MMLite: A Scalable and Resource Efficient Control Plane for Next Generation Cellular Packet Core

Vasudevan Nagendra
Stony Brook University
vnagendra@cs.stonybrook.edu

Arani Bhattacharya
Stony Brook University
arbhattachar@cs.stonybrook.edu

Anshul Gandhi
Stony Brook University
anshul@cs.stonybrook.edu

Samir R. Das
Stony Brook University
samir@cs.stonybrook.edu

Abstract

With increase in cellular-enabled IoT devices having diverse traffic characteristics and service level objectives (SLOs), handling the control traffic in a scalable and resource-efficient manner in the cellular packet core network is critical. The traditional monolithic design of the cellular core adopted by service-providers is inflexible with respect to the diverse requirements and bursty loads of IoT devices, specifically for properties such as *elasticity*, *customizability*, and *scalability*. To address this key challenge, we focus on the most critical control plane component of the cellular packet core network, the *Mobility Management Entity (MME)*. We present `MMLite`, a *functionally decomposed* and *stateless* MME design wherein individual control procedures are implemented as microservices and states are decoupled from their processing, thus enabling elasticity and fault tolerance. For SLO compliance, we develop a *multi-level load balancing* approach based on *skewed consistent hashing* to efficiently distribute incoming connections. We evaluate the performance benefits of `MMLite` over existing approaches with respect to scaling, fault tolerance, SLO compliance and resource efficiency.

CCS Concepts

• **Networks → Middle boxes / network appliances**; **Packet scheduling**; **Network performance analysis**;

Keywords

Cellular Networks, EPC, NFV, MME, Functional Customization, Microservices, Load Balancing.

SOSR '19, April 3–4, 2019, San Jose, CA, USA

ACM ISBN 978-1-4503-6710-3/19/04…$15.00
https://doi.org/10.1145/3314148.3314345

1 INTRODUCTION

One of the grand challenges in the design of future cellular core network is its *resource-efficient scaling* with the projected growth of signaling or control traffic. Much of this growth is expected to come from the tremendous rise in IoT devices (≈12 billion by 2022 [14, 17]). Compared to traditional smartphones, IoT devices generate at least twice the volume of control messages, growing 50% faster than data traffic [3, 15, 27, 43, 63]. This represents a significant overhead as control messages do not directly contribute to the service provider's revenue. Moreover, the traffic characteristics and performance requirements of cellular-based IoT devices have much greater diversity than traditional user equipments (UEs) like smartphones or laptops [21, 35, 63]. Efficiently managing resources in the presence of this diverse traffic is challenging [41, 52].

An immediate concern now is the scalability and efficient resource utilization in the cellular core network (also called *Evolved Packet Core* or EPC in connection with LTE networks). Our focus in this work is on the *Mobility Management Entity* or MME, which is the most intensive control plane component that handles five times more control messages than any other entity in EPC [45, 70]. Designing an efficient and scalable MME requires addressing at least the following key challenges:

(1) *Elasticity*: IoT applications create bursty traffic [29, 63], necessitating dynamic capacity provisioning. Insufficient capacity at MME may lead to connection failures and rejections, triggering retry messages and further increasing the load on the MME [21]. Worse, UEs and all entities inside the EPC maintain stateful contextual information (*static bindings*), making it difficult to migrate connections to other MMEs in case of scale-out or scale-in. Not surprisingly, the current practice is to simply over-provision the MME, resulting in an expensive and wasteful design [54, 65].

(2) *Flexibility*: IoT devices can have very different control and data traffic characteristics and performance requirements [21, 35, 52, 63, 68]. For example, IoT devices in smart cars require stringent Service Level Objectives or SLOs to react to changing traffic conditions, while smart home IoT devices may simply require IP connectivity. Unfortunately, today's cellular networks make use of monolithic MME devices which are rigid and do not offer any functional or performance flexibility. Even recently proposed network function

virtualization (NFV) based EPC architectures [8, 11, 41, 54, 55] lack the capabilities to handle differentiated SLO goals.

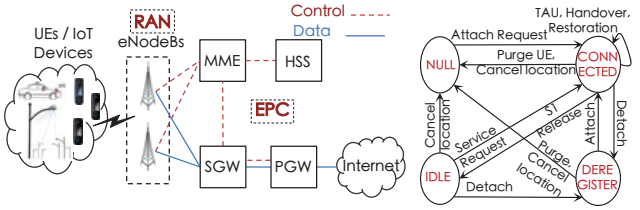
(3) *Scalability*: A key bottleneck for large-scale networks is the centralized load balancing mechanism that must immediately assign incoming connections to an MME. Given the heterogeneity in the entire ecosystem, traditional approaches such as round-robin or least-used servers is no longer effective [32]. While recent approaches based on consistent hashing (CH) distribute connections uniformly [4], they are unable to quickly scale resources in response to bursty IoT traffic (i.e., unaware of the actual traffic load conditions at the MME), making them vulnerable to "hot spots", where a few MME hosts are overloaded (discussed in §8). Meeting user-specified SLOs while being scalable and resource efficient thus requires a careful reconsideration of load balancing decisions in the network.

To address the above challenges, we propose `MMLite`, an agile MME architecture that exploits recent advances in NFV. The key enabler of `MMLite` is its *stateless* and *functionally decomposed* design. The statelessness is achieved by externalizing each UE-specific state in shared memory inside the MME host, thus decoupling the MME from the UE contextual information. This stateless design enables fault tolerance and dynamic provisioning of MMEs responsive to traffic changes, without incurring the overhead of state migration.

To address customizability, we *decompose* the MME functionality into a set of microservices (or NFs) based on the specific control procedure they handle, such as attach, service, handover, migrate, etc. This control procedure-specific decomposition, facilitated by our stateless design, allows us to cater to specific functional and SLO requirements of individual UEs in a resource-efficient manner. This is in contrast to existing protocol-based decomposition approaches [36, 61] that allow flexibility but fail to provide fine-grained (UE-specific) SLO control.

To address elasticity and scalability, we re-architect the MME into a multi-stage forwarding engine that divides the MME functionality into MME-load balancer, MME-forwarder and stateless MME processing entity. Specifically, we introduce a multi-level, SLO-aware MME load balancer and MME forwarder that optimizes the resource utilization within and across MME hosts. Unlike existing approaches that aim to balance connections across MMEs [4, 12, 32], we purposely unbalance load to meet SLO requirements and facilitate dynamic scaling. We evaluate the benefits of our SLO-aware load balancer in the context of stateless and functionally decomposed MMEs, and contrast it with traditional stateful models.

We implement `MMLite` using the DPDK [24] and OpenNetVM [48] frameworks that provide high performance and low latency packet processing capabilities. We exploit the zero copy capabilities of DPDK to build stateless MME microservices (by externalizing their states from processing) and fast path components such as MME load balancer and MME forwarder. Our `MMLite` architecture is compatible with



(a) LTE architecture with key components. (b) MME state machine.

Figure 1: Overview of LTE architecture and MME states.

3GPP protocols, making it incrementally deployable in the existing cellular packet core network.

Our experimental results show that `MMLite` provides much higher throughput compared to existing open-source frameworks, including OAI [13, 47] and OpenEPC [69]. Further, while `MMLite` provides a raw throughput at par with the stateful design (implemented on DPDK platform) under stable traffic conditions, we outperform the stateful design by about 18.4% under bursty traffic. Importantly, `MMLite` can satisfy stringent SLO requirements, provides near-optimal load balancing (within 1%), and results in better resource utilization (by about 30–50%) compared to existing approaches, all without adding significant overhead.

In summary, we make the following contributions:

- We demonstrate the performance limitations of the current generation monolithic, stateful MME design (§2.2).
- We present the design of a stateless and functionally customizable MME that provides service differentiation while handling the diverse traffic of cellular IoT devices (§4).
- We develop a multi-level load balancing approach based on skewed consistent hashing that meets SLO requirements and facilitates dynamic scaling of MME (§5).
- We build a prototype of our MME using DPDK and OpenNetVM (§6), and demonstrate our performance benefits over existing and recently proposed frameworks (§7).

