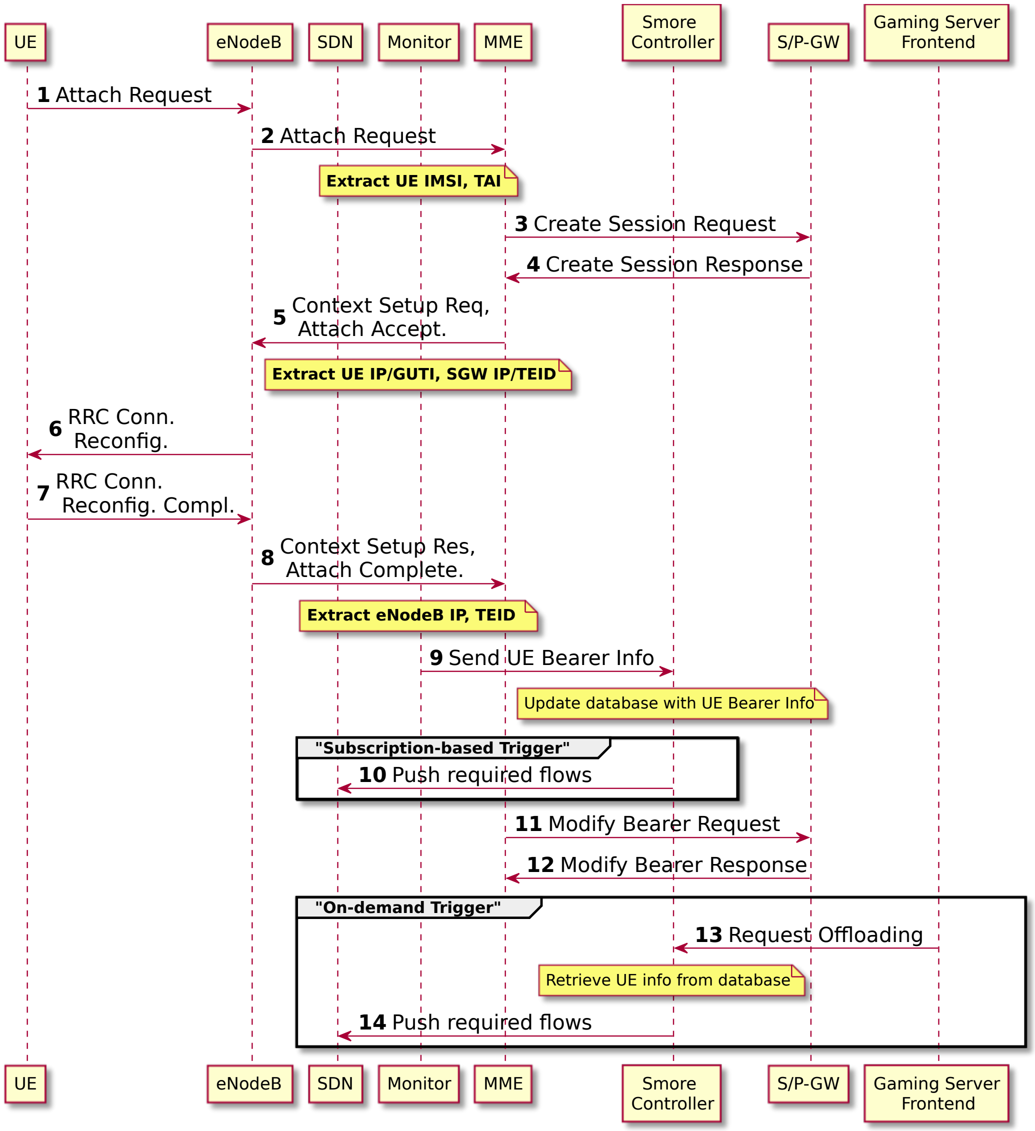


Figure 3.3. Attach call flow

sponds with the ‘context setup response’ message (#8). At this point, the SMORE monitor has all the information required to trigger offloading and sends it to the SMORE controller (#9) which, in turn, updates its database. In the case of a subscription-based use case, at this point the SMORE controller pushes the required flows to the SMORE SDN (#10), which enables offloading of game server specific traffic to the game server hosted in the in-network cloud platform. For the on-demand use case, when the SMORE controller receives the offloading trigger from the gaming frontend server (#13), it retrieves the UE’s bearer information from the database and pushes the required flows to the SMORE SDN

(#14) to enable offloading.

• **Handover.** Figure 3.4 shows the steps required to enable offloading in presence of user mobility. Specifically we consider X2-based handover between eNodeBs without MME and SGW relocation. (This is the common handover case in LTE/EPC.) As seen in the figure, when the source eNodeB decides to hand over the UE (based on measurement reports) to the target eNodeB, it sends a handover request (#2) to the target which, in turn, does the necessary resource allocation and sends back a handover acknowledgement message (#3) to the source. The target also sends a path switch request message (#4) to the MME that contains the target eNodeB information (IP address and TEID), which will subsequently be used by the SGW to forward downlink packets. The SMORE monitor extracts this bearer information, and sends it to the SMORE controller (#5), which proactively constructs a new rule to be pushed to the SMORE SDN to enable processing of downlink packets corresponding to the target eNodeB. Note that all the other UE bearer information remains unchanged (including the SGW identifiers) and the SMORE controller obtains the other required information from its database. When the SGW updates the user bearer to forward downlink packets to the target eNodeB instead of the source eNodeB, the MME sends back a path switch acknowledgement message (#10) to the target indicating the success of the path switch request. When the SMORE monitor sees this message, it sends a path switch trigger request (#11) to the SMORE controller, which, in turn, pushes the previously constructed rule to the SMORE SDN and deletes the previous rule (#12). Subsequently, the SDN substrate routes (after proper encapsulation) all downlink packets from the cloud based game server to the target eNodeB.

3.1.5 Implementation

We developed a prototype implementation of the SMORE architecture and deployed it in a local LTE/EPC testbed based on the OpenEPC LTE/EPC architecture implementation [55]. As shown in Figure 3.5 the RAN part of our current setup is based on an emulated implementation from OpenEPC. Each component shown in the figure is realized as a physical machine instance in the Emulab [148] facility. Our implementation did not support the handover mechanism described in the design because our LTE/EPC testbed did support handover.

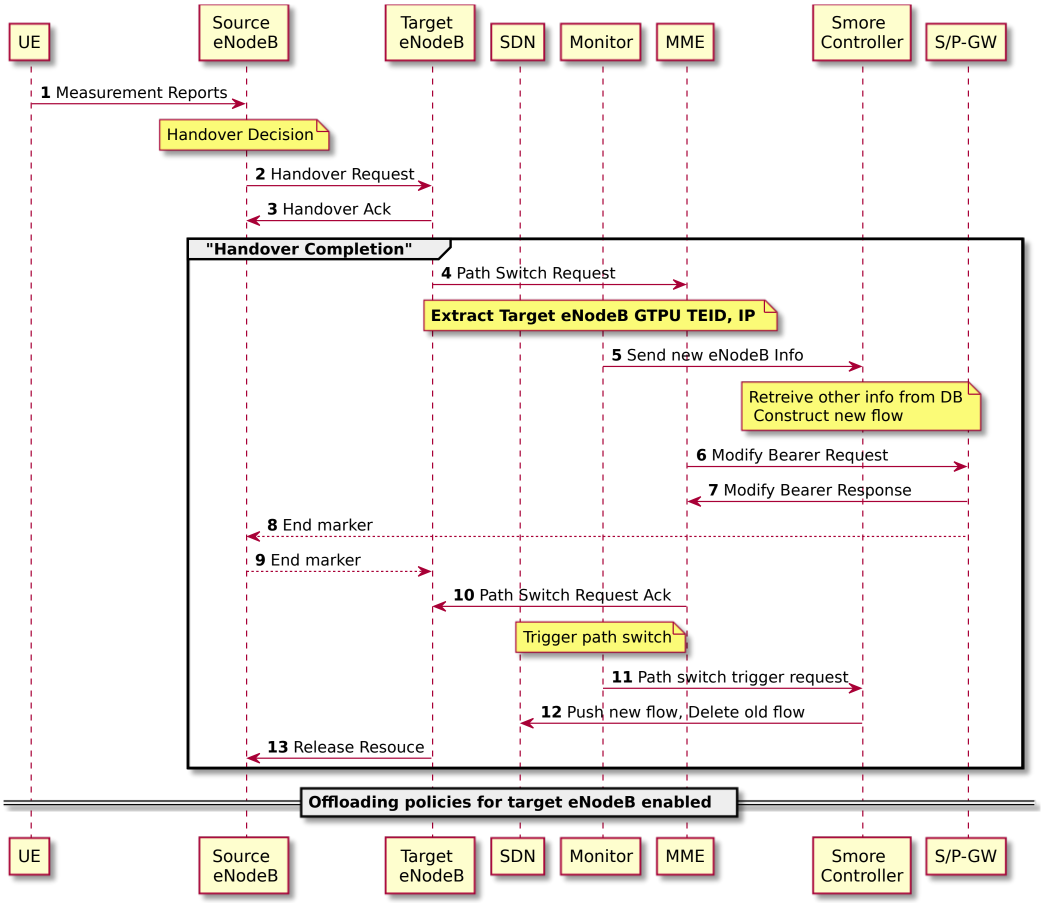


Figure 3.4. Handover call flow

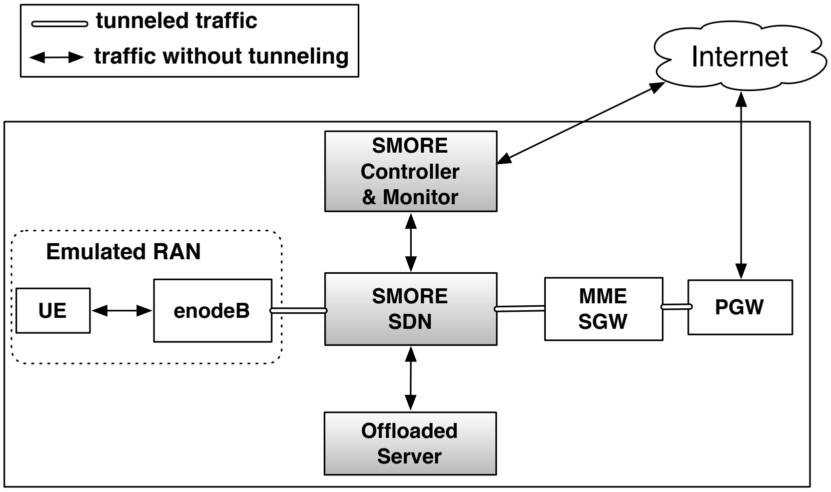


Figure 3.5. Experiment setup.

Below we consider the prototype implementations of each SMORE component in more detail.

• **SDN.** To implement SMORE SDN, we used Open vSwitch (OVS) 2.0 which supports the OpenFlow 1.3 standard. Tunneling in OVS is implemented using a virtual port (vport) abstraction which simplifies header manipulation [88]. As shown in Figures 3.6 and 3.7, we added three vports to realize offloading, GTP decapsulation, and GTP encapsulation. By default the SMORE SDN component works as a standard layer 2 switch. It forwards packets from the eNodeB to the SGW/MME and vice versa. However, when it receives a traffic offload request from the SMORE controller, flow rules are setup to divert UE connections to alternate destinations.

Packet processing in the uplink direction is shown in Figure 3.6. The SMORE SDN first checks whether the incoming packet is GTP data traffic (whether it has 2152 as UDP port number). If so, the offloading vport extracts the SGW TEID in the GTP header and the source and destination IP addresses from the inner IP layer (i.e., the UE source and destination IP). If this information matches a SMORE OVS table entry, then the packet is destined for an offloading server and is sent to the GTP decapsulation vport. Otherwise, the packet is sent along the default path to the SGW. When the GTP decapsulation vport receives a packet, it removes the GTP header, replaces the destination MAC address, and forwards the packet. The replacement MAC address is that of the next hop along the route to the offload destination.

Downlink packet processing is shown in Figure 3.7. Packets arriving from the SGW/MME are forwarded toward the eNodeB as per normal. However, if the packet is from an offloading server, the SMORE SDN uses the IP addresses (src: offloaded server IP, dst: UE IP) to find the correct encapsulation information for this exchange. The packet is forwarded to the GTP encapsulation vport along with the associated tunnel information (outer IP addresses (src: SGW IP, dst: eNodeB IP) and GTP (eNodeB TEID) identifiers). It is also necessary to replace the destination MAC address with the next hop's MAC address (in the direction of the UE). With encapsulation complete, the GTP packet is forwarded on toward the eNodeB.

• **Monitor.** We implemented the SMORE monitor using tshark, a terminal-based version of wireshark. The monitor listens to the interface between the eNodeB and the MME

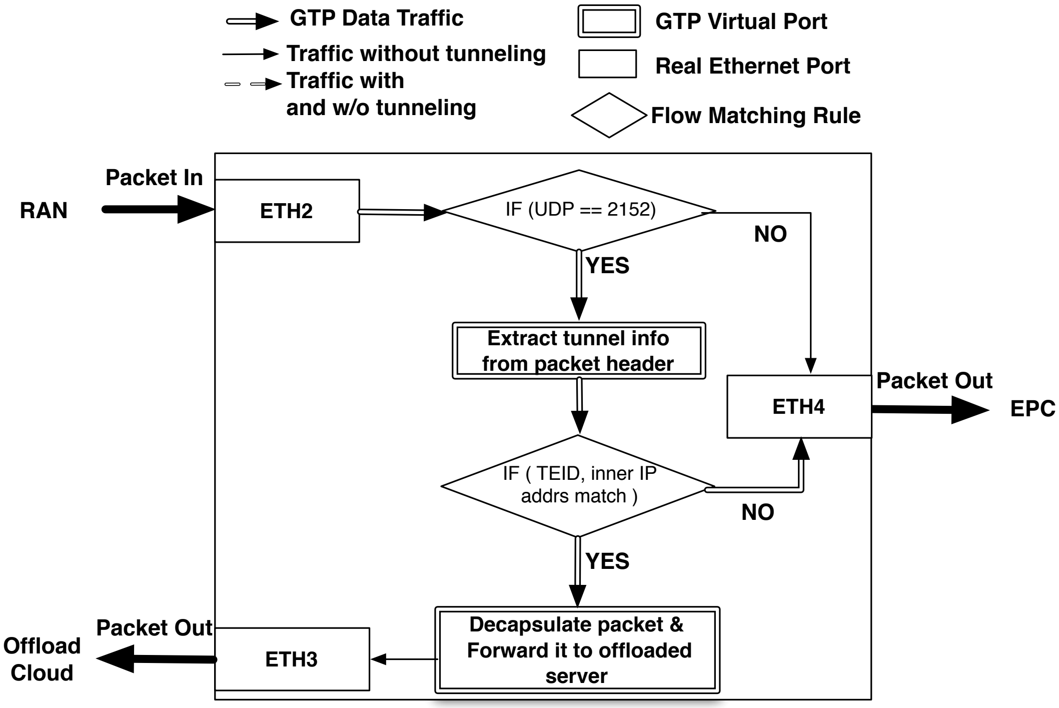

GTP Data Traffic
Traffic without tunneling
Traffic with and w/o tunneling
GTP Virtual Port
Real Ethernet Port
Flow Matching Rule
Packet In
RAN
ETH2
IF (UDP == 2152)
NO
YES
Extract tunnel info from packet header
ETH4
Packet Out
EPC
IF (TEID, inner IP addrs match)
NO
YES
Packet Out
Offload Cloud
ETH3
Decapsulate packet & Forward it to offloaded server

Figure 3.6. Handling uplink packet.

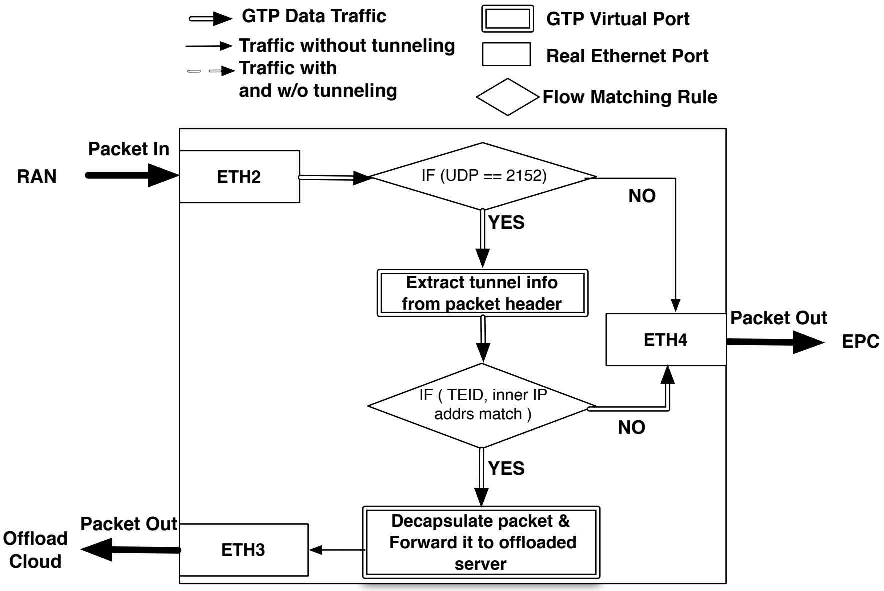

GTP Data Traffic
Traffic without tunneling
Traffic with and w/o tunneling
GTP Virtual Port
Real Ethernet Port
Flow Matching Rule
Packet In
RAN
ETH2
IF (UDP == 2152)
NO
YES
Extract tunnel info from packet header
ETH4
Packet Out
EPC
IF (TEID, inner IP addrs match)
NO
YES
Packet Out
Offload Cloud
ETH3
Decapsulate packet & Forward it to offloaded server

Figure 3.7. Handling downlink packet.

and processes all messages that are exchanged to detect bearer setup and tear-down events. Specifically the monitor implements the information extraction and triggers as described in the *attach* description in Figure 3.3.

• **Controller.** We implemented a prototype of SMORE controller using the Ryu OpenFlow controller. The SMORE controller has a simple database storing registered offloading server IP addresses and IMSIs of UEs that subscribed to offload services. We extended the Ryu API to support GTP flow modification control messages. The SMORE controller modifies the flow table of the OVS using the extended API of Ryu controller. The SMORE controller compares the bearer setup information provided by the SMORE monitor with its database of registered UEs and offloading service addresses to decide whether to push offloading flows for a specific UE to the OVS flow table.

3.1.6 Evaluation

We measured the SMORE SDN processing time, the responsiveness and latency of the SMORE monitor and controller, and illustrated the potential for latency reductions with our approach. Our evaluation was performed using machines with a single 3 GHz CPU and 2 GB RAM.

3.1.6.1 Smore SDN

OVS has two tables, smorely, the OpenFlow flow table in userspace and a simple flow table called datapath in the kernel. The data path table is essentially a cache for active flows. The userspace table is consulted when no match is found in the datapath, typically for the first packet of a flow. We evaluated both the datapath and Openflow table processing of our SDN implementation. We measured the processing time for each port and vport using the *getnstimeofday()* kernel function.

• **Datapath processing.** A breakdown of the processing time when a flow entry exists in the data path table is shown in Figure 3.8. SMORE (Cloud) means the path between the RAN and Offload cloud. SMORE is the path between the RAN and the EPC when SMORE rules are installed. OVS is the same path when no SMORE rules are installed. Compared to (native) OVS, SMORE (Cloud) and SMORE increased uplink processing by 5.366us and 1.911us and downlink processing by 2.105us and 0.058us respectively. SMORE shows slightly longer processing time in ETH input than OVS because more complex matching

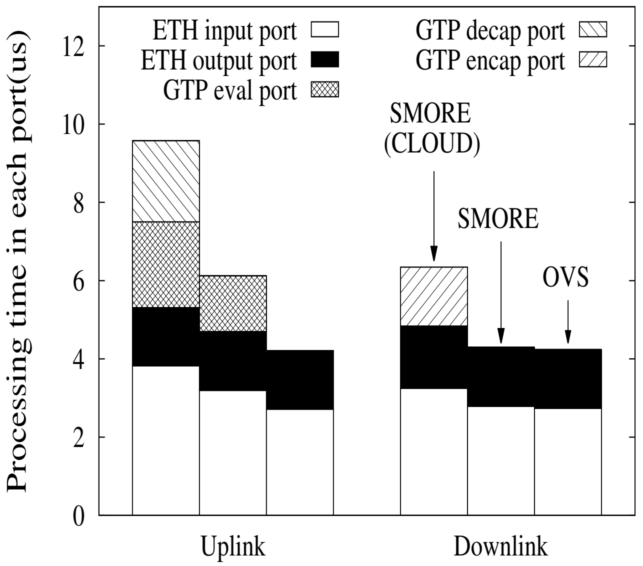


Figure 3.8. Breaking down processing time of each port in datapath

rules are used and there are more flow entries in the datapath.

• **Openflow table processing.** Figure 3.9 shows a similar breakdown for the case where a flow entry does not exist in datapath but in only in the OpenFlow flow table. As expected, the processing times are much longer because of the userspace processing involved. However, this is only incurred on the first packet of a flow. In both cases the overhead is modest compared to the end-to-end delay in LTE.

3.1.6.2 SMORE monitor and controller

• **SMORE controller microbenchmark.** We measured the time the SMORE controller takes to install offloading flows into the OVS when it receives a trigger (i.e., a trigger for either on-demand or subscription use cases). The average processing time is 4.677ms.

• **SMORE monitor microbenchmark.** We generated a series of synthetic UE attaching events to measure the time taken to detect a UE attach event. The average detection time is 2.973ms (not including packet reading time incurred by tshark). Our current packet capture realization based on tshark is quite inefficient, contributing an additional 850ms.

• **Subscription use case evaluation.** We evaluated the responsiveness of SMORE for the subscription use case described in Section 5.4. Since an attached UE might immediately start to interact with the offloading server, this case demands fast response from the

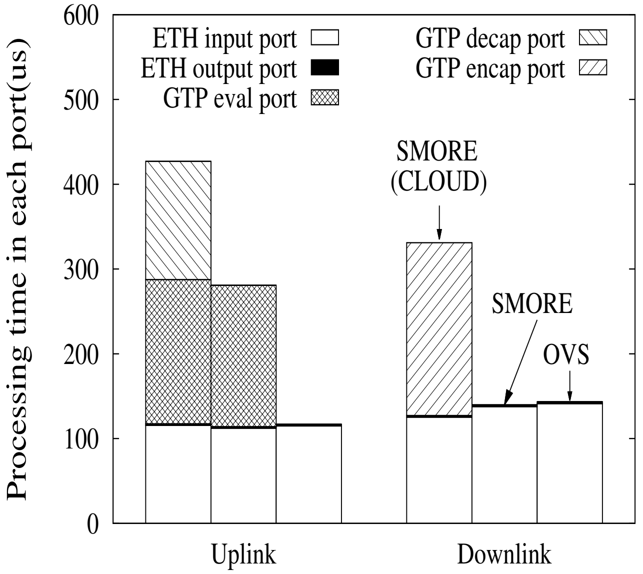


Figure 3.9. Breaking down processing time of each port in OpenFlow table

SMORE monitor and controller.

We measured the elapsed time (t_{res}) between the moment when the 'attach complete' message (#8 in Figure 3.3) arrives at the OVS and when the offloading flows are successfully installed (in the OVS) by the SMORE monitor and SMORE controller. We measured t_{res} for 10 different UE attach events and found the mean value for t_{res} to be 1.056s with a standard deviation of 18ms. This is an acceptable delay as it is of the same order of magnitude as application startup delay on a mobile device.

3.1.6.3 Rtt improvement

We measured the RTT between a UE and the top websites in the U.S. [27]. We derived a synthetic delay from previous measurement works as the delay of the cellular core network [42]. Specifically, the core network delay is calculated as: *core network's delay = UE-server's delay - PGW-server's delay - radio's delay*. The one-way delay from UE to server was set to 35ms as reported in [79]. The PGW-server delay was set to 8ms as a typical "regional" Internet delay reported in [116]. The delay on the radio link was 4ms one-way [42]. In total, the synthetic delay of the core network (from OVS node to PGW) was set as 23ms. The delay between the OVS node and the offloading server was set to a typical "local" link and it was almost negligible, i.e, less than 1ms.

We pinged each server 50 times and reported the average RTT in Figure 3.10. As expected and shown in the figure, the delay when using the offloading server is significantly smaller than using the servers in the Internet.

3.1.7 Related work

The issues associated with current hierarchical routing approach in the cellular networks have been identified in works like [57, 152]. To alleviate the burden of the core network and enable efficient routing, the 3GPP standards body has proposed Local IP Access (LIPA) and Selected IP Traffic Offload (SIPTO) – for offloading mobile traffic to a local network using femtocells and, to the Internet via regional gateways closer to the user locations, respectively [9]. However, these methods require significant changes in the current standard and expensive additional infrastructure for their realization. Our aim, in this work, is to enable efficient offloading without requiring any changes to the existing standard. Other works have proposed opportunistic offloading of delay tolerant cellular traffic using alternative access technologies like WiFi [33, 75]. Our work is focused on efficient offloading strategies in cellular networks specially for traffic with strict delay requirements (e.g., real time gaming).

We draw inspiration from research works exploring the possibility of leveraging SDN to enable more flexibility in the core mobile network [84, 88, 95] and using cloud resources

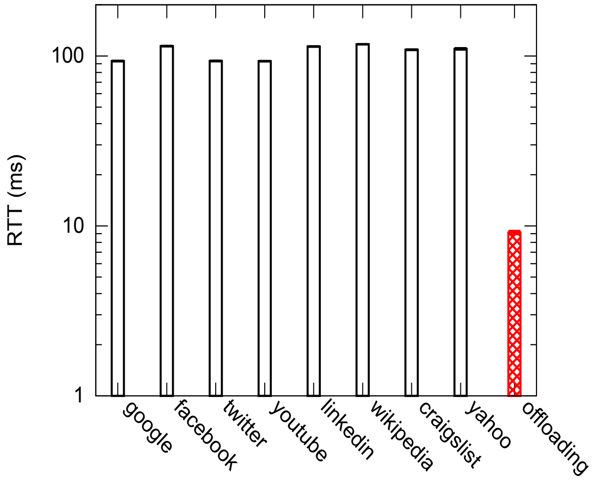


Figure 3.10. RTT of different Internet services and the offloading server

in user proximity to enable low latency applications [132]. Our work is grounded in the reality that it is not economically feasible for the mobile network operators to partake in a major overhaul of the existing infrastructure built on standards compliant closed-source vendor specific equipment. Our proposed architecture can be deployed in strategic locations to co-exist with the current infrastructure and enables much needed flexibility in the mobile network using emerging technologies like SDN and cloud, without requiring any changes in the existing architecture.

Our work is strongly inspired by MOCA [34]. MOCA proposed an SDN-based architecture for cellular networks to offload selected traffic to a cloud-based local S/P-GW (service and packet gateway) instance and an offloading server inside the core network. In MOCA architecture, although modifications in the current network were intentionally minimized, the MME still played a central role and needed to be modified. With SMORE we designed and prototyped similar functionality without requiring modification to *any* LTE/EPC network element.

3.1.8 Conclusion

We leveraged SDN to introduce an offloading architecture for cellular networks. The architecture allows traffic to be intercepted and rerouted to offloading servers located inside the cellular core. The SMORE approach allows us to realize offloading with no modifications to the current network architecture.

3.2 SIMECA: SDN-based IoT mobile edge cloud architecture

3.2.1 Introduction

Future networks, including future mobile networks (i.e., 5G), will need to support a large number of Internet-of-Things (IoT) devices; around 26 billion by 2020 according to some estimates [67]. Most existing IoT devices are using short-range radio access technologies, e.g., IEEE 802.15.4, but emerging IoT devices are expected to also use cellular radio access technologies to take advantage of mobile network features, such as wider coverage, better support for mobility, well-managed and secured [25]. The massive number of devices implies a fundamental question of how well the current cellular architecture would support IoT and what might need to change to both IoT applications, without adversely affecting the network.

While some expect the 5G architecture to be radically different from 4G [31], it will nonetheless be informed and derived from 4G principles and approaches [35, 36, 38, 85, 105]. In this work we follow a similar approach by considering the limitations of 4G networks, in particular the EPC core network, in supporting IoT and then combining evolved and refactored 4G mechanisms with software-defined networking (SDN) and network function virtualization (NFV) to realize an IoT friendly mobile (core) network architecture.

The 4G network architecture is heavy-weight and optimized for human-to-human and human-to-machine communication. This is, however, a poor fit for IoT devices, where we might expect a much larger number of devices with different traffic patterns. For example, millions of simple sensors that might send only a few bytes of data every half hour. Such applications might prefer low overhead over high quality data delivery.

Further, the control signaling in 4G networks has grown significantly and already present significant pressures on network components even just to support conventional devices and services. Control traffic grows 50% faster than data traffic while generating no revenue [14]. MME (a main control entity in LTE/EPC network) already experienced up to 1500 messages per User Equipment (UE) per hour under adverse conditions [56]. Scaling the control plane in 4G [35] would help mitigate the load problem. However, significant overhead in the protocols remains, especially when a large number of IoT devices attach to the network. This suggests fundamental changes in both network control and data planes in order to support IoT devices and services.

In terms of service abstractions, current 4G networks offer connectivity services for user devices to connect to other networks via a centralized gateway (packet data network gateway - PGW). Although this abstraction is suitable for human-centric devices where the network mostly needs to deliver an "Internet" service (e.g., web browsing), it is ill-suited for machine-to-machine or machine-to-human type communications. For example, peer-to-peer (P2P) communications, which are expected to be prevalent in IoT space, are not natively supported by the architecture and must be enabled by mechanisms such as NAT traversal on the gateways (PGWs) [70, 133]. The centralized nature of current 4G deployments [90] adds extra latency to the end-to-end path, especially for IoT P2P traffic whose end points might be geographically close. Also, since current mobile networks offer only network connectivity service via their mobile infrastructure, end-services are typically deployed in the Internet. This hurts the end-to-end latency and forces IoT devices to use the LTE service despite the unsuitable service abstractions.

Mobile network operators are considering (or in some cases already deploying) highly distributed small-sized data centers and clouds at the network edge [18]. Industry efforts are also underway to deploy distributed cloud platforms close to the mobile edge to support cloud based radio access network (RAN) functionality [49], as well as mobile edge computing efforts in support of traffic-offloading for low latency applications [63, 121]. This trend of highly flexible platforms close to the mobile network edge offers a new opportunity for network operators to more efficiently provide existing or emerging services and is specifically well aligned with emerging IoT services, some of which support an enormous number of devices, with low network latency requirements.

In this work we present an SDN-based IoT Mobile Edge Cloud Architecture (SIMECA). SIMECA leverages SDN for forwarding and NFV for network function deployment to realize a new service API that replaces LTE/EPC and is geared towards IoT devices and services. The design of SIMECA is based on following insights: (1) we argue that *best-effort* forwarding is more suitable for IoT devices instead of high QoS delivery in LTE/EPC. Best-effort forwarding eliminates core network components (i.e., SGW/PGW) and therefore reduces control plane overhead involving end-to-end connectivity; (2) Instead of extra tunneling overhead in LTE/EPC, SIMECA proposes a *packet header translation* mechanism realized by SDN forwarding that reduces packet header overhead and favors small payload traffic; (3) Separating end-point identity and routing identity together with a proper address resolution mechanism enable *seamless mobility* and *peer-to-peer* communications in SIMECA; and (4) Services and mobility functions are hosted inside edge clouds to improve network latency.

We make the following contributions:

- We propose a novel mobile edge cloud architecture, SIMECA, that leverages NFV, SDN and cloud computing to enable a new service abstraction that is more suitable for IoT devices and services. Our service abstraction enables direct P2P communication for IoT end devices. Our distributed architecture reduces the EPC core network latency significantly while being able to scale to multiple geographic areas.

- We propose novel control and data planes which scale better and have lower overhead to better support IoT services, compared to the current LTE/EPC control plane. Our design involves an IP header translation forwarding scheme which reduces data plane overhead by up to 20% for small sensor payload sizes (e.g., <100 Bytes). Our SDN-based control plane reduces control plane overhead by up to 37% while performing up to 76% faster.

- We validate our architecture by prototyping and evaluating our approach by refactoring precommercial EPC software and using an SDR-based eNodeB to realize SIMECA. Using applications running typical IoT protocols, our evaluations show promising results of SIMECA compared to LTE/EPC.

3.2.2 Motivation and overview

With reference to the current LTE/EPC architecture [117] our architecture targets three limitations in the current *EPC core* architecture, i.e., limitations in the radio access (including power consumption dues to radio protocols) and security aspects are not in our current scope and will be considered as future works.

- **Inappropriate service abstraction and inflexibility.** While built according to well-defined standards, the current LTE/EPC data network was designed to provide *Internet service* and its equipment is proprietary and closed. This results in several draw-backs for IoT devices: (1) These devices *must* use the LTE/EPC abstraction even when the abstraction is not well suited to its characteristics, i.e., LTE/EPC incurs high latency and overhead for IoT devices [80], (2) devices cannot establish a (native) peer-to-peer (P2P) communication, as the architecture does not offer a direct way to do it, and have to rely on additional techniques such as NAT traversal [70, 133].

We argue that there should be a more suitable service abstraction for IoT devices that should: replace the *Internet* abstraction in LTE/EPC by a new abstraction that is *more suitable* for IoT communication models, e.g., supports peer-to-peer communications natively. The abstraction should also be realized by control/data planes that favor IoT devices, e.g., less signaling and lower data overhead. Moreover, the network should apply NFV/SDN to allow flexible new service deployment and resource management, e.g., 3rd-party service providers could deploy their own services on top of a shared infrastructure provided by an operator.

SIMECA proposes a new IoT service abstraction (ISA) based on typical IoT communication models. *ISA* hides the complexity of supporting *mobility* and *connectivity* and exposes a simple interface for IoT devices to initiate both client-to-server and P2P communications. The *ISA* is realized using NFV/SDN technologies that run on a local edge cloud to enable flexible resource scaling and full end-to-end control (§ 3.2.3.2).

• **Heavy weight data and control plane.** The current LTE/EPC network was designed to support devices streaming large volumes of high quality data/voice (e.g., a smart phone streaming a video or making a voice call) rather than *massive* IoT/M2M devices with sporadic, best effort traffic (e.g., millions of meters sending a temperature sample every 20 minutes.) For example, data packets are delivered by EPS bearers in the EPC core network which are GTP tunnels. Although those tunnels support QoS (via different QCI indexes [23]) which might be needed for human devices (smart phones), setting up or maintaining them incurs significant overhead [90]. For example, a standard LTE initial attach procedure incurs up to 28 control messages, while a service request procedure incurs up to 14 control messages [24]. On the other hand, a sensor sending a temperature sample of 2 Bytes prefers a lighter control plane to reduce traffic delivery overhead. Moreover, an EPS bearer (which consists of 4 GTP tunnels uplink and downlink) also corresponds to 8 tunnel end-points (i.e., 2 at eNodeB, 4 at SGW, and 2 at PGW). This would add significant overhead to network elements when large numbers of IoT devices attach to the network.

In the data plane, the GTP tunnels add data plane overhead to IoT traffic. This overhead becomes relatively more significant if the packet's payload is small. For example, a body temperature sensor generates 1.5 Bytes of data per sample requires 36 Bytes of additional GTP/UDP tunneling overhead. Moreover, as the RAN capacity is expected to dramatically increase with emerging access technologies in 5G [58], increases in the amount of traffic would exacerbate the problem. For example, our analysis shows that if 90% of the LTE/EPC traffic consists of large packets (e.g., smart phones streaming videos with 1400B TCP packets) and the rest is small-payload traffic (e.g., meters sending 2B temperature samples), the packet header overhead (GTP/UDP) in the EPC core to deliver the total traffic is only 7%. However, if 90% of the traffic is small-payload traffic, the EPC core network header overhead increases to about 47%.

