Rethinking LTE Network Functions Virtualization

Muhammad Taqi Raza*§, Dongho Kim†, Kyu-Han Kim†, Songwu Lu* and Mario Gerla*
*University of California – Los Angeles (UCLA), †Hewlett Packard Enterprise Labs
Email: *{taqi, slu, gerla}@cs.ucla.edu, †{dongho.kim, kyu-han.kim}@hpe.com

***Abstract*—LTE Network Function Virtualization (LTE-NFV) scales user services in a low cost fashion by transforming the centralized legacy LTE Core architecture to a distributed architecture. This distributed architecture makes multiple instances of LTE Network Functions (NFs) and virtualizes them on commodity data-center network. The functionality of LTE-NFV architecture breaks however, since the distributed NF instances connected via unreliable IP links delay the execution of critical events. The failure of time-critical events results in users' quality of service degradation and temporary service unavailability. In this paper, we propose a new way to virtualize LTE core network. We argue that *logic-based* NFs segregation should be done for NFV, instead of *instance-based* NFs segregation done in current NFV implementation. Our approach of 'logic-based NFs segregation' combines the logic of an event into a single NF, thus localizing the execution of critical events to one virtual machine. This way, only the localized entities exchange signalling messages, and the events do not experience large delays. We further reduce the delays by exploiting the parallelism in LTE network protocols; and partition these protocols such that their signalling messages run in parallel. In addition, we eliminate unnecessary messages to reduce the signalling overhead. We build our system prototype over OpenEPC LTE core network in virtualized platform. Our results show that we can reduce event execution time and signalling overhead up to 50% and 40%, respectively.**

I. Introduction

In this paper, we analyze the impact of virtualization on LTE Evolved Packet Core (EPC) functionality and service provisioning. We find that the legacy LTE EPC architecture is designed for fewer, powerful, and dedicated Network Functions (NFs). However, we argue that the functionality of EPC architecture breaks when this architecture is virtualized and scaled to thousands of NFs, where each NF is implemented in data-center network.

Challenges and impact: We discover two major challenges in virtualized EPC (vEPC).

1. Virtualized network is not designed for LTE: LTE EPC is virtualized over data-center network which suffers from long queueing delays in switches, packet losses, timed out retransmissions, and out of order packets delivery [1]. Because of these characteristics, virtualized NFs (VNFs) implemented over commodity data-center network only provide flow level guarantees, whereas LTE standard requires packet level guarantees (100ms and 300ms delays for voice and data packets, respectively) [2].

2. LTE EPC is not designed for virtualized network: In legacy EPC, there are fewer NF boxes, which are connected through dedicated fiber links. The round-trip-time (RTT) over direct link is stable, and determines NF reachability and packet retransmission counters [3]. In vEPC however, some network signalling packets take longer over congested IP links, triggering unnecessary packet retransmissions at sender. The timeout and retransmission of signalling packets for one NF causes 'time-out domino effect' at the NFs that follow. This higher signalling failure rate while executing certain network events [4] have direct impact on user traffic (e.g. voice and data) continuity. We call these events 'mission critical events'.

Goal: We want to protect mission critical events from delay and failure. We identify three mission critical events; *handover* event during device mobility, *paging* event during device idle mode, and *service request* for gaining network resources. Since these three events cause 50% of LTE network signalling [5] [6], we aim to isolate these events from the vEPC to reduce network signalling load as well as execute critical events in a timely manner.

Design: In legacy EPC, few dedicated NFs can handle millions of customers. In vEPC, whereas, thousands of NFs instances are initiated to handle same number of customers. These NF instances are distributed across data center network. The use of multiple NF instances in data-center achieves scalability, and provides simplified and inexpensive approach. However, it breaks the basic functionality of EPC when an event experiences long and unpredictable delays between these NFs. The distributed NF instances jeopardize the timely execution of events.

In contrast, we took a holistic approach to design a robust and scalable EPC virtualization architecture. Our design theme is to perform logic based NFs segregation for an event, rather than the instance based NFs segregation in vEPC. In our design, we extract an event's logic from each NF in the form of a module, and then assemble the extracted modules of the same event from several NFs into a Fat-proxy. This Fat-proxy – comprised of an event's complete execution logic – acts as a standalone execution engine for that event. Let's take an example of the "handover event" Fat-proxy that extracts modules from the Mobile Management Entity (MME), P-Gateway (PGW), and Serving Gateway (SGW) NFs, and assembles them to make independent handover event facilitator. The concept of Fat-proxy or thick client is very useful in distributed community, where server delegates *some* processing logic to the client [7]. Our design is motivated from similar concept but under different setting; we want to make proxy a powerful component that *completely* executes a particular event.

When vEPC receives an event request, it forwards the request to that event-specific Fat-proxy; it takes on the responsibility of executing that event and finally flushes the updated event status and device session information to vEPC. This way, all

§Corresponding and student author.

978-1-5090-6501-1/17/$31.00

event execution logic remains local to one entity (Fat-proxy), thus solving issues incurred by distributed vEPC architecture, and keeping greater number of signalling messages away from vEPC. Nevertheless, extracting event specific logic from multiple NFs is a challenge as we have to keep track of different events, and resolve logic and data dependencies among those events.

The second part of our design exploits the inherent parallelism in LTE network protocols by identifying and partitioning the mutual exclusive logic of an event. In the "handover event" example, the MME logic can be split into two mutually exclusive partitions, which are able to run S1AP protocol operations and GTP protocol operations in parallel. Traditionally, these two types of operations execute one after the other. By partitioning, the network operations run in parallel and speed-up the execution of the event, thus mitigating the timeouts and the need to re-transmit packets.

Results: We show that our design (1) reduces more than 40% signalling load by skipping and parallelizing the messages for network operations, (2) reduces up to 50% event execution time, and (3) improves voice and data applications performance.

II. Background: LTE Architecture

LTE network consists of three main components; User Agent (UE), Evolved Node Base-station (eNodeB), and Evolved Packet Core (EPC), as shown in Figure 1. These components are both logically and physically distributed. The eNodeB anchors as a radio interface between UE and EPC. EPC communicates with packet data networks in the outside world such as the Internet, private corporate networks or the IP Multimedia Subsystem (IMS) and facilitates user communication. LTE EPC is comprised of a number of LTE Network Functions (NFs), which are Mobility Management Entity (MME), Home Subscriber Server (HSS), Serving Gateway (SGW), Packet Data Network Gateway (PGW), Policy and Charging Rules Function (PCRF), and a few others. These NFs handle control-plane and data-plane traffic through separate network interfaces. As shown in Figure 1, control-plane traffic from radio network is sent to MME (via S1-AP logical interface), whereas data-plane traffic is forwarded to SGW. MME acts as a central management entity that authenticates and authorizes UE, handles network events (such as *device Attach*, *Handover*, *Service provisioning*, and *Paging* events), and maintains SGW and PGW connections for data-plane traffic.