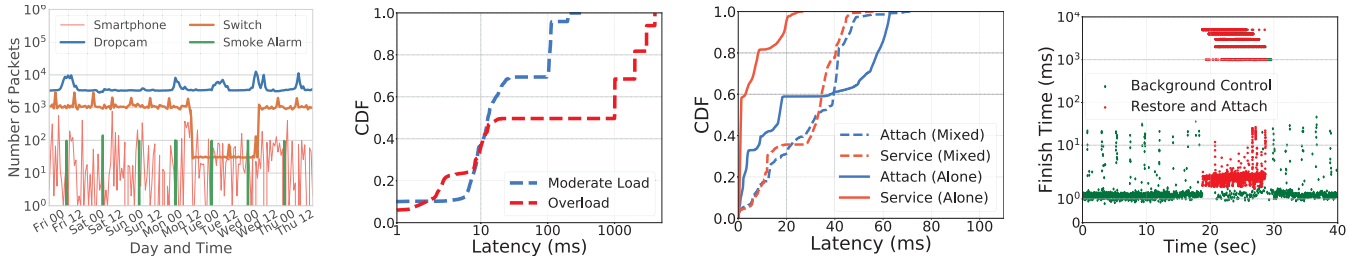
2 BACKGROUND & MOTIVATION

This section provides a brief primer on the current generation LTE architecture and highlights the key challenges involved.

2.1 Overview of LTE Architecture

Present day LTE network has two main components: Radio Access Network (RAN) and Evolved Packet Core (EPC); see Figure 1a. The User Equipment[1] (UE) communicates with Internet through eNodeB (another name for the Base Station) of RAN via EPC. The Serving Gateway (SGW) is responsible for forwarding the packets between the eNodeB and Packet data network Gateway (PGW). PGW provides packet data services to UE such as QoS (Quality of Service), packet filtering services, and billing. The Mobility Management Entity (MME) acts as the centralized controller module providing control plane functionality, such as establish and release data sessions, verifies the subscription details of a user and maintains the control channel with eNodeB for exchanging the information.

[1]We use User Equipment and Cellular-enabled IoT device interchangeably.



(a) Diversity in cellular-enabled IoT and UE traffic characteristics. **(b) Delays in data transfer instantiation during MME overload.** **(c) Interference among LTE control procedures at line rate traffic.** **(d) MME failure & snowball effect during session restoration.**

Figure 2: IoT traffic characteristics and experiments demonstrating the limitations of existing stateful MME design on DPDK-based industrial-grade prototype.

MME Control Procedures and Bindings: Figure 1b shows the key MME states and the control procedures associated with each state. The *Attach request* is issued by UE to register itself with EPC for Internet connectivity, which is an infrequent procedure that is invoked a few times per day per UE. The *Service request* procedure is performed when an inactive UE in Idle state wishes to send or receive data. This is a frequent procedure in LTE [34]), especially for IoT devices [15, 27, 63]. The *Handover* procedure handles the mobility and *TAU* (Tracking Area Updates) procedure is responsible for migrating each device's associated states to other MMEs, and MME scaling (in response to overload or failures). In an ideal scenario, these procedures on an average takes few hundred milliseconds to complete; in case of overload or failures, these procedures can take *seconds or at times even minutes* to complete [5, 30, 42].

The UE and MME retain these association details (called *static bindings or associations*) until the UE is completely detached from the core network. In this association, each UE maintains identifiers such as GUTI (Globally Unique Temporary UE Identifier), which contains: (*i*) TMSI (Temporary Mobile Subscriber Identification) for temporarily identifying the current UE session until it is detached from the network, and (*ii*) MME identifier (i.e., MMEID-UE-S1AP). Similarly, the MME maintains the necessary contextual state information specific to that connection (such as authentication and security keys), TMSI, and other session details. The TMSI and MMEID information is used by MME to subsequently identify the UE connection. These static bindings *hinder* elastic scaling. This is because only the MME host that has the state for a specific UE can handle all control signaling for the UE.

2.2 Challenges and Motivation

Several characteristics of IoT devices pose a challenge for MME design: (*i*) the diverse nature of IoT traffic, including traffic that is sporadic, periodic, high-frequency, and bursty [63, 68]; Figure 2a illustrates this diversity in control traffic for specific IoT devices obtained from real data [25] in comparison with UE traffic [38], (*ii*) the different SLO requirements of IoT devices depending on their functionality, and (*iii*) scale – a large number of IoT devices must be supported by the MME at low per-unit cost. We now discuss the key performance challenges that must be addressed by MME, thus motivating our work.

To evaluate the limitations of cellular core, we build a DPDK-based traditional stateful MME architecture. We generate IoT-based cellular control traffic by replaying real-world packet capture traces from publicly available data sets and LTE control procedures following specific traffic distributions [25, 38] in our local testbed using LTE UE emulator [37] (details in §6 & §7). While prior work has analyzed data plane performance issues using DPDK-based EPC implementation [55, 57], our work focuses on *control plane* performance issues, as discussed below.

Overload Protection: The bursty nature of IoT traffic [29, 63] can lead to frequent MME overload. Current overload protection methods [7, 9] include: 1) *Migrating connections* from an overloaded MME to other lightly loaded MMEs, and 2) *Rate throttling* at the overloaded MME by dropping or rejecting control messages beyond a certain limit. Unfortunately, both approaches incur significant overhead due to the stateful static bindings in MME. Figure 2b highlights the significant increase (by almost 50×) in data transfer instantiation times when an overloaded MME operating at 80% CPU utilization attempts to migrate some of its connections to another MME (experimental setup detailed in §7); note the log scale on the x-axis. This is because of the large number of control messages that are needed to reestablish the UE's context following a connection migration request. We observe similar results using the *rate-throttling* approach due to the reconnection attempts made by the UEs following dropped messages. Similar issues also arise in the case of scale-in, in response to low load.

Functional Decomposition and Isolation: Control procedures from one device can interfere with the processing of control procedures from other devices, resulting in unpredictable SLO violations. Figure 2c illustrates the impact of this interference. Here, the heavier *Attach Request* increases the latency of the lighter *Service Request* by over 50% (compared to isolated execution). Thus, an IoT device with frequent *Service Request* messages can affect the performance of other IoT devices or UEs that share the same MME. The need for performance isolation between device traffic is particularly important for use cases such as virtual reality (VR) or smart-transportation networks, where latency requirements are in the 1–10 millisecond range [71].

SLO-aware Load Balancing: Common approaches for distributing UE connections among MMEs to improve performance include round robin and consistent hashing [4, 32]. Due to the diversity in IoT traffic, however, the load on individual connections can be vastly different, resulting in hot spots and SLO violations. We show, in §7, that the above approaches can lead to more than 20% SLO violations in the presence of UE and diverse IoT traffic loads.

Fault Tolerance: MME failures may cause service outages of up to tens of minutes [42]. Current LTE networks address fault tolerance in two ways [23, 33]: (*i*) Active-Passive High Availability via $N + 1$ (N Active, 1 Passive) resiliency, and (*ii*) session restoration procedures. The $N + 1$ resiliency approach requires additional hardware and cannot scalably handle multiple MME failures. In the second approach, a *session restoration server* maintains UE session information of each of the MMEs [33]. In case of MME failures, the session restoration server redistributes the UE session information pertaining to the failed MME servers among other active MMEs. The active MMEs trigger the affected UEs to re-associate with new MME servers through session restoration procedures, resulting in the flooding of the core network with a large number of control procedures. Any session restoration procedure that fails to complete within 5s will retry with reattach procedures, resulting in additional attach floods (*snowball effect* [21]). Figure 2d illustrates this effect, showing that UEs can take seconds, and even minutes, to reattach to the network; note the log-scale on the y-axis.

3 SYSTEM OVERVIEW

From above description, it is apparent that there are two core issues: (1) the stateful nature of MME and the static binding it engenders significantly impact performance when moving UE connections across MMEs, and (2) the current monolithic design of MME is contrary to the need for functional decomposition and performance isolation. Our `MMLite` architecture thus fundamentally uses two core design principles - (*i*) *Statelessness*, and (*ii*) use of *functionally decomposed microservices*. These principles are used in conjunction with *slicing* - a unit of physical resource that procedures needing specific SLO requirements are mapped to.