SIMECA proposes a *light-weight IoT control and data plane* to realize the IoT service abstraction (ISA). Specifically, the SIMECA design eliminates the notion of bearers in the EPC core network. Instead, SIMECA proposes to use light-weight, best-effort data forwarding in the data path right after the RAN. SIMECA leverages an intelligent-edge dumb-core principle: packet classification happens only at the base stations, while the dumb-core network simply forwards the packets. Moreover, instead of adding tunneling headers to packets, SIMECA's packet classification scheme *translates/replaces* packet header at the network edge (i.e., base stations) for forwarding, thereby reducing packet header overhead (§ 3.2.3.3). By eliminating tunnels inside its network, SIMECA reduces the control plane signaling to maintain or modify them. Removing tunneling also reduces the number of fine-grain forwarding states (i.e., tunnel IDs) in the network (§ 3.2.3.4).

• **Centralized deployment of core mobile network functions.** In current LTE/EPC networks, core mobile network functionality, e.g., data-path elements that forward traffic to the Internet (i.e., SGW and PGW), as well as control plane elements dealing with mobility (i.e., MME) and policy (i.e., PCRF), are specialized hardware-based equipment or virtualized NFVs deployed in a limited number of physical locations. Traffic between endpoints needs to travel to these centralized locations, adding extra latency to the end-to-end path. The vision for 5G networks [25, 58] and recent efforts [53, 147] call for services that will require significantly lower latencies (e.g., <5ms for tactile Internet), which the current LTE/EPC architecture will not be able to support.

On the other hand, IoT/M2M devices are mostly static or have a low degree of mobility [58, 138]. This gives an opportunity to move away from a centralized network architecture (e.g., a large geographical area shares a PGW) to a more distributed architecture with computation and connectivity mostly local (e.g., spanning a metro area). A more "local" environment would allow local deployment of computational resource (i.e., services) and therefore reduce end-to-end latency.

We argue for a more distributed architecture where core mobile network functions, as well as compute resources, are placed closer to the mobile network edge, i.e., a *mobile edge network and edge cloud* architecture. The SIMECA architecture consists of regions, each covering a relatively small geographical area. The distributed edge cloud and edge network reduce network latency as mobility functions and services are deployed closer to users and help solve the scalability problem associated with centralized data centers.

3.2.3 SIMECA architecture

We present our work on SIMECA, an SDN-based IoT Mobile Edge Cloud Architecture to enable efficient and flexible support of IoT services in future mobile networks. This architecture is based on three main assumptions/design principles: i) A mobile infrastructure consists of a set of SDN-enabled base stations, an SDN-enabled mobile edge network, and an edge cloud (Fig. 3.11). Both services and virtualized mobility functions (NFVs) are deployed in the edge cloud. (ii) A new service abstraction, i.e., *IoT service abstraction (ISA)*, is designed with fully functional, low overhead and scalability goals in mind, running on top of the infrastructure to provide connectivity and mobility that is optimized for IoT devices. (iii) The infrastructure and the abstraction are able to scale to a large geographical area by dividing an area into multiple smaller regions each controlled by a separated control entity.

3.2.3.1 SDN-based mobile edge network and cloud infrastructure

Figure 3.11 depicts the infrastructure that SIMECA runs on. The infrastructure is distributed and consists of smaller regions. (We note that our assumed infrastructure is aligned with broad industry vision towards an infrastructure in support of mobile edge computing [63, 121].) Each region consists of a set of SDN-enabled base stations, an SDN

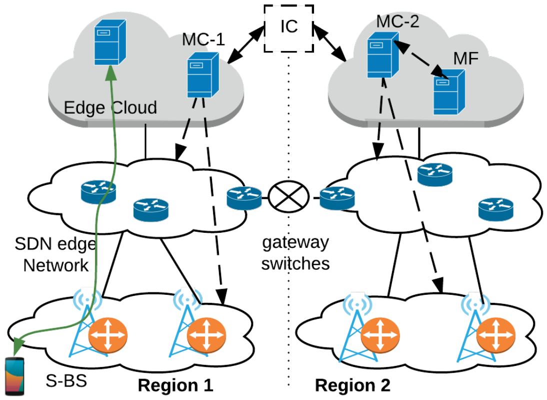


Figure 3.11. Mobile edge network and cloud infrastructure

edge network, and a local edge cloud. Each region has a gateway switch and regions are interconnected by programmable optical links with low latency [10]) (§ 3.2.3.5). We envision a region could correspond to one or several metropolitan areas and provide a low latency network. Specifically, the infrastructure consists of:

• *SDN-enabled base stations - S-BS*: In SIMECA base stations are SDN-enabled, allowing them to apply the SDN match/actions paradigm to modify flows as dictated by an SDN controller. The SDN-enabled Base stations are responsible for packet classification (i.e., modifying packet's source and destination IP) for packets forwarding inside the SDN edge network.

• *SDN edge network*: The SDN edge network consists of multiple SDN switches that connect the S-BS and a local edge cloud. This SDN edge network has several gateway switches that connect to neighboring regions SDN. intraregion traffic is forwarded by this SDN edge network based on header destination address, while interregion traffic goes across regions via gateway switches.

• *Edge cloud*: The edge cloud consists of multiple computation nodes that are capable of running *both* virtualized mobility functions (NFVs) and IoT services. The edge cloud is connected to the SDN edge network via an SDN switch to enable end-to-end SDN control.

We envision the edge cloud infrastructure explained above would be managed by a single *infrastructure operator*. The infrastructure is shared by multiple IoT *service providers* who deploy and manage their own IoT applications. Infrastructure resource allocation and isolation are performed by the infrastructure operator (e.g., network slicing). A service provider requests a *slice* of network and compute resource of the shared infrastructure to run their service on demand. For example, an IoT service provider may request an additional slice from the infrastructure operator to host their new Health Monitoring Service. The service provider can also scale the resource when the number of patients grows.

Compared to the current 4G infrastructure, SIMECA's infrastructure would need an additional distributed mobile edge cloud infrastructure and SDN programmability in base stations. The distributed mobile edge cloud is well aligned with broad industry vision of a highly distributed edge infrastructure to support mobile edge computing [63, 121]. The SDN programmability in base stations could be implemented either in the base station architecture or by adding a SDN-enabled switch colocated with the base station. SIMECA could transparently coexist with current 4G networks as long as there is a mechanism at the Radio Access Network to distinguish and forward SIMECA traffic to SIMECA infrastructure while leaving normal 4G traffic untouched [52].

3.2.3.2 IoT service abstraction (ISA)

In this section we provide details on how ISA abstracts the mobility and computation infrastructure to provide a simple interface to IoT devices.

• **ISA exposed interface.** Unlike conventional smart phones, there are two communication models for IoT applications: client-server and publisher-subscriber. The client-server model is suitable for one-to-one communications, e.g., client controls an actuator. The publisher-subscriber model is suitable for many-to-many communications, e.g., multiple publishers publish data to a broker and multiple subscribers get data from the broker. These two models suggest both device-to-edge cloud communication (i.e., we call this C2S communications) and device-to-device communication (i.e., we call this P2P communications) [17]. The ISA therefore exposes to the IoT devices computational resources and an interface to request *end-to-end connectivity* between end IoT devices (i.e., P2P connectivity), or between devices and services (i.e., C2S connectivity). Table 3.1 shows the interface that an ISA instance exposes to a service provider.

To manage computational nodes (VMs) for its service, an IoT service provider could instantiate a number of VMs in the edge cloud using the same interactions as it would normally do in existing clouds (e.g., Amazon EC2), i.e., the network operator exposes a north-bound API through its Infrastructure Controller (IC in Fig. 3.11 - The IC is out of scope of this work. Therefore we depict it as a dashed box) and lets the service providers request resources (VMs) dynamically. The IC then provisions and exposes the VMs back to the service providers. Having instantiated its VMs, the service providers could deploy

Table 3.1. Interface exposed by an *ISA*

Interface	Effect	Example
C2S-attach	Connect a device to a server	*Device A attaches to server 1*
P2P-attach	Connect a device to another device	*Device A attaches to device B*

their services on the VMs and let IoT devices connect to the services (VMs). Similarly, they can scale up/down the VMs as needed.

The ISA is realized by two control plane entities: Mobility Network Functions (MF) and Mobility SDN Controller (MC). Unlike LTE/EPC, the two control entities in SIMECA could run on commodity servers and therefore the control plane could scale relatively easily.

• **Mobility functions (MF).** The MF mainly deals with mobility specific functionalities such as: tracking the location of a device, performing mobility procedures with devices and base stations. As shown in Figure 3.12, the MF consists of 3 components (depicted in yellow boxes): *Mobility Interface, Device Tracking, and Identity Management*. The *Mobility Interface* component speaks standard protocols with the devices and base stations (i.e., NAS and S1AP protocols in 3GPP [20]). Using the interfaces provided by this entity, an IoT device can request end-to-end connectivity in the network (i.e., Attach procedure). This enables SIMECA to operate with base stations with normal 3GPP control plane. The *Device Tracking* component keeps track of the attachment point of devices that move around, i.e., it keeps track of which base station a device currently attaches to. By tracking the attachment points of devices, SIMECA is aware of where to forward traffic to when devices move. The *Identity Management* component manages identities for devices. Specifically, a device in SIMECA has two identities: one for end-point application use and one for network routing. This identity separation allows seamless handover in SIMECA (details about identity, device tracking will be described in more detailed in § 3.2.3.3).

For example, a device turns on, authenticates with the network using existed protocols, and sends an "Attach to Server A" request to the *Mobility Interface* (step 1 in Fig. 3.12). The *Mobility Interface* then requests identities with the *Identity Management* and replies to the device. The *Identity Management* updates the *Device Tracking* component with the current location of the device (step 2).

• **Mobility SDN controller (MC).** While the MF deals with mobility functionalities, the MC mainly deals with end-to-end path implementation via an OpenFlow API, i.e., after dealing with mobility requests and assigning identities, the MF notifies MC to implement paths accordingly. As shown in Figure 3.12, the MC consists of 4 components (depicted in green boxes): *Connectivity, Handover, OpenFlow (OF), and Network Information Base (NIB).*

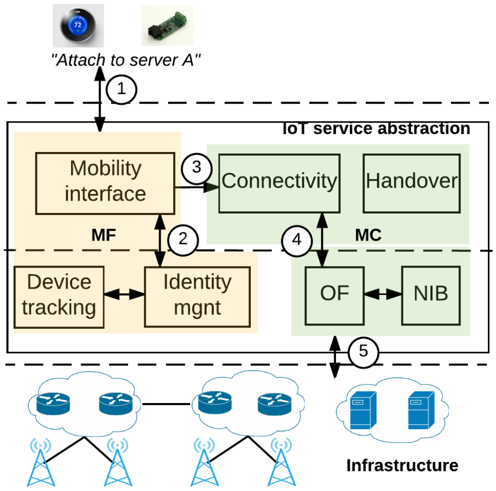


Figure 3.12. IoT service abstraction (ISA)

The *Connectivity* component is specific to implement end-to-end paths (e.g., set up a path from a device to a server when the device attaches), while the *Handover* component is responsible for modifying paths during mobility events (e.g., forward traffic to a new base station if the device moves). The two components use OpenFlow interfaces (*OpenFlow* component) in combination with network topology information (i.e., *NIB* component) to implement the paths. The NIB includes information such as on which base station OpenFlow rules should be installed to realize end-to-end connectivity.

For example, after receiving the "Attach Request" from the device, the *Mobility Interface* tells the *Connectivity* component in the MC to install a path from the base station (e.g., cell-ID) to server A (step 3 in Fig. 3.12). The *Connectivity* component uses the *NIB* to translate the base station ID to the SDN data path object and use *OF* to realize the end-to-end connectivity (step 4, 5).

3.2.3.3 Lightweight IoT data plane

In contrast to heavy-weight tunneling with QoS guarantees in the standard EPC core, SIMECA's best-effort forwarding data plane could benefit IoT services. It: (1) reduces packet header overhead for IoT devices, (2) reduces the number of forwarding states inside the network, while (3) enabling seamless mobility.

- **Device identity and forwarding identity.** A device in SIMECA is identified using two identities: *device identity* (DI) and *device routing identity* (RI). Each device has a unique DI assigned by the network when the device attaches to the network. This DI is associated with the device and *does not change* until the device detaches from the network. As this DI does not change upon device mobility, applications running on the device use this DI to enable continuous operation, i.e., seamless mobility. In the SDN edge network, packets are forwarded using a separate RI (different from the DI). This RI is also allocated by the network when a device attaches to the network. Unlike DI, RI is specific to a base station and *changes* when a device moves to another base station. Under a specific base station, there is a unique mapping between DI and RI.

Note that SIMECA's device identity is simply an identity that is compatible with the end-device's network stack. This allows SIMECA to work regardless of the IP technology used for the device identity (i.e., a device identity could be an IPv4 or IPv6 address, or even a hash string). Similarly, the routing identity used in SIMECA is also flexible; the only constraint for routing identity is that it must have two parts as described in Figure 3.13. An operator might choose to allocate different sizes for each part (BID and EID) in a routing identity depending on network topology and other optimizations for an IoT service.

• **Lightweight header translation.** Unlike in EPC where extra tunnel overhead is added to packets at eNodeB and S/PGW for delivery over a transport network, SIMECA does not add *any* extra overhead to packets. SIMECA instead uses header translation mechanism to translate DI-RI at the network edge (i.e., base stations). Inside the SDN edge network, packets have source and destination RIs in their header for forwarding. When the packets reach end-points, they have DIs as their source and destination for seamless mobility. When a device moves across base stations, it has a new RI and therefore the DI-RI mapping also gets updated accordingly. To realize this DI-RI mapping, an SDN controller installs SDN rules to base stations in a *proactive* manner during device attach/handover.

• **Seamless mobility.** SIMECA supports seamless mobility as it separates device identity, which is unchanged by mobility, and forwarding identity which changes with mobil-

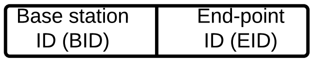


Figure 3.13. Routing identity header structure

ity. When a device moves to a new base station, it is assigned a new RI that has the new base station's BID. Note that the DI of the device does not change. Therefore from the end-points (device, server) perspective, the RI change is *transparent*.

For example, device A with DI of *192.168.3.33* attaches to base station 1 and gets RI *192.168.10.10/24*. When it moves to base station 2 it gets a new RI *192.168.20.10/24*. Packets inside the SDN edge network are now forwarded to base station 2 where device A is currently attached using the new BID *192.168.20.0/24*. However, packets arriving at device A and server still have device A's DI *192.168.3.33* in the header.

• **P2P forwarding and device location tracking.** Unlike EPC, SIMECA naturally enables peer-to-peer communication in which a device is reachable via its DI. This type of communication is enabled when a device requests a *P2P-attach* to the other device with the destination device's DI specified (e.g., "Device DI1 attaches to device DI2", Section 3.2.3.2). When receiving a P2P-attach request, the SIMECA control plane needs to know the RI of the destination device to install SDN rules in the source base station for DI-RI header translation. This DI-RI mapping information is dynamic as the peer device moves across base stations. SIMECA control plane stores this DI-RI mapping in a centralized table (location tracking table) and updates the table accordingly when a device changes its attachment point (hands over to another base station).

Table 3.2 shows an example of location tracking table in SIMECA. A new entry is created by SIMECA control plane when a device attaches *and* registers itself as a reachable service (Section 3.2.3.2). It is updated when the registered device moves across base stations (more details in Section 3.2.3.4).

• **Example of packet forwarding.** Figure 3.14 shows the translation of device identity to routing identity at base station for intraregion forwarding: P2P forwarding on the left and C2S forwarding on the right. For P2P forwarding, device 1 initiates a flow to device 2 in the *same* region. The packet header at device 1 is $[DI1, DI2]$ where $DI1$ and $DI2$ are

Table 3.2. Attachment point tracking table for P2P communications

Device identity	Routing identity
192.168.3.33	192.168.10.10/24
192.168.3.34	192.168.11.11/24

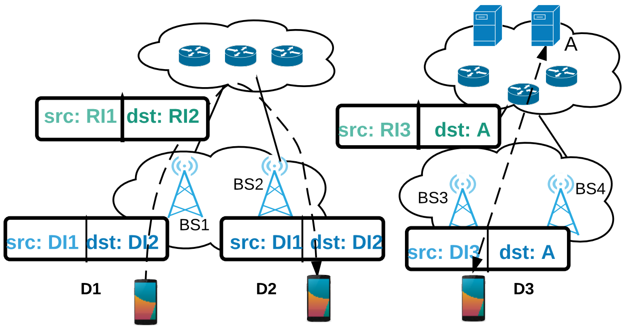


Figure 3.14. Packet header translation: packet forwarding example

device 1 and 2's *device identity*. Base station 1 has an SDN flow rule that *replaces* uplink packets header with device 1 and 2's *routing identity* and forwards the packets to the SDN edge network. The SDN edge network has preinstalled SDN rules that matches on base station ID in packet header (i.e., $[RI1, RI2]$).

Upon arriving at destination base station 2, the packet header is translated back to $[DI1, DI2]$ and delivered to device 2.

Similarly for C2S forwarding, base station 3 translates packets source address from $DI3$ to $RI3$. The destination address which is the server address does not change (i.e., A in Figure 3.14). The SDN edge network forwards uplink packets based on server destination address. For downlink packets from the server, the SDN edge network simply forwards packets by matching the destination address in the packet which is embedded in the uplink packets (i.e., $RI3$).

• **Reduce number of forwarding states.** To reduce the number of forwarding states inside the SDN edge network, SIMECA uses an intelligent-edge dumb-core principle. In the SDN edge network, RI has a predefined structure that aggregates end-points in the same base station. Specifically, as shown in Figure 3.13, an RI consists of a *Base station ID-BID* and a *End-point ID-EID*. The BID specifies a base station in the network and EID specifies a device in that base station. Inside the SDN edge network, packets are forwarded by matching on BID *only*. When packets arrive at the edge (base stations), the base stations demultiplex packets to corresponding devices using EID. Therefore, the number of forwarding states in the SDN edge network only depends on the number of BIDs allocated regardless of the number of devices.

For example, if using the IP address structure, a RI would look like *192.168.10.10/24* where *192.168.10.0/24* is the BID and *192.168.10.10* is the EID. Inside the SDN edge network, packets are forwarded using the BID *192.168.10.0/24*. Upon reaching a base station, the EID *192.168.10.10* is used to deliver the packets to the corresponding device (using DI-RI mapping in the base station).

For each end-to-end connection, there are only 2 SDN rules installed at base station in SIMECA (the rules in the SDN edge network are shared among devices of the same base station), while there are 8 GTP tunnel IDs maintained per EPS bearer in the EPC core network (i.e., 2/4/2 GTP tunnel IDs at eNodeB/SGW/PGW per bearer).

3.2.3.4 Lightweight IoT control plane

• **Eliminate tunneling to reduce control plane overhead.** By eliminating EPC tunnels in SDN edge network and allowing packet classification at the edge, SIMECA eliminates control signaling overhead to set-up/maintain the tunnels and reduces the number of forwarding states in the data path. Compared to EPC, SIMECA's control plane is more lightweight for multiple reasons: (1) packet classification happens only at the network edge (base stations) and therefore requires fewer interactions to set-up a path (i.e., the SDN controller only needs to push OpenFlow rules into the base station as opposed to interactions between eNodeB, SGW, PGW and MME in EPC); (2) SIMECA's dumb-core eliminates the overhead to set-up forwarding states in SDN edge network as compared to GTP-U tunnels in EPC, which also results in a lower number of forwarding states in the data plane in SIMECA; (3) unlike in LTE/EPC networks the forwarding states in SIMECA do not expire due to radio bearer release, therefore eliminating the control signaling overhead incurred by re-establishing the data path when devices become active again after an idle period (i.e., Service Request and Paging), and (4) OpenFlow control messages are more lightweight than EPC bearer creation/modification messages.

Figure 3.15 shows forwarding states in the data path of SIMECA (upper) and LTE/EPC (lower). As SIMECA classifies packets at the network edge, only 2 OpenFlow rules (red arrows) are installed in each base station per C2S-attached *device* as opposed to 4 GTP-U tunnels (*uplink and downlink*) per *EPS bearer* (i.e., according to 8 forwarding states or tunnel IDs) at eNodeB, SGW, and PGW in EPC. Installing OpenFlow rules at base stations also incurs less control signaling than setting up tunnels between multiple components as in

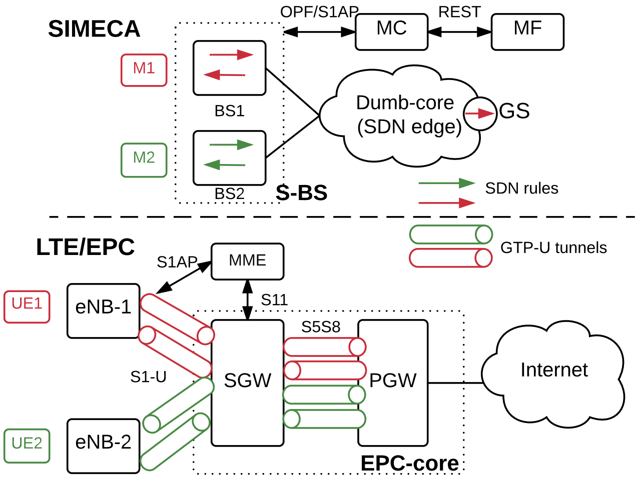


Figure 3.15. Tunneling in EPC vs. SDN rules in SIMECA

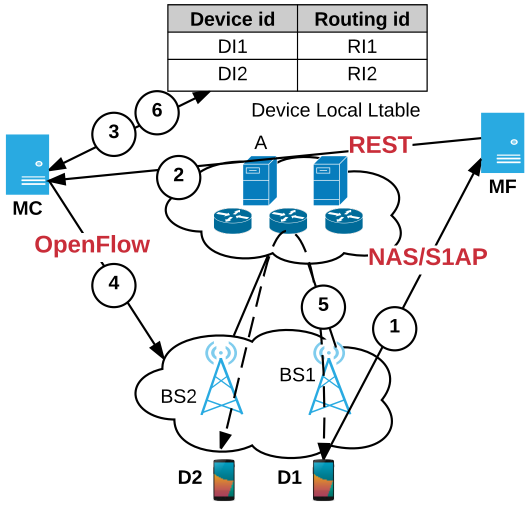
EPC: only 2 OpenFlow *flow_mod* messages and a RESTful message are needed for each C2S-attached *per device* as opposed to 8 GTP-C and S1AP messages exchanged on S11, S1AP, S5S8 interfaces as in EPC [24] to set up both ends of each tunnel.

• **Example of OpenFlow rules installed in a P2P attach procedure.** Figure 3.16 shows how SIMECA controllers interact and install OpenFlow rules to implement a P2P path between device D1 and D2 when D1 requests a P2P attach to D2. Table 3.3 shows OpenFlow (OF) rules installed in base station 1 (BS1) and base station 2 (BS2) after each control plane step. To be specific, in this example let's assume (DI,RI) of D1 is *(192.168.3.33, 192.168.10.10/24)* and (DI,RI) of D2 is *(192.168.3.34, 192.168.11.11/24)* (*x* in the table represents the prefix *192.168*).

At the beginning, D2 is already attached to the network. Its (DI,RI) is recorded in

Table 3.3. OpenFlow rules installed at base stations to implement a P2P connection in SIMECA

Step	Base Station	OpenFlow Rules on Base Station
4	1	1.**Match**(from_RAN, ip.src==x.3.33, ip.dst==x.3.34) **Action**(ip.src←x.10.10, ip.dst←x.11.11, to_SDN) 2.**Match**(from_SDN, ip.src==x.11.11, ip.dst==x.10.10) **Action**(ip.src←x.3.34, ip.dst←x.3.33, to_RAN)
	2	1.**Match**(from_SDN, ip.src==x.10.10, ip.dst==x.11.11) **Action**(ip.src←x.3.33, ip.dst←x.3.34, to_RAN) 2.**Match**(from_RAN, ip.src==x.3.34, ip.dst==x.3.33) **Action**(ip.src←x.11.11, ip.dst←x.10.10, to_SDN)



Device id	Routing id
DI1	RI1
DI2	RI2

Figure 3.16. Control plane interaction during a P2P attach request

the *Device Location Table (Ltable)* (we omit OF rules installed when D2 initially attached in the table). At step 1, D1 requests a P2P attach to D2 using DIs (i.e., 192.168.3.33 attaches to 192.168.3.34). As this is the first time D1 is seen, the MF allocates a RI for D1 (i.e., 192.168.10.10/24). MF then tells MC to install a path between D1 and D2 (step 2). As D2 could be located anywhere in the network, MC refers to Ltable to obtain D2's RI (i.e., 192.168.11.11/24) and its location (i.e., BS2, step 3). After having all information, MC pushes OF rules to the base stations to implement a P2P path between D1-D2 (step 4). After this step, OF rules installed in BS1 and BS2 are shown in Table 3.3: the path D1-to-D2 is implemented by flow rules #1, while the reverse path, D2-to-D1, is implemented by flow rules #2. After the MC installs the path, it updates (DI,RI) of D1 in the Ltable (step 6).

Compared to Locator/Identifier Separation Protocol (LISP) [64], SIMECA's data plane and control plane are different in several points. First, in the data plane instead of encapsulating packet header with an additional header (i.e., map-n-cap mechanism) or adding additional routing identifier into the packet header (i.e., address rewriting mechanism), SIMECA *replaces* the entire header (DI) with a routing identifier (RI). This allows SIMECA to use DI and RI that have flexible formats, e.g., IPv4 or IPv6 or even a hash string, while not adding additional header overhead. Second, the mechanism that triggers a data path modification in SIMECA is different from LISP; unlike LISP where the Endpoint Identifier-to-Routing Locator (EID-to-RLOC) mapping is fetched from a mapping database when the first (unknown) data packet of a flow arrives, SIMECA's control plane *proactively* pushes OF rules into data plane as a side effect of a control plane request from the end-device (e.g., a P2P attach request). This proactive path implementation helps switch a data path faster compared to the LISP's mechanism especially when the end-point moves rapidly in a mobile environment.

• **Control plane overhead analysis of SIMECA procedures.**

• *Device attach.* Compared to the LTE/EPC attach procedure, SIMECA's attach procedure does not involve signaling to set up bearers between multiple components (i.e., eNodeB, SGW, PGW and MME). Instead, the SDN controller of the region (MC), only needs to install OpenFlow rules at base stations. In term of forwarding states (OpenFlow rules) in the data plane, SIMECA also has fewer states compared to EPC: there are 2/4 OpenFlow rules (as shown in Table 3.3) to realize an intraregion C2S/P2P path *per device*, while EPC needs 8/16 GTP-U tunnel states for *each bearer* (i.e., 2 at eNodeB, 4 at SGW, and 2 at PGW).

In terms of control signaling, for an intraregion connection setup, SIMECA incurs 2/4 $flow\_mod$ OpenFlow messages to set up an end-to-end C2S/P2P path, respectively (and a table lookup that is supposed to be local at the MC). Compared to LTE/EPC which involves 8 messages (i.e., 6 GTP-C messages for S11 (MME-SGW) and S5S8 (SGW-PGW) tunnels set up and 2 S1AP messages for tunnels set up at eNodeB [24]), SIMECA saves 42% and 37% of signaling traffic per C2S/P2P attach, respectively (as shown in Table 3.4). Note that SIMECA reuses the authentication procedure of LTE/EPC so that it has the same control signaling overhead for authentication.

• *Service request and paging.* In LTE/EPC, when a device does not have data to send/receive, it will go idle and the network releases its radio resources at the eNodeB (i.e., a device enter EMM-Registered and ECM-idle state after about 15s in LTE). As the network maintains EPS bearers across the EPC core and the RAN, when the radio bearer (i.e., Data Radio Bearer - DRB) is released at the eNodeB, its associated GTP tunnels to the SGW (i.e., S1-U/S1-C tunnels) also get released. This results in control signaling overhead to re-establish the tunnels when the device has data to send (i.e., Service Request procedure). Similarly, in the downlink direction if there is traffic to the device, the MME needs to trigger the Paging procedure to page the device before re-establishing the tunnels as in Service Request.

Table 3.4. Control plane message size, LTE/EPC vs. SIMECA

Event	Interface	Message	LTE/EPC	SIMECA
Attach	S11	Create Session Req./Res.	564B	x
	S5S8	Create Session Req./Res.	369B	x
	Control	flow_mod	x	48B×2
	S1AP	Auth, Security	1060B	1060B
	Total		**1993B**	**1156B** (42% ↑)
Handover	S11	Modify Session Req./Res., Create Fwd. Tun. Req./Res., Delete Fwd. Tun. Req./Res.	1302B	x
	Control	flow_mod	x	48B×3
	S1AP	HO Req./Res./Noti.	872B	872B
	Total		**2174B**	**1016B** (53% ↑)

On the other hand, SIMECA does not maintain the EPS bearer notion and offers best effort delivery. This allows the data path at base stations and the SDN edge network to be reused when the device has data to send/receive without incurring extra control signaling to set up the path. This reduces 5 control messages per Service Request. As Paging includes a Service Request, this saving also holds for Paging. Moreover, if the DRB information exists at the base station for the device being paged, SIMECA would require no paging request for that device and therefore it could save 7 control messages.

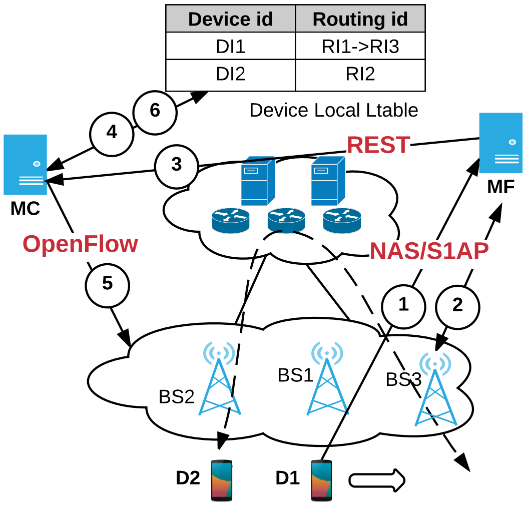
• *Device handover.* Similarly, to modify the flows between source/target base station and the destination, SIMECA incurs 4/5 *flow_mod* messages to maintain a C2S/P2P connection of the mobile device. In EPC, to realize path switch and forwarding tunnels to support lossless handover, there are 8 control messages incurred (i.e., 6 GTP-C messages on S11 interface and 2 S1AP messages for S1 handover [24]). Compared to LTE/EPC, SIMECA reduces 53% and 49% of control traffic for the two types of handovers (as shown in Table 3.4.)

• **Control plane interaction.** As discussed in § 3.2.3.2, the control plane consists of following logical components: *Mobility Functions - MF*, *Mobility SDN controller - MC*. Also, to keep track of device attachment point, a *Device location table-LTable* is used. We will only describe the interaction of SIMECA control components during a *Handover* procedure.

Figure 3.17 shows interactions when device 1 (D1) handovers and maintains an end-to-end connection with device 2 (D2). At step 1, D1 notifies the MF that it is handing off to base station 3 (BS3). MF makes a request to BS3 and notifies the MC about the handover (step 2). MC looks up D1 and D2's current RIs in the local Ltable (step 4) and installs OpenFlow rules to realize the path switch (step 5): (1) at BS2, packets heading to D1 are now forwarded to BS3 (originally BS1); (2) at BS3, a new pair of OpenFlow rule (downlink/uplink) is installed for D1 so that packets arriving at BS1 will be forwarded to D1; (3) in-flight packets that are heading to BS1 are forwarded to BS3 by a triangular path set up between BS1 and BS3. The MC then updates the table with the new RI (i.e., RI3, assigned by the MF at BS3) for D1 (step 6). This completes the handover procedure.

Note that SIMECA's control plane uses SDN technology because of the inherent programmability of SDN and the well-defined protocol (i.e., OpenFlow). However, SIMECA architecture itself is independent of protocols used. If other protocols, e.g., I2RS1 [76], provide this programmability, SIMECA can operate over those protocols. Also, if SDN-enabled switches support BGP and MPLS-TE, SIMECA could control these protocol behaviors by appropriate North Bound/South Bound interfaces. Moreover, other alterna-

Figure 3.17. Control plane interaction during a handover



Device id	Routing id
DI1	RI1->RI3
DI2	RI2

tives would be recent proposals that enable SDN to communicate with legacy protocols, e.g., SDN and BGP in ONOS's SDN-IP use case [13].

3.2.3.5 Multiple-region communications

A single Mobility controller and Mobility Function can only support a limited number of devices in a limited area. To scale to a large geographical area, multiple instances of MC and MF could be deployed in different smaller regions. We refer to connectivity that extend through two regions as *Interregion* communications. The following describes Interregion forwarding and the interaction in the control plane to make this possible.

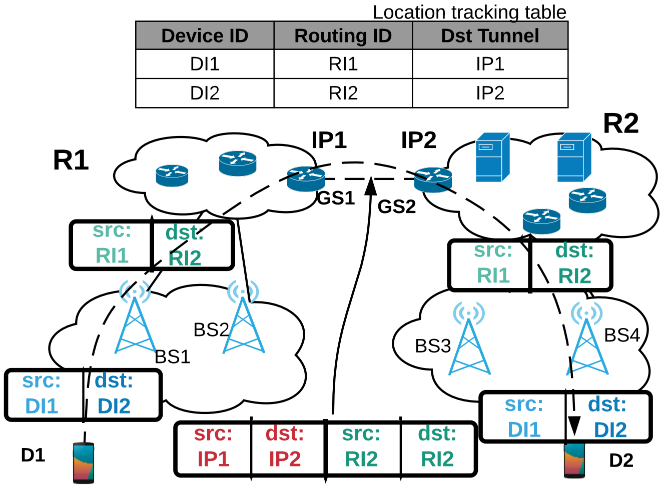
• **Interregion forwarding.** Between regions, SDN edge networks are interconnected by gateway switches (GSs). Gateway switches are able to deliver IP packets amongst them. As shown in Figure 3.18, when device 1 initiates a connection to device 2 in another region, the destination *RI* address of its packet does not match any base station within its region. The SDN edge network forwards that packet to its interregion gateway switch (GS1). The gateway switch encapsulates the packets with $[IP1, IP2]$ IP addresses to deliver packets to region 2. The gateway switch of region 2 (GS2) then decapsulates the IP packets and matches on the inner *RI* to deliver the packets to device 2[1].

To find the destination tunnel IP for a device for interregion delivery (e.g., as shown $IP2$ in Figure 3.18), SIMECA controller looks up the DI-RI mapping table (i.e., as shown in column 3 in the table in Figure 3.18).

For example, an end-to-end packet processing for traffic initiated from device 1 looks as follows: at base station 1 (BS1), packet header is translated from $[src : DI1, dst : DI2]$ to $[src : RI1, dst : RI2]$. At GS1, packets heading to RI2 are encapsulated using IP2. When packets arrive at GS2, they are decapsulated and forwarded to BS4 using RI2 matching. At BS4, packet headers are translated from $RI1, RI2$ to $DI1, DI2$. Similarly, the reverse processing happens for the returning direction.

• **Interactions between control components.** Figure 3.19 shows the interaction between components when a device requests P2P connectivity to another device in *a different region*. D1 requests an attach to D2 (step 1). The MF assigns a new routing identity

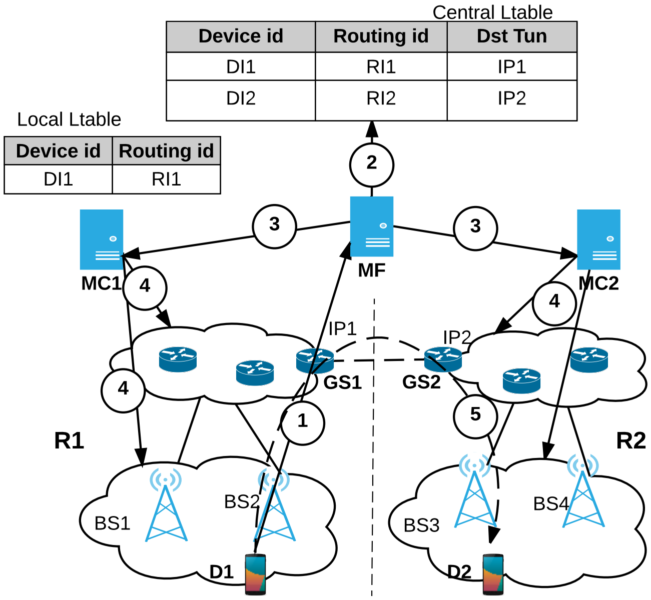
[1] Note that our design is independent of the transport network between SDN gateway switches. Therefore it can run other protocols such as MPLS for scalability.