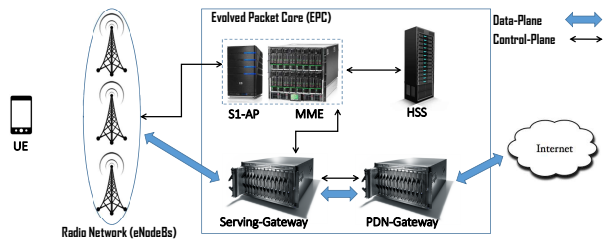


Figure 1: LTE architecture: an overview

EPC NFs are static in nature and are connected or chained in a certain way to achieve desired LTE network functionality/service. These NFs exchange a number of control messages to execute a specific network event. For example, during *device Attach* event, MME obtains device security keys from HSS, authenticates the device, and creates device session information at SGW and PGW. These SGW and PGW establish data bearer connection with the device and configure specific QoS profile, thus registering the device with LTE network. The delay or failure in one control-message results in complete event failure [8]. To mitigate these failure, the NFs – implemented over vendor specific software and hardware – guarantee message level reliability and high availability.

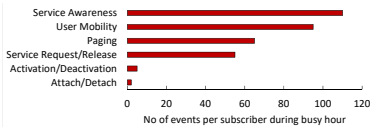


Figure 2: Number of events per user during busy hours

Figure 3: One legacy MME is decomposed into multiple virtualized MME instances

III. Motivation and Problem Scope

Instance-based vEPC implementation is not right: To understand deployment strategies of virtualizing LTE by research and industry communities, we surveyed ETSI (the LTE standardized body) documents on NFV [9], white papers from industry [10], and recent work on LTE NFV (discussed in related work section VIII). We find that most of these efforts talk about *instance-based* vEPC implementation, where vEPC NFs (MME, SGW, PGW etc.) are installed as virtual machine instances. In this paper, we stimulate the discussion that although such contemporary NFV implementation (which is highly distributed) suits the web-based applications' needs well, it may not be a good choice for implementing vEPC.

We address two major issues in the context of *instance-based* vEPC implementation, i.e. (1) signalling storm during peak hours, and (2) timely execution of mission critical events.

Controlling network signalling storm: LTE devices frequently interact with LTE network to execute their events. These events are *Device Attach, Service Request and Release, Handover, Paging, Bearer Activation, Modification and Deactivation, Detach Request*, and many others. Out of these events, some are executed more frequently than others. As shown in Figure 2, *Handover* event is executed at least 50 times more than device *Attach* incident during busy hours [5] [6]. To execute one such event, different NF components interact with each other and generate a greater number of signalling messages. Some events produce more signalling messages than others. For example, one *handover* event can generate upto 32 signalling messages compared to *paging* event that produces only 6 signalling messages. When all events are combined from all devices during busy hours, a signalling storm is generated at EPC NFs. We are motivated to provide a solution that controls the signalling storm at LTE core without restricting devices' network access (the solution operational LTE networks use to control signalling storm [11]).

Signalling messages within vEPC incur latencies: LTE eNodeBs are connected with vEPC over a dedicated fiber link. The latencies within vEPC are caused by VM hypervisor

as well as switching contention and port queuing [1], which reach upto hundreds of milliseconds [12]. For each event request from eNodeB, a greater number of signalling messages exchanged among different VNFs in vEPC suffer network delays. For example, a single *handover request* message from device generates up to 32 signalling messages (including those related to SGW and MME relocation) within vEPC, and suffer delays. Therefore, in this paper, we limit our scope to signalling messages within vEPC, which are prone to higher latencies as compared to fixed latencies over the eNodeB-vEPC fiber link.

Administrating mission critical events: Our preliminary study on LTE operational network discloses that average events completion time at EPC is significantly high during rush hours, reaching up to 3 seconds (as shown in Figure 4)[1]. This higher latency directly affects user QoS experience; from Voice over LTE (VoLTE) call drop and voice jitter, to affecting TCP based services (refer to Table 6.1.7: Standardized QCI characteristics in [2] that provides latency requirements for different applications). We are also motivated to provide timely execution of mission critical events during higher service load at LTE NFs.

Defining mission critical events: We classify those events as mission critical events whose delay or failure has a direct impact over ongoing user services (i.e. voice, data, and multimedia services). These events are:

- *Handover event* that ensures seamless user traffic flow during user mobility.
- *Paging event* that wakes device from idle state when voice/data traffic is pending at LTE network.
- *Service Request event* that provides on-demand network resources to device.

Interestingly, these 3 events make up 50% of all LTE network signalling traffic [5]. By targeting these events, we not only ensure timely execution of user service sessions but also address highly occurring network signalling messages.

Assumptions: This work neither assumes special data center network topology and high performance server boxes nor requires changes in LTE standard. We address timely execution of critical events on commodity data center network (with no dedicated/express links) while obeying LTE standard to provide plug and play solution for any carrier network.

IV. Challenges in Virtualizing LTE-EPC

We discover multiple challenges from our implementation of LTE-EPC virtualization, and from our study on LTE standard documents and virtualized network infrastructure.

A. On data-center network characteristics

NFV infrastructure consists of commodity servers running Virtualized Network Functions (VNFs) over cloud platforms. In contrast to legacy LTE NFs implementation, NFV implementation introduces a number of changes. First, VNFs

[1]We gather LTE traces at device and ignore radio retransmissions (at both MAC and RLC LTE layers), and also excluded device and radio RTT from results.

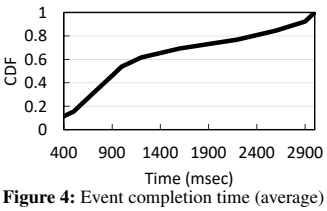


Figure 4: Event completion time (average) during busy hours in operational EPC network

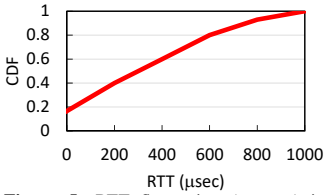


Figure 5: RTT fluctuation (average) in virtualized network

may be located over multiple hops unlike the traditional NFs which are directly connected. Therefore, during congestion, long queueing delays in switches introduce high latencies [1] [12]. Second, during high data-center utilization, packet loss probability increases that can adversely affect traffic flows, where the loss of an ACK may cause TCP to perform a timed out retransmission. Third, data-center network traffic exploits the inherent multi-path nature of data-center networks [12] resulting in out of order packet delivery. Fourth, data-center network is designed to meet application deadlines, which provide mechanisms to meet traffic flow deadlines (e.g. mice flows), rather than packet level guarantees [12].

In short, data-center network is designed to meet Service Level Agreement (SLA) by protecting execution bounds on traffic flows. However, in LTE, service guarantees are made by timely execution of mission critical events.