The `MMLite` architecture introduces the following functional components, as shown in Figure 3:

- *Stateless MME Microservices*: We decompose the MME functionality into individual network functions (NFs) that handle specific LTE control procedures; these NFs are implemented as *microservices*. The state is maintained externally making these microservices *stateless* (see §4).
- *Slices*: The microservices are bundled into 'slices', with each slice hosting a given number of microservices for different control procedures. Slices are units of physical resources, such as a fraction of a logical execution unit of a processor (lcore [58] in DPDK-speak). Multiple slices can run on a single MME host machine, and many such hosts may exist within the carrier's datacenter.

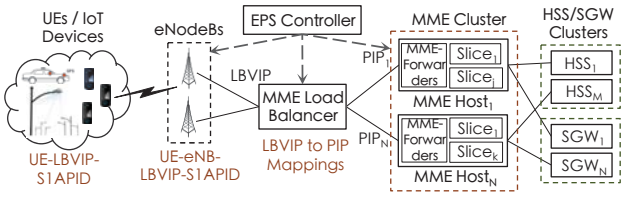


Figure 3: `MMLite`: LTE cellular EPC system architecture.

- *Load Balancing*: The load balancer leverages the above components to enforce SLO compliance in a resource-efficient manner. It has two functional components: (*i*) *MME Load-Balancer*: An external entity that distributes control messages from UEs across multiple MME hosts on the basis of their resource and SLO requirements; and (*ii*) *MME Forwarder*: An NF-based forwarding entity on each MME host that distributes control messages to NFs in a slice-aware manner and on the basis of the SLO requirements of UE's control connections. The details of our inter-host and intra-host load balancing are presented in §5.

For providing the necessary logic and infrastructure support for the above mentioned components, `MMLite` supports two different controllers: (*i*) *NF Controller*: A controller local to each MME host to manage the externalized states, state migration, and NF scaling; and (*ii*) *EPS Controller*: A centralized controller that manages the MME hosts scaling on the basis of SLO violations and resource requirements.

Overview of Operation: All control procedures are implemented as independent microservices with states externalized in the shared memory inside the MME hosts. The load balancer steers all control packets to the right MME host. The EPS controller then helps the `MMLite` architecture to evaluate the performance of the control messages served at different hosts and scale accordingly. Inside each MME host, a dedicated forwarder and a set of NFs are assigned to each slice. Control messages for the same invocation of a control procedure are processed in the same slice; this is tracked using `sliceID` by the forwarder. The NF controller manages the NF scaling and uses NF prioritization for SLO compliance. We discuss the key components of `MMLite` in detail in the following sections.

4 STATELESS MME MICROSERVICES

In this section, we describe the core of our MME design that provides the key functionalities of functional decomposition and elasticity. These are achieved by decomposing the MME NFs into dedicated control procedure-specific microservices and by decoupling the states from the NFs to make the MME stateless. We also discuss our design for state migration and fault tolerance, which makes `MMLite` more robust in practice. The description below refers to Figure 4 that describes our architectural components.

4.1 Functional Decomposition

As discussed in Section 2.2, the bursty IoT traffic results in NFs inducing interference among control procedures and possibly impacting critical messages. A popular approach to mitigate this issue is protocol-level decomposition, that

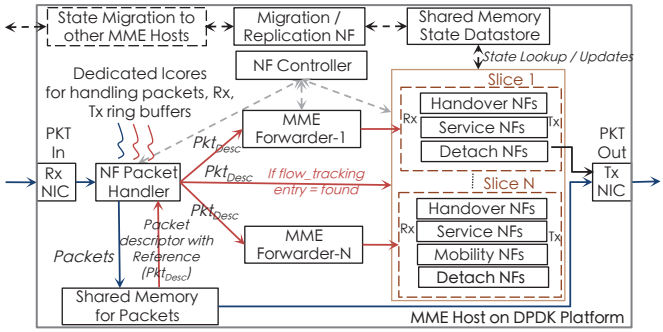


Figure 4: Internal system architecture of `MMLite` components running on a single MME host.

is, decomposing MME on the basis of the functional modules (i.e., code blocks) and protocol layers. For example, MME could be decomposed and pipelined as NAS security module, authentication modules, and S1AP layers separately with this technique. On the other hand, the devices have control traffic characteristics distributed over time with no two types of control procedures from a UE overlapping or arriving at the same time at MME. Therefore, decomposing the MME into microservices specific to control procedures (i.e., vertical decomposition) allows us to scale MME in fine-grained and resource efficient fashion in accordance with the control procedure inter-arrival time.

Our functional decomposition targets two issues. First, the control procedures that need to be invoked are temporally distributed in an unpredictable fashion. Some procedures such as *Attach* are infrequent, while *Service* procedures could be more frequent. The latter may even exhibit periodic, synchronous or semi-synchronous behavior (e.g., IoT sensors) [2]. Second, some IoT devices have limited functional needs and do not require certain types of control functions such as Handover, TAU-based state migration, and QoS procedures (e.g., stationary IoT sensors). Other IoT devices may be very dependent on certain types of control functions (e.g., IoT devices on smart transport platforms may invoke significant mobility related control functions). Mapping of *individual control procedures* to microservices (*procedure-level decomposition*) allows for specific microservices to be instantiated and independently provisioned depending on the load conditions and SLO requirements.

Thus, `MMLite` decomposes the traditional monolithic MME into following set of microservices targeted to handle specific control procedures: *a*) *Attach request*, *b*) *Service request*, *c*) *Detach request*, *d*) *Handover request*, and *e*) *State management* microservice-based NFs; we plan to support other control procedures as part of future work. This functional decomposition also enables seamless scaling and load balancing features by making it easy to place/move specific microservices, as discussed in §5.

4.2 Statelessness

`MMLite` externalizes the states of all MME NFs inside a host, i.e, the states are maintained *outside* the NF, in the shared memory of the MME host. We choose to store the states within each MME host, as opposed to a centralized data-store [59], because of frequent state updates triggered by control procedures. The state replication and migration procedure facilitates the necessary fault tolerance and scaling capabilities required for our architecture (see §4.3). We illustrate the benefits of this design choice in §7.1. We use *NF Controller* to allocate two shared memory pools – one for storing packets for zero-copy architectural support, and another for storing the UEs' states. These memory pools are later used by the MME NFs to store the packets and to get the UE context information.

When a control packet arrives at the NIC of the MME host: (*i*) The *NF Packet Handler* interfaces with the DPDK platform's poll mode driver to bypass the operating system to DMA, which can be readily accessed by all the NFs. (*ii*) The other *NF Packet Handler* threads access the packets stored in the shared memory to create a *packet descriptor* for each packet, which includes the handler to the packet in the shared memory and details on how the packet needs to be handled inside the host by different services. The *packet descriptors* are then placed onto the RX queues of the MME-forwarders for distributing the packets further across MME NFs depending on the slice the packet belongs to. This is facilitated by a set of hash tables maintained in the host that maps `sliceIDs` to forwarders, control message types to SLOs, and UEs (identified by TMSI) to their state.

4.3 State Migration

We invoke state migration across hosts, facilitated by the *state migration utility*, in the event of host failures or host scaling. The migration utility helps maintain up-to-date copies of the states on other hosts (replica hosts). Replication of states and number of replicas may be limited to those with tighter SLO requirements or other priorities in order to conserve resources. We leverage existing work [28, 48, 56] to develop a partitioning of the state space, and enhance this state partitioning specifically for our functionally decomposed and stateless NFs. In particular, we partition the state specific to each UE on the basis of the type of the control procedure and the lifetime of the state into: (i) *perpetual* (i.e., does not change during the lifetime of each control procedure), and (ii) *ephemeral* (i.e., state information that changes with each message exchanged within a control procedure). Upon completion of a control procedure, we discard obsolete state information (e.g., tunnel information, timers) and only retain necessary state that might be required for the next set of control procedures. This allows us to optimize and consolidate the state space, in contrast to existing approaches [4, 55].