Location tracking table

Device ID	Routing ID	Dst Tunnel
DI1	RI1	IP1
DI2	RI2	IP2

Figure 3.18. P2P packet header translation: interregion



Central Ltable

Device id	Routing id	Dst Tun
DI1	RI1	IP1
DI2	RI2	IP2

Local Ltable

Device id	Routing id
DI1	RI1

Figure 3.19. Interregion routing

RI1 for D1 and looks up the *centralized Ltable* for device 2's routing identity (RI2) and the destination tunnel IP (i.e., IP2) (step 2). The MF then selects ingress/egress GSes in the source/destination regions. After having the ingress/egress GSes, the MF notifies the region's SDN controllers, MC1 and MC2, about the request (step 3). The MC then implements (1) paths to/from GSes (i.e., BS2 to GS1, BS3 to GS2 in Figure 3.19), and (2) packet encapsulation at GSes (step 4).

• **Recursive device tracking for scalability.** As the number of devices increases, the number of entries in Location Tracking Table could also increase and the workload of SIMECA's controllers (MC/MF) will increase. In order to scale to a large network, SIMECA divides the network into multiple smaller regions each controlled by independent controllers. To maintain a large number of device identities, i.e., Device ID and Routing ID, SIMECA adopts a hierarchical approach to structure the Location Tracking Table: the table is organized as a tree structure with the leaf tables containing entries of the smallest regions and the root table containing entries of the entire network. At the leaf regions, each region (a metropolitan area or smaller) maintains a local Location Tracking Table keeping track of devices and their mobility in that local region. If a device moves across regions, the parent table that covers the two regions will be updated. If a device ID is not found in the local region's Tracking Table, the SIMECA controller (MC) will recursively search the parent region's tables until the ID is found. This hierarchical structure enables fast look-up and update at the leaf regions while allowing SIMECA to scale to multiple regions in the meantime.

3.2.4 Prototype implementation

• **SDN-enabled base station.** Figure 3.20 shows our base station architecture. We implemented the SDN-enabled base station (S-BS) by combining a node running a refactored OpenAirInterface (OAI) software-defined radio eNodeB implementation [11], with a node running OpenvSwitch (OVS) [119]. We reuse OAI radio stack and radio control plane (i.e., RRC, PDCP, RLC, MAC/PHY) and modify the core network data path. Specifically, in the uplink direction, the radio stack PDCP (Packet Data Convergence Protocol) task associated with a radio bearer places IP packets in an Intertask Interface (i.e., ITTI) queue. A raw socket task reads the queue and forwards IP packets to the node running the OpenvSwitch, which in turn connects to the SDN edge network. Similarly, downlink packets from the raw socket task will be placed in the ITTI queue. There is a relay demultiplexer that keeps the mapping between radio bearers (or radio task) and packets destination IP addresses (the mapping is recorded during bearer setup and device attach/handover). This relay associates the downlink packets to corresponding PDCP radio tasks based on the packet destination IP address. The mobility SDN controller interacts with the base station Open-

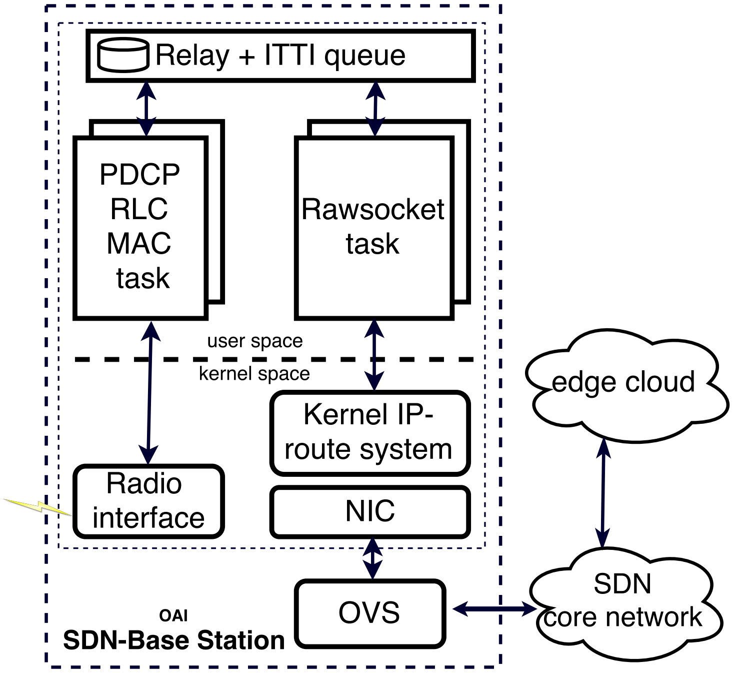


Figure 3.20. SDN-enabled base station

vSwitch via the OpenFlow protocol. Note that this SDN-enabled architecture could also co-exist with conventional LTE/EPC eNodeBs as it has similar relay functionality in the data path. Figure 3.21 shows our prototype. We used an unmodified Nexus 5 device that speaks the normal LTE protocols to interact with SIMECA as SIMECA preserves the S1 interface for backward compatibility.

• **Mobility function component.** We implemented the MF component by refactoring an OpenEPC [55] MME implementation. The MF reuses the MME authentication functionality but implements SIMECA's simplified device attach and handover procedures: The MF shortcuts the attach and handover procedure by removing control messages to set-up/modify tunnels at SGW and PGW. When it receives an "Attach request", the MF notifies MCs to install an end-to-end path and sends "Attach accept" to notify the device when OpenFlow setup is complete. For communication between MF and MCs, we have designed RESTful APIs.

• **Mobility SDN controller.** We implemented the Mobility SDN controller using the Ryu controller. MCs constantly listen for HTTP requests from the MF and install OpenFlow rules onto SDN-enabled base stations accordingly. For intraregion routing we used a

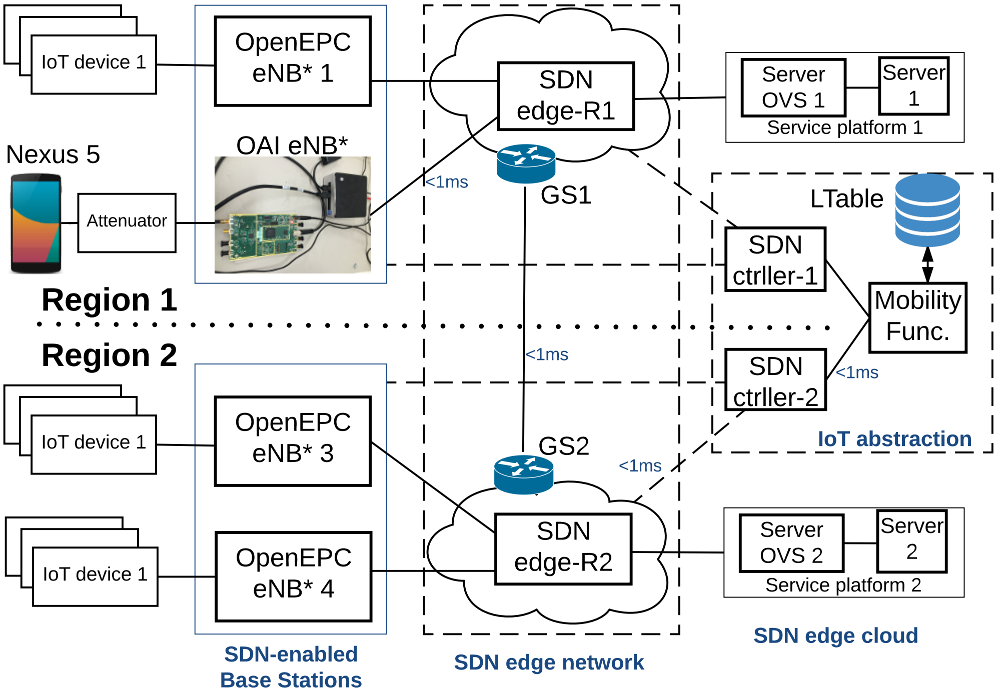


Figure 3.21. SIMECA prototype implementation

simple shortest path implementation that performs forwarding based on matching on an IP prefix: the SDN controller computes shortest paths based on a known topology of the prototype. For interregion forwarding we use GRE tunneling for simplicity (MPLS should be used in actual deployment for better scaling). In our implementation, we used IPv4 addresses for SIMECA device identity.

3.2.5 Evaluation

We used the PhantomNet testbed for our evaluation [51]. Figure 3.21 shows the evaluation topology. It consists of two regions: each has two SDN-enabled base stations (both SDR-based and emulated), an SDN edge network consisting of Open vSwitches (e.g., SDN edge-R1), and a server platform mimicking an edge cloud. The two SDN edge networks are connected via two gateway switches that are also Open vSwitches (i.e., GS1, GS2). The emulated SDN-enabled base stations were developed by refactoring an OpenEPC eNodeB implementation, while the OAI implementation described in Section 3.2.4 executed on a USRP B210 SDR platform. The two emulated base stations each connect to a node that emulates multiple IoT devices. The OAI base station connects to a Nexus 5 smart phone via a programmable radio frequency attenuator. In the control plane, the IoT abstraction consists of two MCs (i.e., SDN controllers, each per region), and an MF (i.e., Mobility Function, shared by the two regions). The location tracking table is implemented using a MySQL database. The SDN core network consist of OVSs that performs shortest path forwarding based on IP header prefix. To mimic a local deployment (e.g., an optical ring as a metro access connecting multiple Central Offices), the links between the RAN, SDN edge network, edge cloud, and regions are set to submillisecond one-way latency with 1Gbps link capacity.[2]

We compared SIMECA with an unmodified EPC network instance (i.e., UEs, eNodeBs, SGW, PGW etc.) which was also set up in the PhantomNet testbed. To mimic a realistic LTE/EPC deployment, we configured the links between eNodeBs and SGW to have 1Gbps capacity and 15ms one-way delay, while the links between the SWG and PGW were configured with to 1Gbps capacity and 5ms delay [80]. The total end-to-end latency for legacy EPC core is therefore around 20ms. We use machines with a 3.0 GHz core and 2GB RAM for OVSs, machines that host MF and MC have eight 2.4 GHz cores and 12 GB RAM.

We chose a noncritical health monitoring service with a large number of sensors for our evaluation. We implemented a simple CoAP [139] client using CoAPthon library [144]. As multiple CoAP clients run on a single PC, we assigned multiple IP addresses to the PC and let each CoAP client (which is a process) bind to a unique IP address. The clients then send/receive payload using the IoT CoAP [139] protocol each with a different IP address. We scaled up to 2000 clients (devices) in our emulated experiments as there are limitations in OpenEPC memory management. We did not increase the number of devices further (using simulations) because the trends were already shown in experiments with 2000 devices.

We mimicked four types of health sensors with payload size and sample rate [124] as shown in Table 3.5. For control plane evaluation, the mean device session duration is 15 minutes (i.e., devices reattach after each 15m) and with 50% chance of mobility in an hour [138] (i.e., a device has 50% chance that it is stationary). For Service Request and

[2] This latency depends on network topology and deployment scenarios. The reference deployment in our evaluation is that a region is the combination of a single central office and connected network devices, and network topology is built on optical metro networks with a submillisecond transmission delay [43].

Table 3.5. Workload of health monitoring sensors

Device	Data rate (bps)	Sample size (B)
Body Temperature	2.4	1.5
Blood pressure	1920	2
Cardiac Output	640	2
ECG	98304	2

Paging, we assumed devices release radio resource (i.e., go idle) after 15s of idle time. We ran control plane experiments for 60 minutes.

3.2.5.1 Micro benchmark

• **Control plane time.** We trigger different requests and measure the time for SIMECA to process those requests. We compare this with the processing time of an EPC core network. A request completion time is the elapsed time between a request and a reply *including* network latency (e.g., between base stations and MF). For example, the processing time of a device attach event is the elapsed time when the device sends an "Attach request" until the network (EPC or SIMECA) returns an "Attach complete".

Figure 3.22 shows the processing time of SIMECA's control plane (i.e., ISA) for different requests: device attaches to a server (CS2 ATTACH), P2P attach, device handover while having a C2S connection (C2S HO), and device handover while having a P2P connection (P2P HO). For intraregion events (*SIMECA*), SIMECA processing time is 76% and 64% lower for device attach and handover requests as compared with EPC. For P2P requests, intraregion SIMECA processing time is also small (about 35ms). For interregion requests (e.g., device in a region attaches to another device in another region, denoted as *SIMECA-I*), SIMECA is 55%/34% faster than EPC for attaches/handovers. The performance improvement of SIMECA is because (i) network functions (i.e., MF,MC) are now closer to users so that control plane latency is small (network latency - black box), (ii) less processing time on control plane components (processing time-crossed box). Interregion SIMECA has higher latency because there are interactions between regions and database look-up/update for devices routing identity (grey box) involved.

• **OAI eNodeB processing time.** To understand how SIMECA affects data plane performance in OAI eNodeB, we compare packet processing time in the data plane of OAI

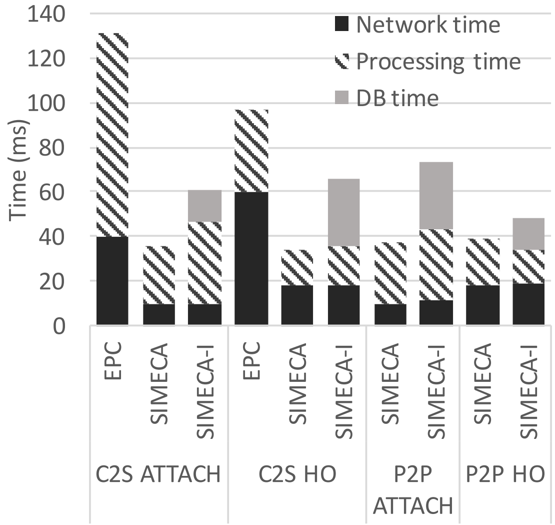


Figure 3.22. Control plane time: SIMECA, LTE/EPC

eNodeB with and without SIMECA. To allow for a fair comparison, in SIMECA eNodeB we measure the time a raw socket task reads a PDCP packet in the ITTI queue and finishes sending it to the socket. Similarly, in unmodified OAI eNodeB, we measure the time a GTPU task reads a PDCP packet and its UDP socket finishes sending the packet.

Figure 3.23 shows data plane processing time in OAI eNodeB with and without SIMECA. SIMECA processing time is similar to normal GTPU processing time which is about 45 *usec* uplink and 15 *usec* downlink.

• **Handover functionality.** To test seamless mobility functionality in SIMECA we conducted an experiment with a server sending a UDP flow from an edge cloud sever to a client. We conducted the experiment using emulated base station as OAI eNodeB does not support handover control signaling. The UDP sender sent packets with a 5ms interval and 1000B packet. In the middle of a flow, the base station performed handover to handoff the client to another station.

Figure 3.24 shows packet sequence number and packet interarrival time at the receiver. At about sequence 800 the handover happens. After the handover, the flow is not disrupted and the handover was seamless.

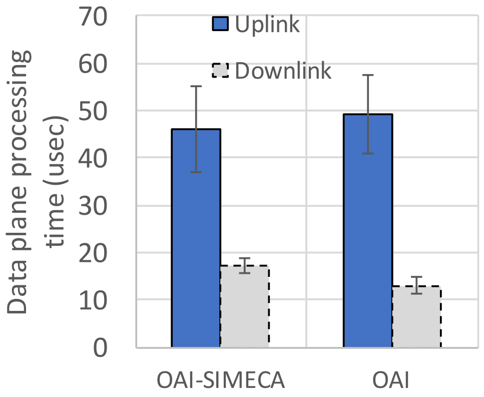


Figure 3.23. Data plane processing time: OAI eNodeB with and without SIMECA

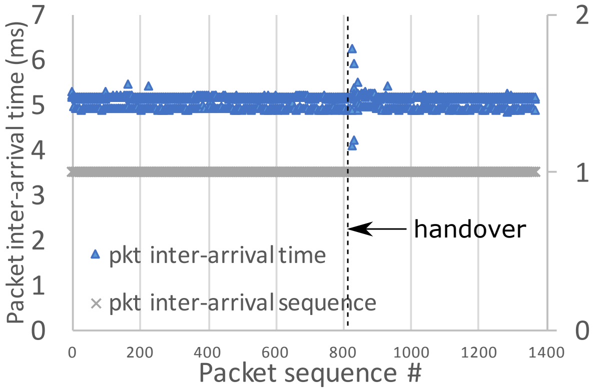


Figure 3.24. Seamless handover in SIMECA

3.2.5.2 System evaluations

• **End-to-end latency improvement.** End-to-end network latency in SIMECA should be smaller than regular EPC because of its distributed deployment close to the edge of the mobile network. Figure 3.25 shows a CDF of the RTT between the Nexus 5 and SIMECA edge cloud, i.e., C2S. and between Nexus 5 and another client, i.e., P2P. Average RTT for C2S is 12ms and for P2P 24ms. We further break down the latency to separate the latency incurred in OAI radio and in SIMECA network. The result shows that OAI radio dominates the end-to-end latency (i.e., 12ms for C2S and 24ms for P2P), while SIMECA incurred latency is less than 1ms. This suggests SIMECA is suitable for low-latency applications in future networks with improved RAN latency.

To show the end-to-end network latency improvement in SIMECA with a large num-

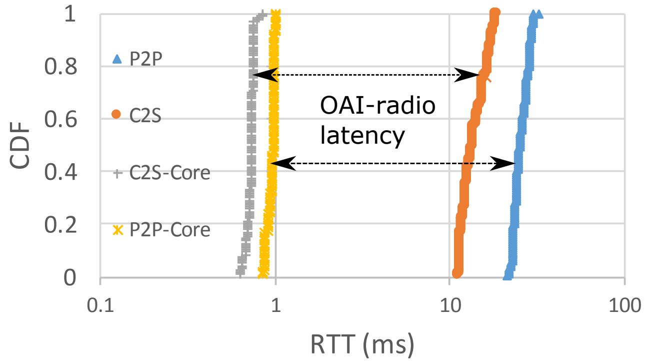


Figure 3.25. End-to-end latency between Nexus 5 and service via OAI eNodeB

ber of devices, we conducted an experiment with a large number of health sensors sending traffic to the edge cloud and to another peer in SIMECA and to a server in the Internet in EPC and measured the round-trip time for packets. As shown in Figure 3.26, in SIMECA client-to-server RTT is about 5ms and client-to-client RTT is about 10ms. As both SIMECA and LTE/EPC have the same radio delay in the experiments, this is almost 10x improvement compared to EPC. Note that this latency gain is mostly because services in SIMECA are deployed in edge cloud which is much closer to users compared to centralized data centers in LTE/EPC deployments.

• **Control plane overhead.** To compare control plane overhead in SIMECA and EPC, we conducted an experiment with multiple sensors performing attach and handover procedures and measured the amount of traffic in the SIMECA edge network (i.e., S1AP,

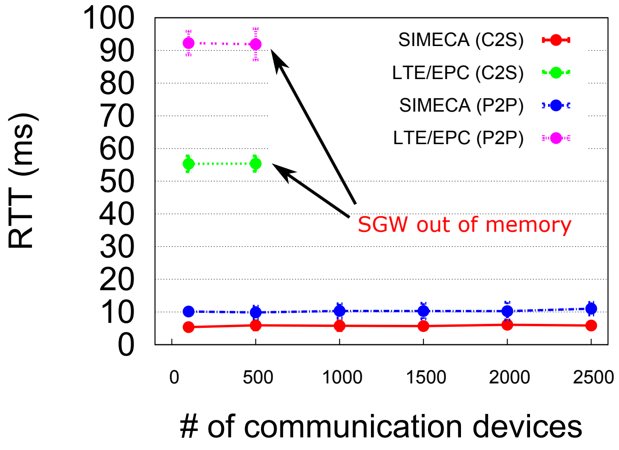


Figure 3.26. End-to-end latency: SIMECA, LTE/EPC

REST, and OpenFlow interfaces) and the EPC core network (i.e., S1AP, S11, S5S8 interfaces). For the intraregion experiment we assumed 100% of events are intraregion *only* while in the interregion experiment 30% of devices requested an interregion attach.

Figure 3.27 shows the amount of control traffic in SIMECA (*SIMECA* means intraregion scenario, *SIMECA-I* means a mixed of intra- and interregion scenario) and EPC as a function of the number of devices. In total, SIMECA has approximately 37% less control traffic for 1000 sensors (i.e., according to 4000 attach events as there are 4 events per hour and one-hour experiment) for intraregion scenario. We further break down the control signaling into 2 parts: *RAN-Core* is the signaling between base stations and the SDN edge network (i.e., or between eNodeBs and the EPC core in LTE/EPC), and *Core* is the signaling within the SDN edge network only (i.e., or within the EPC core in LTE/EPC). As shown in Figure 3.27, SIMECA reduces the control signaling in both the SDN edge network (i.e., between MF, MC, and SDN - red box), and the interface with base stations (i.e., simplified Attach procedure in SIMECA- green crossed box). During the experiment with the regular EPC setup, the SGW ran out of memory when handling around 500 sensors, requiring us to increase the memory footprint of the SGW node 10 times to complete the experiment. With SIMECA the evaluation worked fine on the original node for up to 2000 sensors, illustrating the scalability advantage of our approach.

For Service Request and Paging, Figure 3.28 shows the amount of control plane traffic for 1000 devices each sending and receiving periodic traffic (i.e., every 10s, 15s, 30s, etc). We set the LTE/EPC radio release time to 15s (i.e., if a device goes idle for >15s and

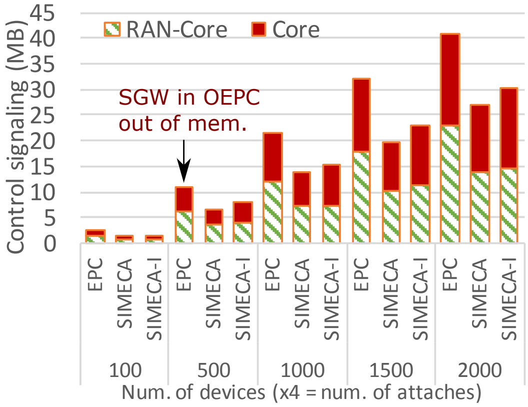


Figure 3.27. SIMECA and EPC control overhead: Attach/HO

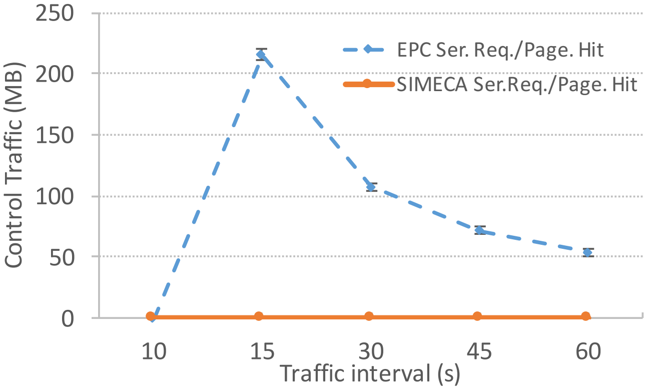


Figure 3.28. SIMECA and EPC control overhead: Service Request/Paging

sends traffic again it will need to re-establish the radio bearers and trigger Service Request/Paging). As we can see, if devices wake up after each 15s and send a small amount of traffic (many meters would fall into this category), the Service Request in LTE/EPC could incur about 230MB of control signaling per hour while SIMECA *does not* incur any control traffic. If devices send traffic more frequently (after each 10s), they won't trigger Service Request. On the other hand, if devices send traffic less frequently (e.g., after each 30s), they will trigger less Service Request per minute and therefore less control plane signaling incurs.

• **Data plane overhead.** To compare the header overhead incurred in SIMECA and EPC, we allowed 500 CoAP sensors to send traffic in both the SIMECA and regular EPC configurations. We vary the payload of the application from small (2B) to large (500B) and measure the packet overhead (i.e., $\frac{payload}{payload+header}$). We also measure the number of forwarding states (i.e., number of SDN rules in SIMECA and GTP tunnels in EPC) as a function of number of devices attached. We assume there is 1 bearer per device in EPC and 60% of the devices are C2S devices, 30% of the devices have interregion communications.

Figure 3.29 shows packet overhead (i.e., payload/(payload+ header)) of CoAP clients with different payload sizes in a POST request in SIMECA and EPC. SIMECA has about 20% less packet header overhead than EPC to deliver a CoAP POST request with a payload that is smaller than 100B. For CoAP POST requests with larger payload, e.g. 500B, SIMECA's data plane incurs about 10% less overhead than EPC.

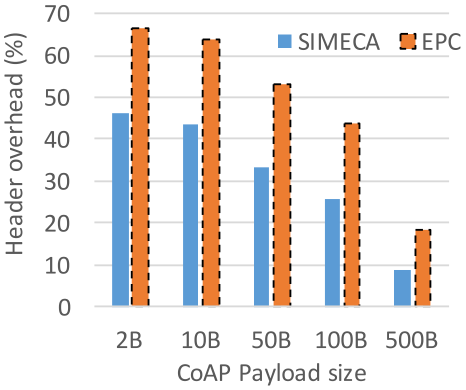


Figure 3.29. Packet header overhead: SIMECA and EPC

3.2.6 Discussion

Since SIMECA is designed specifically for IoT devices, it has some trade-offs compared to the LTE/EPC architecture. We discuss the trade-offs here and their implications.

First, removing the GTP tunnels in the data plane will disable the associated QoS support for IoT devices. IoT devices will not be able to set up “dedicated channels” (dedicated bearers) with guaranteed QoSes in the core network as in LTE/EPC. Also, unlike in the LTE/EPC where each device can have multiple bearers to the same PDN network on top of a single IP connection, each IoT device in SIMECA can only have a single “channel” (e.g., an IP connection) with the same QoS support for all possible applications running on the device. However, removing the fine-grained channels on IoT devices was a deliberate decision to reduce network overhead, and more importantly most IoT devices do not require QoS supports and multiple dedicated channels.

Moreover, if IP forwarding is used in SIMECA’s data plane, QoS support could still be implemented using existing QoS mechanisms for IP networks. For example, the SIMECA’s SDN controller could adopt DiffServ [4] to add an IP Differentiated Services Code Point (DSCP) to the IP packet header at base stations. The SDN-edge network switches then perform Per Hop Behavior to realize end-to-end QoS. Comparing with the mechanism in the LTE/EPC, this QoS mechanism is still cheaper; in SIMECA, packet classification and per-device states are stored *only* at the network edge (i.e., base stations), while in LTE/EPC the classification and per-device states are kept deep inside the core network (e.g., on the SGW, PGW).

Second, SIMECA was not designed to be compatible with other 3GPP architectures. For example, SIMECA is not compatible with 2G/3G and LTE in terms of supporting inter-3GPP mobility. Without 3GPP compatibility, the network coverage for IoT devices is limited. For example, when an IoT device moves out of the coverage of SIMECA's base stations it loses the connection even when other 3GPP access networks are available. To enable 3GPP compatibility, the SIMECA's mobility controller needs to interface with other control components in other 3GPP networks (e.g., SGSN in 3G via an S3 interface) to exchange data plane information upon a handover. On the data plane, the target 3GPP network (e.g., SGSN) needs to forward packets coming from the new radio access network to the service hosted inside SIMECA. One way to enable this is to have an access gateway in SIMECA before the edge cloud. The gateway will serve as an anchor point for IP sessions and the SGSN can forward packets to the gateway using tunneling (similar to an S4 interface in LTE/EPC). Note that the gateway in SIMECA maintains GTP tunnels only with a target 3GPP technology merely for compatibility during handovers. Inside its architecture SIMECA still eliminates tunnelings to reduce network overhead.

3.2.7 Related work

Leveraging SDN programmability to improve flexibility in cellular networks have been proposed in [85, 96, 105]. SoftMow addressed the limitation in the number of Internet egress points and proposed a scalable recursive SDN control plane for cellular networks. SoftMow proposed a design to compute and install routes for devices in the regions to access to the Internet via the desired Internet gateway. SIMECA has a similar vision of edge cloud and edge network with smaller regions to reduce network latency (also aligns with industry proposals for edge cloud [2, 63, 121]). However, SIMECA targets device-to-local cloud and device-to-device communications that are required and specific to IoT devices.

A scalable SDN-based control and data plane to support fine-grained policies for a large number of mobile devices using middle boxes have been proposed [85, 96]. These works addressed the scalability problem of chaining middle boxes in cellular core networks to realize traffic policing at flow granularity. SIMECA uses a similar approach to use SDN to address scalability and overhead problems in conventional cellular networks. SIMECA, however, changes the cellular protocols and abstraction for low overhead, low latency communication for IoTs.

Decoupling of control and data plane and using SDN to set up paths in cellular networks have been proposed in [123, 130]. These works proposed to use an SDN controller as the control entity and use SDN forwarding rules on the data plane for packet forwarding. The works preserved the cellular protocols and data plane. SIMECA has a similar model with SDN controller as a control entity and SDN forwarding rules on the data plane. However, SIMECA proposes to remove cellular protocols and data plane (i.e., GTP tunneling) and replace them with light-weight control/data planes.

Offloading of control and data plane using SDN is proposed in [52, 106]. Procel [106] targeted data and signaling overhead in LTE/EPC by offloading disruption tolerant traffic to a separate IP network, while SMORE [52] proposed to intercept GTP traffic and offload traffic with low latency requirements to cloud platforms the cellular network. 3GPP offloading architectures (LIPA, SIPTO [1]) proposed to place gateways (i.e., SGW, PGW) at eNodeBs to reduce the latency incurred by the EPC core. SIMECA proposes a clean slate design for such an offloading architecture without preserving the current cellular protocols. Unlike the other offloading architectures, SIMECA works independently of the current LTE/EPC core network.

Flat cellular network architecture such as [45] enables base stations to connect directly to the Internet without going through the cellular core. This architecture would potentially improve network latency as it eliminates the hierarchical routing problem in cellular network and sends traffic directly to the Internet without going through the EPC core. SIMECA's architecture is even flatter as its services that are hosted in edge clouds could be accessed directly without going through the public Internet. Moreover, SIMECA supports mobility while other flat cellular architectures that make local breakout must rely on other mechanisms to support IP level mobility (e.g., Mobile IP [125]). Other wireless architectures for IoT devices have been proposed that target large number of IoT devices in wireless networks [131]. SIMECA targets the same problem but in cellular network with scalability and mobility challenges.

Future Internet architecture proposals such as MobilityFirst [136] and routing schemes proposed by industrial research [64] proposed to separate location and end-point identity for better support mobility and multihoming. SIMECA uses a similar separation of end-point identity and forwarding identity to support mobility. However, SIMECA leverages SDN for address translation to reduce data plane overhead and applies it in cellular networks.

3.2.8 Repeatable experiment with SIMECA

To enable repeatable experimentation, the SIMECA prototype is published as a profile in the PhantomNet testbed [12]. Instructions to access the profile is at [15].

3.2.9 Conclusion

In this work, we presented SIMECA, a mobile edge cloud architecture that enables a new network service abstraction aiming to suit IoT devices communication models better compared to the LTE/EPC architecture. The SIMECA architecture leverages NFV and SDN to enable a more distributed, flexible, and fully controlled deployment of IoT services. SIMECA realizes the abstraction by a lightweight control and data planes that significantly reduce signaling and packet header overhead while support seamless mobility. Through evaluations with precommercial EPC software, SIMECA shows promising improvements that favor a large number of future IoT devices in cellular networks.

CHAPTER 4

ENHANCE RELIABILITY IN MOBILE NETWORKS USING DATA ANALYTICS

4.1 Introduction

The proliferation of sophisticated mobile devices like smart phones, tablets and wearable devices [54] have made them an integral part of today's society. The growth in the number of mobile devices, the data usage of each device and the types of mobiles devices of course implies an increased reliance and dependence on mobile networks. To address this demand, mobile operators are continuously investing in new mobile networks and technologies. In recognition of the importance of the underlying network, mobile operators are building redundancy into nearly all components of their infrastructure and developing sophisticated systems to monitor the health of their networks and to rapidly respond to any customer impacting events [87].

Despite these efforts, the inherent complexity of mobile networks and their environments (customer devices, applications) may result in service disruptions that go undetected by monitoring the network elements. Hence, in addition to *network* monitoring, modern mobile operators adopt the strategy of deploying *service* monitoring. Service monitoring is designed to continuously monitor the end-to-end experience that customers receive from their network-based services. This contrasts with network monitoring, in which the status of individual network elements and links are monitored for failures and impairments (e.g., link losses). Service monitoring is vital as a second line of defense – capturing network, customer device or application issues as well as interaction issues among them which may not be detected by the network/applications/devices themselves. Service monitoring is also crucial to quantifying the service impact of known network problems for prioritizing issue resolution.

Somewhat counter-intuitively, network elements that support the service functions are not always able to alarm on conditions which are in fact service impacting. This may be the result of, for example, software bugs in the network elements' firmware, or in the EMS (element management system) for the network elements, or due to configuration errors. It is possible that, even though all network metrics indicate a healthy network, customers might be experiencing degraded service or a complete service disruption. For example, the deployment of a new service feature or a software upgrade to address a bug might trigger an unintended side effect (or indeed a new bug) that the monitoring system is not equipped to detect. We define such service disruptions that are not captured by network monitoring as *silent failures*. Furthermore, it is difficult to infer service quality perceived by customers using the status of the network. There is a complicated relationship between the status of the network and the service quality the users experience. For example, because of redundancy mechanisms within the network, a particular network failure does not necessarily imply customer impact. For example, users associated with a failed cell tower could be picked up by neighboring towers as long as they are in the coverage range of the neighboring towers and the neighbors still have enough resources to handle the users' traffic [155].

However, monitoring service performance across a mobile network is extremely challenging. Traditional active monitoring approaches – techniques which send test traffic across the network – simply don't scale, courtesy of the very large number of cell towers (end points) that need to be monitored and the diverse set of services supported by mobile networks. One could alternatively naively imagine looking for service disruptions by looking for drops in traffic volumes on network elements. However, the inherently dynamic nature of a mobile network environment makes it difficult to infer service impact by monitoring individual network elements, e.g., routers or base stations, to distinguish between changes in traffic volume that are simply the result of the normal operation of the network, and changes that are the result of anomalous network behavior.

In this work we present our work on the ABSENCE system to address the detection of service disruptions in mobile networks in a proactive manner. Our key insight is that service disruptions often impact the traffic consumed by customers, which very likely reflects in customers' usage. This seemingly obvious observation, combined with a suitable mechanism to monitor customer usage, allows ABSENCE to rely on customer usage data to detect the possible presence of service disruptions. Specifically, ABSENCE uses aggregated (e.g., zip code level and handset manufacturer/model level) usage data for different mobile services (e.g., voice call, data, and short message) calculated from anonymized call detail records (CDRs). ABSENCE uses the historical aggregated usage data to predict the expected customer usage for different mobile services at appropriate aggregations of customers, and compares this with real time customer usage data. A deviation from the predicted customer usage is highly likely an indication of a service disruption.

We make the following contributions:

• We present the design of ABSENCE, a novel service disruption detection system for mobile networks that infers service disruptions by monitoring aggregate customer usage. Our design is informed by a data driven exploration of the problem domain using data from an operational mobile network.

• We present a scalable Hadoop-based implementation of our approach which is capable of performing service disruption detection by processing huge volumes of anonymized CDR data (e.g., hundreds of millions of records every hour for mobile data service) in a streaming manner, for all the mobile services associated with an operational mobile network.

• Using data from the same operational mobile network, we perform a systematic data-driven evaluation of our approach by introducing a comprehensive synthetic set of both network and mobile device failure scenarios. Our results show that: (i) Our variable-scale temporal aggregation improves detection by an order of magnitude over fixed interval aggregation. (ii) We achieve overall detection rates of 88%, while we achieve 98% or better detection rates for service disruption that have over 10% usage impact within the corresponding aggregation (e.g., zip code and handset device model).

• We compare our results with ground truth from actual service disruption events and present a number of case studies showing the effectiveness of our approach. For a set of confirmed service disruptions, ABSENCE achieves 100% detection rate.

4.2 Motivation

In this work we define a *service disruption* to be a scenario in which customers become unable to utilize the offered service(s) that they would normally utilize. A service disruption can be either due to the network/application failing to complete customers' service requests, or because customers give up making service attempts due to unacceptable service performance. Service disruptions are typically the result of a network, device or application outage or severe performance degradation. The vast majority of service issues are rapidly detected via the network and/or application. However, there are a small number of issues – typically those resulting from complex software bugs – that may remain undetected by the network and/or application.