B. On inter-VNF delay

LTE-NFV framework provides the flexibility and network scalability by decomposing original NFs into multiple VNFs [9]. In order to ramp up the original capacity of a NF, multiple VNF instances are needed. For example, hundreds (if not thousands) of MME-VNF instances are initiated over commodity servers in order to facilitate 10 millions subscribers as supported by conventional MME function [13]. As shown in Figure 3, legacy MME is decomposed into multiple MME instances, where each MME holds the profiles of subset of customers. These VNF instances are distributed within data center. Ideally, related EPC VNF instances (e.g. MME, SGW, PGW etc.) are placed *within the same rack* that eliminates network delays between two EPC NFs. However, during mobility, device switches from source eNodeB to target eNodeB – connected to different MME instance. As a result, the device session migrates from its source MME to target MME during handover. Thereafter, new serving MME and rest of old serving EPC NFs (e.g. SGW and PGW etc.) end up residing at *different racks*. Now network delays play important role on timely execution of network signalling messages. We find that LTE-NFV framework is not able to cope with varying delays among different VNF instances because of following reasons:

Expiry of a timer at any NF may lead to event failure: LTE was designed for fewer EPC NFs which are directly connected over dedicated fiber link. Therefore, in legacy LTE network, the variation in RTT values is negligible. This motivates LTE network designer to use RTT for two purposes (1) path management, and (2) calculating message retransmission timer between a pair of EPC NFs.

Path management: As a matter of fact, all EPC NFs and the connection between them must always be active to serve users. To determine that a peer NF is active, the NFs exchange echo-request and echo-response messages [3]. This exchange of the echo-request and echo-response messages between two NFs allows for quick path failure detection [3].

Retransmission timer: Echo-request and echo-response also help in calculating packet retransmission time at EPC NF. Retransmission timer is calculated based on RTT measurements (i.e. time difference between echo-request and echo-response) [3]. Although, such timer value incorporates arbitrary RTT value delays, it does not include larger RTT value variations because network communication delay does not vary for directly connected legacy LTE NFs.

However, virtualized EPC implementation needs to address significant RTT variations. Data-center network's link redundancy provides multiple paths for each distributed pair of NF [12]. This means echo-request and echo-response packets may traverse through two different paths for each RTT calculation. This can potentially cause a significant variation between two subsequent RTT measurement readings. To make things worse, data-center network congestion can cause RTT spikes to tens of milliseconds making EPC retransmission timer calculation even harder. Figure 5 shows variation of RTT values in a virtualized network [12]. RTT varies from few microseconds to 1000 microseconds under normal network load. This 1000X RTT difference converts into hundreds of different timer values.

When a NF selects smaller timer value based on smaller RTT, the signalling messages from that NF are unnecessarily retransmitted, as shown in Figure 6. Unnecessary signalling messages retransmission lead to overall delay in event execution; and expiry of event-timer running at device results in event failure.

Expiration of a timer has a domino effect: For one event execution, multiple EPC NFs are chained such that one NF output is an input of second NF, and so on. For example, in *handover* event execution, signalling messages are exchanged between 5 different NFs (i.e. source MME, source SGW, target MME, target SGW, and PGW). Each pair of NF is running a different retransmission timer value. When one timer value expires, it produces a domino effect that causes expiration of preceding NF timer. This has been shown in Figure 7, where source MME sends handover signalling message (e.g. *Forward Relocation Request message*) to a target MME. Target MME sends another handover message (e.g. *Create Session Request message*) to SGW. Even in the presence of mild network congestion, the response from SGW is delayed and the timer at target MME expires. Because source MME is waiting for a response from target MME, eventually the timer value at source MME expires. This can potentially create a domino effect to chained NFs for *handover* event execution.

In short, EPC by design is not only sensitive to network delays but also does not tolerate even mild delay variance. It is challenging to provide both constant and smaller network delays in virtualized data-center network, where packets may face network congestion and take multiple packet traversal paths, which were not the case in legacy EPC.

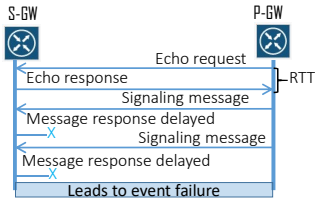


Figure 6: Signalling messages expire prematurely when a timer based on RTT value is too short

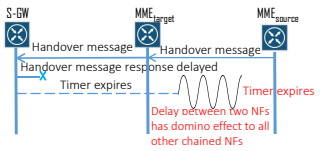


Figure 7: Expiry of timer between two NFs has a domino effect that leads to an event failure

We should also clarify that engineering efforts, such as offloading traffic from busy servers, do not work for vEPC. This is mainly because current data-center networks are designed using web service applications in mind; whereas vEPC (by EPC design rule) requires that all active user sessions remain within same EPC NFs, otherwise the GTP-tunnel [14] between two LTE NFs breaks and incurs further delays.

V. System Design

Design overview: The comparison of our design with contemporary EPC architectures is shown in Figure 8. Network Functions (NFs) pass messages among one another, and perform operations to execute an event. We see in Figure 8 that the legacy LTE EPC has dedicated NFs (for example, MME, SGW, and PGW), connected via fiber links. Hence the message passing between NFs experience minimum delays, and these NFs almost always guarantee event execution. As we move from legacy EPC to Virtualized EPC network, we see multiple instances of the same NFs distributed in data-center networks and joined via unreliable IP links. These NFs in vEPC are designed to support few thousands users as compared to million of users being supported by legacy network; yet vEPC NFs are simple, scalable, and provide plug and play solution to the service providers. However, the main culprit of EPC virtualization is high delay between consecutive NFs for a single event. When delay guarantees are not provided, the system fails to execute events and provide services.

First contribution: We propose an alternative approach for EPC virtualization that solves the above problems. We base our design on logic based NFs segregation instead of Instance based NFs segregation for vEPC. For every critical event, we extract the logic of that event from each NF in the form of a module. Then we assemble the event-based modules extracted from all NFs into a Fat-proxy as shown in Figure 8. This Fat-proxy acts as a NF for that event. Note that we make Fat-proxys only for three critical events (*handover*, *paging*, and *service request*), since we have already established that these critical events constitute 50% of the control signalling traffic. The advantage of extracting logic based modules and assembling them into a Fat-proxy is three-fold. First, the Fat-proxy acts as a single NF and is made to handle only a single type of event. This reduces delays and avoids timeouts. Second, huge storm of critical events' signalling traffic is diverged from the EPC to the Fat-proxy, and the EPC can

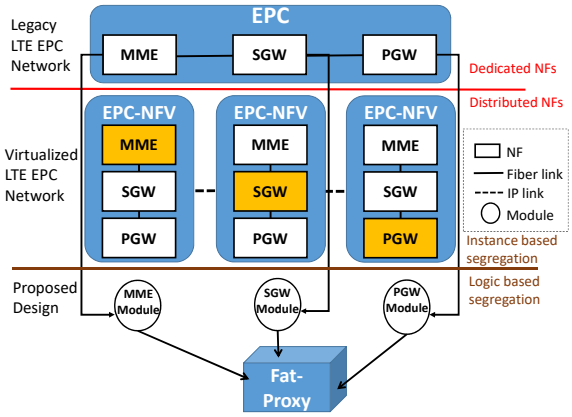


Figure 8: Design overview

handle all the other event requests on time while ensuring the timely execution of critical events through the Fat-proxy. Third, a Fat-Proxy that contains complete execution logic of an event can easily scale-out as the demand increases. We need to simply spin a new instance of Fat-Proxy and configure its incoming/outgoing interfaces. In short, our approach of logic based segregation is not only distributed and scalable but also mitigates the inherent disadvantages of distributed solutions.