The perpetual state is only migrated (across hosts) when the UE attaches or detaches to the network; this includes failures and scaling events. The ephemeral state is migrated more frequently via one of the following approaches:

(*i*) *Cold Migration*: The UE contextual information from within a MME host is migrated to other replica hosts only upon completion of the entire control procedure. Upon completing the procedure, the MME marks each UE

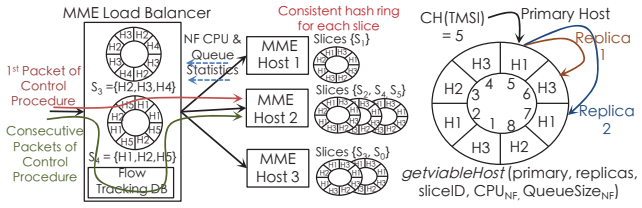


Figure 5: `MMLite` slice and SLO-aware MME LB architecture.

contextual information for migration. The state migration utility consolidates the states belonging to the same slice together and sends it to the other replica hosts.

(*ii*) *Hot Migration*: In case of hot migration, each time a message is handled by MME, the specific context (this is just a part of the state) is marked for migration because it could be updated. The state migration utility continuously polls and migrates only those parts of the state that are marked for migration. This approach provides better fault tolerance compared to cold migration while increasing the volume of migration traffic inside the core network.

4.4 Fault Tolerance and Scaling

As mentioned in §2.2, in case of conventional stateful MME failure, the UEs that are already attached to the failed MME host are redistributed to new MME hosts. This triggers an avalanche of restoration and reattach procedures to change UEs' MME associations, significantly affecting performance. `MMLite` avoids this with the following FT mechanisms.

(i) Host Failure: EPS Controller continuously monitors the MME hosts (i.e., with heartbeat messages) for failures and performs following tasks: (1) updates the load balancer about such MME host failures, and (2) provisions each slice of the failed MME host by adding the same amount of resources from existing MME hosts to the consistent hash ring of that slice. If existing set of hosts does not have sufficient resources, new MME hosts are brought up from an idle pool. If state replicas are available corresponding to the slices of the failed hosts, then these replicas are migrated to the appropriate (slice-aware) newly added hosts. Otherwise, the UEs must invoke reattach.

(ii) NF Failure: The NF controller helps to instantiate new MME NFs from an idle pool of NFs. The NF controller invokes the *NF Packet Handler* thread that is dedicated to handle the failure scenario. This handler registers an MME NF in the idle MME pool with the NF controller and reassigns the packets already in the RX queue of the failed NF to that of the new NF (note that the packets themselves are in shared memory).

5 LOAD BALANCING

To improve resource efficiency, we design an inter-host load balancer that determines the *slice- and resource-aware MME host* for the incoming packets. We then present our intra-host forwarder that selects the *SLO-aware MME NF* within the host for serving the packets.

ALGORITHM 1: Inter-Host Load Balancing.

```
1  pkt → sliceID, mmeID, GUTI.TMSI, msgID;
2  msgType ← getProcedureType(msgID);
3  track_entry ← Hash database of (LTMSI, hostID) ;
4  if track_entry[LTMSI] exists then
5      mmeHostID ← track_entry[LTMSI] ;
6  else if TMSI ≠ 0 then
7      if msgType ≠ "attach" then
8          key ← pkt.GUTI.TMSI;
9          sliceCHRef ← getSliceCH(sliceID);
10         hostReplicas ← lookupHosts(key, sliceCHRef);
11         hostMMEID ← getViableHost(hostReplicas, sliceID);
12         track_entry[LTMSI] = mmeHostID ;
13 else
       /* First message of Attach Procedure.        */;
14     LTMSI ← rand();
15     mmeHostID ← CHGetHost(LTMSI);
16     track_entry[LTMSI] = mmeHostID ;
17 mmeHost ← getHost(mmeHostID);
18 send packet to mmeHost;
```

5.1 Inter-Host MME Load Balancer

Our inter-host load balancer uses a *skewed consistent hashing* (SK-CH) mechanism to distribute incoming connections to hosts. For fine-grained (slice-level) resource management, we maintain a *separate* consistent hashing ring for each slice based on its `sliceID` (see Figure 5). Thus, each slice can be served by a subset of all hosts; however, multiple slices can be served by a single host. Within the slice, we assign a subset of hosts (depending on number of replicas needed) to each UE, based on their `TMSI`. To serve the connection, a specific host is chosen from this subset based on its load conditions and the required SLO.

Our Algorithm: Algorithm 1 details our slice-aware inter-host load balancing. The first procedure for any UE is the attach procedure. For the first packet of attach (for which the `TMSI`, `mmeID`, and `sliceID` values are not assigned), the load balancer assigns a TMSI, i.e., `LTMSI`. The `LTMSI` is used to calculate the MME host, say $host_0$, to which this message will be sent for service using the hashing ring reserved for `sliceID=0`. After service, $host_0$ assigns the `mmeID` to this connection as part of the reply. The load balancer uses this `mmeID` to directly send subsequent packets of attach to $host_0$.

After successful completion of the attach procedure, the `sliceID` of the UE is resolved. This `sliceID` is then used to select the specific slice-aware hash ring. Within the hash ring, the hash of the `TMSI` value (same as `LTMSI`) is used to select the primary host and replica hosts, as shown in Figure 5. The number of replicas can be decided based on the expected load for each UE; in our experiments, we initially assign 2 replicas for each UE. $host_0$ then migrates the UE context to the primary and replica hosts so they can serve subsequent procedures from the UE based on the `TMSI`.

For subsequent procedures (after attach), from among the primary and replica hosts, we derive the set of *viable hosts*, i.e, hosts that satisfy the procedure's SLO requirements.

Calculation of viable MME hosts: At a high-level, we say that a host is viable if it contains at least one NF that meets the SLO requirements of the procedure, say T_{SLO}. To obtain the set of viable MME hosts, we compute the *total estimated time* required by a (primary or replica) host to handle the incoming control procedure. This, in turn, requires the current load statistics at each host. Each MME host propagates the CPU utilization and queue sizes of their NFs to the load balancer. To minimize the overhead of communication, hosts periodically send the *moving average* of CPU utilization and NF queue sizes. In our implementation, this period is set to a few hundred milliseconds, resulting in only a few hundred kilobytes of data overhead on the network.

Let t be the type (e.g., service) of the incoming control procedure. Let T_t be the total completion time required for a type t procedure when handled in isolation on a core (obtained via profiling). Let m be the total number of NFs that handle type t procedures across all primary and replica hosts of `TMSI`. For each NF i, the load balancer is aware of the moving average of queue sizes, q_i, and CPU assigned, c_i. The waiting time for the procedure at NF i is then estimated as:

$$W_i = (q_i \cdot T_t)/c_i \ \forall i = 1,\ldots,m \tag{1}$$

Assuming that the moving average is stable, every message of the procedure assigned to NF i will see a backlog of q_i. If there are p_t messages in a type t procedure, the total backlog experienced by the procedure is $p_t \times q_i$. Since T_t is the time for a procedure when run in isolation (on a full core), we can approximate the time per message as $T_t/(c_i \cdot p_t)$, and thus the wait time is $p_t \times q_i \times T_t/(c_i \cdot p_t) = q_i \times T_t/c_i$. The completion time is then computed as $T_i = W_i + T_t = (\frac{q_i}{c_i} + 1) \times T_t$.

We now find the *viable hosts* as those that contain at least one NF i for which $T_i \leq T_{SLO}$. If no viable hosts exist, then the messages are sent to the host that has the NF with the least q_i/c_i ratio. Such violations are reported to the EPS Controller, which can then add hosts to the slice (see §5.3).

Selection of final host from viable hosts: From among the set of viable hosts, we select the host that is *most loaded*, i.e, the host that has the highest CPU usage. While this may seem counter-intuitive, recall that we are only picking from among viable hosts, each of which has at least one NF that can satisfy the SLO. We prefer the most loaded host to maintain lighter load on other hosts, which can then be scaled down during low cluster usage. Instead, if we select the least loaded host, over time we will have balanced hosts, making it difficult to identify less loaded hosts to drain connections from in case of a scale down. Prior works have shown this *load unbalancing* technique to facilitate server scaling in web clusters [6, 16], while we explore this in the context of stateless MME architectures with states distributed across multiple MME hosts (see §7.2).