Given the challenges of scaling active service monitoring techniques, one could imagine instead simply relying on customers to inform a mobile service provider of service disruptions as is performed in other industries, such as the power industry. However, given that individual customer concerns may relate to a large number of underlying causes – individual customer device issues, customer user error or broader service disruptions – identifying a significant service disruption would typically require detecting a pattern in the customer feedback across a number of different customers. This is inherently slow, and thus a highly undesirable approach to detecting service issues. Figure 4.1 shows an example of customer ticket volume resulting from a service disruption. The figure shows a time series of the number of tickets in the customer care system. The actual event occurred around 16:38 UTC, but was only evident via an increase in customer ticket volumes at 21:00 UTC (i.e., 4.5 hours later), when the number of customer calls rapidly increased. Relying on customer complaints to detect such failures is thus clearly undesirable – customers are simply too slow at calling in for this to be a timely approach for detecting service issues.

Given the relative maturity of network management and operation practices [87], and significant research efforts associated with failure detection [26, 72], network and application interaction [57, 78, 107, 127], mobile network performance [44, 68, 112, 113, 153] and service monitoring [29, 141, 154], the obvious question is: *Why are these service disruptions so difficult to detect?*

Active end to end service monitoring is used extensively across mobile and wireline networks to test service integrity and performance. Active monitoring uses probes placed

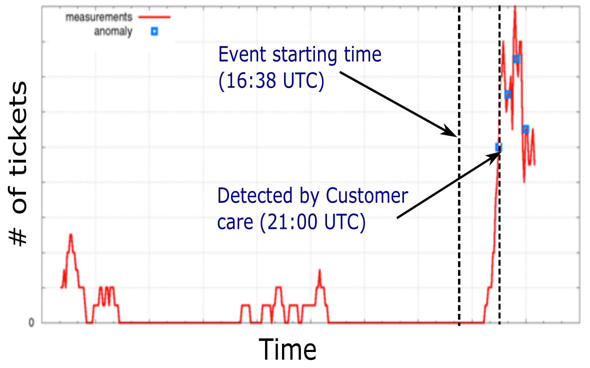


Figure 4.1. Customer care tickets indicating a service disruption

strategically across the network to send test traffic. However, the major challenge with active monitoring is scale – ideally probes must be deployed so that every combination of service path, customer device type and application is actively tested on an ongoing basis. But this is clearly an unrealistic expectation. There are simply too many different types of customer devices in the market place, a multitude of different applications and a huge geographic environment to probe. In a wireless environment, service performance can vary considerably across a very local region even for customers connected to a common cell site. Thus, even placing a dedicated probe associated with each individual cell site does not provide a comprehensive view of service experience across the entire region associated with that cell site. Thus, active monitoring in a mobile network provides only a sampling of service experience and, depending on the extent of the deployments, will likely not be able to detect all service impairments.

One may alternatively use passive monitoring of customer traffic to detect service impairments. Such monitoring can be performed on traffic aggregates, and thus does not need visibility into individual customer experience. However, service outage detection cannot be achieved by observing customer traffic – by its very nature customer traffic is expected to disappear during an outage. Thus, detecting a service outage using passive monitoring entails looking for an *absence* of expected traffic. Thus, one could natively assume that we could look for drops in load on network elements to identify service outages. However, where in the network to look for such reductions in carried load is an intriguing challenge – traffic is regularly shifting around a mobile network, typically without any service impact. For example, activities such as load balancing or planned maintenance events could cause load changes on network devices, yet has no impact on customers' service experience. Figure 4.2 shows a load change on a Serving Gateway (SGW) node in an SGW pool during a load balancing event. Despite the load change on the individual SGW, there was no impact on users. Simply looking at the load of this SGW alone is insufficient to determine customer impact.

While detailed network performance metrics have been defined for mobile networks by 3GPP [21], and are being implemented by equipment vendors, these key performance indicators (KPIs) are not always sufficient to detect user impact either. For example, 3GPP defines *accessibility* as a KPI to measure the probability that a user will be provided with radio access network (RAN) resources (technically with a radio access bearer) on request. This metric can clearly provide insight concerning resource shortages in the radio access network. However, users might still be impacted, even when the RAN accessibility KPI is good, because users that lost radio coverage are not even accounted for in the KPI calculation or when the root cause of the service problems are beyond the RAN (e.g., congestion on a core network element). In other words, service disruption occurs when accessibility is bad, but accessibility being good does not necessarily imply that service is good.

Finally, in order to deliver the end-to-end service successfully, handset device, mobility network and application need to work together seamlessly. Thus the root cause of a service disruption could be well beyond the mobility network. For example, a firmware upgrade on a certain customer device model could result in incompatibility between equipment in

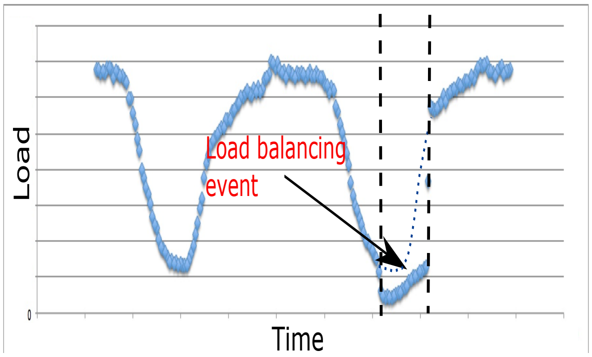


Figure 4.2. Load reduction caused by load balancing

the RAN (e.g., a radio network controller (RNC)), and the mobile devices that performed the firmware upgrade, thus resulting in a large number of devices not being able to access the network. Similarly, an update in the delivery protocol of a video streaming application could cause interoperability issues between application and network, which further leads to service disruptions. These types of service disruption are challenging for mobility network operators since there is little evidence of it on the network side.

In this work, we argue that users' usage, or lack thereof, is a reliable indicator of service outages and severe performance degradations in a mobile network. By monitoring and analyzing users' usage, we are able to detect service disruptions that could be challenging for other event detection mechanisms.

4.3 Approach

With ABSENCE we propose to use historical customer usage data to predict expected usage that should be generated from customers under normal conditions. Any deviation from expected customer usage indicates an anomaly, which might be indicative of a service disruption. While this basic approach conceptually seems to make sense, it is not obvious that the method would be feasible in practice. For example: would customer usage predictions based on usage data be sufficiently accurate to allow anomaly detection? Given the number of mobile devices, the variety of services offered on mobile networks and the complexity of the mobile network infrastructure, is there a level of customer usage aggregation that would provide fidelity of detection fine-grained enough to detect silent failures at a granularity that is practically useful? Given that users in a mobile network are by design using the network while moving around, how should we deal with mobility?

In this section, we present our exploration of these questions using customer usage data from a large mobile network provider. Before describing our exploration we briefly describe the nature of Call Detail Records (CDRs) which, when aggregated, constitute the customer usage data used in ABSENCE.

4.3.1 Customer usage data

ABSENCE aggregates metrics from Call Detail Records (CDRs) to measure customer usage at different locations and application levels. These aggregates are calculated within

ABSENCE using individual, anonymized CDRs. Note that these individual CDRs are only used internally to the system during the aggregation process, and no customer specific data (anonymized or otherwise) is ever exposed to a user of the ABSENCE system. Call Detail Records contain metadata about executed transactions across the mobile network (i.e., phone call, data session, access to voice mail etc.). Each record captures information that is needed for charging and debugging such as a time-stamp of the activity, device specific information (e.g., the international mobile station equipment identity (IMEI)), network related information concerning the activity (e.g., the sector(s) of a cell tower that the device is connected to), the duration of the activity (for voice services) or the volume of data the device downloads/uploads (for data services). Of critical importance for our approach, CDRs are generated in near real time: for Voice service a CDR record is generated right after a call finishes, for Data service a CDR record is generated whenever a PDP context is created and a new CDR record is created every hour if the data connection spans multiple hours. This allows CDRs to be used as a timely indicator of customer activity (or inactivity). Note that CDRs do not contain actual customers' short message, voice call or data content and ABSENCE only uses anonymized CDRs.

4.3.2 Usage prediction and aggregation size

The primary challenge in calculating aggregate customer usage information is to determine what level of aggregation is most effective to address the problem at hand. In considering the question of accuracy of predicting traffic volumes versus fidelity of anomaly detection, there exists an intuitive tradeoff: At one extreme one might attempt to use the usage data of each individual. While the network usage pattern of an individual user might show fairly predictable patterns, at this granularity a deviation from an expected pattern is clearly not a reliable indication of a service impairment. The user in question might simply have a change in their normal behavior, e.g., going on vacation. At the other extreme, the usage aggregated across all customers in the U.S. is highly predictable. However, at this aggregation level, a service disruption that only impacts a relatively small number of users, e.g., those associated with a particular cell-tower, would not be visible at such a coarse grained data aggregation level. The challenge therefore is to find an aggregation level of usage data that is small enough that it can provide high fidelity of detection, but at the same time large enough to render stable usage patterns to allow for accurate prediction.

For example, Figure 4.3 and 4.4 show the amount of voice calls, respectively, made by a group of 70 and 3020 randomly chosen devices over the course of three weeks. The amount of usage on the smaller group is significantly less that in the larger group and the larger group also shows more stable day-to-day usage pattern between the different weeks.

To understand how the amount of usage affects the tradeoff between the stability of the usage patterns versus the fidelity of detection regardless of device specific, we conducted experiments to predict future usage based on historical usage with different amounts of aggregated data.

• **Experiment description.** We selected a uniform random sampling of users to form different groups (with size ranging from 20 to 150,000 users) for Voice and LTE data service. We assumed that the aggregated service usage follows a weekly seasonal model and we used 16 weeks of data for our training. We constructed a weekly seasonal usage pattern

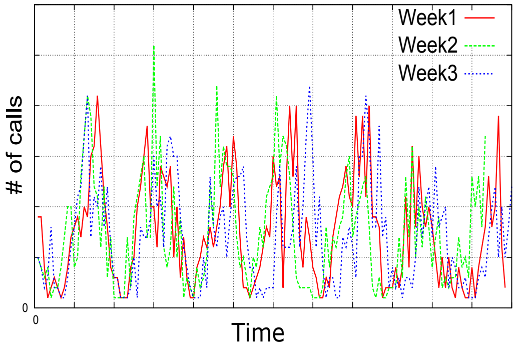


Figure 4.3. Usage of group of 70 devices

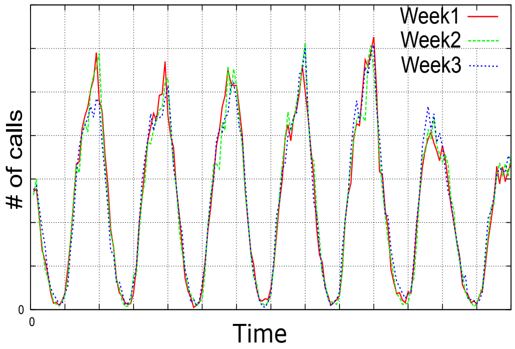


Figure 4.4. Usage of group of 3020 devices

using the additive decomposition technique described in Section 4.4. The seasonal pattern is the predicted usage for the future usage and the "noise" is the absolute distance between the usage and its seasonal data point. We then used the above seasonal usage pattern to predict another week of usage.

- **Metric.** To quantify the prediction accuracy, we used the normalized noise ratio as the metric. We formally define the noise ratio in Section 4.4. In short, the noise ratio is the "noise" between the testing data and the training data normalized by the training data. Intuitively, if the usage of an hour deviates too much from the seasonal pattern, the noise of that hour is high and therefore results in a higher normalized noise ratio. We sampled the noise ratio at two regions of a time series (i.e., during the peak usage period 17:00-23:00 UTC and during the low usage period 03:00-11:00 UTC), and plot the noise ratio as a function of usage.

- **Results.** Figure 4.5 shows the noise ratio as a function of the usage for LTE and voice. Overall, for a sufficient aggregation (i.e., above 1,000 of usage), the usage is quite predictable (i.e., the noise is about 10%). Moreover, the noise ratio is high for a small usage and reduces when the amount of usage increases. This matches the intuition that the usage of an individual user is less predictable than the aggregated usage of a group of users.

The figure also suggests that after a certain amount of usage data, the predictability of an aggregation does not increase significantly. This suggests that the size of an aggregation should not be too large for both good predictability and high sensitivity, e.g., if we monitor usage of an entire city as an aggregation, a failure that impacts only a single ZIP code area

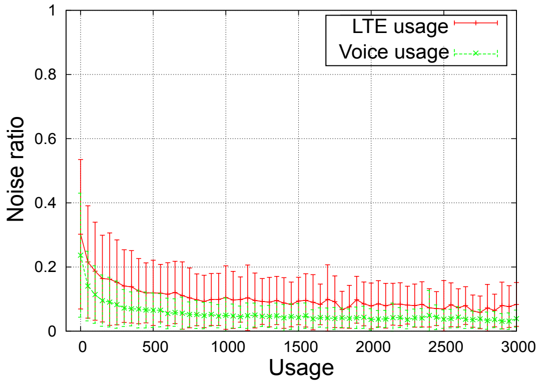


Figure 4.5. Noise ratio of LTE and voice usage

might not cause a significant enough drop on the total usage for a system to detect.

4.3.3 Practical user aggregation

In this section we consider the question of how groups of users can be selected in practice. While network and service failures can be highly diverse in their impact scope, the mobile network design and operational practice would inherently cause the service failures to be localized to geographically close-by regions and/or user devices with some common hardware or software. For example, rolling out a software upgrade on Radio Network Controllers (RNCs) would typically take place in a few geographical regions, and the upgrade may introduce an unexpected compatibility problem with certain phone models that was not captured by the RNC equipment vendor during lab testing. As another example, a software bug in a packet data network gateway (PGW) may cause the service that the PGW supports (e.g. visual voice mail) to become unusable, and all service requests originated from certain geographical regions that are routed toward this PGW are affected. Hence, grouping users geographically and by device hardware and software and tracking usage by different service features would have the best chance of capturing service failures.

• **Geographical hierarchy.** In this grouping method, we utilize the geographical hierarchy in the ZIP-code system to group users. We use the ZIP-code hierarchy for three reasons: (i) the ZIP-code system was designed for efficient postal delivery and therefore each ZIP-code naturally covers a sufficient and relatively equal amount of users; (ii) the ZIP-code hierarchy is geographically driven and the structure of the ZIP-code has geographical meanings; (iii) by utilizing the ZIP-code hierarchy, the system can quickly scale up and down the size of aggregation based on the structure of the ZIP-code, i.e., groups of states, states, large cities etc. Moreover, a ZIP code area is relatively large enough for sufficient usage prediction and small enough to obtain good sensitivity A ZIP code area also often belongs to either an urban or a rural area and therefore users in a same ZIP code often have the same usage pattern. The detailed ZIP-code hierarchy is presented in Figure 4.6.

• **Device type hierarchy.** Under each geo-group (i.e., a node in Figure 4.6), we divide devices into smaller groups based on operating systems (i.e., Android, IOS, Windows,

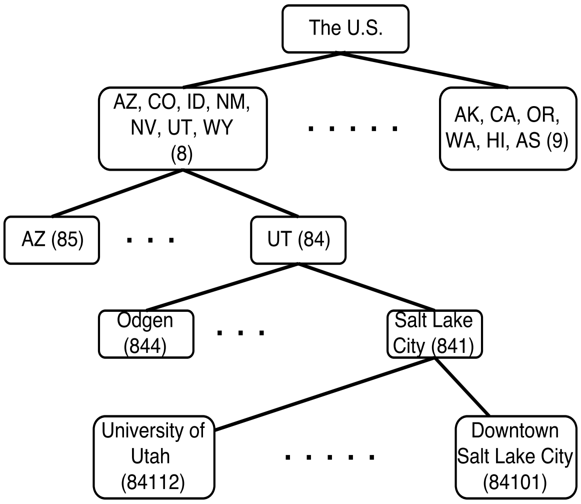

The U.S.
AZ, CO, ID, NM, NV, UT, WY (8)
AK, CA, OR, WA, HI, AS (9)
AZ (85)
UT (84)
Odgen (844)
Salt Lake City (841)
University of Utah (84112)
Downtown Salt Lake City (84101)

Figure 4.6. Geographical hierarchy

BlackBerry OS), device make (i.e., Samsung, Apple, Nokia, etc), and device type (i.e, Samsung Galaxy S5, iPhone 4, Nokia Lumia 512, etc). This way the system can monitor not only geographical aggregations (e.g., Salt Lake City) but also specific device types in the area (e.g., all Samsung Galaxy S4 devices in Salt Lake City). As shown in Figure 4.7, this device hierarchy can be applied at different levels in the geographical hierarchy.

4.3.4 Temporal usage aggregation

Recall that in Section 4.3.2 we found that in order to get good predictability the aggregations being monitored should have a large enough usage (Figure 4.5). Daily network usage follows a well known diurnal pattern with well established busy and quiet times. During the network quiet time (i.e., after midnight), hourly usage is typically small and might not be sufficient for good predictability. Figure 4.8 shows the CDF of hourly voice usage during low usage period (i.e., 03:00 UTC to 11:00 UTC) and peak usage period (i.e., 17:00 UTC to 23:00 UTC) of all ZIP codes. Almost 95% of the hourly usage measurements during low usage period are smaller than 500 which reduces the accuracy of prediction over hourly aggregations, i.e., at the ZIP code level, simply grouping usage into hourly bins results in insufficient usage to obtain a good prediction. In contrast, 48% of the hourly usage measurements during peak period are smaller than 500. Moreover, usage can also be low if, when using the geographical or device aggregation, the chosen aggregation level

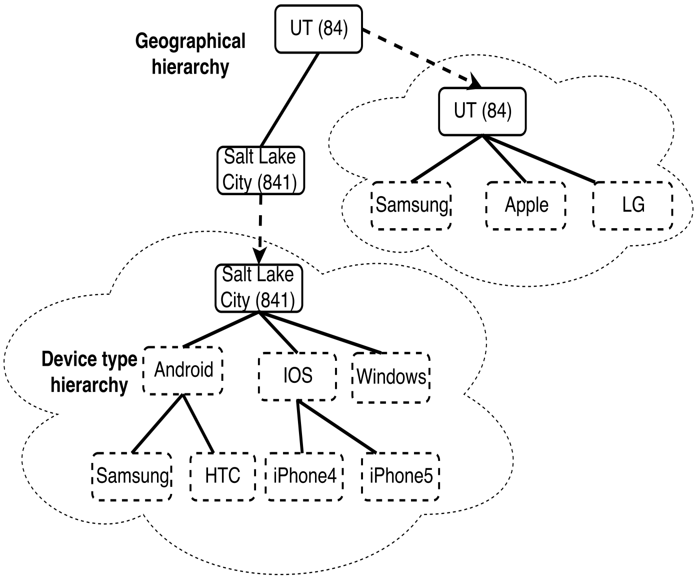


Figure 4.7. Device hierarchy

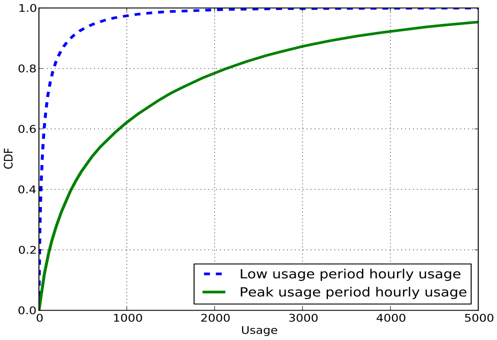


Figure 4.8. CDF of hourly usage during low usage hours and peak usage hours across ZIP code aggregations

only has a small number of users (e.g., a ZIP code in a rural area or a ZIP code with a small number of subscribers).

This suggests the utility of grouping multiple hours during low usage period or of small spatial aggregations into a single temporal aggregation. Note that using longer time periods over which to do the usage aggregation would present higher accuracy at the cost of increasing the potential detection time, i.e., when the usage is low this technique increases the likelihood of detecting a failure after several hours while the conventional technique cannot detect it. Based on this observation we employ a *variable scale aggregation* strategy in Section 4.4 to improve the accuracy of prediction in ABSENCE.

4.3.5 Usage aggregation

ABSENCE uses aggregate service usages to detect service disruptions. In order to generate these aggregates, ABSENCE groups individual anonymized CDRs from a set of similar users. In ABSENCE, users are considered similar to each other if their mobile devices are from the same manufacturer/model or from the same ZIP code. Aggregation usage from users with the same mobile device manufacturer/model is straight-forward. In the section, we only focus on how to group users based on the ZIP code that they are in.

The inherent challenge stems from the mobility nature of mobile network users. Thus the set of users in a particular ZIP code area is changing over time. For example, the number of mobile users in a ZIP code covering a business or educational campus might vary by thousands, or even tens of thousands, over the course of a day as workers/students arrive for the work day and leave again at the end of the day. In this section we explore mechanisms to deal with this inherent variability in ABSENCE.

Our approach hinges on the observation that for our purposes in ABSENCE, the exact location and mobility patterns of specific users at a particular point in time are not relevant. Rather, we are interested in knowing these properties with sufficient accuracy for a statistically meaningful aggregate of users. Moreover, approximated users' location could be used to localize geographical location of a failure to benefit root cause analysis. Thus, to calculate the ZIP code level usage aggregates we simplify the internal operation of our ABSENCE system by simply aggregating over those customers that are typically in the given ZIP code for that time of day. Thus, within ABSENCE, for each (anonymous) user, we derive a *user ZIP code profile* which approximates the user's mobility pattern at ZIP code level over time. This user ZIP code profile is derived within ABSENCE using the anonymized CDRs. After calculating an individual user profile, the usage of a user is counted toward *his/her profiled ZIP code* regardless of the current ZIP code of the user. For example, during summer some students leave their campus for internship. Those students' usage will be counted toward their approximated ZIP code, i.e., their campus.

• **ZIP code level profiling.** We evaluated a number of strategies to derive the user profiles. We assumed users' mobility pattern follows a daily pattern during weekdays and weekends (i.e., weekends are treated differently). To obtain a user's location, we observe in the CDR where the user uses the network. For example, if the user makes a call in a ZIP code, the ZIP code is recorded in the CDR. We explored 2 parameters used to approximate users' ZIP code profile: how many ZIP codes in a day a user has and the length of training data used for the approximation.

• **Accuracy of location approximation and complexity trade-off.** Intuitively, the finer-grain the user location profiling the more accurate the location estimation is. However, the finer-grain the location profiling results in multiple ZIP codes for a user and more resources required to store the historical data and extract usage in real time. For example, if the location estimation is done on an hourly basis, then a user is in 48 ZIP codes during weekdays and weekends (i.e., 24 hours for weekdays and 24 hours for weekends). With 48 locations, each user will have 48 different historical usages. Given hundreds of millions of users, maintaining 48 historical profiles per user is expensive. Moreover, the user's accurate location information is not available even in an operational network; per-hour location report is not available due to privacy and overheads. Therefore, we are interested in knowing the trade-off between accuracy of location estimation and the computational complexity.

Somewhat surprisingly, our findings show that per-hour location estimation in fact does not predict a user's location significantly better than other coarse-grain estimations. In contrast, predicting users' location based on their *home and work hours* yields the most accurate estimation. The home/work estimation is also cheapest in terms of computational complexity and is what we use in ABSENCE. An explanation is that users tend to not have a predictable mobility pattern on a per-hour basis, e.g., 1-2PM on Monday and 1-2PM on Tuesday a user tends to be at different locations. On the other hand, users tend to be in their home/work locations during their home/work hours. Moreover, note that because we use CDR to derive users' location, the estimation is less accurate if users are mostly silent.

With the home/work profile approach we make the simplifying assumption that user mobility can be approximated as follows: Depending on the time of day and day of week, a user is typically either at home or at work. Specifically, during week days and working hours, i.e., 9 a.m. to 7 p.m., the user is assumed to be at work, while during weekends and the remaining week day hours, the user is assumed to be at home. Given this assumption, what is required to derive a home/work profile is to determine the user's home and work base station. We make use of historical CDR data over a relatively long period of time, i.e., a couple of months, and simply use the most frequent base stations for the appropriate time (i.e., work or home) to determine the respective base stations for home and work hours. For example, if a user uses a base station most frequently during 9 a.m. to 7 p.m in a month period, his/her work hour ZIP code profile will be that base station's ZIP code. The home/work profile is clearly quite scalable requiring only two ZIP codes (and associated historical information) to be maintained for each user.

• **Experiment.** We evaluated the accuracy of the home/work ZIP code level profile. We first derived the users' ZIP code profiles as described above using historical CDRs. We varied the amount of the training data used to evaluate its impact on our approximation by 1, 2, 4, and 8 weeks. We then used the derived profiles to predict users' ZIP code using the same amount of future CDRs. We measured the percentage of users' ZIP codes that were correctly predicted (i.e., the hit rate - whether he/she makes calls in the estimated home/work ZIP code) using the user profiles.

• **Results.** Table 4.1 shows the results of this evaluation. As shown the duration of the historical (training) data does not significantly improve the accuracy of the approximation. Given the computational cost to consider this suggests that the home/work ZIP code profile, with monthly updating, will be the most suitable for ABSENCE. Note that this location estimation exercise is more useful for *root cause analysis*, e.g., to get the exact location when an anomaly is detected. The estimation, however, would not affect the anomaly detection because ABSENCE only uses the statistical feature (i.e., usage) of an aggregation *regardless* of where the aggregation is. For example, ABSENCE is always able to detect when an anomaly happens to an aggregation regardless if the aggregation is labeled as "downtown Salt Lake City" or "downtown Los Angeles".

Table 4.1. Hit rate of ZIP code approximations

	1-week training	**2-week training**	**4-week training**	**8-week training**
Home/work	57.03%	57.92%	*58.91%*	56.05%

4.4 System overview

In this section we describe ABSENCE and how the data are processed. The logic blocks of the system are shown in Figure 4.9. From left to right, "raw usage data" (anonymized call detail records) go through a pipeline, which consists of multiple stages: spatial aggregation, variable-scale temporal aggregation, event detection and false-alarm removal before the detected events are shown to the operators. While we show this step for completeness in Figure 4.9, our current focus is on developing an effective usage-based detection system and we therefore do not consider this aspect.

• **Spatial aggregation.** ABSENCE groups users based on their profiled zip-codes: users that are geographically close are grouped together using their zip-code profile. Users in the same area are often being served by the same set of network elements and therefore likely to be impacted as a group. Moreover, events that impact a group of geolocated users are more likely to be actionable to network operators.

• **Temporal profile generation.** After grouping the users, ABSENCE needs to extract the usage data of the group over time in order to detect abnormal usage for the group. Simply grouping usage into hourly bins is not optimal for usage predictability and anomaly detection. Instead, if the hourly usage is smaller than a predefined threshold (e.g., after

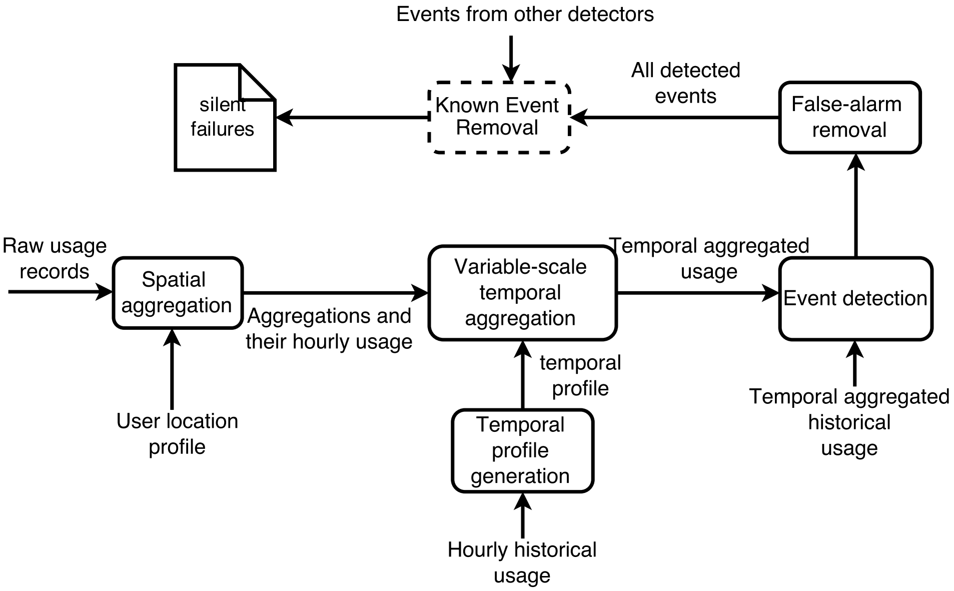

Events from other detectors
silent failures
Known Event Removal
All detected events
False-alarm removal
Raw usage records
Spatial aggregation
Aggregations and their hourly usage
Variable-scale temporal aggregation
Temporal aggregated usage
Event detection
User location profile
temporal profile
Temporal profile generation
Hourly historical usage
Temporal aggregated historical usage

Figure 4.9. Data processing pipeline of ABSENCE midnight or the spatial aggregation is small), ABSENCE groups multiple hours of usage into a single bin in order to satisfy the usage threshold.

To realize this, ABSENCE needs a *temporal profile* for each spatial aggregation to do the grouping. ABSENCE assumes a weekly seasonality for the aggregations, i.e., the usage of an aggregation repeats every week. To generate the temporal profile, ABSENCE first uses hourly historical data to find a regressed weekly time series such that every data point in the regressed weekly time series is the *median* of the historical data points. Note that the historical time series and the regressed weekly time series consist of hourly usages. Having calculated the regressed weekly time series, ABSENCE then runs a greedy algorithm (Algorithm 1) that groups consecutive hours together until the total usage is larger than a predefined threshold (i.e., K in algorithm 1) and repeats this until all the hours in the weekly time series are grouped. If the last temporal bin of a week appears to be too small then it will be combined with the first hours of the following week until the threshold K is satisfied. The output of the algorithm is a *temporal profile* consisting of multiple bins each having at least an amount of K of usage. Note that ABSENCE needs to run this training process only once every several months given that the *temporal profile* of the aggregation is usually stable.

- **Variable-scale temporal aggregation.** After obtaining the *temporal profile* for each spatial aggregation, ABSENCE uses the profile and the hourly time series from the spatial aggregation to create a variable-scale time series. The output of the *variable-scale temporal aggregation* is a time series which has multiple temporal granularities and each data point satisfies a predefined usage threshold. This time series is used for *event detection*. We compare this variable-scale approach with the plain hourly temporal aggregation. Note that ABSENCE currently aggregates usage data on an hourly basis at the finest granularity.[1]

- **Event detection.** After generating the *variable-scale* usage time series of a group, ABSENCE appends the usage with the corresponding *variable-scale* historical usage and feeds the entire time series into a time series decomposition and event detection module that analyzes the time series and outputs abnormal events. Due to the large number of time series that need to be processed, ABSENCE adopts the additive time series decomposition

[1] While an hour might seem long from the perspective of detecting a network outage, it does represent a reasonable tradeoff in the context of network operations scale.

Algorithm 1 Generate temporal profile

Input: Weekly regressed usage time series $T = (t_0, t_1, ..., t_{167})$ for 168 hours in a week, threshold K.
Output: Temporal profile j and $P = (p_0, p_1, ..., p_j)$ as the starting hour of j continuous segments of each profile bin

```
Initialize: $i \leftarrow 0$, $accumulate\_usage \leftarrow 0$, $j \leftarrow 0$, $tag \leftarrow 0$
while $i < 167$ do
    $accumulate\_usage \leftarrow accumulate\_usage + t_i$
    if $\sum(accumulate\_usage) \geq K$ then
        $p_j \leftarrow tag$, $accumulate\_usage = 0$, $j \leftarrow j + 1$, $tag \leftarrow i$
    end if
    $i \leftarrow i + 1$
end while
if $j == 0$ then
    return 0, ()
else
    if $accumlate\_usage == 0$ then
        return j, P
    else
        {remainder wrapping around to the beginning of the week}
        $p_0 \leftarrow tag$
        return j, P
    end if
end if
```

approach, which is a light-weight time series analysis algorithm and has been found very effective in modeling economic data and recently in network traffic as well [129]. At a high level, the time series decomposition technique deconstructs a given time series into the secular trend component ($\{T_t\}$), the seasonal component ($\{S_t\}$), and the noise component ($\{N_t\}$) [48]. In the additive model, the original time series ($\{V_t\}$) is the summation of these three components.

Figures 4.10 (b), (c), (d) show the corresponding components for the time series in Figure 4.10 (a). The general idea of time series decomposition is very simple – with a specified seasonality window W, secular trend can be obtained through smoothing over long term (multiples of W), i.e., by *centered moving average*:

$$T_t = \sum_{i=-W}^{W-1} V_{t+i}/2W$$

Note that to be able to decompose the *variable-scale* time series, the seasonality window W here is set to the number of temporal aggregations of the *temporal profile* generated, i.e., j in Algorithm 1, and W varies for different spatial aggregations.

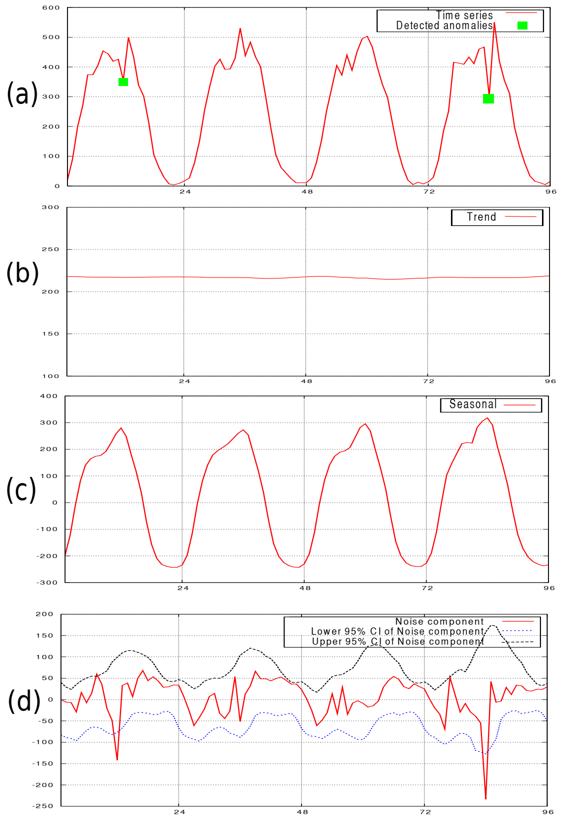


Figure 4.10. Trend, seasonal, and noise components

The seasonal trend can be obtained by averaging the phase value (after removing secular trend) across seasons, i.e., by *seasonal moving average*:

$$S_t = \sum_{i=0}^{K} (V_{t-iW} - T_{t-iW})/K$$

where K is the number of seasonal windows contained in the historical data. And the remainder becomes the noise component:

$$N_t = V_t - T_t - S_t$$

Note that time series decomposition can be applied to analyzing both long range historical data and in a moving window fashion for the recent data (as new data is appended to the time series).

In our approach, we further model the noise components, N_t, at different phases as zero-mean Gaussian variables with different variance, $\sigma^2_{t|W}$, where the phase $t|W$ represents $t \bmod W$. We tag the corresponding time series value, V_t, as anomalous (critical value 1.96 at 95% confidence interval) if

$$|N_t / \sigma_{t|W}| > 1.96 \tag{4.1}$$

This is consistent with classic anomaly detection techniques. We also apply an iterative process such that we remove the anomalous points in the previous iteration from the trends and noise variance computation, which makes our approach robust to bad data/known anomalies.

An example of how the event detection works is shown in Figure 4.10. The two dips (green dots) in Figure 4.10 (a) correspond to the two dips in the noise component in Figure 4.10 (d) (red solid line) and those two dips are smaller than the lower 95% confidence interval of the noise component at the points (blue dashed line). This results in two detected anomalies in the time series.

4.5 Implementation

Processing the anonymized CDRs is computational intensive (e.g., hundreds of millions of records every hour for data service and tens of millions of records every hour for voice service) and normal serial processing methodologies will not be scalable. Since CDRs can be processed independently, we use a Hadoop Map-Reduce cluster to process the data in parallel to speed up the process.