Second contribution: We identify the mutually exclusive logic inside one module and partition that logic. A single logic partition runs one network operation. We can run multiple network operations in parallel since the logic in their respective partitions is independent of the other. This further speeds up event execution and helps us choose lower timeout values.

A. Fat-Proxy Design

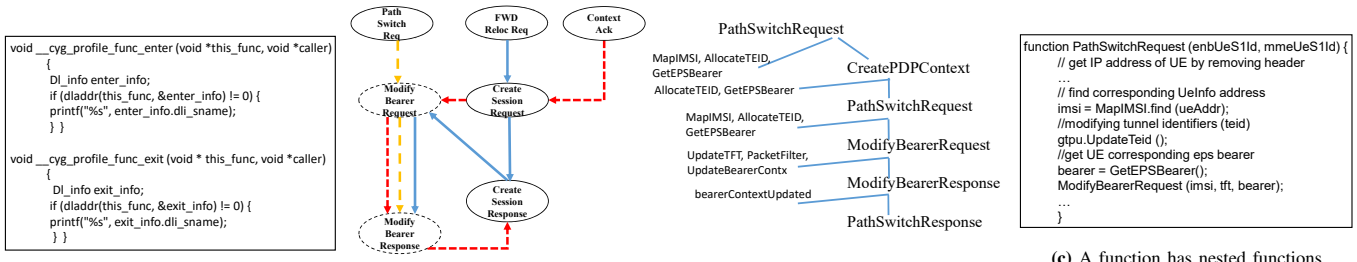
Our goal is to develop a Fat-proxy for an event by assembling all the event specific logic from different EPC NFs. For example, Handover Management (HoM) Fat-proxy is made for handover event by extracting logic components from MME, SGW and PGW NFs. To develop event-specific Fat-proxy, we perform three major steps.

1) Functional decomposition: Decomposing event specific functions from the source code.
2) Event logic extraction: Extracting critical events' functions by resolving decomposed functions' dependencies.
3) Logic-based partitioning: Partitioning the mutually exclusive logic of critical event.

1) Functional decomposition The first step in our design is to decompose a NF into its logic components. We use OpenEPC source code [15] to construct function call graph. We extended etrace [16], a run-time function call graph tool, to extract function call information and global variable usage. Our goal is to automatically identify functional dependencies over complete source code. The function call graph captures the caller-callee relationship. Suppose there are two functions represented as nodes A and B in a function call graph. We add an edge from A to B if a function A calls function B, and/or if a function A accesses a global variable whose value is manipulated in function B. To construct this graph, we use gcc feature called "instrument-functions". We add *__cyg_profile_func_enter()* at the start of a function and *__cyg_profile_func_exit()* at the end of a function, and collect the function call traces in a text file. These "instrument-functions" write the function pointer addresses of functions, in which they were called, to the trace file. This data does not contain any symbol/function names. To resolve the function pointer addresses to their human readable names, we use BSD library function *dladdr()*. It takes a function pointer and tries to resolve its name and file where it is located. The source code of above mentioned procedure is shown in Figure 9.

Our functional decomposition methodology captures dynamic linking of function calls (at runtime). The dynamic call graph records functions chain resulting from calling an event and under specific scenario. This is important for correctness of true functional dependencies among different events (step 2). We show this in Figure 10a where different functions interact differently depending upon event execution logic. Figure 10a captures part of *handover* and *TAU* event execution. We have intentionally omitted certain functions from this call graph to highlight the fact that same functions can be chained differently depending on the event they are executing. First, we show that there are two different ways same event (*handover* event) can execute depending upon two different scenarios. In first scenario, eNodeB_{source} and eNodeB_{target} are not directly connected (that is two eNodeBs are not connected over LTE *X2* interface). In this case, the downlink user packets, while the UE is in the handover process, are tunneled through EPC. The PGW forwards the packets to SGW_{source} that forwards them to eNodeB_{source}. However, eNodeB_{source} is unable to forward to eNodeB_{target} because there is no *X2* interface. Then eNodeB_{source} reflects these packets back to SGW_{source} that uses "indirect" tunnel and forwards these packets to SGW_{target}. SGW_{target} finally forwards them to eNodeB_{target}. This *ForwardRelocationRequest()* function requires that MME_{target} creates a session with SGW_{target} by calling *CreateSessionRequest()* and *CreateSessionResponse()* functions. Create Session procedure sets-up a new device entry (that includes IMSI, APN name, Link EPS Bearer ID, PGW S5/S8 Address for Control Plane) for tunneling downlink packets. Thereafter, device bearers are updated through modify bearer functions. In Figure 10a solid blue arrows show chain of call graph (FWD Reloc Req $\rightarrow$ Create Session Req $\rightarrow$ Create Session Resp $\rightarrow$ Modify Bearer Req $\rightarrow$ Modify Bearer Resp). The second scenario of handover takes place when eNodeB_{source} and eNodeB_{target} are connected over LTE *X2* interface. In this case the user downlink data packets do not require to be forwarded through EPC tunnel. The eNodeB_{target} sends the path switch request message (by calling *PathSwitchRequest()* function) to MME and informs that the device has moved away from eNodeB_{source}. The SGW needs to forward the incoming packets to a different destination. So the MME invokes *ModifyBearerRequest* procedure to the SGW and updates the downlink tunnel identifiers. This handover procedure has been shown in Figure 10a through yellow dashed arrows (Path Switch Req $\rightarrow$ Modify Bearer Req $\rightarrow$ Modify Bearer Resp).