Once the final host is selected for a control procedure, we maintain the connection tracking information for this `TMSI` to forward subsequent messages of this procedure to the same host without having to recalculate the viable hosts. This information is refreshed each time a new control procedure (not message) is seen by the MME load balancer.

ALGORITHM 2: Intra-Host Load Balancing.

```
1  sliceID, mmeID, GUTI.TMSI, msgType ← pkt;
2  sID ← getServiceID(msgType);
3  nfIDs ← getNFInstances(sID, sliceID) ;
4  track_entry ← Hash database of (TMSI, mmeNFID) ;
5  if track_entry[TMSI] exists then
6      mmeNFID ← track_entry[TMSI] ;
7  else
8      if sliceID = 0 then
9          mmeNFResources ← getNFQueue(nfIDs) ;
10         mmeNFID ← min{mmeNFResources} ;
11     else
12         mmeNFID ← getOptimalNF(mmeNFLoads) ;
13         track_entry[TMSI] = mmeNFID ;
14 send packet to NF [mmeNFID] ;
```

5.2 Intra-Host Load Balancing (Forwarder)

Each slice inside an MME host is assigned a dedicated MME forwarder and a number of MME NFs of different service types (e.g., attach, detach, handover). For each service type, there can be multiple NFs depending on the load conditions of that specific control procedure. Once a host is chosen for handling an incoming message (inter-host load balancing), the next task is to decide the specific NF within that host that will serve the request. *MME Forwarders* provide an effective means to load balance the control traffic across multiple MME NFs inside each MME host.

Our intra-host load balancing algorithm is presented in Algorithm 2. Control messages that do not have any `sliceID` details (i.e, attach procedure's messages), will be put on the receive ring buffer of the forwarder that is dedicated to `sliceID=0`. This forwarder distributes the messages to the attach MME NF that has the least queue size. Consecutive messages from control procedures that have a `sliceID` (i.e., procedures from UEs whose attach is successful), will be sent to the *optimal* MME NF, as discussed next.

Determining the optimal MME NF: A common approach of intra-host load balancing is round robin or consistent hashing [4]. However, these approaches do not provide any performance guarantees, and may create hot spots, resulting in SLO violations (see §2.2).

Our approach, by contrast, selects the NF that provides the lowest latency for the incoming procedure. Specifically, we use the same latency model as in Eq. (1), except: (i) we use current queue size at the NF, say Q_i, instead of moving average, and (ii) we use the current lcore allocation (based on updated priorities, see §5.3), say C_i, as opposed to moving average. We use the current values (i.e., obtained from within the host without any network overhead) to enforce SLO requirements as they provide more accurate and timely estimates of current load conditions at the NF. The estimated latency is then:

$$T_i = (Q_i/C_i + 1) \times T_t, \ \forall i = 1,\ldots,r \tag{2}$$

where r is the number of NFs in the host. Based on Eq. (2), finding the optimal NF for a given procedure type, t, and a given slice that minimizes the estimated latency is equivalent to finding the NF *opt* such that:

$$opt = \underset{1 \le i \le r}{\arg\min}\, Q_i/C_i \tag{3}$$

If the estimated latency violates the SLO, i.e., $T_i > T_{SLO}$, then, in addition to sending the packet to the computed *opt* NF, the predicted violation is reported to the NF Controller within the MME host. The Controller then instantiates additional MME NFs as needed (see §5.3).

In general, the number of NFs within a host for a given procedure and for a given slice is not too large (on the order of 10's). Thus, the optimal NF can be determined without much overhead. However, the overhead of centrally computing the queue and cpu statistics for NFs in the kernel is infeasible due to the high packet arrival rate and the required polling of NFs to obtain the necessary statistics. We reduce this overhead by maintaining (computing) the required statistics within each NF itself. To amortize the overhead, we maintain the flow tracking entry for each procedure with TMSI and MME instanceID (`TMSI::instanceID`) once its optimal NF has been determined. Subsequent packets of this procedure bypass the forwarder and go to the *opt* NF.

5.3 NF Prioritization & Resource Scaling

The MME must handle heterogeneous traffic from diverse IoT devices with a wide range of SLO requirements. We thus employ prioritization and resource management to avoid performance interfere and achieve SLO compliance.

Prioritization: Multiple NFs on a host may be assigned to the same core, creating contention. To provide performance isolation, our `MMLite` architecture leverages CPU-based prioritization and processor scheduling. When a core is assigned to n MME NFs with different priority levels, P_i, we compute, for each NF i, the CPU core allocation fraction, C_i, as:

$$C_i = \frac{P_i}{\sum_{j=1}^{n} P_j} \forall i = 1,\ldots,n \tag{4}$$

P_i ranges from 0 to 1, with higher values representing higher priority. If only one active NF exists on a core, it will be allocated the entire core. Idle NFs are not considered in the C_i calculation. To enforce the C_i allocation, we schedule the core's time slices across resident NFs in proportion to their C_i values in a round robin manner. The priorities are recalculated every time an NF becomes idle or when an NF is added to the core for packet processing.

To assign priorities, we first note the minimum SLO value across users, say SLO_{min}. We then set the priorities for each NF *inversely proportional* to their respective SLO values, normalized by SLO_{min}. Thus, for an NF with SLO value SLO_i, we set $P_i = SLO_{min}/SLO_i$. For example, consider two NFs a and b that share a core with $SLO_a = 5$ms and $SLO_b = 10$ms. If $SLO_{min} = 1$ms, we set $P_a = 1/5$ and $P_b = 1/10$. This gives us, from Eq. (4), $C_a = 2/3$ and $C_b = 1/3$. Prior work on shared storage workloads has shown that priorities that are set inversely proportional to performance requirements work well in practice [75]. We evaluate the impact of our priority assignment on performance in §7.2.

Lower and upper limits on C_i may be predefined for specific slices. If SLO violations occur for an NF, we reactively increment its priority by a small fraction (e.g., 0.01). If violations continue to persist, we inform the NF Controller, which may then instantiate additional NFs, as discussed below.

Scaling: Hosts or NFs can be dynamically added for each slice (or user) reactively in response to overload or failures. On the other hand, resources (NFs or hosts) can be removed in response to low utilization. NF-level scaling is carried out by the NF controller within a host (see Figure 4), whereas host-level scaling is carried out at the EPS controller.

We add NFs at a host (or at other hosts if the CPU is saturated) in response to persistent SLO violations that are reported to the NF controller. To minimize the impact of waking up MME NFs, we maintain a pool of idle NFs that can be quickly instantiated, as needed; the overhead of idle NFs is negligible in our experiments. Likewise, we add a new MME host in response to persistent SLO violations that cannot be addressed by NF scaling alone. Further, we also add hosts if the load on all existing hosts is high. For scaling down, we first transition NFs to the idle state and return them to the pool of idle NFs if there are no outstanding messages in the queue. If the pool of idle NFs contains more than a threshold (say, 1, as in our experiments) number of NFs of a given type, then additional idle NFs of that type are turned off. We scale down hosts (or move to sleep or idle state) if the total available CPU across MME hosts is high, say significantly greater than 100%.

When the number of hosts changes, the EPS controller triggers the hosts to recalculate the slice-specific consistent hashing to determine the new set of hosts to migrate the states. When the number of NFs change in a host due to scaling, our *viable* and *optimal* NF selection strategies will gradually redistribute the packets among existing NFs as they prefer smaller queues. However, we can redistribute the packets in the receive queues more aggressively, e.g., in response to an NF failure.

6 IMPLEMENTATION

We use the OpenNetVM [48] integrated with DPDK platform [24] to build our stateless MME microservices and other components of `MMLite`. OpenNetVM provides the ability to process packets directly from the NICs allowing them to be DMA'd (Direct Memory Accessed) into a shared memory region. NFs can thus directly access packets with no additional copies (i.e, zero copy I/O). To overcome the 100% CPU utilization with DPDK poll mode driver (PMD), we use a hybrid polling-interrupt driven technique to achieve both performance and resource efficiency [73]. We next describe the functional components that developed in C for our prototype.