• **Running environment.** As shown in Figure 4.11, ABSENCE consists of four components: historical usage retrieval, hourly usage retrieval, time series processing and user location profile retrieval. ABSENCE runs on two environments: usage retrieval is done on a Hadoop cluster and time series processing is done “locally”. The Hadoop cluster hosting ABSENCE consists of 100 nodes each with 32 cores CPU and 128GB RAM and runs on an HDFS file system. The local environment is a single node in the Hadoop cluster.

• **Historical/hourly usage retrieval component.** We use Apache Pig (pig.apache.org), a platform that offers a high-level language for expressing map-reduce programs, for the usage retrieval components. The hourly usage retrieval component wakes up every hour to extract the usage of the last hour while the historical usage retrieval component is triggered every month to extract the usage of the last months that is used as a historical

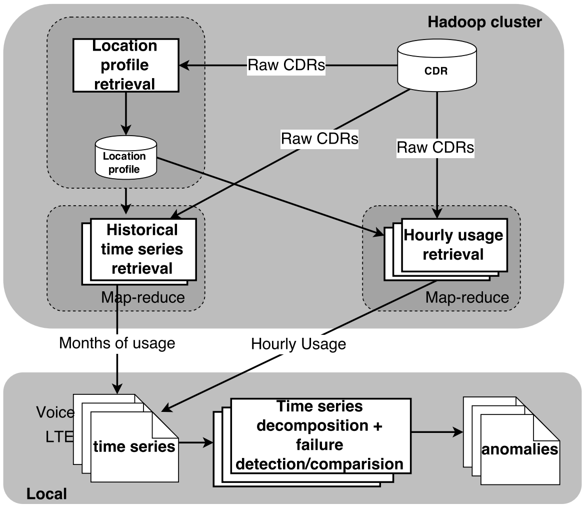


Figure 4.11. ABSENCE components

usage baseline of the coming month. The output of the historical/hourly usage retrieval component is transferred to a local machine for time series processing.

• **User location profile retrieval.** We build the user location profile retrieval using native Hadoop Map-reduce. The location profile retrieval component is triggered every month to construct the latest location profile that is used for the following month. The user location profile is stored on the Hadoop HDFS file system as the usage aggregation components need the information (Figure 4.11).

• **Time series processing component.** This component is located locally on a single node in the Hadoop cluster. The component consists of light-weighted modules such as the time decomposition module, the anomaly detection module, false-alarm removal module etc.

4.6 Evaluation setup

Obtaining ground truth about service disruptions is inherently difficult. To allow a systematic evaluation of our approach we introduced synthetic service disruptions into data obtained from a U.S. mobile provider.

• **Data overview.** We used CDR data collected in a large US mobile network from July 2014 to December 2014 in our evaluation. We used all 6 months worth of data to build up the historical data used in ABSENCE. With 6 months of historical data and the weekly seasonal model, we maintain $6 * 4 = 24$ usage histories for each of the nodes in our geographical hierarchy to use as reference points for anomaly detection in current usage data. This amount of historical usage data is sufficient to maintain acceptable confidence intervals for the time decomposition algorithm. The total amount of data used in our evaluation is 45 TB. The volume of collected CDR data varies from 15-20 GB per hour, depending on user activity, with the monthly volume of 7-8 TB. For the synthetic evaluations presented below, we excluded a week's worth of data from the 6-month data set and used those days as "current usage data" in which the synthetic failures were introduced.

• **Synthetic service disruptions.** To allow for a systematic evaluation of ABSENCE we emulate service disruptions due to both the network and the device failures. ABSENCE performs service disruption detection by identifying changes in the expected usage data. As such, to emulate both network and device failures, our approach is to *remove* the corresponding data (i.e., data that would disappear if the failure had occurred) from the CDR data for each synthetic scenario.

We mimic *network* failures at the granularity of a base station, i.e., when a base station is down, the service at that base station is lost. In the CDR data, every call record is associated with a list of base stations that served the call, i.e., the "serving list". When we emulate the failure of a base station, we use the serving list to remove call records associated with the base station in question.

We similarly emulate *device* failures by removing all call records associated with the emulated device failure. For example, a firmware bug could affect all devices from one manufacturer after a firmware update and prevent users from making calls even when the network is healthy. To introduce this type of failures, we remove call records with the device make/model associated with the emulated failure.

To introduce different failure scenarios for our evaluation, we combine the basic network and device failures described above with *geographic* information at different granularities and a *severity* factor to be applied. We vary the severity of a failure by failing different numbers of base stations or devices in the chosen aggregation, e.g., 10% of base stations in a ZIP for less severe failures and 100% of base stations in a ZIP for large outages.

• **Sensitivity to failure impact.** ABSENCE detects anomalies based on variations in the expected normal usage patterns. A key question to answer with our evaluation of this approach is what degree of impact ABSENCE will be able to detect. To answer this question we investigated two factors: (i) failure impact ratio and (ii) absolute impact. Failure impact ratio is defined as the ratio of the total amount of usage reduction during the failure over the total amount of a normal usage, or

$$impact\ ratio = \frac{total\ usage\ reduction}{total\ normal\ usage}.$$

Absolute impact is defined as the total usage reduction during an injected or detected event. For example, if during a 5-hour event, 4,000 out of 5,000 calls were lost, the absolute impact is 4,000 and the impact ratio is 80%. The smaller the impact ratio is, the more challenging it is for an anomaly detection system to identify it. The larger the absolute impact is, the more important it is for operators to pay attention to it. We use both metrics in evaluating ABSENCE regarding the sensitivity and performance in detecting anomalies.

• **Failure scenarios.** We consider two different geographical aggregation levels: city and ZIP-code area. In terms of devices, we consider two popular mobile-device manufacturers namely A and B and two popular mobile-device models, A-1 and B-1. Combinations of the two geographical aggregations and two specific device types allow a variety of test scenarios: city (e.g., all phones in Los Angeles), city+device make (e.g., all A phones in Los Angeles), city + device model (e.g., all A-1 phones in Los Angeles), ZIP code (all phones in ZIP code 07921), ZIP code + device make (e.g., all B phones in ZIP code 07921), etc. To come up with our final failure scenarios we consider three additional attributes: the type of service, the time, duration and the severity of the event (i.e., failure impact ratio).

In order to thoroughly evaluate ABSENCE we generate failure scenarios randomly based on different aspects we want to evaluate. Table 4.2 shows all aspects and the evaluation values from which we randomly selected to make up our failure scenarios. The table also shows an example scenario for each aspect.

We randomly chose 100 ZIP codes and 10 cities for the geographical aggregations to evaluate. We generated failures with different impacts by varying the amount of failed base stations when generating the failures, i.e., the impact ratio. There are 11 impact ranges, and each is 5% of impact ratio wide, i.e., [0%-5%], [5%-10%] etc.

We randomly picked 100 failures (i.e., 100 samples) for each impact range, e.g., we picked 100 failures that have [10%-15%] of impact, 100 failures that have [15%-20%] of

Table 4.2. Aspects and evaluated values of generated failures

Aspect	**Evaluated values**	**Example of a failure**
Geographical aggregation	100 ZIPs, 10 cities	All devices in L.A. fails
Device make	A, B	All A devices in ZIP 07921 fails
Device model	A-1, B-1	All A-1 devices in L.A. fails
Service	Voice, LTE	All devices in ZIP 07921 can't make calls or can't access the Internet
Start time	quiet period (06:00 UTC), busy period (20:00 UTC)	All devices in ZIP 07921 fails starting from 20:00 UTC
Duration	1, 2, 3, 6, 12 hours for busy; 8,10,12 hours for quiet	Voice service in ZIP 07921 outages for 8 hours starting from 06:00 UTC
Severity impact	0% to 55% of the total usage	A failure that causes 20% reduction of the normal usage in ZIP 07921

impact, etc., until all the impact ranges are covered. Note that for each impact range the randomly generated failures should be uniformly distributed across attributes. For example, 100 failures in the (ZIP code + device make) aggregation could happen either to all A devices or B devices and could last for 1, 2, 3 or 6 hours etc. This way, the set of generated failures should uniformly cover many failure types across attributes and therefore ABSENCE would have a set of diverse failure scenarios to evaluate against.

After generating this "pool" of failure scenarios and using ABSENCE to detect them, we gather the results and break them down into different dimensions based on the aspects in Table 4.2. In this manner we generated a total of 11,000 synthetic failures across our evaluation space. We present our evaluation results in Section 5.6.

• **Evaluation metrics.** We evaluated ABSENCE using two metrics: detection rate (%) and loss ratio (%). Detection rate is defined as the ratio of correctly detected failures (true positive,TP) over the total amount of introduced failures (true positive,TP + false negative,FN).

$$Detection\ Rate(\%) = \frac{TP}{TP + FN}.$$

Detection rate quantifies how effective ABSENCE is. The higher the detection rate, the more effective ABSENCE.

Loss ratio is defined as the ratio of the total net-loss *until detection* over the total amount of normal usage during the failure.

$$Loss\ Ratio(\%) = \frac{netloss\ until\ detection}{normal\ usage\ during\ failure}.$$

For example, if the normal usage during a failure is 5,000 and ABSENCE detects the failure when 1,000 calls get dropped then the loss ratio will be 1/5 (20%). In short, for a long lasting failure, the lower the loss ratio means the faster ABSENCE detected the failure.

4.7 Evaluation results

4.7.1 Variable-scale decomposition

The variable-scale decomposition technique is designed to ensure that ABSENCE uses sufficient usage data to enable accurate detection. To evaluate the effectiveness of this approach we evaluated ABSENCE with and without the variable-scale decomposition technique. We focused our evaluation on ZIP code level aggregations during quiet hours.We introduced failures starting from 03:00 UTC (typically the start of network quiet time) and lasting for 8 hours for voice service. The impact of the failures is in the range of 5% to 100%. We ran the two decomposition techniques (with and without the variable-scale mechanism) over the same set of failures and compared the detection rate of the two.

As shown in Figure 4.12, the variable-scale decomposition technique improves the detection rate during the low usage period by *8-10x*, e.g., ABSENCE detected 91% as opposed to 10% of failures that affect more than 2,000 calls in 8 hours, respectively, with and without the technique.

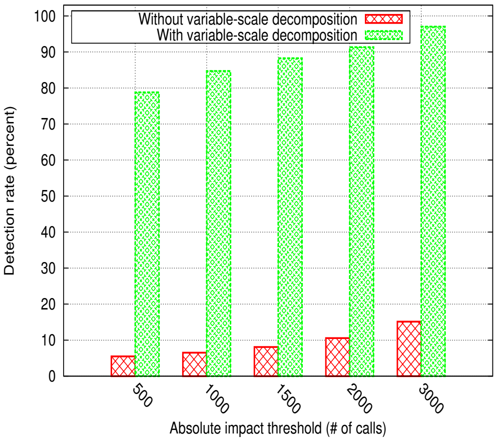


Figure 4.12. Failure detection during low usage: with and without the variable-scale decomposition

4.7.2 Synthetic failure evaluation

• **Overall results.** As shown in Table 4.3 first row, out of 11,000 introduced failures for both Voice and LTE service, ABSENCE was able to detect 9,676 with a detection rate of 88.0%. For failures with an impact larger than 10%, ABSENCE was able to detect 97.7% (8,064 out of 8,254 failures), and for failures that are larger than 20% of impact, ABSENCE detected 99.0% (6,189 out of 6,254 failures).

Table 4.3 breaks down the detection rate by different aggregations. Overall ABSENCE detected 98% of failures that have more than 10% impact across different aggregations from large aggregations (e.g., city level - all users in LA) to smaller aggregations (e.g., (ZIP code + make) level - all Device A or B devices in a ZIP code). Next, we look into different factors (i.e., failure impact ratio, absolute impact) that affect the detection rate of ABSENCE. With each factor, we also break down the results into different aggregations.

• **Failure impact ratio and detection rate.** We would like to understand the effectiveness of ABSENCE as a function of the severity of failures, i.e., the impact ratio, for different failure scenarios.

• ***Overall results.*** Figure 4.13 shows the overall detection rate as a function of the impact ratio of failures across all aggregations and service types. ABSENCE was able to detect 96% of failures that have a 15%-20% of impact across all aggregations, services and device types. For outages (with 50% of impact or more severe), ABSENCE detected 100% of them. For failures that are less severe (i.e., smaller than 10% of impact) ABSENCE detected 20%-67% of them.

Table 4.3. Overall results break down by different aggregations

Break down	**Aggregation**	**All failures**	$\geq$ 10% **of impact**	$\geq$ 20% **of impact**
Total	All aggregations	88.0%	97.7%	99.0%
Geographical break down	City	89.6%	98.1%	99.0%
	Zip code	87.5%	97.8%	99.4%
Service break down	Voice	85.2%	96.5%	98.7%
	LTE	90.7%	98.9%	99.3%
Geographical + device break down	City + device make (e.g., A devices in LA)	88.9%	99.0%	99.3%
	City + device model (e.g., A-1 devices in LA)	88.3%	97.3%	98.6%
	ZIP code + device make (e.g, A-1 devices in 07921)	85.5%	96.3%	98.6%

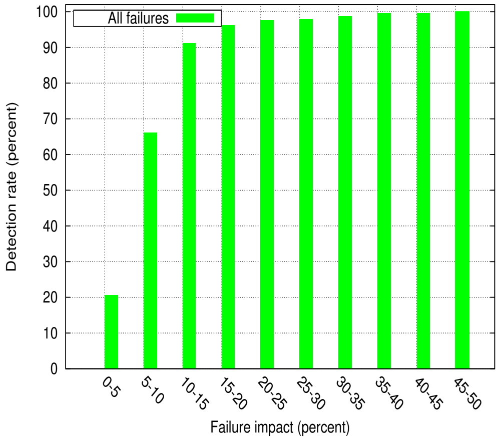


Figure 4.13. Overall detection rate vs. impact ratio

• ***Failures at different aggregation levels.*** Figure 4.14 shows ABSENCE's detection rate *for failures with small impact* at different aggregation levels: city, city + device make, city + device model, ZIP code, ZIP code + device make. Overall, for failures with more than 15% of impact, ABSENCE is equally effective across aggregations with a detection rate of 94% or better. This trend continues for failures with higher impact. For failures with lower impact (0%-5% and 5%-10%), ABSENCE's detection rate reduces to between 5% and 80%. For those failures, ABSENCE was slightly more effective for large aggregations such as *city* and *city + device make*.

• ***Failures happen to different service types.*** Figure 4.15 shows ABSENCE's detection rate for failures occurring in LTE and voice services. For failures with more than 15% of impact, ABSENCE's detection rate is high for both voice and LTE (i.e., around 97%). For less severe failures (i.e., less than 15% of impact), ABSENCE detected failures to LTE service slightly better than voice service.

• ***Failures happen to different mobile device types.*** Figure 4.16 shows ABSENCE's detection rate for failures associated with two popular device makes (A and B) and two popular phone models (A-1 and B-1) at city level. ABSENCE detected around 94% of failures with 15%-20% of impact associated with those mobile device makes and models.

• ***Failures breakdown by duration.*** Figure 4.17 shows ABSENCE's detection rate for failures with different durations. Overall ABSENCE is equally sensitive across durations of the failures: it detected about 95% of failures with $15-20\%$ of impact for both short and

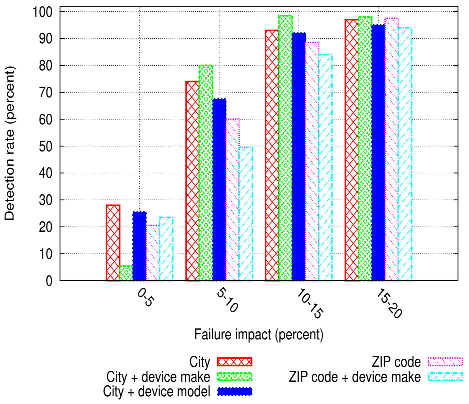


Figure 4.14. Small failures at different aggregation levels

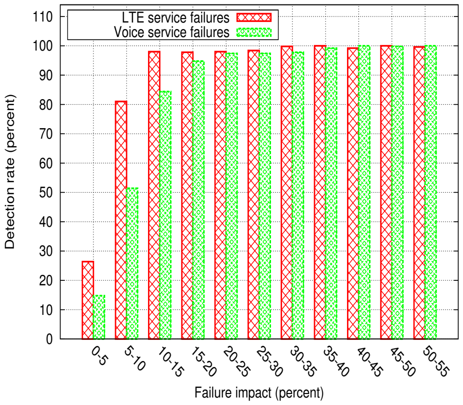


Figure 4.15. Failures for different service types

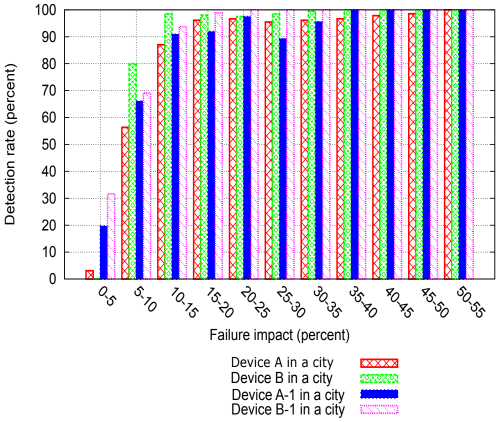


Figure 4.16. Failures for popular device make/model (city level)

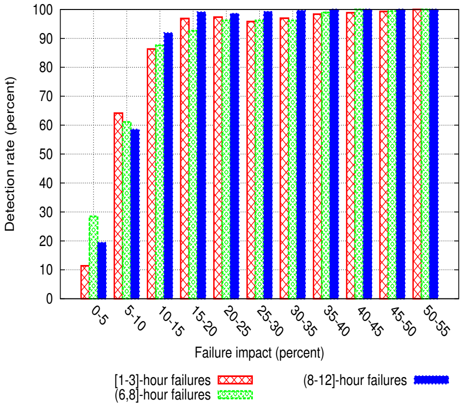


Figure 4.17. Failure duration

long-lasting failures. For failures with 0%-10% impact, ABSENCE detected 10%-63% of them.

• **Absolute impact and detection rate.** To understand how effective ABSENCE is in detecting failures ranked by the absolute size, we looked into ABSENCE's detection rate as a function of the absolute impact of the failures. Figure 4.18 shows that for LTE ABSENCE detected 94% of failures that cause more than 2,000 PDN connections to be dropped at the ZIP code level and 96% of failures at the (ZIP code + device make) level.

• **Loss ratio of detected failures.** To understand how quickly ABSENCE can detect long-lasting failures, we obtained the loss ratio (Section 4.6) of detected failures that last at least 6 hours for all services across different aggregations. We are interested particularly in

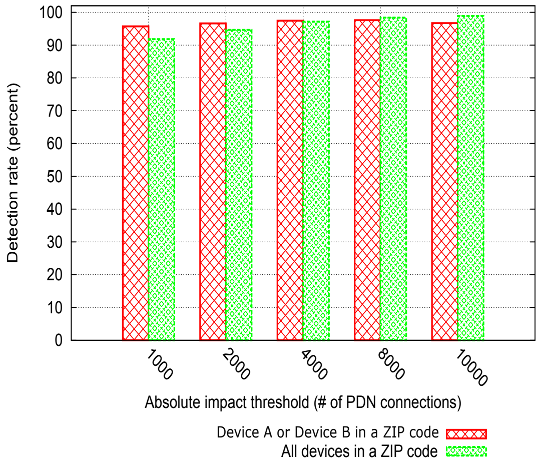


Figure 4.18. Absolute impact - LTE service

long-lasting failures because those failures often cause larger impact and early detections mean lower impact.

Figure 4.19 shows the CDF of the loss ratio of detected failures during quiet hours (i.e., low usage period) and busy hours (i.e., peak usage period). In general, during busy hours 98% of the failures were detected when less than 10% of the total loss happened (i.e., if the issue is fixed at the detection time, 90% of the usage would be recovered for 98% of the failures). During quiet hours, 90% of the failures were detected when there was less than 10% of loss. Given that the detected failures are at least 6 hours, 10% loss suggests that the failures are mostly detected right after the first hour of the failures.

• **Impact based event prioritization.** Because of resource constraints, in an operational setting providers typically need to prioritize the events that they investigate. Because it is inherently usage based, ABSENCE lends itself to an "operational knob" that operators can tune to distinguish large impact events from the small impact ones, so that they can rapidly respond to more severe conditions. Here we evaluate the tradeoff through such a knob, in the form of a "cut-off" threshold – the threshold above which events are defined as of high priority – between the number of high priority events and the detection rate (rate of high impact events to be included in the high priority list).

Figure 4.20 shows the detection rate (Y1-axis) and the fraction of synthetic events that are of high priority (Y2-axis) as functions to the threshold for the ZIP code level. We observe that if we only focus on events that have an absolute impact of over 4,000 records,

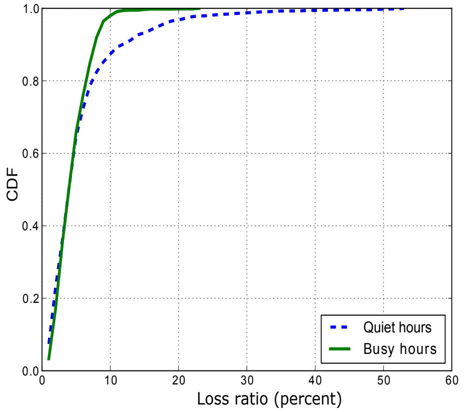


Figure 4.19. Loss ratio of detected failures that last for at least 6 hours

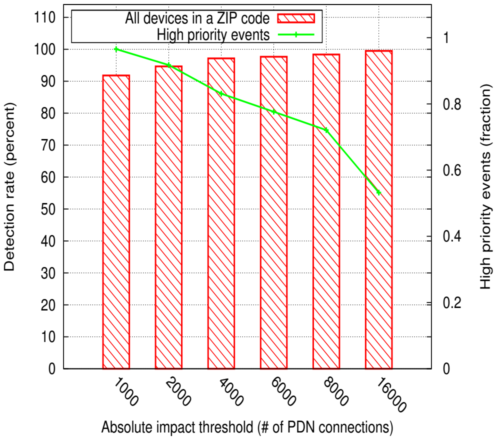


Figure 4.20. "Cut-off" threshold - LTE service at ZIP code level

ABSENCE achieves the detection rate of 97% among the 82% of the events.

• **Implications.** Our evaluation of ABSENCE shows that: (i) ABSENCE has a very high detection rate across all scenarios for failures with medium to high impact (i.e., above 15%). (ii) ABSENCE can detect failures relatively quickly (i.e., has a low loss ratio), thus reducing the impact of failures. (iii) Prioritizing events based on the absolute number of missing records provides a simple operational knob that enables operators to tune the number of high priority events generated by ABSENCE. These results suggest the practical feasibility of using ABSENCE to perform service disruption detection at the scale of modern mobile networks.

4.8 Operational validation

In this section, we validate service impacting events detected in the operational data via ABSENCE with known historical service outages. Specifically, we use events that resulted in anomalous volumes in customer care calls as our known customer impacting network events. Customers can call into customer care centers to report service issues and to discuss other concerns. The vast majority of customer calls relate to individual customer concerns; they may be the result of individual customer device issues or user errors, for example. However, in some situations, these customer calls may be the result of a broader service impacting event. Network, device type or application disruptions can thus result in a spike in the number of customer complaints. These events – spikes in customer care calls – are captured in a database along with their associated underlying network/device/application root cause, and are used here to provide a source of ground truth of service disruptions that we use to compare with those detected by ABSENCE.

We first attempt to validate service disruptions detected by ABSENCE with customer complaint events over a corresponding time period. We then investigate a number of specific use cases in more detail to verify ABSENCE functionality.

4.8.1 Comparison with customer complaints

From the customer care database, we can extract customer complaint events at a market level, i.e., which market the event is in and the event's start/end time. We obtained a list of 19 such events from the customer care database that happened to Voice and LTE data services. Operations had confirmed that each of these was a true service disruption. We then attempted to detect these with ABSENCE using CDR aggregates for the corresponding dates.

We ran ABSENCE to get a list of detected events at the ZIP code level and compared the detected events with the 19 customer complaint events mentioned above. Customer complaint events were aggregated at the market level (i.e., a market typically consists of several geographically close cities), while we were detecting events at the ZIP code level. As a result, before performing the comparison, we mapped the ZIP code of ABSENCE detected events to the corresponding market. We considered a *match* between an event detected by ABSENCE and a customer complaint event if the two events are spatially and temporally matching. Using this approach, ABSENCE was able to detect *all* 19 customer complaint events. We noted that ABSENCE also detected possible service disruptions that are not included in the customer care database. Due to lack of ground truth available in this study, it is challenging to investigate these further.

4.8.2 Alarm rate and true positive rate

ABSENCE detected events that are not included in the customer care database. Due to lack of ground truth it is challenging to determine whether those events are false positives. However, to make ABSENCE practical, the number of events per day (i.e., alarm rate) should be reasonable for an operations team to handle while the true positive rate should be maintained. To adjust the alarm rate, ABSENCE uses different cut-off thresholds (i.e., amount of usage impacted per hour) to filter out events with relatively small impact, i.e., if the impact of an event is smaller than a certain threshold, the event will not trigger an alarm. We used the 19 events in the customer database as the ground truth for the true positive rate and varied the cut-off threshold to observe the alarm rate of ABSENCE. Figure 4.21 shows the alarm rate and the true positive rate as a function of the cut-off thresholds. As we can see, both the alarm rate and the true positive decrease as the cut-off threshold increases. To protect proprietary information, we show the alarm rate and the cut-off threshold as relative numbers. If ABSENCE only triggers alarms for events which impact greater than 4*n* calls/PDN connections per hour, then the operations team will need to investigate around *m* such events per day and ABSENCE detects all of the 19 events above (i.e., 100% true positive). We confirmed that *m* is manageable by Operators and thus ABSENCE is practical in an operational environment.

4.8.3 Use cases

In this section we explore a number of specific use cases where ABSENCE was able to detect anomalies that also showed up in the customer care database. In this section, ABSENCE used 4*n* as the cut-off threshold for the number of calls or PDN connection per hour.

• ***Voice service in a large metropolitan area.*** This failure was on the voice service in several ZIP codes of a large metropolitan area. Users in these areas were not able to receive calls from landline devices. The event started at 16:00 UTC on a given Tuesday according

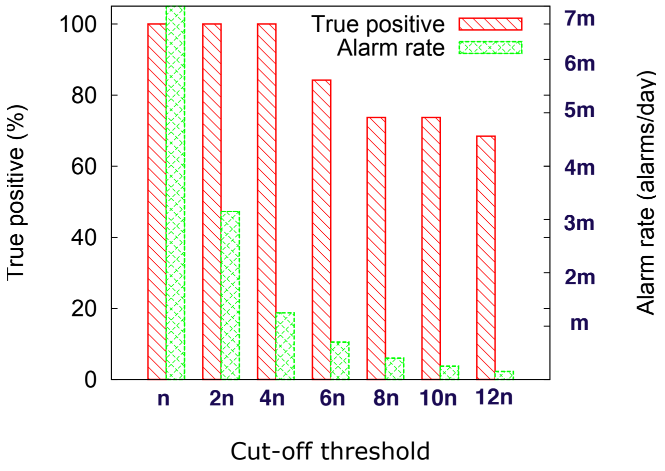


Figure 4.21. Alarm rate and true positive of ABSENCE

to the customer ticket data. Figure 4.22 shows the time series in UTC of the historical usage data for voice services of one of the affected ZIP codes (dashed line) for previous Tuesdays and the abnormal usage on the date of the service disruption (red solid line). ABSENCE was able to detect an anomalous event at 16:00 UTC (red point) as the usage falls significantly outside of the range of the normal historical usages.

- ***Voice service in a large metropolitan area.*** This failure was on the voice service in several ZIP code areas in another large metropolitan area. The failure was first evident in customer ticket volume at 12:00 UTC on a Friday according to the customer complaints and the reports by the operator. Figure 4.23 shows the historical usage (dashed lines) of previous Fridays and the usage of the day that the failure occurred (red solid line) for one of the affected ZIP code areas. As we can see, there are drops in the usage on the day of the failure and ABSENCE can detect the anomaly at 14:00 UTC (red point) – two hours after it commenced. Note that the sudden drop later in the day is due to the change in the number of users in the area according to the change of the zip-code profile.

- ***Voice mail.*** In this use case the failure happens to a much larger area (i.e., all ZIP codes in a large metropolitan area). This failure started around 18:00 UTC on a Monday and was associated with one of the operator's Voicemail services – a data service that is available to only a subset of user devices. This use case is an example of the failure of a particular service that impacted only particular device types within a given area. ABSENCE detected a series of anomalies happening to devices in the metropolitan area commencing at 18:00 UTC (when the failure itself commenced) (Figure 4.24 solid red line).

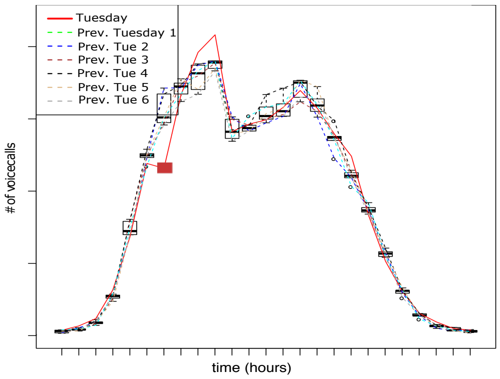


Figure 4.22. Voice service anomaly: metro area 1

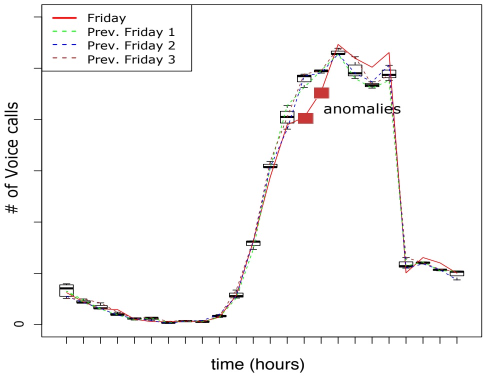


Figure 4.23. Voice service anomaly: metro area 2

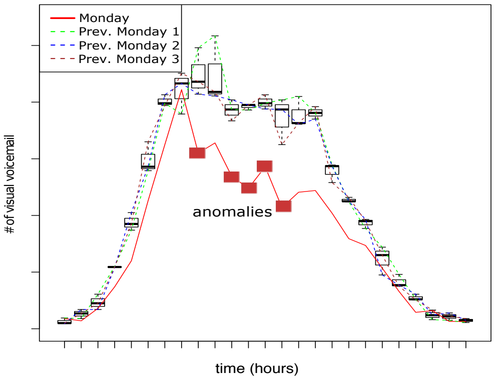


Figure 4.24. Voicemail service anomaly

4.9 Discussion

• **Special events.** Special events and holidays may affect users' usage and mobility pattern, which can be challenging for a passive monitoring approach. For example, people make fewer calls on holidays, or during a sport event. Correlated usage reduction in large scale can cause ABSENCE to generate false positives. If users' service consumption does not change but their mobility pattern changes (e.g., gathering at a stadium and using the phones as normal), ABSENCE will not generate false positives because the location profile will handle this and their usages are counted toward the profile location regardless of where they are.

• **Metrics to use.** ABSENCE uses number of calls and number of PDN connections as the metric for detecting service disruptions. Those two metrics could quantify users' experience in most cases, e.g., if a base station fails and users could not attach or make calls/create PDN connections. Potentially, there are other metrics that could be used such as duration of calls or number of Bytes downloaded/uploaded. These metrics would capture other types of failures. For example, a failure at the routing system in the network may cause voice calls to be routed to voice mail instead of reaching the callee. In this case the total duration of calls would be a better indicator of a potential service impairment worthy of investigation.

• **New services in data network.** Network operators may introduce new data services such as Wi-Fi offload. Such new services may affect the usage captured in the CDRs, as load is (deliberately) reduced. Since ABSENCE uses multiple months of historical usage as the baseline, if Wi-Fi offload skews the usage, the baseline will be rebuilt. After a sufficiently long period of training (typically 3-4 weeks if a weekly seasonal model is used), the new baseline will include the Wi-Fi offload and ABSENCE will be able to operate in the new environment.

4.10 Related work

There are quite a lot of related works in the area of service disruption detection including both commercial systems such as Keynote [8] and Gomez [5] and various efforts by the research community [66, 91, 156, 158]. All of them share two limitations. First, their effectiveness is typically limited by the coverage of deployed probes. Second, they all need to inject unnecessary probing traffic into the system, which could affect legitimate users. By contrast, ABSENCE detects service disruptions in a nonintrusive (passive) manner by purely depending on the existing traffic from real users.

Our work also relates to various mobile network performance studies. For example measurements from mobile devices have been used to study the performance of mobile networks [112]. Several studies have investigated protocol level performance aspects of mobile networks [78, 107, 127]. While these detailed performance aspects of mobile networks are related to our work, ABSENCE is a network management tool dealing with the operational health of a mobile network. As such ABSENCE is most related to various network operations tools [100, 101, 141, 154, 157]. A framework for network anomalies based on a principal component analysis approach has been proposed in [157]. A performance troubleshooting tool [100] and a service quality assessment mechanism [141] for

IPTV networks have been developed. Service anomaly detection [154] and troubleshooting tools [101] have been developed for ISP networks. In contrast ABSENCE is focused on service disruption detection in mobile networks and deals with the specific mobility and scalability challenges of that environment. Production tools such as [47] are installed on user devices to collect users' service experience yet they are not available on most Samsung and iPhone popular models due to privacy issues. ABSENCE in contrast works for all device models without installing any software on the device.

4.11 Conclusion

We presented our work on ABSENCE, a service disruption detection system for mobile networks. ABSENCE makes use of customer usage data, in the form of aggregated and anonymized call detail records, to derive historical usage patterns for groups of customers. Appropriate selection of these groups results in stable and accurate predictions of usage patterns, allowing ABSENCE to detect deviations as possible service disruptions. We presented a data driven exploration of the design space. We performed a systematic evaluation of ABSENCE by introducing synthetic failures in data from an operational mobile network and compared ABSENCE's detection results with known ground truth events from the mobile network. ABSENCE is currently operating in a preproduction environment. Our future plans include integration of ABSENCE with the operator's production environment and fine tuning the parameters to improve the accuracy and utility of our approach in an operational setting.

CHAPTER 5

ENHANCE RELIABILITY IN MOBILE NETWORKS USING DISTRIBUTED SYSTEM TECHNIQUES

5.1 Introduction

Recent years have seen a tremendous uptake of cloud computing. More and more companies move their services to the public cloud to take advantage of the economies of scale, the resource elasticity and scalability that the cloud offers. In stark contrast, the telco industry today faces major challenges in equipment upgrading, scalability, and introducing new services [61]. Cellular core networks are largely still based on custom-built hardware mandated by the strict reliability requirements posed by running a network core.

To alleviate these challenges, telcos and cellular operators are attempting to virtualize their core networks through network function virtualization (NFV). Typically, this is in the form of a move to a private-cloud setting, where the telco provider has full control of the infrastructure and can optimize the whole stack for its particular services. Indeed, owning the whole cloud stack can provide direct mechanisms for fault tolerance and management – open source cloud software stack OpenStack and its OpNFV layer provide such services (e.g., see Vitrage [118] and Doctor [120]). However, such a deployment model still cannot take full advantage of the economies of scale a public deployment can offer. Telco providers will have to manage and maintain the new private cloud deployments, while at the same time, super-optimized cloud stacks for a particular core service might not be able to scale to the size of a public cloud, and may be at odds with the requirements of a new service to be introduced.

Instead, the question we address is whether it is feasible to implement a cellular core network on top of a *public* cloud, such as Amazon AWS or Microsoft Azure. To achieve this, one has to address two main challenges. First, reliability – a cellular core network today requires “five 9s” reliability (i.e., availability of 99.999%) [59, 115]. Typical public cloud availability SLAs are four 9s or less, which means an order of magnitude more expected outages. Second, service abstractions mismatch. Naturally, public clouds are optimized for general workloads, offering basic network abstractions such as a network node, a private network, or a load balancer. Cellular core networks are complex, with multiple different components implementing distributed state machines that will need to be redesigned atop the cloud’s abstractions.