We next show how same functions interact differently de-



(a) Dynamic functions call graph

(b) Call graph as a tree data structure

(c) A function has nested functions

Figure 9: Generating functional call graph through gcc instrument functions

Figure 10: Function decomposition: (a) Different functions chain differently based on the event logic. (b) The function call graph consists of those functions (i) which are part of an event (right nodes) and (ii) which are called within a particular functions (left nodes). (c) In other words, functions are nested and only runtime call graph can identify these functions (i.e. ii)

pending on two different events execution logic. Similar to *handover* event, the *TAU* event also requires *Session Creation* and *Bearer Modifications* procedures to be executed, as shown by TAU function call graph in Figure 10a (red dotted lines). In *TAU* event, when MME selects a new SGW, it sends a *Create Session Request* message per PDN connection to the selected new SGW. The PGW address and traffic flow template are indicated in the bearer *Context Request* message. The SGW informs the PGW(s) about the change by invoking *ModifyBearerRequest()* per PDN connection to the PGW(s) concerned. The PGW updates its bearer contexts and generates *ModifyBearerResponse()*. Finally, SGW generates a *Create Session Response* message to MME. Note that *Modify Bearer Req/Response* is sandwiched between *Create Session Request/Response* (which is different from *handover* event execution call graph). This has been captured in Figure 10a through red dotted lines (Context Ack → Create Session Req → Modify Bearer Req → Modify Bearer Resp → Create Session Resp).

Our function call graph also considers global variable usage as a reason for functional dependency. This has been shown through an edge from *PathSwitchRequest* to *CreatePDPContext* in Figure 10b, where former function accesses global variable (named *bearer*) modified in the latter function. Moreover, through source code snippet in Figure 10c, we show there are other functions which are although seemingly not part of event execution function, yet they create a dependency for that event. For example, *PathSwitchRequest()* function needs to update TEID (Tunnel Endpoint Identifier), and gets corresponding IMSI and EPS bearer before invoking *ModifyBearerRequest()* function.

2) Event logic extractions The second step in our design is to extract critical event execution logic from respective vEPC NFs and combine them as that event's Fat-Proxy. This requires us to first identify those critical event functions on which other events rely too; because extracting these dependent functions would make original vEPC failure prone. Through our functional decomposition procedure (step 1), we construct an execution graph for each event using function call graph. We noticed two kinds of dependencies, logic and data dependencies. The logic dependency occurs between two events when both events need that logic to execute; when not identified and handled properly, it can affect the functionality of the events. Data dependency occurs during the execution of the event, when it needs to exchange user data with an external entity. Another variant of this dependency is when the start of an event depends on the end of other event. Once we have all events' execution graphs, we compare the graph of critical events with all the other events' (critical and non-critical) graphs. Our goal is to find the Common Subgraph Isomorphism (CSI) between critical events and other events' graphs; this subgraph reveals the shared components between events, which infact are the logic dependencies between the two events. Since the CSI problem is NP-Complete, we use an improved back-tracking algorithm [17] from the CSI literature. Even though backtracking algorithms are not efficient in terms of computational time, we argue that our event logic extraction is a one-time and offline procedure, hence the time complexity of CSI does not effect our approach.

For all the common components in the execution graphs of two events, we retain a copy of those common components in the NF while extracting the independent components of the critical event. If there is no logic dependency for a component used by a critical event, we extract it without maintaining its copy in the NF. The identification of these logic dependencies help us maintain functional correctness for all the other events (such as *Attach*, *Detach*, and *Paging* etc.). In Figure 11, we show execution graphs of 'Handover' event and 'Service Request' event. The highlighted components in Figure 11 in step 1 (*create session request, modify bearer request, create session response, and modify bearer response*) are the common components between the two events. This implies that we cannot extract these components for handover event, unless we maintain their copy in the NF to be used by service request event as well.

Data dependency arises when the handover event is in 'modify bearer' component[2]. Device bearers are to be modified at actual SGWs and PGW of EPC. This is shown in Figure 11 in step 2, where handover event's 'modify bearer' component needs to interact with the EPC. In our design, we restrict any communication with EPC unless the event is complete, and remove such mid-way data dependency by always generating fake 'modify bearer success' response for device. Once the handover event has concluded and returns user state to vEPC,

[2]Bearer modification procedure is used to modify device QoS and/or TFT (Traffic Flow Template) of an EPS bearer.

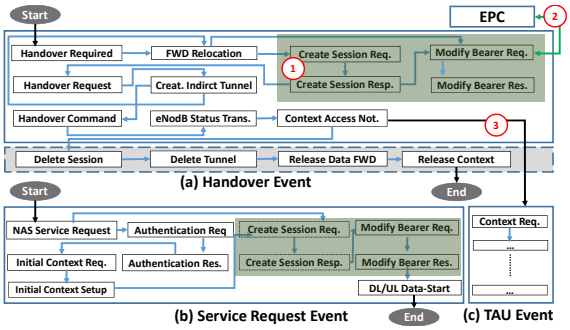


Figure 11: Identifying logic dependencies through Common Subgraph Isomorphism (ICS) and data dependencies through protocol analysis

the vEPC runs the actual 'modify bearer' request. It is possible that such bearer modification step fails (e.g. one of SGW or PGW fails), even though the device is already notified of a successful bearer modification. This is not a problem because the vEPC can simply initiate the re-attach event in this case.

In Figure 11 step 3, we also see the second kind of data dependency between the handover event and the tracking area update (TAU) event. The TAU event waits for the handover event to finish before it starts its execution. Such a dependency between two events is mitigated when events execute in blocking mode (see Figure 5.1.3.2.2.7.1: EMM main states in the UE [8]. The Figure shows that no other event can execute if one event is being executed).

We follow the above procedure for the other two critical events, i.e. service request event and paging event. At this step, we have working Fat-Proxy for all three critical events.

3) Network protocols' logic-based partitioning The third and last step is about optimizing Fat-proxy execution (i.e. our second contribution). We recall that message execution delay of a critical event disrupts timely execution of that event; and to be worse its failure aborts the critical event procedure. Therefore, there is a need to speed-up event execution by executing some messages in parallel. We propose network protocols' *logic-based* partitioning to achieve this goal. We partition the mutually exclusive logic of different protocols in a module (module is event-based logic from one NF). We identify the opportunity of *logic-based* partitioning through analysis of these standardized protocols.

We explain this through *handover* event example. The *handover* event triggers coordination between a series of NFs (MME, SGW, and PGW). Such coordination takes place between different NFs within EPC, and between EPCs and the radio network (eNodeB). The MME NF requests eNodeB to establish secure radio connection with UE, instructs eNodeB to establish device context, initiates the connection between SGW NF for user uplink/downlink traffic, and many more. The signalling messages exchange between MME and eNodeB are carried out by S1AP protocol, while the communication between MME and other NF (i.e. SGW) is carried out using GTP protocol. The legacy EPC infrastructure tightly couples S1AP protocol with other protocols like GTP protocol in MME. However, these protocols are designed for different purposes and their messages do not interleave with each other. The MME's interaction with eNodeB via S1AP protocol is mutually exclusive to MME's communication with SGW through GTP protocol. Therefore, we partition the logic of S1AP, MME core function, and GTP protocols inside MME NF's module. This design choice results in faster communication between eNodeB and vEPC using S1AP protocol, and between vEPC's MME and SGW NFs using GTP protocol, even during data-center network congestion.