UE Emulator: The UE emulator is built as a multi-threaded program, with each individual thread generating control traffic of a UE. We bypass the radio to generate traffic for direct

handling by the MME. Our UEs generate control procedures such as attach, service, handover, detach, restoration, and TAU-based procedures. It allows the user to configure threads to generate traffic with specific characteristics.

MME Load Balancer & Forwarder: We implement our skewed consistent hashing along with round robin (RR), consistent hashing (CH) and Maglev-based LB mechanisms on DPDK. We use Jenkins hash function [26] as the baseline hashing mechanism for CH and our skewed CH methods.

MME as Microservice: To further improve scalability, we remove the existing network dependencies from Linux networking stack and build it as an application-level networking module using DPDK. We implement each control procedures as microservices with their states externalized to shared memory. This reduces the NF instantiation time to the order of a few milliseconds compared to tens of seconds using OAISIM [13] and OpenEPC NFs [69].

State Migration Utility: We implement state migration as a standalone module interfaced with the EPS controller to trigger bulk and device-specific state transfers. The states marked for migration are transfered to other MME replica hosts (§4) using separate threads assigned with dedicated cores. These threads continuously monitor the state updates to perform state transfers to other MME hosts. This utility is designed to perform both hot and cold migrations.

Controller infrastructure: The EPS controller and NF controller modules interface with the MME NFs, load balancer, and forwarder modules. The controller infrastructure is built with following key capabilities: (*i*) initiates state migration across hosts, (*ii*) performs scaling of resources (MME NFs and hosts) based on observed SLO violations and cpu usage, (*iii*) updates load balancer with the MME failure and recovery states, and (*iv*) assigns the globally unique LTMSI values to UEs when attached to the network for the first time.

7 EVALUATION

In this section, we evaluate the performance of `MMLite` and compare it with conventional MME architectures and also with recent approaches. We use the following platforms: (*i*) DPDK Compatible Intel Ethernet 10G 2P X520 NIC cards, (*ii*) Dell R710 servers with 48GB RAM, 12 cores (2.6GHz) with Ubuntu 4.4.0-97-generic kernel used as MME hosts, and (*iii*) Dell R710 servers with 48BG RAM, 12 cores and 10G Mellanox InfiniBand adapter integrated with RAMCloud infrastructure for centralized data store. The testbed has UE emulator hosts interfaced with multiple 10Gbps NICs to MME load balancer. The load balancer interfaces with multiple MME hosts and EPS Controller using 10Gbps NICs.

7.1 Throughput Comparison

We compare the throughput of `MMLite` with the following prototypes: (1) *OpenAirInterface (OAI)*: OAI is the most widely used open source EPC implementation [60]. We benchmark the performance and scalability of OAI in the OAISIM mode [13], where the UE and eNodeB are integrated together into a single node, bypassing the radio interface. (2) *OpenEPC*: We benchmark OpenEPC using the PhantomNet testbed [69]. (3) *MME with Centralized data-store (MME-RC)*: We customize our DPDK-based MME code to use the RAMCloud-based centralized data store model. RAMCloud [59] stores all the data in DRAM allowing the remote servers to access the RAMCloud data objects with low latency (as little as $\approx 5\mu$sec). (4) *Stateful DPDK-based MME*: This prototype implementation uses the same code base as `MMLite`, but is stateful.

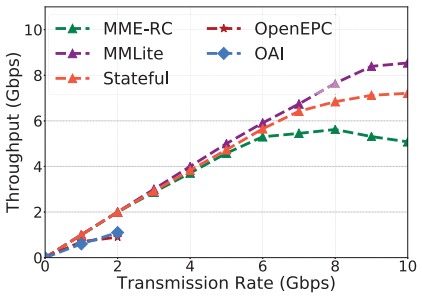


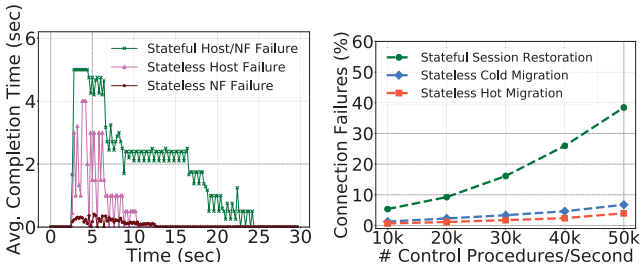
Figure 6: Throughput of different MME prototypes.

Figure 6 summarizes the throughput for the above approaches relative to our stateless DPDK-based approach in `MMLite` as a function of load. For each load level, we generate realistic IoT traffic, similar to Figure 2a. As shown in Figure 6, we find that the OAI and OpenEPC/PhantomNet platforms have limited throughput scalability, and saturate at around 1 Gbps due to MME application's binding with the kernel, unlike infrastructures such as DPDK which bypass it. MME-RC scales well initially, but saturates at around 5.6 Gbps, and gradually drops to 5.1 Gbps due to the overhead of state storage, retrieval and transfer to the target host. The stateful DPDK-based model performs better than MME-RC, but still saturates around 7.2 Gbps; this is because of the combined effects of queue processing delays induced by MME and other EPC nodes. By contrast, our functionally decomposed and stateless model that is free of static bindings allows MME to scale almost linearly with load by effectively sharing the load across stateless NFs, thus saturating only close to the line rate. It requires further investigation to understand the intricate component-level benefits, which we defer to our future work. Compared to the stateful model, our stateless design provides ≈16% better performance at peak load. In summary, `MMLite` effectively prioritizes and distributes control traffic load across stateless NFs to provide higher throughput than stateful and MME-RC models.

7.2 Performance of `MMLite`'s Components

We first highlight the benefits of `MMLite` with its fault tolerance, dynamic scaling capabilities and its ability to effectively control the interference across control procedures. We then end with an evaluation of SLO compliance and resource efficiency.

We use the following traffic characteristics generated using our UE emulator: `T1`, a constant rate of control procedures; `T2`, a steadily increasing rate of control procedures; `T3`, traffic rate from each UE using a Markov modulated (time-varying)



(a) Latency of control procedures after failure. **(b) % Connection drops while handling failure.**

Figure 7: Demonstrating fault tolerance.

Poisson process; and T4, a sporadic traffic pattern (Pareto distributed [49]), representative of traffic surges.

Fault Tolerance: Figure 7a demonstrates our MME failure handling with three MME hosts using T1 traffic. We suspend one of the MME host's NFs at about the 2 second mark to emulate MME failure. The traditional stateful MME uses the same state restoration procedure for NF or host failures. We thus observe the same performance in case of NF and host failure for stateful MME. We see that the control procedure latency shoots up to 5 s (which is the maximum UE retry time) in response to failures for the stateful MME. This is because during failures, the UE retries for the connection every second. If the UE fails to get a response to its retries within 5s, it generates reattach procedure and drops existing data connections associated with it.

Unlike stateful MME, MMLite handles host and NF failures differently as discussed in §4. MMLite needs state migration in case of host failures, but this does not involve UEs. Figure 7a shows that MMLite is far more responsive after failures and recovers quickly. The average latency of the control procedure is < 0.5 s with NF failure and up to 2.5 s with host failure. Clearly, the performance recovery is significantly better for MMLite.

To further analyze performance, Figure 7b shows the number of connection failures after the fault; we perform multiple experiments at different loads to generate data for this figure. With stateful MME, we see numerous connection drops. At 50K connections/s load, we observe $\approx$ 40% connection drops; this is due to congestion that occurs at MME, eNodeB and UE during MME failure. By contrast, MMLite significantly reduces the number of failures in all cases, for both cold and hot migration approaches (§5).