In this work, we introduce a distributed cellular network architecture for the public cloud. We focus on the evolved packet core (EPC) [19], which provides core network control and data plane functionality for all cellular radio technologies (2G, 3G and 4G/LTE), and we present ECHO, our proposal for a distributed EPC cloud-based architecture. While operator networks consist of the EPC and middleboxes, ECHO focuses on the core EPC as much effort has already been devoted to virtualizing conventional middleboxes ([65, 69, 135, 140]).

ECHO provides the same properties that EPC guarantees today. The role of the EPC is to manage user devices (Section 5.2.1); this is driven by a distributed state machine that stores user device states across multiple components, as well as on the user’s mobile device. All these states need to be consistent in spite of potential component and network failures. Inconsistencies can lead to long service outages (Section 5.3.1). This is fundamentally different from the requirements of conventional middleboxes where state is typically shared only across multiple instances of the same appliance.

To achieve consistency, our design leverages the “necessary” reliability of access points – mobile devices are only connected to the network as long as their associated access points are operational. ECHO introduces a thin software layer (entry point agent) on access points which enforce consistency between the state at the mobile device and corresponding states on components in the cloud. The entry point serializes all state updates from a single transaction across all EPC components in the cloud. This scales well since each user only executes a limited number of transactions, and the scalability is required across users. This effectively means that EPC components in the cloud can be replicated by refactoring their functionality into a stateless front-end and persistent backend storage in which state is stored after each operation. ECHO further takes care of a nontrivial task of keeping interactions among these cloud components (which we call side-effects) consistent. In doing so, ECHO avoids modifying the standard core protocols and provides abstractions that nicely overlay with the existing cellular standards.

In spirit, ECHO is similar to other reliable distributed cloud services, and it relies on well-known techniques like redundant stateless components, external state storage, and state machine replication for high availability and strong consistency. The key challenge in ECHO is dealing with distributed side effects, which must appear to execute atomically and in order (linearizably) from the client's perspective. Ordinarily, this is achieved through mutual exclusion, but ECHO must be fully nonblocking to provide high availability.

Our contributions can be summarized as follows:

• We propose ECHO, an EPC architecture for the public cloud. ECHO is based on the observation that EPC runs user-specific distributed state machines in parallel. We identify the main building blocks for a reliable implementation of the distributed state machine, and we then map these blocks to the original EPC system components.

• We demonstrate the feasibility of the proposed architecture by implementing it in full. We implement the entry-point agent software and deploy it on a COTS LTE small cell [82]. Additionally, we implement the required EPC modifications into OpenEPC [55] and deploy ECHO on Azure.

• We perform an extensive evaluation of the system using real mobile phones as well as synthetic workloads. We show that ECHO is able to cope with host and network failures, including several data-center component failures, without end-clients noticing it. ECHO shows performance comparable to commercial cellular networks today. Compared to a local deployment, ECHO has reasonable overheads that are introduced by the extra-needed reliability – less than 10% latency increase and throughput reduction on control procedures if replicated within one data center.

To the best of our knowledge, ECHO is the first attempt to run an EPC on a public cloud. We hope that it will be a step towards relieving telcos the burden of managing their own infrastructure and will further inspire the next generation of 5G cellular networks, which will require even more scale and more decentralization than the current architecture.

5.2 Background

This section presents a brief overview of today's mobile core architecture and makes the observation that, effectively, the network core implements multiple distributed state machines, one per user.

5.2.1 Mobile core network architecture

A cellular network consists of a wireless radio access network and a wired mobile core network. The core network consists of a control plane and a data plane.

• **Control plane.** The main component of the control plane in LTE/EPC is the Mobility Management Entity (MME) which is responsible for handling registration/authentication, connection setup, device mobility, etc., for mobile clients (also called User Equipment – UE). MMEs do not participate in packet forwarding but only modify routing entries in the gateways on the data plane.

• **Data plane.** The data plane consists of a Serving Gateway (SGW) and a Packet Gateway (PGW). They are responsible for routing and forwarding the data packets from the UEs to and from an external IP network (e.g., the Internet). When it receives a connection setup request from the control plane (MME), a data plane gateway installs a state locally and in some cases, triggers requests to other gateways, which in turn create local state associated with the request.

• **UE context.** The MME keeps track of a *UE context* for each *attached UE*. The UE context consists of subscriber information (authentication key, UE's capability), the current state (connected or idle), and the data connection information the UE has on the data plane. The UE context on the MME reflects the "real-world" states on the data plane gateways – it contains a data connection profile (called an Evolved Packet System bearer or EPS bearer) that consists of a QoS profile and end-point IDs that are set up on-demand.

Most of the UE context information is created and exchanged between the MME and the UE the first time the UE attaches to the network. However, the UE context can change after attachment depending on whether the UE is active or not; when the UE is idle for a certain amount of time (e.g., half a minute), the eNodeB releases its radio resource and triggers a resource release procedure on the gateways. If the UE has data to send, it requests the MME to re-create the data connection again via a Service Request procedure

(Figure 5.1). Note that in all cases, it is required that the UE context be updated to match the actual states on the data plane.

5.2.2 Mobile core: a distributed state machine

The cellular control plane implements a distributed state machine for each UE, as illustrated on the Service Request example in Figure 5.1. As shown in the figure, once a control channel has been established across the Radio Access Network (RAN) (i.e., RRC setup), a request from the UE - *Service Request* - triggers the MME to make changes to the UE context and sends a *Modify Bearer Request (MBR)* as a side effect request to a Serving Gateway (SGW). This side effect message requests to set up a data bearer for the UE on the SGW. When it receives the *MBR*, the SGW sends another *MBR* to Packet Data Gateway (PGW) to set up a tunnel endpoint for the UE on PGW. If this request fails, a timer on the MME expires and as a result the MME retries (message #3, 4). If the retry goes through successfully, the PGW acknowledges the SGW which acknowledges the MME. At this point, the MME knows that a data bearer is created for the UE (messages #6, 7). The MME then informs the eNodeB with information of the data bearer (message #8). The eNodeB then sets up a E-UTRAN Radio Access Bearer (ERAB) with the UE acknowledges the MME (message #9) when the ERAB is created.

The state machine is distributed across multiple components and a transition may involve communications and state changes across multiple other components. The control

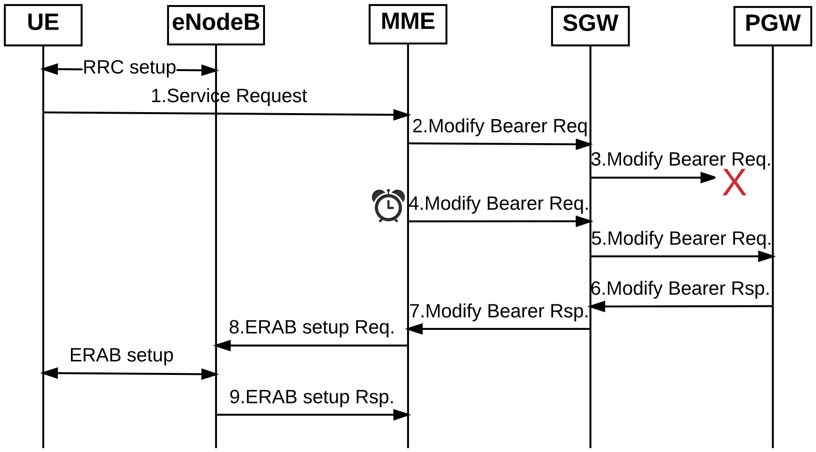


Figure 5.1. Service Request procedure in LTE/EPC.

plane runs many such state machines in parallel, *one for each UE*. A generalized depiction of this distributed state machine, which is common across all mobile core operations and components, is depicted in Figure 5.2.

Specifically, the distributed state machine deals with the following events and messages:

• **Request from UE.** Most of the changes in the state machine are triggered by a client or mobile device. For example, when an idle UE has data to send, it sends a Service Request (message #1 in Figure 5.1)) to the MME.

• **Side effect request.** Upon receiving a request from a UE, the MME may alter the states at other components. In the Service Request example above, the MME must set up a bearer in the data plane. The MME sends a bearer setup request to the SGW (message #2), which sends a bearer setup request to the PGW (message #3). We call these two messages *side effect requests*. They are generated by components in the cellular core; they are indirectly triggered by the main request that originated in the UE.

• **Timers.** A state transition can also be triggered by a time-out. For example, if an SGW does not respond to the bearer setup request, the MME will trigger a retry when its timer expires (message #4). This retry generates another side effect request to the system. A timer can be set and triggered by any component, if so required by the protocol.

• **Control messages.** Components in the system communicate through various control messages, such as messages that trigger requests, ACKs, NACKs and other state update messages (message #1 – #9 in Figure 5.1).

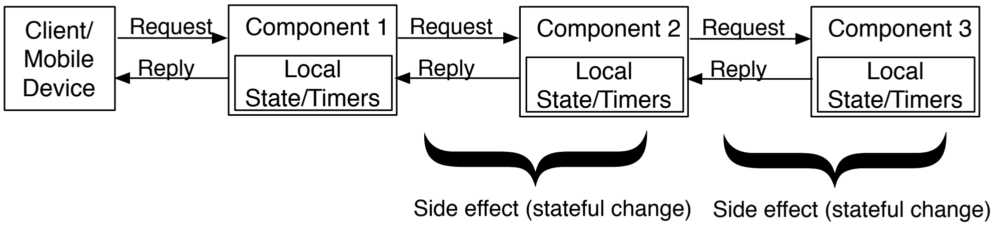


Figure 5.2. Distributed state in core mobile network. Components 1, 2 and 3 map to the MME, the SGW and the PGW in Figure 5.1.

5.3 Reliability in cloud-based EPC

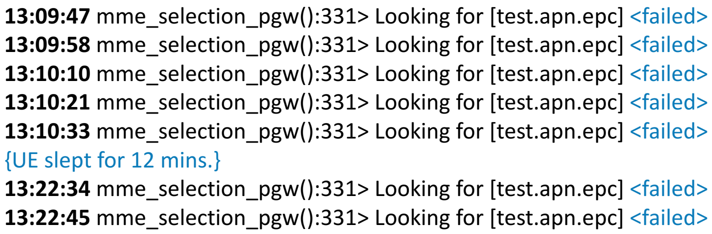
Through examples, we highlight the strict reliability requirements of the cellular core network. We then present the state of the art of reliability in the current mobile core network using hardware. We contrast today's cloud availability with hardware reliability through a 3-month-long study.

5.3.1 Mobile network reliability requirements

We conducted experiments with a real mobile device (Nexus 5), an LTE eNodeB (IP.Access smallcell) and the OpenEPC core network to demonstrate the core network's sensitivity to failures and its reliability needs. The results motivate ECHO's key design requirements (§5.4).

• **High availability.** An MME outage would immediately cause a service outage on many UEs. Moreover, a service outage on an MME would also be interpreted as a congested mobile network, so UEs are required to back off from the network. We demonstrate this with an example scenario in which, after 5 unsuccessful Attach attempts lasting 1 minute in total, the UE entered silent state for 12 minutes before it retries to attach again. Hence, a short MME outage can result in disproportionate experienced outages in UEs. To illustrate this behavior, we triggered the UE to attach to the LTE network and left a bug in the MME so that the UE failed to attach. Figure 5.3 shows the MME's log with timestamps illustrating this experiment. This suggests the network must be highly available.

• **Persistent state.** The UE context exchanged during the attach procedure is kept in the MME. If this context is lost, the MME cannot process UE requests, leading to an outage for the UE. We show experimentally that when the MME loses the UE context, the UE loses connectivity for 54 mins!

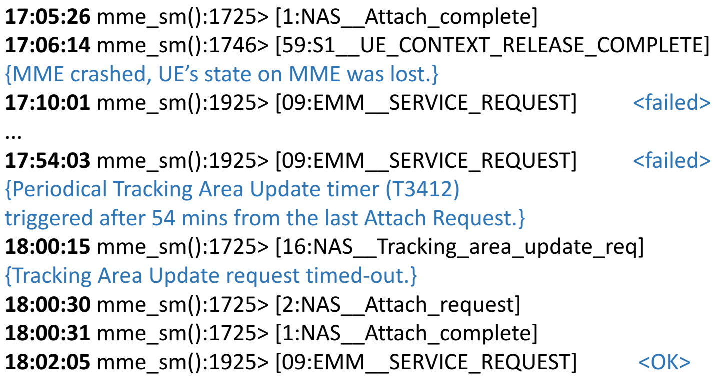


```
13:09:47 mme_selection_pgw():331> Looking for [test.apn.epc] <failed>
13:09:58 mme_selection_pgw():331> Looking for [test.apn.epc] <failed>
13:10:10 mme_selection_pgw():331> Looking for [test.apn.epc] <failed>
13:10:21 mme_selection_pgw():331> Looking for [test.apn.epc] <failed>
13:10:33 mme_selection_pgw():331> Looking for [test.apn.epc] <failed>
{UE slept for 12 mins.}
13:22:34 mme_selection_pgw():331> Looking for [test.apn.epc] <failed>
13:22:45 mme_selection_pgw():331> Looking for [test.apn.epc] <failed>
```

Figure 5.3. Examples of real-world outages: 5 consecutive Attach failures caused UE to sleep for 12 mins

Figure 5.4 shows the MME's log with timestamps when the MME loses context for an attached UE. The UE attached to the network (*17:05:26*), and after a period of inactivity, the UE released a portion of the connection (at *17:06:14*, note that the UE context should be kept by the MME). After this the MME crashed and the UE context was lost. The UE then requested service (i.e., it had data to send) but did not get any service (from *17:10:01* to *17:54:03*). Approximately 54 mins after the Attach, the UE performed a periodic Tracking Area Update (TAU) procedure (*18:00:15*). This TAU also failed because the MME does not have any context of the UE. The TAU timed-out after 15 seconds. The result of this unsuccessful TAU is that the UE is moved to the EMM deregistered state and as defined in the protocol [20] it performed a new Attach Request (*18:00:30*). This Attach Request was performed successfully and the UE exchanged its context with the MME. After having the UE context, the MME was able to serve the UE as normal (*18:02:05*), ending the extended UE service outage. This suggests that the EPC network must persist UE's state to guarantee continuous operation.

• **In-order message delivery and execution.** To maintain state consistency, requests from the same UE should be executed in *First-In-First-Out (FIFO)* order on the MME. We demonstrate this with an experiment which shows an example where the FIFO execution is violated, resulting in state inconsistency and service outage for the UE. In the experiment, a sequence of requests $\langle R_1, R_2 \rangle$ of the same UE arrive at the MME. However, request R_1 was delayed and executed *after* R_2. The result was that the stale request R_1 overwrote the effect of request R_2 which causes inconsistency between MME's state and UE's state.



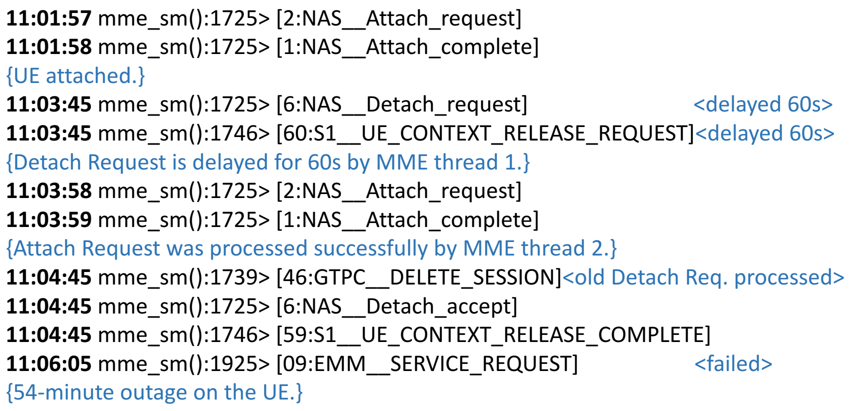
```
17:05:26 mme_sm():1725> [1:NAS__Attach_complete]
17:06:14 mme_sm():1746> [59:S1__UE_CONTEXT_RELEASE_COMPLETE]
{MME crashed, UE's state on MME was lost.}
17:10:01 mme_sm():1925> [09:EMM__SERVICE_REQUEST]        <failed>
...
17:54:03 mme_sm():1925> [09:EMM__SERVICE_REQUEST]        <failed>
{Periodical Tracking Area Update timer (T3412)
triggered after 54 mins from the last Attach Request.}
18:00:15 mme_sm():1725> [16:NAS__Tracking_area_update_req]
{Tracking Area Update request timed-out.}
18:00:30 mme_sm():1725> [2:NAS__Attach_request]
18:00:31 mme_sm():1725> [1:NAS__Attach_complete]
18:02:05 mme_sm():1925> [09:EMM__SERVICE_REQUEST]         <OK>
```

Figure 5.4. Examples of real-world outages: UE did not have service for 54 minutes because MME crashed and UE context was lost

Figure 5.5 shows the MME's log with timestamps describing this experiment.

In this experiment, after attaching to the network, the radio interface of the UE was turned off to trigger a detach (11:03:45). That detach was processed by a *MME1 thread* which is a slow MME thread. We intentionally caused a delay of 60s on this thread through a sleep timer while at the same time releasing the session lock. Later the Nexus 5 was turned on to trigger another attach request which arrived at *MME2 thread* (11:03:58), updating the state of the UE Context with the *Attached* state. This was successfully verified by the MME2 and replied to (11:03:59). However, the slow MME1 thread later was executed and updated the UE Context with *Detached* state (11:04:45). The Detached Accept message was ignored by the UE. At this point, the UE state recorded in the UE Context and the actual UE state is inconsistent: recorded *Detached*, while the actual state is *Attached*. This caused a UE outage of 54 mins as in our previous experiment. This suggests that the EPC network must execute requests in the order that they are generated from the UE.

• **Summary.** The above experiments imply that the mobile network must be highly available, must persist UE context (state) and maintain state consistencies both between components and between the mobile device and components. The state consistencies mean that the core network must ensure (i) *in order execution* per mobile device – the distributed components must appear to process the requests in order from the mobile device's perspective and (ii) *atomic execution* – the distributed components must be in either a "before" or "after" state.



```
11:01:57 mme_sm():1725> [2:NAS__Attach_request]
11:01:58 mme_sm():1725> [1:NAS__Attach_complete]
{UE attached.}
11:03:45 mme_sm():1725> [6:NAS__Detach_request]                    <delayed 60s>
11:03:45 mme_sm():1746> [60:S1__UE_CONTEXT_RELEASE_REQUEST]<delayed 60s>
{Detach Request is delayed for 60s by MME thread 1.}
11:03:58 mme_sm():1725> [2:NAS__Attach_request]
11:03:59 mme_sm():1725> [1:NAS__Attach_complete]
{Attach Request was processed successfully by MME thread 2.}
11:04:45 mme_sm():1739> [46:GTPC__DELETE_SESSION]<old Detach Req. processed>
11:04:45 mme_sm():1725> [6:NAS__Detach_accept]
11:04:45 mme_sm():1746> [59:S1__UE_CONTEXT_RELEASE_COMPLETE]
11:06:05 mme_sm():1925> [09:EMM__SERVICE_REQUEST]                  <failed>
{54-minute outage on the UE.}
```

Figure 5.5. Examples of real-world outages: Violation of FIFO order execution caused state inconsistency and 54 minutes outage

5.3.2 Reliable EPC: state of the art

The conventional way to maintain high availability is to introduce redundancy in hardware. Telecom-grade reliable hardware is built with $N + M$ redundancy (N active blades have M back up blades) [59, 114, 146]. Active-standby techniques [94] allow for state synchronization (e.g., UE context) between the active and standby instances with the active one switching over to the standby one in case of a failure. This technique is extended to an NFV setting where a resource scheduler can quickly detect a fault and migrate service from a faulty component [118, 120].

Further redundancy is introduced at the protocol layer. The standard EPC architecture supports a pool of MMEs [24]. An eNodeB can connect to any of the MME instance in the pool. The MME instances share a common Session Restoration Server (SRS) [97, 98], which acts as persistent storage for UE context in real time. If one MME instance fails, the eNodeB will notice that its Stream Control Transmission Protocol (SCTP) connection to MME is broken. It will then attempt to reconnect, and will be connected to another MME instance.

To maintain atomic and in order execution, there is an SCTP connection between the eNodeB and the MME that provides retransmission, de-duplication, and request ordering. The reliable MME then maintains a FIFO queue of the requests and executes them in order, one at a time until the states are consistent across EPC components. In order for the MME pool mechanism to work, the failed MME instance must crash cleanly; after the SCTP connection is broken all of the requests that are already in the crashed MME's queue are completely discarded so that no stale requests exist.

There are several aspects of the existing designs which do not map well to the public cloud infrastructure. Unlike public cloud, hardware appliances and VNFs offer a fine-grain availability information and scheduling control (active standby or service migration). This allows for almost instantaneous fault isolation and repair, which is impossible in a public cloud [74]. A public cloud EPC deployment has to deal with failures proactively, before the software is certain that a fault has occurred, as we illustrate with public cloud measurements in the next section. Furthermore, due to higher inherent reliability of conventional nodes, the types of faults that can occur are different. Public clouds run all software on VMs that can delay executions (e.g., due to an upgrade), causing stale requests and inconsistent side-requests (as explained in the examples above). Again, a public cloud EPC deployment has to deal with these issues in software, as most of today's distributed systems deployed in the cloud do.

5.3.3 What does public cloud provide?

Typical telco appliances like the ones described in the previous section provide availability of 99.999% [115] ("five 9s" availability). The five 9s availability corresponds to an overall outage of 1 minute in 2 months. Instead, cloud offerings today, such as AWS and Azure, advertise VM availabilities of "four 9s", or outages of 1 minute every week – an order of magnitude larger total outage compared to reliable hardware platforms.

Besides the overall availability in the number of 9s, the mobile network reliability requirements outlined in Section 5.3.1 highlight that the duration of an outage can be critical. For example, the system may be able to recover from many short 1-second outages using transport or other mechanisms, but a few outages lasting minutes can be catastrophic (example 5.3). It is thus crucial to understand the availability properties (total outage instances and their duration) of public clouds in practice, beyond advertised SLAs.

To this end, we perform a 3-month-long measurement study in a major public cloud provider. We expect our findings to be indicative of other providers as well. We monitor the VM uptimes as well as the reachability of VMs at multiple levels: data center (DC) cluster, single DC, and across DCs. A DC cluster consists of three VMs in different availability zones behind a load-balancer within the same regional data center. There are two clusters per DC. In total, we use 3 DCs, two in Europe and one in the U.S. and perform TCP pings every 1 second from each VM to all other VMs. Further, we use Azure's Application Insights service [104] to monitor the reachability of our VMs from the public Internet. The service initiates web request to all VMs every 10 minutes from 10 locations across 4 continents. The cloud is available if *at least* one VM in a cluster is available.

• **Results.** Our results are summarized in Table 5.1. Each row in the table shows the observed availability constrained on an outage duration (e.g., in row $>$ 1 min we only account for outages that are longer than 1 min; at least some of these cannot be handled by MME retransmissions, as illustrated in example 5.3). We observe that the advertised SLAs of four 9s are generally met by the cloud. Most of the outages are very short, and

Table 5.1. Inter-DC availability in a major cloud provider

	Cloud				World-wide			
Outage	VM	DC Clus.	DC	DCs	VM	DC Clus.	DC	DCs
All	99.9947%	99.9998%	99.9999%	100%	99.988%	99.991%	99.9921%	100%
> 10 sec	99.9947%	99.9998%	99.9999%	100%	99.988%	99.991%	99.9921%	100%
> 1 min	99.9948%	99.9998%	99.9999%	100%	99.988%	99.991%	99.9921%	100%

can possibly be attributed to network congestion, or other instantaneous problems. We observed intra-cloud outages of more than 1 second, 2,400 times during our study. The Cumulative Distribution Function (CDF) of the durations of such outages is depicted in Figure 5.6. In all, there are 7 outages that last more than 1 minute and they can all be attributed to VM failures. However, VMs in the same DC cluster do not tend to fail at the same time.

The picture is significantly different as observed from hosts in the Internet ("World-wide" in Table 5.1). Availability is roughly an order of magnitude less compared to intra-DC measurements, implying that most "outages" are due to public Internet connectivity problems reaching the cloud. The CDF of the durations of the outages longer than 10 minutes (measurement interval) is depicted in Figure 5.6, and we can see that more than 20% of them last 20 minutes or more.

• **Implications:** In summary, we observe that our key requirements (high availability and state persistency) can be achieved with five 9s only if the service is replicated across multiple VMs across availability zones in a single DC; additionally, coping with public Internet reachability problems requires service presence across multiple regional data centers unless a dedicated connectivity service to the cloud [28, 103] is deployed which can

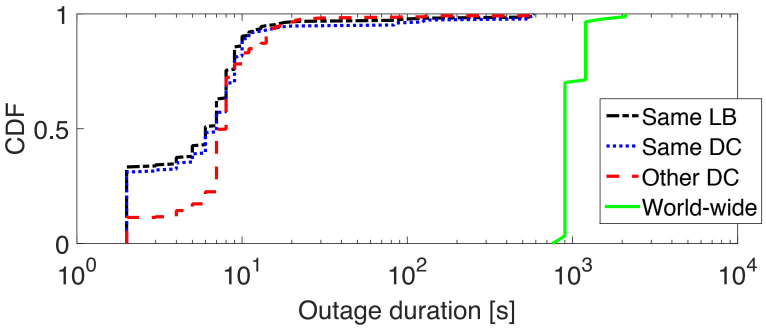


Figure 5.6. Outage duration distribution across all pairs of VMs

incur extra cost.

We also note that other studies, such as [73] that gives account of public cloud reliability over a 7-year period, point out that a median reliability across all public clouds (32 public clouds studied) is below 99.9% and that the median duration of an outage varies between 1.5 and 24 hours, depending on the type of failure. This reinforces the need for software replication and proactively dealing with faults.

5.4 ECHO design

Figure 5.7 presents a high-level depiction of ECHO's operation. ECHO moves the main components of EPC into the cloud, replicating them for reliability while connectivity to the UEs is through the base stations which implement our entry point serialization. Existing public cloud load-balancers provide load-balanced connectivity to the "regular" Internet and to the Internet-based access "backhaul" between base stations and the public cloud. We now discuss in more details the problem space, ECHO's architecture and operation.

5.4.1 Problem space

As discussed in §5.2.2, the EPC can be viewed as a distributed state machine comprising multiple components. Each component stores state for each user. ECHO must assume nodes and the connections between them can fail; the VM or container hosting a component could crash and restart or it could be arbitrarily slow, and connectivity between components is not reliable. Alternatively a node might reply with a correct response, but the replies could be late or lost.

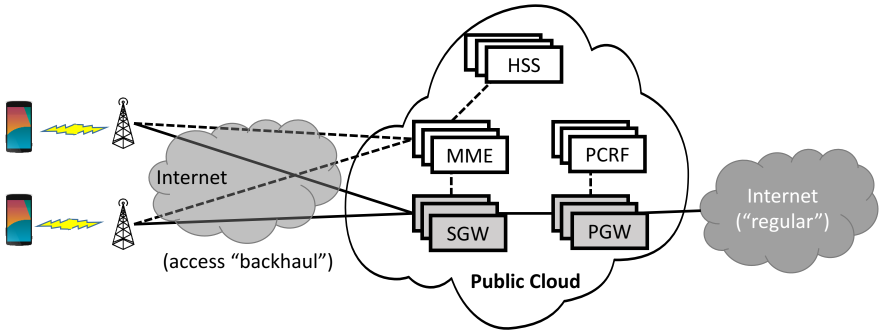


Figure 5.7. ECHO components at a high level

ECHO must continue operation despite these failures while producing the same results as the original core network that assumes components are reliable; ECHO must appear to execute requests atomically and in the order that the (per-device) requests arrive at the base station. ECHO must also scale to support a large number of users. Moreover, the EPC protocols are complex and constantly evolving; we must avoid modifying the protocols or relying on its implicit semantics for correctness. Finally, a particular challenge is that one of the component that stores the state is a user's mobile device, which cannot easily be modified.

5.4.2 ECHO architecture overview

Figure 5.8 depicts an overview of ECHO. Each control plane component (Components 1, 2 and 3) is replicated (instances 1 to *n*) behind a data center load balancer (LB) [102, 122]. Each component instance is refactored into a stateless processing frontend, paired with a high availability persistent storage backend that maintains state for all replicas (and all components). This allows quick replacement of a malfunctioning component and scaling based on demand. At each base station, i.e., eNodeB in LTE/EPC parlance, there is a "necessarily reliable" entry point. This entry point is the key to ECHO's availability and correctness (§5.4.3).

Figure 5.9 shows possible interactions when a request is issued in ECHO. At step (1), a request from the client is queued at the entry point. Step (2), the request is forwarded to the MME (but fails to be processed). The agent resends the request after a timeout at step (3). The retry at step (3) triggers side effects at steps (4,5). The side effect at step (4) triggers another side effect at step (6). Component 3 acknowledges the side effects (5,6) by ack (7,8). Component 2 after receiving the ack for request (6) sends an ack (9) that acks request (4). Component 1 then send ack (10) to ack request (1). When generating request (6), component 2 generates a timeout event for this request and sends it to the agent at step (11).

In ECHO, each request originates at the UE and is proxied by the entry point at the base station or access point. Each request gets a unique, sequential ID from the entry point, and it is queued until completion. The *sequence ID* captures the order that the requests arrive at the eNodeB from the mobile device. A component *n* acknowledges a request to a previous

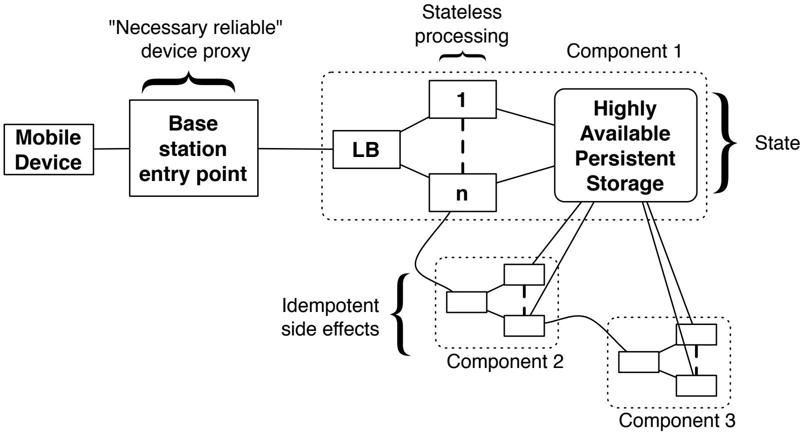


Figure 5.8. ECHO overview

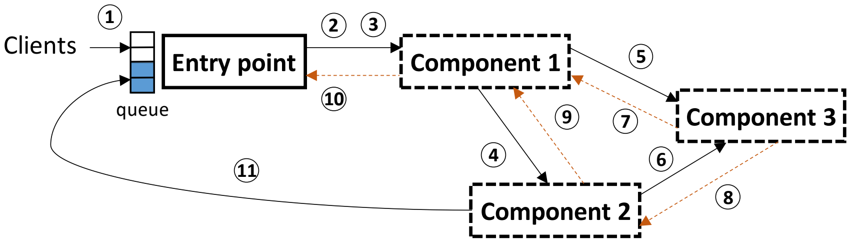


Figure 5.9. ECHO request example

component *n-1* once all of its downstream requests (requests to components *n+1*,*n+2*,...) are acknowledged. An acknowledgment to the entry point means the request from the mobile device has been applied at all components. After a timeout, the entry point retransmits the request until it is acknowledged, only after which will it move to the next request.

5.4.3 Necessarily reliable entry point

The ECHO entry point approach relies on the fact that the base station (eNodeB) is a *necessarily reliable* component. Because the network connectivity of a mobile device relies on wireless access to the base station, connectivity is lost if the base station crashes; there is no point designing the system to deal with base station failures. Therefore, since the entry point is as reliable as the base station, it is seen as a "reliable" component of the system. Moreover, note that because the number of users served by each entry point is limited by its wireless resources, its scalability constraints are different from core network components.

The entry point agent is a thin software layer deployed on a base station. In spirit, it is similar to a sequencer in other distributed databases [145]. However, it enhances an eventual completeness property and relies on the base station to guarantee reliability (instead of using the state machine replication technique). It provides the following functionality.

• **Sequential request IDs.** The entry point assigns a sequential ID to each request from a given UE; different UEs have independent ID sequences. The request is queued locally and forwarded to the next component (the MME). The entry point serializes the requests using a FIFO queue: the oldest unacknowledged request is resent until it is acknowledged and removed from the queue. The sequential IDs are used to ensure that requests are processed at components in the same order as the UE issued them.

• **Eventual completeness.** After queuing a request, the entry point persistently retries until the request is acknowledged before moving to the next one. This ensures a component failure in the cloud won't be visible to the mobile device; if an instance of a component crashes in the middle of an operation, the entry point transparently issues a retry and the retry will reach another instance of that component to recover from the crash. As the entry point is the "reliable" component, its retries ensure a request is eventually processed and is processed by *all* core components regardless of failures.

• **Reliable timers.** As in other protocols, components in EPC must set a timer whenever they receive a request. However, if components crash timers could be lost. In ECHO, since the entry point is considered reliable, components' timers are maintained and triggered by the entry point instead of by the components; after receiving a request, the component creates the timer event by sending a *set timer request* to the entry point. The set timer request includes a unique ID of the mobile device that the timer applies to, a unique timer ID, and a timeout value; the request ID of the event is returned. To cancel a timer event, the component sends a *cancel timer request* with the user ID and the previously returned request ID of the timer event.

• **State coordination with clients.** Since request IDs are added (and removed) at the entry point, unlike components, client devices cannot rely on them to reliably receive correctly ordered responses. A failed state update at a component may produce a message that is sent to a client, and a retry may produce another copy of the same message. This must be handled by the entry point. Each client-bound reply is labeled with a request ID and the sequence number of the message within the request. The entry point ignores replies that have already been forwarded to the client. Retries always produce the same responses, but it is important that one and only one gets forwarded. Necessarily reliability means the client and the entry point can be expected to maintain a single, ordered, reliable connection (e.g., TCP connection), which safely deals with message loss on the last hop as long as the entry point correctly orders replies.

• **Handovers.** Occasionally, a client moves between two eNodeBs and requires a handover. Another entry point must handle the client's operations, so state must be transferred between the entry points. This state is very light; it consists of the contents and ID of the last processed request and any registered timers or control packets. The handover procedure is augmented so the old entry point sends the context to the new entry point, which becomes the anchor for the client once the procedure is finished.

5.4.4 Nonblocking cloud components

Given the requests with monotonic request IDs, ECHO needs to guarantee atomicity and in-order execution properties on each component *and* across components. ECHO's operation is different from distributed ACID transactions because it enhances *both* atomicity and in-order execution. Also, ECHO cannot simply use existing state machine replication techniques [134] because ECHO's operation is not atomic and deterministic; components induce side effects on other components, making determinism hard to guarantee.

Algorithm 2 shows how an ECHO component processes a request. Note that the algorithm describes two types of components, with and without side effects (as explained in §5.2.2), in a single algorithm. The algorithm is designed to dovetail with required processing in conventional EPC components; the red lines (14, 17, 18, 19) already exist in EPC components. Note, the algorithm is *nonblocking*; multiple stateless instances of a component can execute the algorithm in parallel without causing any stall on other instances.

• **Component's atomicity.** Replication of components in ECHO and retries from the entry point mean that a single request could be processed by multiple instances of the same component. To prevent inconsistency caused by interleaved processing of the same request

Algorithm 2 Nonblocking cloud component

Input event: Receive a request from eNodeB's entry point (agent) *R*, with UE's ID (*R.UE*) and request ID (*R.ID*).

Output event: Send reply and timeout message to eNodeB's agent.

```
1:  Fetch session from storage: (session, version) = read(R.UE), where version is version number of znode.
2:  if session not found in storage then
3:      Create a session locally. Set session.ID = R.ID
4:      Go to step 14.
5:  end if
6:  if $R.ID < session.ID-1$ then
7:      {Received an obsolete request}
8:      Return
9:  end if
10: if session.reply and session.timer exist then
11:     (Re)send session.reply and session.timer
12:     Return.
13: end if
14: Update session.
15: Increment request ID: session.ID += 1
16: Set request ID in side effect msg: session.side_effect.ID=R.ID.
17: Send side effect message: session.side_effect.
18: Receive side effect reply.
19: Update session.
20: Prepare reply message: session.reply, set request ID in reply message session.reply.ID = R.ID
21: Prepare timeout message: session.timer, set request ID in timeout message session.timer.ID = R.ID
22: Write session to storage: write(session, version)
23: If write OK: Send reply and timeout messages: session.reply, session.timer
```

across instances, ECHO uses atomic conditional writes provided by the persistent storage (we discuss our persistent storage implementation in Section 5.5). When committing changes to the reliable storage (line 22 in the algorithm), each component instance ensures that the stored UE context (session) remained unmodified while it was processing the request by checking the *version* number of the session. If the conditional write fails, then another component instance has already processed the request, so this instance discards the local session state and backs off. This assures that, even though multiple component instances can process the same request, only one instance is able to commit the changes at step 22, guaranteeing atomicity.