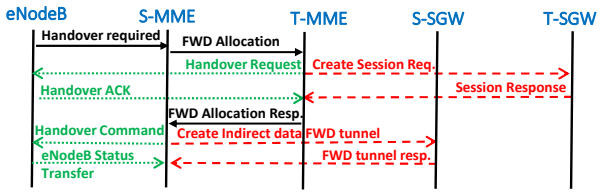


Figure 12: Concurrent execution of messages: S1AP messages (shown as dotted line arrows) and GTP messages (shown as dashed line arrows) are executed in parallel to each other. Sequence execution of messages (S1AP and GTP) are shown as solid line

A protocol defines message exchange procedure between different entities. Figure 12 shows subset of these messages exchanged between EPC NFs during *handover* event. $\text{MME}_{serving}$ (S-MME) receives *Handover Required* message from eNodeB and triggers *Forward Allocation* message to MME_{target} (T-MME). After receiving, T-MME sends "Create Session Request" to SGW_{target} (T-SGW). On successful session response from T-SGW, T-MME sends *Handover Acknowledgement* to eNodeB. Note that the direction of "Create Session Request" and "Handover Request" messages are opposite (both messages are sent simultaneously), where former is a part of GTP protocol, and the latter is a part of S1AP protocol. The messages shown as dotted and dashed lines in Figure 12 are executed in parallel. The simultaneous transmission of these messages is possible since their respective protocols are mutually exclusive. We accelerate handover event (which we chose as an example) by identifying the protocol level modularity in a NF and execute their corresponding messages in parallel. We find that in most network events, there exists 40% to 60% messages that can be executed concurrently to significantly improve the network performance.

There is a chance that concurrent message execution may fail and this failure may provide inconsistent view of network states to eNodeB. For example, on receiving the "Handover Request" message from T-MME, the eNodeB believes that the T-MME has successfully established the device connection with T-SGW. But eNodeB's network states may get incorrect, in case T-MME has failed to establish device session with T-SGW. To handle such network state inconsistencies, we propose transaction rollback by sending network failure message. As stated earlier, message execution failure at any step terminates the entire handover process, therefore, by sending a failure message (even at later step) to eNodeB addresses any inconsistency previously caused by concurrent message execution. We should highlight that network states within different NFs of vEPC remain consistent because handover event executes in blocking mode. Moreover, we partition only mutually exclusive messages of two different protocols. Our design is robust where parallel execution of these messages produce the same result as their sequential execution.

VI. System Implementation

Our system implementation consists of OpenEPC LTE implementation and LTE EPC NFs virtualization.

OpenEPC LTE deployment: Our test-bed consists of LTE eNodeB (nanoLTE Access Point [18]), OpenEPC software EPC platform [15], and Samsung S6 smartphones. The eNodeB is a 3GPP Release 9 compliant LTE small cell on 700 MHz band. Considerable effort, involving code modifications to OpenEPC components, is made to integrate eNodeB (closed-source) with EPC to ensure interoperability with commercially available LTE clients (i.e. Samsung S6 smartphones). Our EPC network consists of MME, HSS, PCRF for control plane, and SGW and PGW for data plane functions. In addition, the Internet gateway provides connectivity to the Internet. Samsung S6 smartphones use USIM cards programmed with the appropriate identification name and secret code to connect with eNodeB. Since eNodeB and the device communicate on T-Mobile's licensed band, we use custom built frequency converters. These converters convert the frequency in both downlink and uplink from 700 MHz to 2.6 GHz, where we have an experimental license to conduct over the air experiments.

For the evaluation of our second design choice (protocols' logic partitioning), where S1AP-MME simultaneously communicates with S1AP-eNodeB and SGW, we require changes at eNodeB-S1AP. Because our eNodeB is closed-source, we use device emulation provided by OpenEPC. The OpenEPC provides *client-Alice* module that emulates user device and eNodeB and interacts with EPC NFs. The *client-Alice* module has basic S1-AP functionality, enough to show performance improvement when protocols' logic partitioning is used.

Virtualizing LTE EPC: After LTE testbed deployment, we virtualize EPC NFs. EPC virtualization includes deployment of decomposed EPC NFs over VMs, and exposing them to real LTE traffic load.

EPC's NFs decomposition and placement: We virtualize EPC NFs over vMware's vSphere, which is a server virtualization platform with consistent management. We first decompose OpenEPC into a number of LTE NFs (i.e. MME, SGW, PGW, HSS, and PCRF). To implement an event-specific Fat-proxy, we first identify the Fat-proxy's logic based on the event's state transition diagram (as shown in Figure 11). We then traverse through OpenEPC NFs source code to locate that implementation logic. This event logic is deployed on a separate VM (after extracting/copying from OpenEPC NFs) and we call it that event's Fat-proxy. Now this Fat-proxy's VM acts as standalone NF. Thereafter, we configure the interfaces for all virtual NFs (LTE NFs as well as Fat-proxy) by changing OpenEPC boot process (init) so that the OpenEPC can discover installed Fat-proxies at start-up and allows relaying packets to and from these virtual NFs (VNFs).

During actual execution of a critical event, OpenEPC stores most of the information needed by a NF (such as device states) in a back-end database. To reduce the overhead of Fat-proxy communicating with the database back-and-forth, we duplicate the device states at Fat-proxy when the critical event triggers. We gather results by changing testbed configurations for two different settings: (a) placing the VNFs (with no Fat-proxy) over different servers, which are then connected through network tunnel, and (b) installing the Fat-proxy VNF within one server.

Considering real data-center network loads: Because research lab's testbed environment does not (a) add round trip time to data-center network (b) consider dynamic loads at servers (c) take data-center network congestion into account; we consider data-center network performance metrics while compiling our results. We parsed system logs provided by HP Helion cloud infrastructure and gather inter-data-center network latency metrics. We measure round trip time from query entering and exiting the data-center network. We understand, it is challenging to precisely find the root cause of latency for each query [19]. Therefore, we apply moving average to cancel random variation in our result.

VII. Evaluation and Results

We evaluate our design based on following aspects: (1) controlling signalling storm, (2) timely execution of mission critical events, and (3) performance impact. We compare our approach against contemporary instance based vEPC LTE design, which is not only deployed by a number of network operators [9], but also discussed in recent research papers [20]–[22]. We run our tests on a local network of servers with 10-core Intel Xeon E5 - 2650 v3 processors at 2.3Ghz, 25MB cache size, and 16GB memory. We build our prototype and tested it using real smartphones (Samsung S6 smartphones) and device emulation mode of OpenEPC. To consider real operator network scenario, we use a network trace as our input packet stream; results are representative of tests we run using these traces.