Scaling: To evaluate scaling, we use T2 traffic to steadily increase traffic until a maximum point (from 2.5 Gbps to about line rate) and then reduce it gradually. We note that our stateless MMLite seamlessly scales the number of NFs and hosts, as needed, in response to the changing traffic (Figure 8). Further, the resulting latency is much lower for stateless compared to stateful; note that the latency is shown on a *log scale*. The latency specifically spikes for stateful MME when the number of hosts is scaled up (around the 90 s mark) or scaled down (around the 260 s mark). This is due to the (resource intensive) TAU-based load rebalancing and state migration across hosts

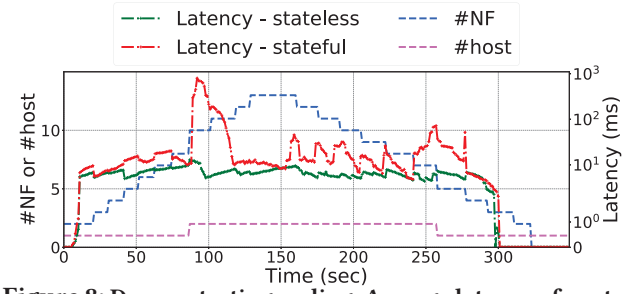


Figure 8: Demonstrating scaling: Average latency of control procedures with scaling of NFs and hosts.

that is required for stateful MME. During these periods, the latency for stateful MME is about *50-100× higher* than stateless MME. Empirically, we found that while handling failures at line rates, MME-RC resulted in higher connection drops and higher latency to complete the control procedures compared to MMLite. Similar observations are made in scaling MME hosts and NFs due to the saturation effect observed with MME-RC at higher throughputs, as discussed in §7.1.

Functional Decomposition & NF Prioritization: In this experiment, we illustrate the benefits of functional decomposition with two different types of control procedures i.e., *handover request* and *service request*. The average latency for handling the *service request*, in isolation, is a lot smaller ($<$ 10 ms) compared to *handover* ($\approx$ 20 ms). We use the T1 traffic pattern with a rate of 1000requests/s.

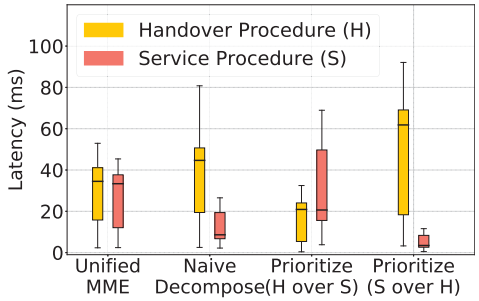
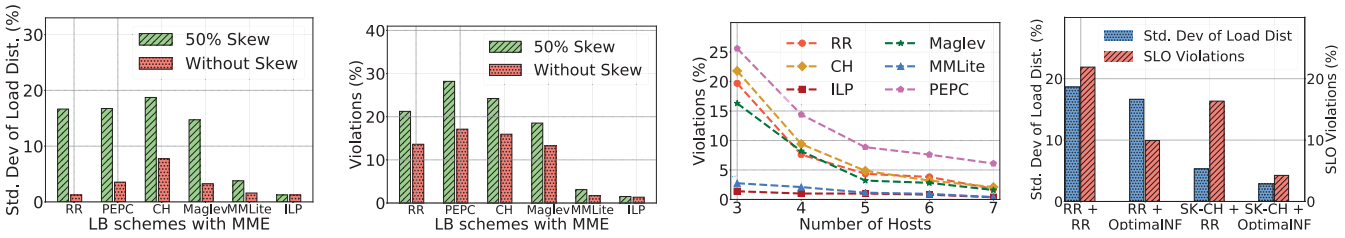


Figure 9: Functional decomposition and prioritization of control procedures at line rate.

Figure 9 shows the latency of the two control procedures in different scenarios. First, without any decomposition (default, unified stateful MME), we get a similar latency for handling each procedure ($\approx$ 35 ms). This is because, under the unified architecture, the handling of the two procedures creates contention and significantly increases the latency of the smaller *service request*. With our naive decomposition, the interference induced by *handover* over the *service request* procedure is alleviated. The service request procedure latency is brought back to the ideal case of $<$ 10 ms. However, the *handover* latency increases since the *service request* NF stays idle after finishing the control procedures assigned to it, wasting resources.

The latency can be further optimized by effectively setting the NF *priorities* when sharing CPU resources. Figure 9 (H over S) shows that the *handover* procedure latency can



(a) Standard deviation of load distribution among MME hosts.

(b) SLO violations under all LB schemes for different skews.

(c) SLO violations for different number of hosts.

(d) Performance of MME with different inter- and intra-host LB.

Figure 10: Evaluation results showing load distribution and SLO violations for different load balancing schemes.

be brought down to ≈ 20 ms while sacrificing the *service* latency by ≈10ms. Alternatively, we can prioritize the *service* procedure NF to get much lower latency, though at the expense of increased *handover* latency (S over H). These results showcase how tighter SLOs can be obtained by appropriately allocating resources (differentiated service) in the decomposed implementation in `MMLite`. Such prioritization is not possible for the default, unified MME design.

Load Distribution and SLOs: To evaluate our load balancing schemes from §5, we use all traffic patterns (`T1` – `T4`), and skew the load among UE connections. We compare the performance of traditional cellular control plane architecture that uses (i) RR (round robin), (ii) CH (consistent hashing), and (iii) Maglev [12], with our skewed-CH inter-host load balancer. Maglev aims to better balance the keys in CH to achieve balanced number of connections across hosts (as verified in our experiments), but it does not take into account the load on each connection or session. We also compare our approach with the PEPC cellular core architecture that we built, which employs the Maglev-based LB scheme, as discussed in [55]. Note that PEPC uses consolidated data and control plane cellular core elements.

We also compare with an unrealistic yet optimal Integer Linear Programming (ILP) solution that solves the load balancing problem, though at a significant computational overhead. The ILP finds the placement of NFs on hosts that satisfies the SLO constraints for incoming procedures (expressed similar to Eq. (2)), with the objective of minimizing the number of hosts. We omit the ILP formulation due to lack of space.

Figures10a and 10b show the standard deviation of CPU utilization across hosts and the resulting SLO violations for different balancers when using 3 MME hosts. For uniform load across UEs (no skew), all schemes perform well, though CH, Maglev and PEPC (enabled with Magle-based LB) do have slightly higher deviation. However, for skewed load (wherein the load for 50% of the connections was increased significantly), RR, PEPC, CH, and Maglev have high deviation (above 15%) in CPU load across hosts. This is because these schemes only try to balance the number of connections per host (i.e., UEs in our case), which is insufficient as the connections themselves have different load. This also results in significant SLO violations (≈18-28%) for these schemes. By contrast, `MMLite` results in about < 4% standard deviation and only about 3-4% SLO violations. These numbers are almost a factor 3-7× lower compared to other schemes. Compared to the optimal ILP, skewed-CH is within 1% of the SLO violations and within 5% of the CPU deviation.

Figure 10c further analyzes the SLO violations when the number of hosts is increased (under skewed load). While the violations decrease with number of hosts for all schemes, we see that Maglev, CH, RR, and PEPC, continue to have significantly higher SLO violations, when compared to `MMLite` . PEPC has the highest SLO violations because of the interference between control and data place traffic, which are consolidated under the design of PEPC. As the number of hosts increase, `MMLite` starts to approach ILP; we believe this is because `MMLite` has more opportunities to find viable hosts with larger cluster sizes.

While the ILP outperforms `MMLite` in the above experiments, it must be noted that the ILP's optimal decisions were calculated offline based on collected workload traces. This is because the ILP takes on the order of seconds to converge to the solution for our testbed parameters, making it infeasible in practice (as the ILP is run for each arriving procedure).

Inter and Intra-host LB Schemes: Finally, Figure 10d evaluates the performance under different combinations of inter- and intra-host load balancing using 3 MME hosts and the skewed load to illustrate the importance of each component. We focus on RR as the alternative scheme to limit the state space. We see that our load balancing scheme (Skewed-CH for inter LB + OptimalNF for intra LB) provides the least SLO violations and deviation in CPU utilization. When we replace our inter-host balancer with RR, the violations increase from 4.25% to 9.92%, but when we replace our intra-host balancer with RR, the violations increase to 16.35%. However, the deviation of CPU usage shows the opposite trend. This is because the inter-host balancer is primarily responsible for resource efficiency, whereas the intra-host balancer is primarily responsible for SLO compliance. Replacing both components with RR provides even worse performance.