- **Component's monotonicity (in order execution).** Each component in ECHO needs to execute requests in the order that they arrive at the entry point. However, concurrent retries of a request issued by the entry point can cause processing of an obsolete message at a component instance. Without care, this could cause the state in the session store to regress, leading to inconsistency, as illustrated in example 5.5 in §5.2.

A component in ECHO uses the monotonic request IDs to filter out obsolete requests. As in line 6 in the algorithm, before processing a request, the component instance checks if the request ID of the request is less than the last executed request ID (which is stored in the persistent storage). If it is, then the request is obsolete and is discarded. When updating the persistent storage, the component increments the request ID of the session (line 15) and acknowledges the request ID to the entry point (line 20).

For example, in example 5.5, the stale Detach Request at *11:04:45* would have been discarded as its request ID would have been lower than the request ID of the Attach Request that is last processed at *11:03:58*.

• **ECHO's atomicity and monotonicity.** Given each single component operates atomically and in order as described, ECHO needs to ensure atomicity and in order execution *across* its distributed components.

A *side effect* is triggered when one component processes a request that generates a message to another component. Consistency must be maintained across components despite side effects, but retries from the entry point can create multiple *duplicated* side effect requests, and slow instances can generate *stale* side effect requests. Without care, duplicated and stale side effect requests could cause inconsistency.

Service Requests example illustrates the inconsistency that can arise from duplicated side effect requests. Suppose an MME instance *A* receives a Service Request. In step 17 of algorithm 2, it sends a *Modify Bearer Request* (request #1) to the SGW component. An SGW instance receives the request #1, creates and installs a tunnel endpoint *TEID1*, stores it in persistent storage and replies to the MME with the information. Meanwhile, suppose that the entry point times out and retransmits the Service Request. Another MME instance *B* receives the retry and sends a *duplicated Modify Bearer Request* (request #2) in step 17. Later an SGW instance receives the request #2, and it *overwrites* and replaces *TEID1* with a new tunnel endpoint *TEID2* and replies. The MME component ignores the second reply because it already moved to a new state when the first reply arrived. In the end, the MME component (and the UE) contains *TEID1* while the SGW records *TEID2*; this inconsistency breaks the data plane.

To keep multiple duplicated side effect requests from mutating component state, retries of a side effect must induce the *same* effect on the target component (i.e., side effects must be idempotent). Algorithm 2 enforces this. When a message is processed, the response is recorded in the session store with its corresponding request ID, so lost responses can be reproduced without repeating execution. If an instance receives a request and the committed session in the persistent storage contains a reply, then another component instance has already executed the transaction, and it only needs to reply (lines 10, 11, 12). Since responses are recorded in the persistent storage, they can be obtained by other instances, in case the current instance crashes before replying.

To solve the inconsistency problem caused by stale side effect requests, a component also passes the request ID of received requests to the side effect requests it generates (line 16). The target component then ensures the side effect requests are executed in the order specified by the request ID. This happens at *every* ECHO component, so no stale side effect requests are processed.

5.4.5 Correctness

Here we give a sketch of why ECHO is safe even though components are redundant and nonblocking under failure. Showing that ECHO *appears* to process operations atomically, in mobile device's FIFO order, one-at-a-time demonstrates safety.

First, we show that a leaf component (a component that does not trigger side effects to other components) operates linearizably (in an order consistent with some total order of the client operations). Next, we show that the total order it is consistent with is the client's FIFO order. Then, using leaf components as the base case, it can be shown that all components appear to process operations atomically in client FIFO order.

Lemma 1 (Leaf Instance Linearizability)**.** *Each leaf component instance appears to process requests atomically in an order consistent with client invocation and response.*

• **Proof.** Figure 5.10 shows the processing steps of a component instance handling a request. The compare-and-swap on the shared storage acts as a linearization point: before

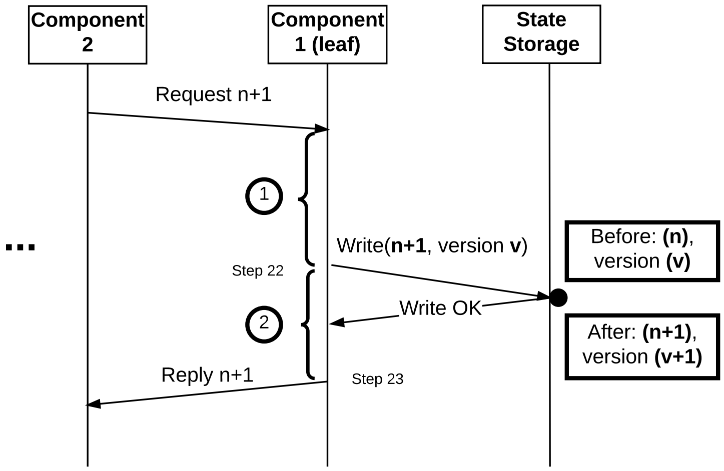


Figure 5.10. ECHO's leaf component is linearizable

it no (local) effect of the component instance can be perceived, after it all its (local) effects are guaranteed to persist. Each processed request with ID $n+1$ results in one of four outcomes: 1) *aborted* – the component is not in state n or $n+1$, so the request is invalid and ignored; 2) *successful* – the operation completes successfully, the component instance updates the state storage, moves the component from state n to $n+1$ at step 22 and replies; 3) *crashed before update* – the operation fails in ① before updating the state leaving the component in state n; or 4) *crashed after update* – the operation fails in ② after updating the state leaving the component in state $n+1$.

In case 3, because of the eventual completeness of the entry point, there must be a retry arrived at another instance that progresses to either case 4 (in-completed) or case 2 (completed). In case 4, when a component instance receives a retry, it simply replies with the recorded reply which eventually results in case 2. Therefore, a component only executes requests in the specified order, either completely successful or completely failed. □

Because each compare-and-swap is issued from a natural number n to $n+1$, only one instance of any component can achieve a successful outcome in the above proof, so Lemma 1 can be strengthened:

Lemma 2 (Leaf Component Linearizability). *Each leaf component appears to process requests atomically in an order consistent with client invocation and response.*

Finally, because agents only issue request $n + 1$ after the successful acknowledgment of request n (that is, operations are synchronous), no component instance can attempt to apply $n + 1$ to shared storage while the state in storage is tagged with a request number less than n. This strengthens Lemma 2 to:

Lemma 3 (Atomic FIFO Leaf Components)**.** *Every leaf component in* ECHO *processes requests atomically in the (FIFO) order client issued them to the agent.*

Given Lemma 3 as a base case, components that make nested calls to other components can be shown to be atomic and FIFO as well using induction.

Lemma 4 (Atomic FIFO Components)**.** *Every component in* ECHO *processes requests atomically in the (FIFO) order client issued them to the agent.*

• **Proof.**

As shown in Figure 5.11, nonleaf components are identical to leaf components except that they may send requests and wait for responses just before attempting to update shared storage. Let M be the *height* of a component, which is the number of nested requests that must succeed before a leaf component is reached. Leaf components have $M = 1$.

Induction hypothesis: Assuming a component at height M operates atomically in client FIFO order, then a component at height $M + 1$ operates atomically and in client FIFO order.

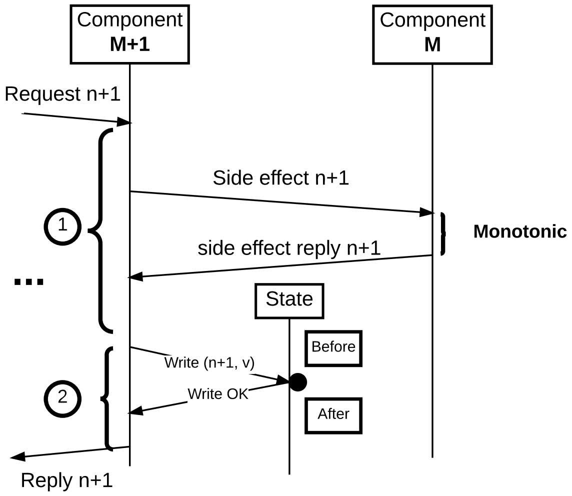


Figure 5.11. ECHO is linearizable

Base case: Lemma 3 proves the case for $M = 1$.

Consider a request arriving at a component at height $M + 1$. Similar to the leaf component proof above, the request has 4 outcomes: *1-aborted, 2-successful, 3-crashed before update* and *4-crashed after update*. The *crashed after update* outcome eventually results in the *successful* case as in the proof in lemma 2. In the *crashed before update* case, the component instance crashes in ① and leaves the component $M + 1$ in state n. Eventually, there is a retry $n + 1$ from the entry point that arrives at another component instance and triggers the side effect again. If there are multiple retries that trigger multiple side effects to component M, the effect on component M is still atomic and in client FIFO order by the induction hypothesis. Therefore, case 4 eventually results in case 2 or 3 (which eventually results in case 2). Therefore, component $M + 1$ operates requests atomically in client FIFO order. □

Finally, since the ordinary, unreplicated protocol precisely processes messages atomically, in mobile device's FIFO order, one-at-a-time, this gives the essential safety property:

Property 1 (ECHO Safety Property)**.** *The set of states observed by ECHO clients is equivalent to the unreplicated protocol.*

5.5 Implementation

Section 5.4 outlines general design principles ECHO uses to provide safety and reliability. Here we discuss specifically how this design applies to a cellular control plane and a public cloud. The summary of changes to the standard EPC architecture is illustrated in Figure 5.12.

5.5.1 ECHO agents

ECHO's agents are lightweight software proxies that provide entry-point functionality on eNodeB and an interface between eNodeB and MME. There are two agents, one that resides on the eNodeB and one on the MME, as illustrated in Figure 5.12. The eNodeB's agent is implemented as a separate user-mode daemon written in standard C, deployed on top of embedded Linux running on a commodity small cell [82]. This allows us to easily port it to any COTS eNodeB without affecting the time-critical LTE radio code. The MME's agent is integrated in the source code of the S1AP processing module of OpenEPC [55].

One of the agent's functions is to proxy S1AP control messages. 3GPP eNodeB and

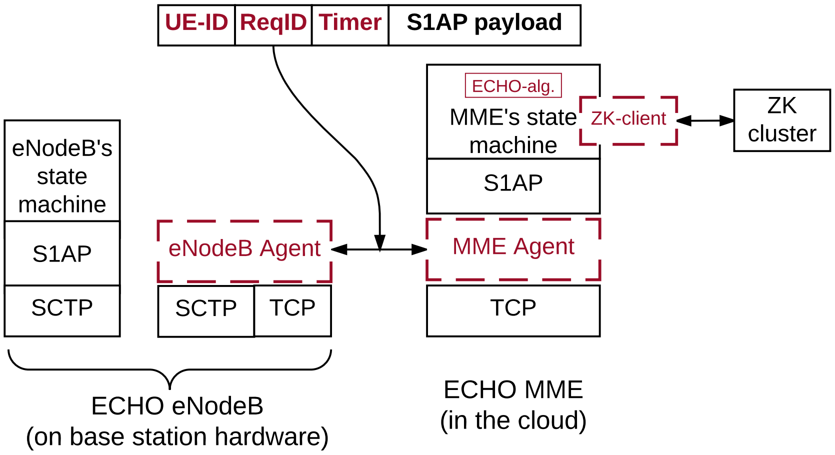


Figure 5.12. Modifications of ECHO in LTE/EPC

MME use SCTP protocol for S1AP messages. However, Azure and other public clouds do not support SCTP protocol, so we implement a proxy agent that replaces SCTP by TCP. The ECHO agent on the eNodeB opens an SCTP connection to the rest of eNodeB software stack on one side (which is unmodified and unaware of the agent's existence) and a TCP connection to an ECHO MME agent on the other side. The eNodeB agent relays messages between the two connections. The agent reestablishes the TCP connection on a failure, in order to attach to a new MME instance (in the same DC or a different DC).

Furthermore, the agent implements the entry point design, described in Section 5.4.3. The agent adds an extra network layer (ECHO or agent layer) into the LTE/EPC control network stack, as shown in Figure 5.12. The ECHO layer header consists of the *Request ID*; a *UE-ID*, a unique identifier of the UE, composed of tunnel identifiers readily available from S1AP messages; and a *Timer value*, used to set up timers and to inform components about timer expiry.

5.5.2 Stateless EPC components

We have augmented the most important EPC components (MME, SGW and PGW) in OpenEPC [55] with ECHO functionality. In the example of MME, our implementation preserved the original implementation that extracts information from a received S1AP message, generates side effects and updates the client's state (e.g., steps 14, 17, 19 in the algorithm). We extended handlers to accept request IDs from the ECHO layer and to add duplicate/stale request checks that adapt processing accordingly (step 6). When the original MME code finishes processing a request, ECHO sends an acknowledgment to the eNodeB agent together with a S1AP reply. We made SGW/PGW operations idempotent by making the SGW reply with a stored message (i.e., with the same bearer information) for duplicate requests from the MME (so, the duplicates don't forward effects to the PGW). In all, ECHO's extensions to OpenEPC required changing 1,410 lines in 12 files.

We added two additional blocks to the conventional EPC: an *agent* (described previously) and a *ZooKeeper client* (ZK-client). The ZK-client provides a *read/write/delete* interface to a ZooKeeper [81] (ZK) cluster that acts as a reliable, persistent storage. ZooKeeper is a reasonable choice of storage because of its consistency guarantees, small amount of stored information (a few KBs per UE context) and relatively low request rate. The UE context (which is extended to include UE replies) is stored as a binary string in a znode in ZK. ECHO uses the *version number* of a znode in ZK to realize an atomic state update at step 22 of the algorithm; ZK only allows updating the znode if the version number hasn't changed since the beginning of the request.

5.5.3 Cloud deployment

Multiple instances of the same component are deployed in a private network in Azure behind a load balancer. The load balancer performs consistent hashing on the connection's 5-tuple, so a connection sticks with the same instance unless there is a failure or a new instance is added. When ECHO is deployed across multiple data centers, requests that time out a few times are retransmitted to another data center by the ECHO agent on the eNodeB.

5.6 Evaluation

We evaluate ECHO in the Azure public cloud across several dimensions. In particular, we examine the correctness of our implementation, the potential latency introduced across various components of the architecture, the observed throughput and simulate potential failure scenarios. Our main findings can be summarized as follows:

• ECHO introduces reasonable overheads as a trade-off for a public-cloud reliable deployment. When replication within a single data center is used, the response latency is increased by less than 10% and there is no visible drop in throughput. Even in more extreme deployments, we show that total latency is well below standard 3GPP timeouts and would not be noticeable by the users. The result shows user-perceived latency is similar in ECHO and T-mobile.

• By emulating typical data center failures, we show that ECHO gracefully handles all such cases without noticeable user experience impact.

• **Evaluation setup.** Our base deployment is given in Figure 5.13. It consists of radio equipment (a UE - Nexus 5, LTE eNodeB - IP.Access small-cell) in PhantomNet [51] and an EPC core (MME pool with 2 MMEs, a ZooKeeper ensemble with 3 nodes, and other EPC components - SGW, PGW, HSS) in Azure. We also use a local OpenEPC deployment in PhantomNet to compare ECHO's performance.

• **Reliability options.** We consider two availability options. One is a single data center deployment, in which all ZooKeeper (ZK) nodes are collocated in the same data center. The other is ZooKeeper deployment across multiple data centers, as depicted in Figure 5.13. The network latency between the eNodeB and Azure is around 22 ms round-trip. The 3 Azure DCs are 20 ms round-trip away of each other. A single DC deployment provides less reliability but also lower latency than a multi-DC deployment. We evaluate both of them as both can be relevant for different application scenarios.

The reliability also depends on ZooKeeper operational parameters. We evaluated three ZK logging configurations: *synchronous disk* (Disk), *asynchronous disk logging* (Disk-nFC, no

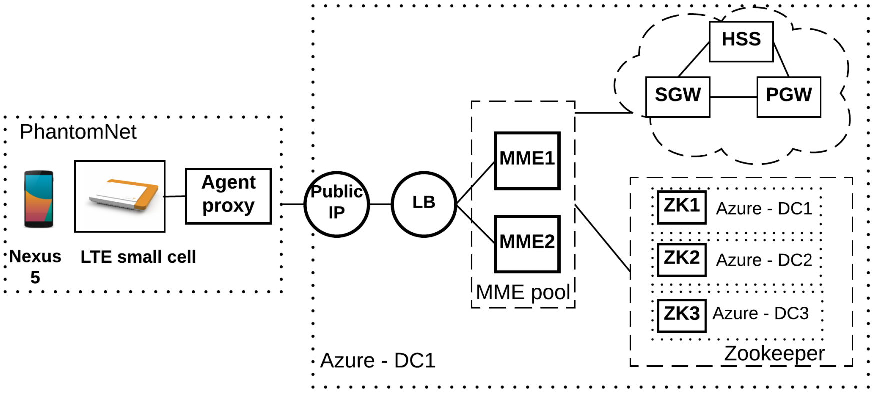


Figure 5.13. ECHO evaluation topology setup

force sync) and *logging to ramdisk* (Ramdisk). Synchronous disk logging is the most robust and quickest to recover, but introduces the most latency. Ramdisk and Disk-nFC (log to disc but don't wait before acknowledging) are two trade-offs that reduce latency but also slightly reduce the ability and speed of recovery. Table 5.2 shows the deployment options and failure scenarios that they can tolerate. Disk-nFC and Ramdisk configurations have smaller latency while 3DCs cloud deployment could tolerate 1 DC failure. We compared ECHO with OpenEPC which stores UE context in memory. We also compared user perceived performance of ECHO and T-mobile. We introduced node crashes to the prototype and illustrated ECHO is robust against failure events.

• **Correctness.** We deployed ECHO on one Azure data center and ran it for 7 days. We generated 6,720 Service and Context Release requests (20,160 messages) from a Nexus 5 device attached to a eNodeB. The system remained stable and all requests were correctly processed. We next randomly introduced node reboot and process crash events on 1% of control messages; ECHO recovered from crashes and all messages were correctly processed.

• **Latency.** Figure 5.14 shows latency of entire Attach (top) and Service Request (bottom) procedures with different ZK configurations running in DC. The latency is broken down into EPC core network - the latency between EPC components (including ZK); Network time - network round-trip time between eNodeB and Azure; and Radio - latency to set up radio bearers on UE and eNodeB hardware. Overall, ECHO introduces about 7% (70ms) more latency for Attach compared to OpenEPC which stores UE context in memory, which is almost negligible. The overall latency is dominated by radio bearer configuration between UE and eNodeB.

• **Individual message overheads.** Figure 5.15 shows the latency overhead ECHO

Table 5.2. ZK configurations and cloud deployment options in ECHO evaluation with their latency and reliability profiles

Option	**Latency**	**Robust against failures**		
		Node	**Avail. Zone**	**DC**
OpenEPC	Low	No	No	No
1DC,Disk	Moderate	Yes	Yes	No
1DC,Disk-nFC	Low	Yes	Yes	No
1DC,Ramdisk	Low	Yes	Yes	No
3DCs,Disk-nFC	High	Yes	Yes	Yes

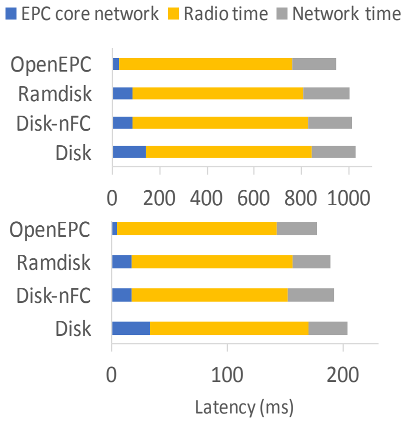


Figure 5.14. Latency overhead of ECHO: Attach (top) and Service Request (bottom) procedures latency on 1DC deployment, observed on eNodeB

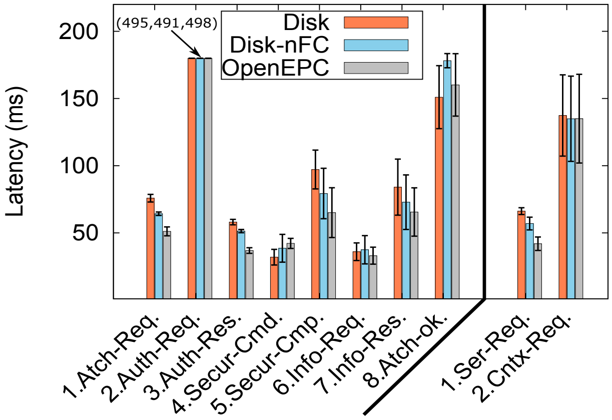


Figure 5.15. Latency overhead of ECHO: Latency of each individual message in an Attach (left part) and Service Request (right part) procedures on 1DC deployment

introduced to each message exchanged between UE and MME in an Attach (left part) and Service Request Procedure (right part). The odd-numbered messages (1-Attach Request, 3-Authentication Response, 5-Security Mode Complete, 7-UE Information Response, 1-Service Request) are sent by the UE and are processed by ECHO. The even-numbered messages (2-Authentication Request, 4-Security Mode Command, 5-UE Information Request, 8-Attach Accept, 2-Context Setup Request) are sent by ECHO and processed by the UE. As shown, this confirms radio setup and authentication processing on UE (msgs. 2-left,8,2-right) dominate total procedure latency. Looking at ECHO latency (i.e., msgs. 1-left, 3, 5, 7, 1-right) we can see a clear latency overhead trend among ECHO-Disk,ECHO-nFC and

OpenEPC. Overall, using disk logging incurs the most latency overhead while using disk without force sync (Disk-nFC) incurs less latency. The overhead ECHO introduced per message is noticeable (about 40%) but the overall overhead is small.

• **Reliability vs. latency trade-off.** Figure 5.16 shows latency of an Attach Procedure with ZK deployed in a single DC and 3 DCs. ECHO with multiple-DC deployments will survive DC failure (Table 5.2) yet incur higher latency because of network latency between ZK nodes (40% or 400ms more for attach procedure). Depending on the reliability characteristics required, one might favor an option as a tradeoff of the associated latency costs. For example, public Internet outages can simply be relayed from reachable data centers if this is a viable option for a particular deployment. However, even with the most extreme deployment, ECHO incurred overhead is still tolerable for UE operating 3GPP protocols. We further probe into this by showing a CDF of the latency of each ZK write (Figure 5.17) and each message on ECHO MME in an Attach procedure (Figure 5.18). Replication to 3 DCs can incur 10X messaging latency as it may invoke several ZK writes. Yet, this is still only a fraction of the total latency and well below the smallest timeout value of a UE – 5s for T3417 (see section 10.2 in 3GPP NAS timers [20], 3GPP S1AP timers [22].) In future, this could be improved by using closer data centers or closer integration of a consensus protocol into Algorithm 2 to reduce the number of writes.

• **UE-perceived latency.** Figure 5.19 shows the latencies of Attach and Service Request

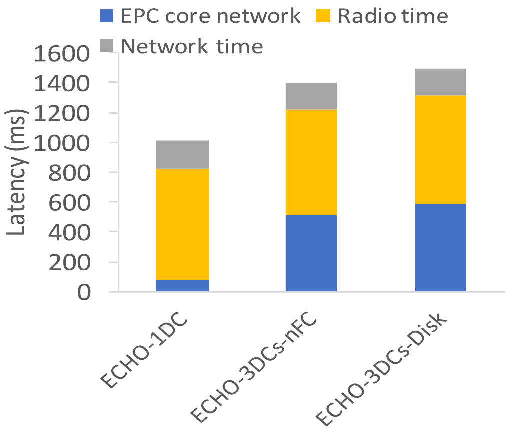


Figure 5.16. Latency overhead of ECHO: Latency of each individual message in an Attach (left part) and Service Request (right part) procedures on 1DC deployment

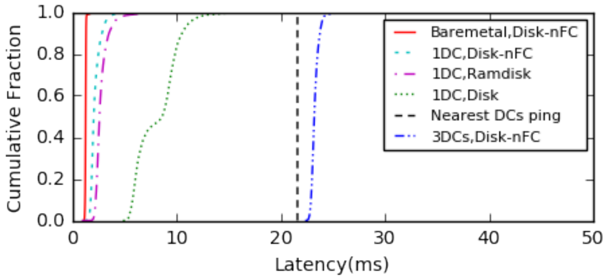


Figure 5.17. Network latency CDF for ZooKeeper write. Baremetal shows optimal, nonvirtualized performance

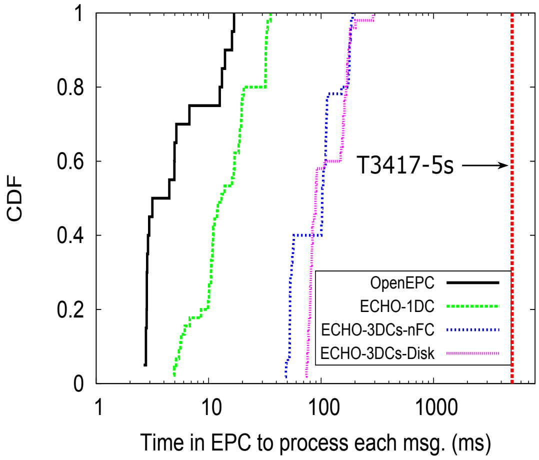


Figure 5.18. Network latency CDF for attach procedure

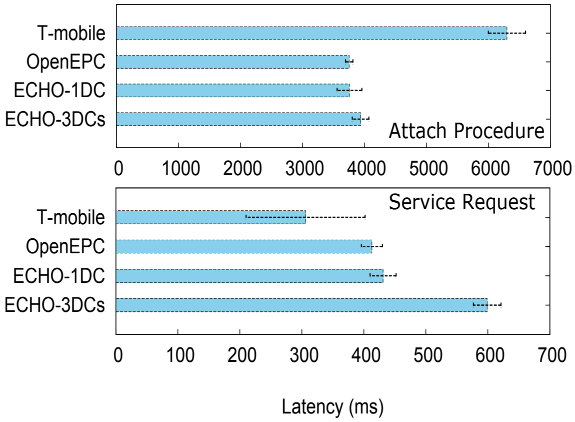


Figure 5.19. UE-perceived latency for attach procedure (top) and UE-perceived latency for service procedure (bottom)

procedures perceived by a UE on ECHO and T-mobile. Since we can't capture T-mobile control messages inside their proprietary EPC deployment, we measured the latency by triggering Attach and Service Request on the Nexus 5, using the same methodology on both platforms for a fair comparison. To trigger an Attach we toggled airplane mode in the Nexus 5. To trigger an Server Request we let the device idle to make sure it releases radio connection, and triggered a Ping request from it to the Internet. We then measured the time it takes for the Nexus 5 to be network-available (from the trigger-time to the first Ping packet gets through.) As these latencies include phone-level overheads, they correspond to end-user perceived latencies, and they are much larger than network measured latencies (Figure 5.14). Overall, ECHO control procedure latency on one DC is comparable to T-mobile, but worse on 3 DCs. However, control events are infrequent, so we expect that this will not affect end-user experience.

• **Throughput.** Figure 5.20 shows ECHO's peak throughput on 1DC and 3DCs for Attach and Service Requests. ECHO throughput is comparable to OpenEPC. Even though the throughput does not look very high, notice that each procedure consists of multiple messages exchanged (e.g., 8 messages for an Attach), and it is comparable with throughput reported in other works [35].

• **Failure scenarios.** Figures 5.21 and 5.22 show OpenEPC and ECHO operation when an MME crashes. We synthetically introduced a crash to both OpenEPC's MME and ECHO's MME. The UE attached to OpenEPC was not able to use the network for 54 minutes because of the crash. The MME lost the UE Context therefore following Service Requests from the UE failed (denoted with red crosses). The service resumed 54 minutes later after a periodical Tracking Area Update (see Section 5.3.1 for a detailed discussion of the example). The UE attached to ECHO continued to have service throughout the MME crash event (note that the 1st attach request on MME2 experienced a slightly higher latency because MME2 had to contact ZK for the UE's Context). This illustrates the advantage of ECHO over OpenEPC in term of reliability against node crashes.

• **eNodeB client.** We deployed our eNodeB's ECHO client implementation on IP Access E40 eNodeB [82] as a user-mode daemon. We configured four mobile nodes to perform data transfers and then sleep over periods of 1 minute, generating 8 requests per minute. A typical small cell can support up to 64 active users so this is an expected number

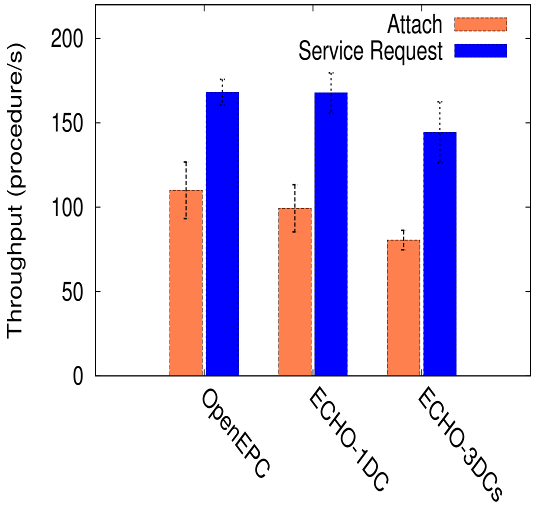


Figure 5.20. Throughput of attach and service request

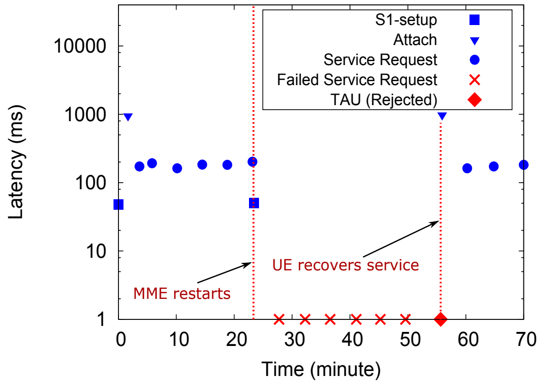


Figure 5.21. Unmodified OpenEPC MME crash results in an outage

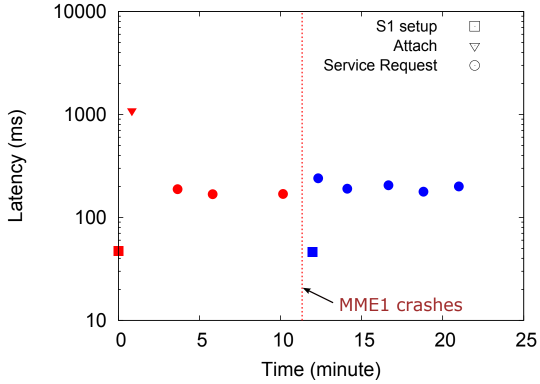


Figure 5.22. ECHO MME crash avoids outages

of requests to get served. The induced CPU load was not noticeable on the embedded Linux monitor.

5.7 Related work

Our work is related to efforts in network function virtualization in general [60, 62, 69, 149, 150], as well as more closely related virtualized mobile network efforts focused on resource management and scalability [85, 126, 128], orchestration of virtualized core network functions [142], and virtualizing specific core network components [37]. Perhaps most closely related to ECHO are the virtualized MME architectures proposed in SCALE [35] and DMME [30]. SCALE and DMME proposed to horizontally scale the MME using load balancing and state replication. However, SCALE and DMME focus only on scalability of a single (MME) component. They do not deal with reliability issues – if an MME instance is slow or crashes, stale requests could cause state inconsistencies.

Various studies have dealt with availability and reliability concerns of cloud platforms [39, 40, 71, 73]. Alternative approaches to our work to address these concerns include mechanisms to make clouds inherently more reliable [143], service abstractions to hide the complexities of dealing with cloud failures from application developers [83] and attempts to add specialized cloud features to deal with cloud fault tolerance [118, 120]. ECHO took a different approach to assume the cloud infrastructure is not reliable and instead used software and protocols to enhance availability.

ECHO's replication strategies relate to state machine replication (SMR) [134], a well-known approach to building fault-tolerant, highly available services [46, 81]. However, as in § 5.4.4, naively reimplementing MME logic in replicated state machines does not work. It also intertwines scaling, partitioning and fault-tolerance, since state machines are stateful. SMR plays a role in ECHO, but in the form of ZooKeeper's [81] fault-tolerant atomic broadcast protocol, Zab [86].

ECHO's enforcement of FIFO and atomicity is similar to virtually synchronous CBCAST from the ISIS toolkit [41]. However, ECHO is the first to combine atomic and FIFO processing over distributed components in a cellular network leveraging the reliable base station. The key challenge is in minimizing changes to the existing EPC protocol and in interactions with the outside UE, which cannot be modified. Others observed this issue with clients in other contexts [93]. The necessary reliability between the UE and its eNodeB simplify this, since the radio control link offers a reliable, ordered connection with the UE. Setty et al. [137] proposed "locks with intent" for building fault-tolerant systems on cloud storage. In ECHO, each client only affects its own state, which eliminates the need for intents.

5.8 Conclusions

Virtualization of cellular core network protocols onto a public cloud introduces new and different challenges. This chapter presents ECHO, a scalable and reliable architecture that can easily be implemented on top of existing core networks. ECHO is proved correct and significantly improves the reliability of mobile core networks with minimal overheads when deployed in a public cloud.

CHAPTER 6

CONCLUSION

6.1 Summary of the dissertation

The next generation (5G) mobile network is expected to host emerging use cases that have a wide range of requirements; from Internet of Things (IoT) devices that prefer low-overhead and scalable networks, to remote machine operation or remote health-care services that require reliable end-to-end communications. However, the (4G) mobile core network has several drawbacks in terms of enabling new technologies and supporting new applications. On the other hand, SDN and NFV are the key technologies that improve flexibility and agility in fixed networks. Applying SDN/NFV together with big-data analytics and distributed system techniques could enable a **flexible, low-overhead, and reliable mobile core network.**

This Dissertation focused on improving scalability and reliability of the mobile network for future applications. The Dissertation presented SMORE [52] and SIMECA [109] which use Software Defined Networking (SDN) and Network Function Virtualization (NFV) to enhance scalability of the mobile core network for a massive number of Internet-of-Things (IoT) devices. It then presented ABSENCE [110] which uses big-data analytics and a novel failure detection technique to detect failures in an operational mobile network. Lastly it presented ECHO [111] which uses distributed system techniques to enhance availability of software-based mobile networks in public clouds.

Chapter 3 focuses on a flexible, low-overhead and scalable mobile core network. 4G mobile core networks are largely based on proprietary and expensive hardware components that limit both new functionalities as well as low latency applications. SMORE is an SDN-based offloading architecture for LTE/EPC network. Using an SDN infrastructure and an SDN controller, together with a proper monitoring mechanism, SMORE offloads a portion of LTE traffic to a local edge cloud to support low latency applications. The challenges of SMORE are to find a practical solution given the current deployment of the mobile network and to be transparent to existing LTE traffic. SMORE intercepts data and control plane traffic at Mobile Telephone Switching Office (MTSO). It monitors the LTE control channel to extract data plane information used in data plane offloading. It offloads registered traffic to the mobile edge cloud and leaves normal LTE traffic untouched.

In terms of network architecture and protocols, the current EPC core network was designed to support a small number of devices streaming high-quality traffic (e.g., a smartphone streaming a Youtube video) rather than a large amount of devices sending small and sporadic traffic (e.g., a massive number of IoT devices sending a few bytes every 15 minutes).