Signalling load at vEPC: As mentioned earlier in the paper, during busy hours, operational LTE core is exposed to signalling storm. First, we show that our design reduces signalling storm by diverting highly occurring network events to Fat-proxy. Although, Fat-proxy is part of LTE core, all execution remains local to Fat-proxy NF. In this way, LTE core NFs (such as MME, SGW ad PGW) are not exposed to high signalling messages exchange and remain functional at all times. Figure 14 shows that for *handover*[3], *service request* and *paging* events, LTE core is exposed to 5X, 6X and 2X less signalling traffic respectively, compared to when Fat-proxy is not used. We see that *paging* event benefits the most from our design which is due to the fact that paging Fat-proxy handles all the paging signalling with eNodeB directly after MME delegates paging execution to Fat-proxy. Whereas, *service request* event requires bearer modifications at actual SGW and PGW which relatively increases vEPC signalling load even in case of *Service Request* Fat-proxy.

Total number of signalling messages: We show that less number of signalling messages are generated by each event with Fat-proxy compared to the case when Fat-proxy is not used. This is mostly because we are able to execute some

[3]Handover event is induced through OpenEPC command line script.

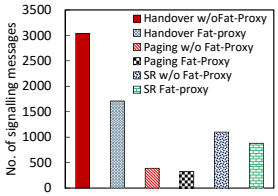
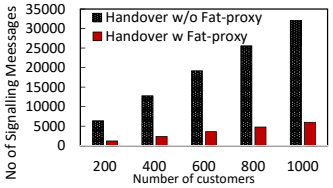
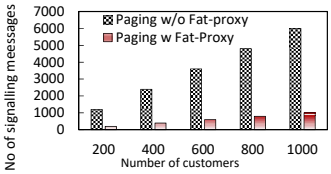


Figure 13: Total no of signalling mesgs per subscrb/hr



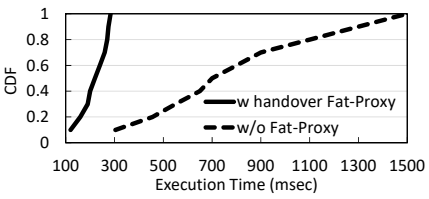
(a) Handover

(b) Service Request

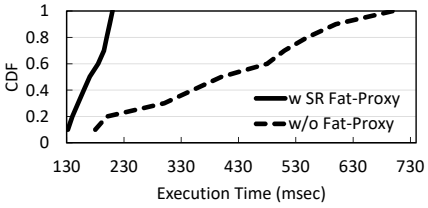


(c) Paging

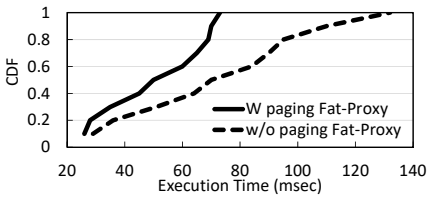
Figure 14: Signalling load vEPC is exposed during peak hours with and without Fat-proxy



(a) Handover



(b) Service Request



(c) Paging

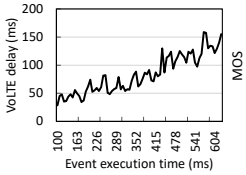
Figure 15: Event execution time during peak hours with and without Fat-Proxy concept

messages in parallel, and also skip few messages from being executed. Figure 13 shows the reduction of signalling message per event (in one hour window) when Fat-proxy is used. Note that, we are mainly interested in determing signalling load in vEPC. Therefore, we count two parallel messages as one, but in actual implementation exactly two messages are generated. The rationale of treating pair of parallel message as one is that these two messages are traveling in opposite direction, i.e. one out of EPC and the other towards EPC NF. Therefore, both of these messages are independent to each other execution. Figure 13 shows that *handover* event produces around 40% less signalling messages, when *handover* event is handled by Fat-proxy. This is mainly because vEPC NFs modules (specific to handover event) implemented in Fat-proxy are local to Fat-proxy. Therefore, these modules do not need to use signalling messages to communicate with each other and avoid unnecessary signalling exchange. The signalling messages that *handover* Fat-proxy skips include *create session request/response*[4], *delete session request/response*[5], *UE context release command/complete*[5], *delete indirect data forwarding tunnel request/response*[5]. Figure 13 shows that *paging* event can only skip one message of *Uplink-Nas-Transport*. This NAS message carries the information about the service that device wants to receive from LTE network.

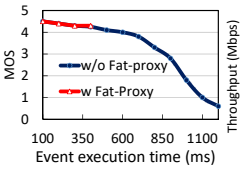
Event execution time: Figure 15 shows CDF of event execution time for *handover*, *service request* and *paging* events with and without Fat-proxy implementation. We see that with *handover* Fat-proxy event latency decreases by the factor of 6X on average. This improvement is observed because (1) all events execution remains local to *handover* Fat-proxy and does not suffer any network delays, and (2) *handover* Fat-proxy executes 6 signalling messages in parallel and skips total of 8 messages. We note that even with *handover* Fat-proxy, handover latency is higher than 100 ms. This is because in our experiment we handle worst case handover scenario in which both MME and SGW are relocated. Although event completion time exceeds data packets QoS time-bounds (100msec for voice over LTE call, and 300msec for TCP based traffic [2]), it does not affect user QoS experience where users' data packets are tunneled from old serving SGW to target SGW and then delivered to user.

Service Request (SR) Fat-proxy, on average can reduce only upto 50% *service request* event execution time mainly because at the end of *service request* event execution, SR Fat-proxy needs to update device bearers with vEPC.

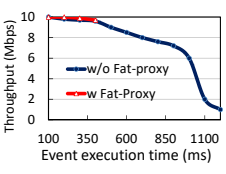
Paging event execution time with and without Fat-proxy is not significantly high. We observe in *paging* event, all of the signalling messages are exchanged between S1APs of MME and eNodeB which diminish intra-vEPC NFs delay. The improvement we see in Figure 15c is mainly achieved by pushing S1AP to the edge of cloud and executing one pair of message in parallel.



(a) VoLTE delay



(b) Mean opinion score



(c) Throughput

Figure 16: Impact of event execution time on voice and data applications

Voice and data applications performance: We are interested to find how an event execution time impact voice and data applications. We deploy voice over LTE (VoLTE) and TCP applications to test voice and data throughput, respectively. We consider *handover* event execution scenario. Figure 16a shows that VoLTE delay increases as the event execution time increases. The VoLTE remains within its QoS requirement of 100ms as far as event execution time remains below 450ms. Even if the event execution time is more than 100ms , the VoLTE packets are kept forwarded via tunnel from eNodeB_{source} to eNodeB_{target}. However, VoLTE packets delay start increasing as *handover* takes more time to complete mainly because of the resource contention at vEPC. To show how VoLTE delay converts into voice quality of service, we

[4]Conventionally it is used to communicate with two distant NFs.

[5]These messages are used to remove states from memory. In Fat-Proxy the states are automatically deleted when the Fat-Proxy responds back to vEPC.

consider Mean Opinion Score (MOS) – a measure of voice quality, as shown in Figure 16b. We use AQuA software tool [23] to calculate MOS value for VoLTE voice. The MOS value are ranged from 1 for unacceptable to 5 for excellent voice quality. VoLTE calls often are in the 3.5 to 4.2 range. We find that when event execution time increases beyond 800msec, the voice quality degrades from fair (MOS value 3) to poor (MOS value 2) and then to unacceptable (MOS value 1) at 1000msec. Because *handover* Fat-Proxy execution always remains below 400msec, therefore, our design provides higher voice quality even during peak load. Figure 16c shows the TCP throughput under event execution time. The throughput exponentially degrades from 6Mbps to 2Mbps when an event execution time goes over 850ms. Through our Fat-Proxy design, the throughput remains stable and does not degrade.

VIII. Related Work

ETSI has provided several documents discussing guidelines and requirements for LTE-NFV [9]. There are several white papers [10] by technology giants, but none of them has demonstrated any system design of LTE-NFV that solves LTE specific issues in virtualized environment.

Closest to our work are CoMb [24], OpenBox [25], E2 [26], OpenNF [27], DPCM [28], and PEPC [29]. CoMb [24] uses network controller that assigns processing responsibilities across the network. OpenBox [25] decouples the control plane of NFs from their data plane. E2 [26] is an application-agnostic scheduling framework for packet processing. OpenNF [27] provides a control plane architecture that allows quick reallocation of flows across NF instances. Our work differs from these by bringing innovation in the method used to decompose the vNF and the algorithm to determine what to consolidate. Our solution is tailored to LTE vEPC design. DPCM [28] proposes low latency LTE data access approach for service request, handover and roaming scenarios. It runs location update procedure parallel to data-plane forwarding. Our work differs from DPCM [28] where we parallelize mutually exclusive signalling messages within a control-plane procedure, rather than parallelizing two different procedures. Moreover, our procedure gracefully handles parallel messages failure and ensures network states consistency among different NFs. PEPC [29] highlights that device states are duplicated across multiple NFs. PEPC [29] consolidates these states to amplify EPC performance. In contrast, our work does not consolidate device states; instead our design brings control-plane events execution logic closer to speed-up their execution.

IX. Conclusion

In this paper, we disclose new major issues in virtualizing LTE core which mainly arise from *instance-based* NFV design. We propose a new way of thinking to virtualize LTE core so that LTE events are executed within time-bounds. We leverage the *logic-based* modularity of NFs, decompose the NFs based on events logic and assemble them into a Fat-proxy. This Fat-proxy takes the message-intensive critical events away from the core network. We further speed up event execution by executing event messages in parallel.

Future work: One of the future research is to co-relate device activity with different events. By doing so, vEPC would be able to predict forthcoming device action well in advance and can thus prepare the resources. Such an approach has promise to improve both (1) device performance and (2) vEPC design.

Acknowledgement

This work was done when the first author was an intern at Hewlett Packard Labs. We thank Fatima Muhammad Anwar at UCLA whose writing and paper presentation inputs made this submission possible. We thank our shepherd, Eric Rozner, and the anonymous reviewers for providing excellent feedback.

References

[1] T. Benson and et al. Network traffic characteristics of data centers in the wilds. In *ACM IMC, 2010*.

[2] 3GPP. TS 23.203: Policy and Charging Control Architecture, 2013.

[3] 3GPP. TS29.281: GPRS Tunnelling Protocol User Plane, year = 2013,.

[4] A. Lucent. LTE Subscriber Service Restoration. In *Tech. Report*, 2014.

[5] NSN. Signaling is growing 50% faster than data traffic. In *Tech. Report*, 2012.

[6] A. Banerjee and et al. Scaling the LTE control-plane for future mobile access. In *ACM CoNEXT*, 2015.

[7] J. H. Howard and et al. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS)*, 1988.

[8] 3GPP. TS24.301: Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3, Jun. 2013.

[9] Carrier NFV Project - One Year Milestone. http://www.layer123.com/download&doc=ETSI-1013-Lopez-NFV_One_Year_Milestone.

[10] Vendor documents: NFV World Congress. https://www.layer123.com/current-events/.

[11] MME Trigger OVERLOAD START Towards ENodeB. http://tech.queryhome.com/103073/different-situations-trigger-overload-start-towards-enodeb.

[12] Mittal and et al. TIMELY: RTT-based congestion control for the datacenter. In *ACM SIGCOMM*, 2015.

[13] Ericsson SGSN-MME. http://www.ericsson.com/ourportfolio/products/sgsn-mme?nav=productcategory004%7Cfgb_101_256/.

[14] 3GPP. TS29.274: Tunnelling Protocol for Control plane, year = 2014,.

[15] Open EPC - open source LTE implementation. http://www.openepc.net/.

[16] Run-time function call tree with gcc. http://ndevilla.free.fr/etrace/.

[17] E. B. Krissinel and K. Henrick. Common subgraph isomorphism detection by backtracking search. *Software: Practice and Experience*, 34(6):591–607, 2004.

[18] nanoLTE Access Points. http://www.ipaccess.com/en/lte/.

[19] Y. Zhu and et al. Packet-level telemetry in large datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 479–491, 2015.

[20] A. Basta and et al. Applying NFV and SDN to LTE mobile core gateways, the functions placement problem. *ACM All things cellular*, 2014.

[21] A. Chiu and et al. EdgePlex: Decomposing the provider edge for flexibilty and reliability. *ACM SIGCOMM Symposium on SDN*, 2015.

[22] W. Hahn and B. Gajic. GW elasticity in data centers: Options to adapt to changing traffic profiles in control and user plane. *IEEE ICIN*, 2015.

[23] AQuA – Audio QUality Analyzer. http://sevana.biz/products/aqua/technology/.

[24] V. Sekar and et al. Design and implementation of a consolidated middlebox architecture. In *USENIX NSDI*, 2012.

[25] B.-B. Anat and et al. OpenBox: a software-defined framework for developing, deploying, and managing NFs. In *ACM SIGCOMM*, 2016.

[26] P. Shoumik and et al. E2: a framework for NFV applications. In *ACM SOSP Conference*, 2015.

[27] A. Gember-Jacobson and et al. OpenNF: Enabling innovation in network function control. *ACM SIGCOMM CCR*, 44(4):163–174, 2015.

[28] Y. Li and et al. A Control-Plane Perspective on Reducing Data Access Latency in LTE Networks. In *ACM MobiCom*, 2017.

[29] Z. A. Qazi and et al. A High Performance Packet Core for Next Generation Cellular Networks. In *ACM Sigcomm*, 2017.