SIMECA is an SDN-based mobile edge cloud architecture that provides a new network service abstraction that is more suitable for IoT devices. Unlike the control and data planes of LTE/EPC that are based on IP tunnels to realize end-to-end communications with QoS supports, SIMECA's data plane is best-effort to reduce overheads. SIMECA's control and data planes are more light-weight (up to 20%/37% lower data/control overhead respectively) compared to LTE/EPC's. Removing IP tunnels in the mobile core, however, raises a challenge of how to support mobility in SIMECA. Moreover, designing data and control planes that are more light-weight while supporting mobility is not straightforward. SIMECA proposed an SDN-based mobile edge cloud infrastructure that is more distributed and SDN-controlled. To reduce control overhead, SIMECA classifies packets at the edge (base stations) while core network switches simply forward packets based on the packet header (i.e., smart edge - dumb core principle). At the base stations, packet headers are translated to prevent adding any extra overhead to the data plane. A device in SIMECA has a device ID used for end-to-end applications and a routing ID used for routing and mobility. This identity separation mechanism together with a routing ID tracking mechanism allow seamless mobility in SIMECA.

In order to improve service experience and availability of mobile networks, network operators have already deployed multiple monitoring systems to detect service disruptions and fix problems when they occur. However, detecting all service disruptions is challenging. Somewhat counter-intuitively, monitoring network components is insufficient to indicate user experience because of a complex relationship between the network status and user-perceived service experience. Moreover, service disruptions could happen because of reasons that are beyond the network itself, e.g., bugs on the customer's device.

Chapter 4 presented ABSENCE, a passive service monitoring system that detects service disruptions by monitoring customer usage. The main idea in ABSENCE is that customer usage, or lack thereof, is a reliable indicator of service disruptions or network problems. Using users' Call Data Records (CDR), ABSENCE obtains a normal usage pattern of an aggregation of customers and infers an anomaly if the current usage is less than expected. ABSENCE uses Map-Reduce and a large compute cluster to extract the real-time usage of different aggregations of users from Tera-Bytes of CDR generated constantly in the network. It then uses the time series decomposition technique to decompose the usage time series and detect anomalies. ABSENCE was able to detect failures with up to 95% accuracy. It also detected real failures that went under the radar in an operational network.

With technology advancements in SDN and NFV, next generation mobile networks are expected to be NFV-based. However, in stark contrast to telecom-grade hardware with built-in redundancy, commodity Off-the-Shell (COTS) hardware in NFV platforms often can't compare in terms of reliability. Availability of Telecom-grade mobile core network hardware is typically 99.999% (i.e., less than 5 minutes of downtime per year) while most NFV platforms only guarantee 99.9% availability (less than 5 minutes of downtime per 3 days). Therefore, an NFV-based mobile core network running on a less-reliable infrastructure requires extra mechanisms to guarantee its availability.

Chapter 5 presented ECHO, a distributed mobile core architecture that solves the availability problem for NFV-based mobile core networks. ECHO preserves the mobile core protocols as they are complex and constantly evolving. ECHO adds extra abstractions on top of the mobile core network to compensate for the fact that nodes in an NFV-based infrastructure could be arbitrarily slow or crashed, and messages could be lost or delayed. ECHO's reliability is based on a *necessarily reliable* agent at each base station that serializes requests and keeps sending a request until it receives a reply. Mobile core components are replicated and their state is stored in a replicated and highly available key-value store. ECHO then uses a nonblocking algorithm to ensure only one stateless instance can modify the control state, and stale requests are blocked to avoid state inconsistency. The algorithm is applied on all components in the mobile core in a recursive manner to achieve an atomic state change across distributed components as if they were on a single reliable component. ECHO was deployed on Microsoft Azure cloud peering with an LTE small cell and a mobile smart-phone in PhantomNet [51]. ECHO adds no significant overhead to the current mobile core network and is more reliable when multiple failure events were introduced.

6.2 Future research directions

This dissertation presented a study on the scalability and reliability aspects of the mobile network. Given the huge shift of the network to the next generation (5G) architecture, there are still a lot of topics to explore. This section gives a few research directions that take into account expected changes to future applications and the network infrastructure to support future requirements. The section first offers some insights about the interaction between applications and the mobile network, and then explores further possible improvements to the architecture and protocols of a software-based mobile networks in a distributed Mobile Edge Computing (MEC) infrastructure.

6.2.1 Bridging the gap between applications and the mobile network

In the current mobile architecture, the end-point device (and applications running on it) treats the network as a black-box that can't be tailored to meet the applications' requirements. However, unlike wired networks, performance of mobile networks largely depends on the radio link performance and the protocol configuration at the base stations. A single wireless configuration is not likely to satisfy the wide range of requirements of all types of applications. For example, a previous work [108] pointed out that real-time interactive applications such as VoIP might prefer seamless handover while bulk-transfer applications such as web-browser or ftp might prefer lossless handover.

Bridging the gap between applications and the network allows applications to configure the network (or the network stack), make use of network information, or interact with the network to satisfy their specific requirements. For example, knowing the current load of the base station could help an application to adjust its sending rate to prevent harmful spurious TCP timeouts [108], or a police body camera application could request the network to reserve a prioritized connection that has enough bandwidth for a crime-site video-streaming when needed.

To enable this, we need APIs to expose network's functionalities, network configurations and the end-point device's network stack. The network's functionality API exposes interfaces to manage resources in the network (e.g., set up or reserve bandwidth of a connection). The network's configuration API enables network control/data plane configuration such as handover type, Radio Link Control (RLC) queue size, RLC modes, etc. The network stack API allows the application to configure the device's network stack such as changing TCP's congestion window, increase TCP's Retransmission Timeout value, etc.

6.2.2 Architectural and protocol-level improvements for NFV-based mobile edge computing

Future mobile networks are expected to be NFV-based. However, moving from a hardware-based mobile network to a software-based solution invalidates many existing assumptions and raises multiple challenges. One challenge is that unlike telco's specialized high-performance hardware, commodity OTS hardware in an NFV platform can't meet the reliability and scalability requirements. For example, a VM clearly can't store the state and serve requests from millions of users as a more powerful hardware box could. Moreover, mobile edge computing makes the network infrastructure become more distributed. Exploring trade-offs in this new infrastructure is an interesting topic. For example, a distributed EPC instance at the edge would have less load per instance (and thus the need for an extreme scaling of the distributed EPC is reduced), but in the meantime the distributed EPC instances can only cover a smaller mobility area.

Firstly, one way to improve reliability of NFV-based mobile networks is to add extra abstractions to the existing LTE/EPC protocols [111]. Another way to enhance reliability is to redesign the protocols and take into account the new NFV's assumptions. For example, given an unreliable NFV infrastructure, a "good" protocol on the end-device should *proactively* check for state inconsistency and fix the state immediately instead of waiting for a periodic Tracking Area update and reattach [111]. Secondly, to scale the NFV-based mobile network given relatively small compute instances, a horizontal scaling approach would help. However, the approach should take into account the architecture of the mobile core and how the components interact. Interesting questions, for example, are: (1) When scaling the data plane, how do the network instances store their state in a scalable manner? (2) How to update the state in the data plane when there is mobility. Lastly, to balance the performance and overhead of a distributed EPC architecture in MEC, a hybrid solution might help: a local EPC could be used to improve performance, while a centralized EPC could be used for a portion of users (or connections) that requires a high degree of mobility. This approach needs a proper separation of functionalities of the distributed EPC and the centralized EPC and how they interact.

REFERENCES

[1] 3GPP 23.829: Local IP Access and Selected IP Traffic Offload (LIPA-SIPTO). http://www.3gpp.org/DynaReport/23829.htm.

[2] Cisco Fog Computing. http://www.cisco.com/c/en/us/solutions/internet-of-things/iot-fog-computing.html.

[3] Cisco White Paper 2015-2020. http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf.

[4] DiffServ – The Scalable End-to-End QoS Model (White Paper). https://tinyurl.com/yat8yj84.

[5] Gomez, inc. website. http://www.gomez.com/.

[6] Google's Marissa Mayer: speed wins. http://www.zdnet.com/article/googles-marissa-mayer-speed-wins/.

[7] High latency affects remote control applications demo. https://twitter.com/BellLabs/status/836611924456521728.

[8] Keynote systems, inc. website. http://www.keynote.com/.

[9] Local IP Access and Selected IP Traffic Offload (LIPA-SIPTO). http://www.3gpp.org/ftp/Specs/html-info/23829.htm.

[10] Open ROADM MSA. http://openroadm.org/home.html.

[11] OpenAirInterface. http://www.openairinterface.org.

[12] Phantomnet. https://www.phantomnet.org.

[13] SDN-IP Architecture. https://wiki.onosproject.org/display/ONOS/SDN-IP+Architecture.

[14] Signaling is growing 50% faster than data traffic. http://tinyurl.com/zl5ckda.

[15] SIMECA: SDN-based IoT mobile edge cloud architecture - PhantomNet profile. https://wiki.phantomnet.org/wiki/phantomnet/simeca-sdn-based-iot-mobile-edge-cloud-architecture.

[16] The impact of latency on application performance. https://www.scribd.com/document/200387963/Latency-White-Paper-from-NSN.

[17] THE INTERNET OF THINGS: aN OVERVIEW WHITE PAPER. http://www.internetsociety.org/doc/iot-overview.

[18] Why distribution is important in NFV? http://tinyurl.com/kyewvqq.

[19] 3GPP. 3GPP TS 23.002 - Network architecture.

[20] 3GPP. Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS). http://www.3gpp.org/DynaReport/24301.htm.

[21] 3GPP. Telecommunication management; Key Performance Indicators (KPI) for Evolved Universal Terrestrial Radio Access Network (E-UTRAN): Definitions. http://www.3gpp.org/DynaReport/32450.htm.

[22] 3GPP. S1 Application Protocol (S1AP)(Release 10) - Network, Evolved Universal Terrestrial Radio Access, 2011.

[23] 3GPP. 3GPP TS 23.203 - Policy and charging control architecture. http://www.etsi.org/deliver/etsi_ts/123200_123299/123203/12.06.00_60/ts_123203v120600p.pdf, 2014.

[24] 3GPP. 3GPP TS 23.401 - General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access. http://www.etsi.org/deliver/etsi_ts/123400_123499/123401/08.14.00_60/ts_123401v081400p.pdf, 2015.

[25] 5GPPP. 5G Vision-The 5G Infrastructure Public Private Partnership: the next generation of communication networks and services. https://tinyurl.com/hlwqy3f, 2015.

[26] Ahuja, S. S., Ramasubramanian, S., and Krunz, M. Srlg failure localization in all-optical networks using monitoring cycles and paths. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE* (2008), IEEE, pp. 700–708.

[27] Alexa(2014). Alexa: top sites in united states. "http://www.alexa.com/topsites/countries/US".

[28] Amazon. AWS Direct Connect. https://aws.amazon.com/directconnect/.

[29] Amrutkar, C., Hiltunen, M., Jim, T., Joshi, K., Spatscheck, O., Traynor, P., and Venkataraman, S. Why is my smartphone slow? on the fly diagnosis of underperformance on the mobile internet. In *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on* (June 2013), pp. 1–8.

[30] An, X., Pianese, F., Widjaja, I., and Günay Acer, U. Dmme: a distributed lte mobility management entity. *Bell Labs Technical Journal 17*, 2 (2012), 97–120.

[31] Andrews, J. G., Buzzi, S., Choi, W., Hanly, S. V., Lozano, A., Soong, A. C., and Zhang, J. C. What will 5g be? *IEEE Journal on Selected Areas in Communications 32*, 6 (2014), 1065–1082.

[32] Balakrishnan, M., Mohomed, I., and Ramasubramanian, V. Where's that phone?: geolocating IP addresses on 3G networks. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference* (2009), ACM, pp. 294–300.

[33] Balasubramanian, A., Mahajan, R., and Venkataramani, A. Augmenting mobile 3G using WiFi. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services* (2010), ACM, pp. 209–222.

[34] Banerjee, A., Chen, X., Erman, J., Gopalakrishnan, V., Lee, S., and Van Der Merwe, J. MOCA: a lightweight mobile cloud offloading architecture. In *Proceedings of the Eighth ACM International Workshop on Mobility in the Evolving Internet Architecture* (2013), ACM, pp. 11–16.

[35] Banerjee, A., Mahindra, R., Sundaresan, K., Kasera, S., Van der Merwe, K., and Rangarajan, S. Scaling the lte control-plane for future mobile access. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies* (2015), ACM, p. 19.

[36] Banerjee, A., Nguyen, B., Gopalakrishnan, V., Kasera, S., Lee, S., and Van der Merwe, J. Efficient, adaptive and scalable device activation for m2m communications. In *Sensing, Communication, and Networking (SECON), 2015 12th Annual IEEE International Conference on* (2015), IEEE, pp. 399–407.

[37] Basta, A., Kellerer, W., Hoffmann, M., Hoffmann, K., and Schmidt, E. D. A virtual sdn-enabled lte epc architecture: a case study for s-/p-gateways functions. In *2013 IEEE SDN for Future Networks and Services (SDN4FNS)* (Nov 2013), pp. 1–7.

[38] Basta, A., Kellerer, W., Hoffmann, M., Morper, H. J., and Hoffmann, K. Applying nfv and sdn to lte mobile core gateways, the functions placement problem. In *Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications, & Challenges* (2014), ACM, pp. 33–38.

[39] Benson, T., Sahu, S., Akella, A., and Shaikh, A. A first look at problems in the cloud. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing* (Berkeley, CA, USA, 2010), HotCloud'10, USENIX Association, pp. 15–15.

[40] Birke, R., Giurgiu, I., Chen, L. Y., Wiesmann, D., and Engbersen, T. Failure analysis of virtual and physical machines: patterns, causes and characteristics. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (June 2014), pp. 1–12.

[41] Birman, K., Schiper, A., and Stephenson, P. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems (TOCS) 9*, 3 (1991), 272–314.

[42] Blajić, T., Nogulić, D., and Družijanić, M. Latency Improvements in 3G Long Term Evolution. *Mipro CTI, svibanj* (2006).

[43] Bobrovs, V., Spolitis, S., and Ivanovs, G. Latency causes and reduction in optical metro networks. In *SPIE OPTO* (2013), International Society for Optics and Photonics, pp. 90080C–90080C.

[44] Booker, G., Torres, J., Guikema, S., Sprintson, A., and Brumbelow, K. Estimating cellular network performance during hurricanes. *Reliability Engineering & System Safety 95*, 4 (2010), 337–344.

[45] Bosch, P., Samuel, L., Mullender, S., Polakos, P., and Rittenhouse, G. Flat cellular (umts) networks. In *Wireless Communications and Networking Conference, 2007. WCNC 2007. IEEE* (2007), IEEE, pp. 3861–3866.

[46] Burrows, M. The Chubby Lock Service for Loosely-coupled Distributed Systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2006), OSDI '06, USENIX Association, pp. 335–350.

[47] CarrierIQ. Carrier IQ, Inc. `http://www.carrieriq.com`.

[48] Chatfield, C. *The analysis of time series: an introduction*. CRC press, 2004.

[49] Checko, A., Christiansen, H. L., Yan, Y., Scolari, L., Kardaras, G., Berger, M. S., and Dittmann, L. Cloud ran for mobile networks, a technology overview. *Communications Surveys & Tutorials, IEEE 17*, 1 (2015), 405–426.

[50] Chen, K.-T., Huang, P., and Lei, C.-L. Effect of network quality on player departure behavior in online games. *Parallel and Distributed Systems, IEEE Transactions on 20*, 5 (2009), 593–606.

[51] Cho, J., Duerig, J., Eide, E., Nguyen, B., Ricci, R., Syed, A., Van der Merwe, J., Webb, K., and Wong, G. Repeatable mobile networking research with phantomnet: demo. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking* (2016), ACM, pp. 489–490.

[52] Cho, J., Nguyen, B., Banerjee, A., Ricci, R., Van der Merwe, J., and Webb, K. Smore: Software-defined networking mobile offloading architecture. In *Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications, & Challenges* (2014), ACM, pp. 21–26.

[53] Cho, J., Sundaresan, K., Mahindra, R., Van der Merwe, J., and Rangarajan, S. Acacia: Context-aware edge computing for continuous interactive applications over mobile networks. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies* (2016), ACM, pp. 375–389.

[54] Cisco Systems. Cisco Visual Networking Index: global Mobile Data Traffic Forecast Update, 2014-2019. *White Paper, February* (2015).

[55] Corici, M., Magedanz, T., Vingarzan, D., and Weik, P. Prototyping mobile broadband applications with the open evolved packet core. In *Intelligence in Next Generation Networks (ICIN), 2010 14th International Conference on* (Oct 2010).

[56] David Nowoswiat. Managing LTE core network signaling traffic. `http://tinyurl.com/zfbwy8e`.

[57] Dong, W., Ge, Z., and Lee, S. 3G meets the internet: understanding the performance of hierarchical routing in 3G networks. In *Proceedings of the 23rd International Teletraffic Congress* (2011), ITCP, pp. 15–22.

[58] El Hattachi, R., and Erfanian, J. Ngmn 5g initiative white paper–executive version 1.0, 2014.

[59] Ericsson. High Availability is more than five nines. https://www.ericsson.com/real-performance/wp-content/uploads/sites/3/2014/07/high-avaialbility.pdf.

[60] ETSI. Network Functions Virtualisation (NFV); Management and Orchestration. ETSI GS NFV-MAN 001 V1.1.1 (2014-12).

[61] ETSI. NFV White Paper. https://portal.etsi.org/nfv/nfv_white_paper.pdf.

[62] ETSI. Network Functions Virtualisation (NFV); Architectural Framework. ETSI GS NFV 002 V1.1.1 (2013-10), 2013.

[63] ETSI, M. Mobile edge computing-introductory technical white paper, 2014.

[64] Farinacci, D., Lewis, D., Meyer, D., and Fuller, V. The locator/id separation protocol (lisp). https://tools.ietf.org/html/rfc6830, 2013.

[65] Fayazbakhsh, S. K., Reiter, M. K., and Sekar, V. Verifiable network function outsourcing: requirements, challenges, and roadmap. In *Proceedings of the 2013 Workshop on Hot Topics in Middleboxes and Network Function Virtualization* (New York, NY, USA, 2013), HotMiddlebox '13, ACM, pp. 25–30.

[66] Feamster, N., Andersen, D., Balakrishnan, H., and Kaashoek, M. Measuring the effects of internet path faults on reactive routing. In *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (2003), ACM, p. 137.

[67] Gartner. Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020. http://www.gartner.com/newsroom/id/2636073, 2013.

[68] Gember, A., Akella, A., Pang, J., Varshavsky, A., and Caceres, R. Obtaining in-context measurements of cellular network performance. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference* (2012), ACM, pp. 287–300.

[69] Gember-Jacobson, A., Viswanathan, R., Prakash, C., Grandl, R., Khalid, J., Das, S., and Akella, A. Opennf: enabling innovation in network function control. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (New York, NY, USA, 2014), SIGCOMM '14, ACM, pp. 163–174.

[70] Ghavimi, F., and Chen, H.-H. M2m communications in 3gpp lte/lte-a networks: architectures, service requirements, challenges, and applications. *Communications Surveys & Tutorials, IEEE 17*, 2 (2015), 525–549.

[71] Gill, P., Jain, N., and Nagappan, N. Understanding network failures in data centers: measurement, analysis, and implications. *SIGCOMM Comput. Commun. Rev. 41*, 4 (Aug. 2011), 350–361.

[72] Goyal, M., Ramakrishnan, K., and chi Feng, W. Achieving faster failure detection in OSPF networks. In *Communications, 2003. ICC '03. IEEE International Conference on* (May 2003), vol. 1, pp. 296–300 vol.1.

[73] Gunawi, H. S., Laksono, A., Suminto, R. O., Hao, M., Adityatama, J., Eliazar, K. J., and Satria, A. D. Why does the cloud stop computing? lessons from hundreds of service outages.

[74] Guo, C., Yuan, L., Xiang, D., Dang, Y., Huang, R., Maltz, D., Liu, Z., Wang, V., Pang, B., Chen, H., Lin, Z.-W., and Kurien, V. Pingmesh: a large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (New York, NY, USA, 2015), SIGCOMM '15, ACM, pp. 139–152.

[75] Han, B., Hui, P., Kumar, V. A., Marathe, M. V., Shao, J., and Srinivasan, A. Mobile data offloading through opportunistic communications and social participation. *Mobile Computing, IEEE Transactions on 11*, 5 (2012), 821–834.

[76] Hares, S., and White, R. Software-defined networks and the interface to the routing system (i2rs). *IEEE Internet Computing 17*, 4 (2013), 84–88.

[77] Henderson, T. *The effects of relative delay in networked games*. PhD thesis, University of London, 2003.

[78] Huang, J., Qian, F., Guo, Y., Zhou, Y., Xu, Q., Mao, Z. M., Sen, S., and Spatscheck, O. An in-depth study of LTE: effect of network protocol and application behavior on performance. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (2013), ACM, pp. 363–374.

[79] Huang, J., Qian, F., Guo, Y., Zhou, Y., Xu, Q., Mao, Z. M., Sen, S., and Spatscheck, O. An In-depth Study of LTE: effect of Network Protocol and Application Behavior on Performance. In *Proceedings of ACM SIGCOMM* (2013).

[80] Huang, J., Qian, F., Guo, Y., Zhou, Y., Xu, Q., Mao, Z. M., Sen, S., and Spatscheck, O. An in-depth study of LTE: effect of network protocol and application behavior on performance. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (2013), ACM, pp. 363–374.

[81] Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. Zookeeper: wait-free coordination for internet-scale systems. In *USENIX Annual Technical Conference* (2010), vol. 8, p. 9.

[82] IP Access. E-40 Access point.

[83] Jhawar, R., Piuri, V., and Santambrogio, M. Fault tolerance management in cloud computing: a system-level perspective. *IEEE Systems Journal 7*, 2 (June 2013), 288–297.

[84] Jin, X., Li, L. E., Vanbever, L., and Rexford, J. Softcell: scalable and flexible cellular core network architecture. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies* (2013), ACM, pp. 163–174.

[85] Jin, X., Li, L. E., Vanbever, L., and Rexford, J. Softcell: scalable and flexible cellular core network architecture. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies* (2013), ACM, pp. 163–174.

[86] Junqueira, F. P., Reed, B. C., and Serafini, M. Zab: high-performance broadcast for primary-backup systems. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)* (2011), IEEE, pp. 245–256.

[87] Kalmanek, C. R., Misra, S., and Yang, Y. R. *Guide to Reliable Internet Services and Applications*. Springer, 2010.

[88] Kempf, J., Johansson, B., Pettersson, S., Luning, H., and Nilsson, T. Moving the mobile evolved packet core to the cloud. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on* (2012), IEEE, pp. 784–791.

[89] Kevin Fitchard. Data now 85% of mobile traffic but 39% of revenue: what gives? `http://gigaom.com/2012/03/20/data-now-85-of-mobile-traffic-but-39-of-revenue-what-gives/`, March 2012.

[90] Kim, B.-J. J., and Henry, P. S. Directions for future cellular mobile network architecture. *First Monday 17*, 12 (2012).

[91] Kompella, R., Yates, J., Greenberg, A., and Snoeren, A. Detection and localization of network black holes. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE* (May 2007), pp. 2180–2188.

[92] Lagar-Cavilla, H. A., Tolia, N., De Lara, E., Satyanarayanan, M., and OHallaron, D. Interactive resource-intensive applications made easy. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing* (2007), Springer, pp. 143–163.

[93] Lee, C., Park, S. J., Kejriwal, A., Matsushita, S., and Ousterhout, J. Implementing linearizability at large scale and low latency. In *Proceedings of the 25th Symposium on Operating Systems Principles* (New York, NY, USA, 2015), SOSP '15, ACM, pp. 71–86.

[94] Leung, K. K. Mobile ip mobility agent standby protocol, Feb. 27 2001. US Patent 6,195,705.

[95] Li, L. E., Mao, Z. M., and Rexford, J. Toward software-defined cellular networks. In *Software Defined Networking (EWSDN), 2012 European Workshop on* (2012), IEEE, pp. 7–12.

[96] Li, L. E., Mao, Z. M., and Rexford, J. Toward software-defined cellular networks. In *Software Defined Networking (EWSDN), 2012 European Workshop on* (2012), IEEE, pp. 7–12.

[97] Lucent, A. LTE Subscriber Service Restoration - Application Note. `http://www.tmcnet.com/tmc/whitepapers/documents/whitepapers/2014/10085-lte-subscriber-service-restoration.pdf`.

[98] Lucent, A. Study of EPC Nodes Restoration - technical report. `ftp://ftp.3gpp.org/specs/archive/23_series/23.857/23857-140.zip`.

[99] Ma, L., and Li, W. Traffic Offload Mechanism in EPC Based on Bearer Type. In *Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference on* (2011), pp. 1–4.

[100] Mahimkar, A., Ge, Z., Shaikh, A., Wang, J., Yates, J., Zhang, Y., and Zhao, Q. Towards automated performance diagnosis in a large IPTV network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication* (New York, NY, USA, 2009), SIGCOMM '09, ACM, pp. 231–242.

[101] Mahimkar, A., Yates, J., Zhang, Y., Shaikh, A., Wang, J., Ge, Z., and Ee, C. T. Troubleshooting chronic conditions in large IP networks. In *Proceedings of the 2008 ACM CoNEXT Conference* (2008), ACM, p. 2.

[102] Microsoft. Azure Load Balancer overview. https://docs.microsoft.com/en-us/azure/load-balancer/load-balancer-overview.

[103] Microsoft. ExpressRoute. `https://azure.microsoft.com/en-us/services/expressroute/`.

[104] Microsoft. Monitor availability and responsiveness of any web site. `https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-web-app-availability`.

[105] Moradi, M., Li, L. E., and Mao, Z. M. Softmow: a dynamic and scalable software defined architecture for cellular wans. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking* (2014), HotSDN '14.

[106] Nagaraj, K., and Katti, S. Procel: smart traffic handling for a scalable software epc. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking* (2014), HotSDN '14.

[107] Nguyen, B., Banerjee, A., Gopalakrishnan, V., Kasera, S., Lee, S., Shaikh, A., and Van der Merwe, J. Towards understanding TCP performance on LTE/EPC mobile networks. In *Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications, & Challenges* (2014), ACM, pp. 41–46.

[108] Nguyen, B., Banerjee, A., Gopalakrishnan, V., Kasera, S., Lee, S., Shaikh, A., and Van der Merwe, J. Towards understanding tcp performance on lte/epc mobile networks. In *Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications, & Challenges* (2014), ACM, pp. 41–46.

[109] Nguyen, B., Choi, N., Thottan, M., and Van der Merwe, J. Simeca: Sdn-based iot mobile edge cloud architecture. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management* (2017), IEEE.

[110] Nguyen, B., Ge, Z., Van der Merwe, J., Yan, H., and Yates, J. Absence: usage-based failure detection in mobile networks. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking* (2015), ACM, pp. 464–476.

[111] Nguyen, B., Zhang, T., Radunovic, B., Stutsman, R., Karagiannis, T., Kocur, J., and Van der Merwe, J. Echo: a reliable distributed cellular core network for public clouds. In *In submission* (2017).

[112] Nikravesh, A., Choffnes, D. R., Katz-Bassett, E., Mao, Z. M., and Welsh, M. Mobile network performance from user devices: a longitudinal, multidimensional analysis. In *Passive and Active Measurement* (2014), Springer, pp. 12–22.

[113] Nikravesh, A., Choffnes, D. R., Katz-Bassett, E., Mao, Z. M., and Welsh, M. Mobile network performance from user devices: a longitudinal, multidimensional analysis. In *Passive and Active Measurement* (2014), Springer, pp. 12–22.

[114] Nokia. Nokia 7750 Service Router - Mobile Gateway - Data Sheet. `http://resources.alcatel-lucent.com/?cid=141247`.

[115] Nokia. Nokia 9471 Wireless Mobility Manager Mobility Management Entity/Serving GPRS Support Node - Data Sheet. `https://resources.alcatel-lucent.com/theStore/files/Nokia_9471_WMM_MME_SGSN_WM9_Data_Sheet_EN.pdf`.

[116] Nygren, E., Sitaraman, R. K., and Sun, J. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review 44*, 3 (2010), 2–19.

[117] Olsson, M., Sultana, S., Rommer, S., and Frid, L. *SAE and the Evolved Packet Core*. Elsevier, 2009.

[118] OpenStack. Vitrage. `https://wiki.openstack.org/wiki/Vitrage`.

[119] OpenVswitch. `http://openvswitch.org/`.

[120] OpNFV. Doctor. `https://wiki.opnfv.org/display/doctor/Doctor+Home`.

[121] Patel, M., and et al. ETSI - Mobile-Edge Computing. `www.etsi.org`, 2014.

[122] Patel, P., Bansal, D., Yuan, L., Murthy, A., Greenberg, A., Maltz, D. A., Kern, R., Kumar, H., Zikos, M., Wu, H., Kim, C., and Karri, N. Ananta: cloud scale load balancing. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (New York, NY, USA, 2013), SIGCOMM '13, ACM, pp. 207–218.

[123] Pentikousis, K., Wang, Y., and Hu, W. Mobileflow: toward software-defined mobile networks. *Communications Magazine, IEEE 51*, 7 (2013), 44–53.

[124] Pereira, C., and Aguiar, A. Towards efficient mobile m2m communications: survey and open challenges. *Sensors 14*, 10 (2014), 19582–19608.

[125] Perkins, C. Ip mobility support for ipv4, 2002.

[126] Qazi, Z. A., Penumarthi, P. K., Sekar, V., Gopalakrishnan, V., Joshi, K., and Das, S. R. Klein: a minimally disruptive design for an elastic cellular core. In *Proceedings of the Symposium on SDN Research* (2016), ACM, p. 2.

[127] Qian, F., Sen, S., and Spatscheck, O. Silent TCP connection closure for cellular networks. In *CoNEXT* (2013), pp. 211–216.

[128] Rajan, A., Gobriel, S., Maciocco, C., Ramia, K., Kapury, S., Singhy, A., Ermanz, J., Gopalakrishnanz, V., and Janaz, R. Understanding the bottlenecks in virtualizing cellular core network functions. In *Local and Metropolitan Area Networks (LANMAN), 2015 IEEE International Workshop on* (Apr. 2015), pp. 1–6.

[129] Roughan, M., Greenberg, A., Kalmanek, C., Rumsewicz, M., Yates, J., and Zhang, Y. Experience in measuring Internet backbone traffic variability: models, metrics, measurements and meaning. In *Proceedings of the International Teletraffic Congress (ITC-18)* (2003).

[130] Sama, M. R., Contreras, L. M., Kaippallimalil, J., Akiyoshi, I., Qian, H., and Ni, H. Software-defined control of the virtualized mobile packet core. *IEEE Communications Magazine 53*, 2 (2015), 107–115.

[131] Sanchez, M. I., Asadi, A., Draxler, M., Gupta, R., Mancuso, V., Morelli, A., De la Oliva, A., and Sciancalepore, V. Tackling the increased density of 5g networks: the crowd approach. In *Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st* (2015), IEEE, pp. 1–5.

[132] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. The Case for VM-based Cloudlets in Mobile Computing. In *IEEE Pervasive Computing* (November 2009).

[133] Saunders, R., Cho, J., Banerjee, A., Rocha, F., and Van der Merwe, J. P2p offloading in mobile networks using sdn. In *Proceedings of the Symposium on SDN Research* (2016), ACM, p. 3.

[134] Schneider, F. B. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computing Surveys 22*, 4 (Dec. 1990), 299–319.

[135] Sekar, V., Egi, N., Ratnasamy, S., Reiter, M. K., and Shi, G. Design and implementation of a consolidated middlebox architecture. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2012), NSDI'12, USENIX Association, pp. 24–24.

[136] Seskar, I., Nagaraja, K., Nelson, S., and Raychaudhuri, D. Mobilityfirst future internet architecture project. In *Proceedings of the 7th Asian Internet Engineering Conference* (2011), ACM, pp. 1–3.

[137] Setty, S., Su, C., Lorch, J. R., Zhou, L., Chen, H., Patel, P., and Ren, J. Realizing the Fault-tolerance Promise of Cloud Storage Using Locks with Intent. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (2016).

[138] Shafiq, M. Z., Ji, L., Liu, A. X., Pang, J., and Wang, J. A first look at cellular machine-to-machine traffic: large scale measurement and characterization. *ACM SIGMETRICS Performance Evaluation Review 40*, 1 (2012), 65–76.

[139] Shelby, Z., Hartke, K., and Bormann, C. The constrained application protocol (coap). `http://coap.technology/`, 2014.

[140] Sherry, J., Hasan, S., Scott, C., Krishnamurthy, A., Ratnasamy, S., and Sekar, V. Making middleboxes someone else's problem: network processing as a cloud service. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (New York, NY, USA, 2012), SIGCOMM '12, ACM, pp. 13–24.

[141] Song, H. H., Ge, Z., Mahimkar, A., Wang, J., Yates, J., Zhang, Y., Basso, A., and Chen, M. Q-score: proactive service quality assessment in a large IPTV system. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference* (2011), IMC '11, ACM.

[142] Syed, A., and Van der Merwe, J. Proteus: a network service control platform for service evolution in a mobile software defined infrastructure. In *International Conference on Mobile Computing and Networking (MobiCom)* (2016).

[143] Taleb, T. Toward carrier cloud: potential, challenges, and solutions. *IEEE Wireless Communications 21*, 3 (June 2014), 80–91.

[144] Tanganelli, G., Vallati, C., and Mingozzi, E. Coapthon: easy development of coap-based iot applications with python. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on* (2015), IEEE, pp. 63–68.

[145] Thomson, A., Diamond, T., Weng, S.-C., Ren, K., Shao, P., and Abadi, D. J. Calvin: fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (2012), ACM, pp. 1–12.

[146] Wale, K. Implementing ATCA Serving Gateways for LTE Networks. `http://go.radisys.com/rs/radisys/images/paper-atca-implementing.pdf`.

[147] Wang, K., Shen, M., Cho, J., Banerjee, A., Van der Merwe, J., and Webb, K. Mobiscud: a fast moving personal cloud in the mobile network. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges* (2015), ACM, pp. 19–24.

[148] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation* (Boston, MA, Dec. 2002), USENIX Association, pp. 255–270.

[149] Wood, T., Ramakrishnan, K., Hwang, J., Liu, G., and Zhang, W. Toward a software-based network: integrating software defined networking and network function virtualization. *Network, IEEE 29*, 3 (May 2015), 36–41.

[150] Xilouris, G., Trouva, E., Lobillo, F., Soares, J., Carapinha, J., McGrath, M., Gardikis, G., Paglierani, P., Pallis, E., Zuccaro, L., Rebahi, Y., and Kourtis, A. T-NOVA: a marketplace for virtualized network functions. In *Networks and Communications (EuCNC), 2014 European Conference on* (June 2014), pp. 1–5.

[151] Xu, Q., Huang, J., Wang, Z., Qian, F., Gerber, A., and Mao, Z. M. Cellular data network infrastructure characterization and implication on mobile content placement. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems* (2011), ACM, pp. 317–328.

[152] Xu, Q., Huang, J., Wang, Z., Qian, F., Gerber, A., and Mao, Z. M. Cellular data network infrastructure characterization and implication on mobile content placement. In *SIGMETRICS* (2011).

[153] Xu, Y., Wang, Z., Leong, W. K., and Leong, B. An end-to-end measurement study of modern cellular data networks. In *Passive and Active Measurement* (2014), Springer, pp. 34–45.

[154] Yan, H., Flavel, A., Ge, Z., Gerber, A., Massey, D., Papadopoulos, C., Shah, H., and Yates, J. Argus: end-to-end service anomaly detection and localization from an ISP's point of view. In *INFOCOM* (2012), IEEE.

[155] Yan, H., Ge, Z., Osinski, M., and Yates, J. When Cell Towers Fail: Quantifying the Customer Impact. `http://www.research.att.com/articles/featured_stories/2013_03/201306_tower-outage-analyzer.html`.

[156] Zhang, M., Zhang, C., Pai, V., Peterson, L., and Wang, R. PlanetSeer: internet path failure monitoring and characterization in wide-area services. In *Proc. USENIX OSDI* (2004).

[157] Zhang, Y., Ge, Z., Greenberg, A., and Roughan, M. Network anomography. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement* (Berkeley, CA, USA, 2005), IMC '05, USENIX Association, pp. 30–30.

[158] Zhang, Y., Mao, Z., and Zhang, M. Effective diagnosis of routing disruptions from end systems. In *Proceedings of USENIX NSDI*, vol. 8.