Resource Utilization: In practice, a more relevant question is to determine the number of hosts needed to meet the SLO requirements. Figure 11 illustrates these results (showing time-averaged number of hosts) for the requirement of ≤ 5% SLO violations under the Pareto traffic distribution with different skews (α = 1, 2 and 3). We see, from Figure 11a, `MMLite`

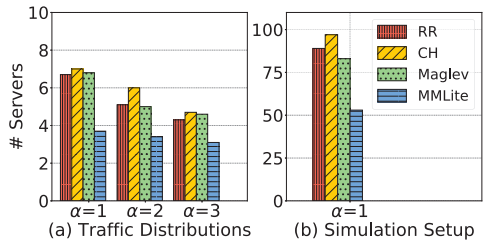


Figure 11: Hosts required for SLO compliance under different schemes using Pareto traffic with varying skews (α).

provides about 34 – 47% reduction in resource requirements for all traffic traces. CH performs the worst, while RR and Maglev do marginally better than CH. This is because, unlike other approaches, `MMLite` is designed for resource efficiency and SLO compliance (see §5). We omit results for PEPC as it fails to meet the SLO requirements even with all the hosts in our testbed; this is because of the additional interference introduced by PEPC with consolidated control and data path elements on the same host.

To evaluate the results for larger cluster sizes, we simulate the various schemes. Figure 11b shows one such scenario ($\alpha = 1$) with 100 hosts. We find that, even with larger clusters, `MMLite` continues to provide significant resource savings.

8 RELATED WORK

Functional Customization: There is a significant body of work on decoupling control and data planes [39, 41, 42]. Recent works have demonstrated the resiliency of such decoupled services implemented as microservices [22, 73]. CoMB [62] demonstrated the composition of NF services into a consolidated middlebox. In the context of cellular networks, techniques are proposed to optimize the control procedure latency by customizing the LTE control messaging architecture [31, 41]. ECHO [42] deals with reliability of EPC in public cloud infrastructures. PEPC [55] proposed functional composition in the context of cellular networks for efficient scaling and state space optimization. Similarly, Softbox [40] proposed scalable LTE core architecture by slicing and composing the core functionality as containerized per-UE EPC. In addition, Softbox supports mobility-aware mechanism to optimize the resource utilization of UE containers, thereby effectively steering traffic through per-UE containers using SDN-based rules. While inspired by the above, our work focuses solely on the control plane and uses statelessness and *procedure-level decomposition* (as opposed to protocol-level decomposition in the above works) to minimize interference among control procedures for fine-grained SLO compliance.

Statelessness: Recent studies have suggested the decoupling of the static bindings of MME with other entities to allow the MME to scale [53, 66, 67]. However, the states held by the MME specific to each UE prevent the MME from scaling and can create hot spots. Our approach to statelessness is motivated by Kablan et al. [28] which proposes completely decoupling the state and processing. We build on this concept to develop a multi-level load balancer that seamlessly moves load across MME hosts owing to statelessness.

SLO-aware Load Balancing: SCALE [4] proposed consistent hashing (CH) based load balancing and dynamic scaling for MME, but the scaling employs analytical approaches that require predictable traffic patterns. However, UE and IoT traffic can be skewed, bursty, and unpredictable [29, 63]. As shown in our evaluation, `MMLite` is able to effectively handle skewed and bursty traffic. Further, SCALE operates at the granularity of VMs, whereas `MMLite` operates at the finer granularity of microservices, allowing control procedures to be effectively prioritized according to their SLO requirements. The Maglev-based load balancing [12] aims to eliminate the skew in key distribution of CH. However, as shown in §7, these techniques are oblivious of the load on each connection and thus result in hot spots and unreliable SLOs. There are also other load balancing and migration techniques [1, 19, 20, 44, 46, 50, 64, 72], but they do not take SLO requirements into consideration when handling connections.

Prioritizing connections has been explored before (e.g., QJump [18], PriorityMeister [76], and SNC-Meister [74]). Building on these works, we integrate priorities into our multi-level load balancer. Another popular approach to SLO-aware resource provisioning is predictive modeling [10, 51]. Given the bursty and at times unpredictable IoT traffic [29, 63], the effectiveness of these models in the IoT space is unclear.

9 CONCLUSION

With the increase of IoT devices in the cellular network, it is now critical for the cellular core to handle diverse devices with varying traffic characteristics and SLOs at a low cost. Given this backdrop, we specifically focus on handling the control traffic effectively at a critical core network entity - the MME. The proposed design, `MMLite`, is a departure from traditional inflexible approaches that use static binding between the state and the processing. `MMLite` uses stateless microservices to decouple this binding and to enable functional customization that is more responsive to SLO requirements and resource availability. We develop a multi-level load balancing approach using skewed consistent hashing to achieve SLO compliance and resource efficiency. Our evaluations using DPDK and OpenNetVM-based prototypes demonstrate the superior performance of `MMLite` over existing approaches with respect to fault tolerance, scalability, resource utilization, and SLO satisfaction. We will open source the `MMLite` framework to enable further work.

Acknowledgments

We greatly appreciate Z. Morley Mao (our shepherd) and the anonymous reviewers for their insightful feedback. Our special thanks to Vijay Gopalakrishnan from AT&T Labs for his continuous support all along the way during our research. This work is partially supported by NSF grants CNS-1642965, CNS-1617046, CNS-1750109 and a grant from MSIT, Korea under the ICTCCP Program.

References

[1] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan, Navindra Yadav, George Varghese, et al. 2014. CONGA: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. ACM, 503–514.

[2] Pilar Andres-Maldonado, Pablo Ameigeiras, Jonathan Prados-Garzon, Juan J Ramos-Munoz, and Juan M Lopez-Soler. 2017. Optimized LTE Data Transmission Procedures for IoT: Device Side Energy Consumption Analysis. *arXiv preprint arXiv:1704.04929* (2017).

[3] R. Archibald, D. Gupta, R. Jana, V. Gopalakrishnan, A. S. Rajan, K. B. Ramia, D. Dahle, J. Cooper, G. Kennedy, N. Rao, S. Sonnads, and M. Mc Donald. 2016. An IoT control plane model and its impact analysis on a virtualized MME for connected cars. In *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. 1–6. https://doi.org/10.1109/LANMAN.2016.7548864

[4] Arijit Banerjee, Rajesh Mahindra, Karthik Sundaresan, Sneha Kasera, Kobus Van der Merwe, and Sampath Rangarajan. 2015. Scaling the LTE Control-plane for Future Mobile Access. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '15)*. ACM, New York, NY, USA, Article 19, 13 pages. https://doi.org/10.1145/2716281.2836104

[5] Call Failures in MME. 2017. https://en.wikipedia.org/wiki/QoS_Class_Identifier.

[6] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. 2008. Energy-aware Server Provisioning and Load Dispatching for Connection-intensive Internet Services. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*. USENIX Association, Berkeley, CA, USA, 337–350. http://dl.acm.org/citation.cfm?id=1387589.1387613

[7] Cisco: ASR5x00 MME Overload Protection Features. 2015. https://goo.gl/LV97b5.

[8] Cisco leads the way to 5G networks, Microservices and Advanced Automation. 2017. https://goo.gl/Sx2xoL.

[9] Cisco: MME Overview (Overload Protection). 2017. https://goo.gl/dyF3x9.

[10] S. Correa and R. Cerqueira. 2010. Statistical Approaches to Predicting and Diagnosing Performance Problems in Component-Based Distributed Systems: An Experimental Evaluation. In *2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. 21–30. https://doi.org/10.1109/SASO.2010.32

[11] Designing and managing VNFs the right way for network functions virtualization. 2017. https://goo.gl/Fnc7Ci.

[12] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. 2016. Maglev: A Fast and Reliable Software Network Load Balancer. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. Santa Clara, CA, 523–535. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eisenbud

[13] End-to-end LTE/EPC network with OpenAirInterface (OAI) simulated eNB/UE and OAI's EPC . 2017. https://goo.gl/kXSvBi.

[14] Ericsson Mobility Report. 2016. http://www.ericsson.com/res/docs/2016/ericsson-mobility-report-2016.pdf.

[15] Lilatul Ferdouse, Alagan Anpalagan, and Sudip Misra. 2017. *Congestion and overload control techniques in massive M2M systems: A survey*. https://doi.org/10.1002/ett.2936 e2936 ett.2936.

[16] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A. Kozuch. 2012. AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers. *ACM Trans. Comput. Syst.* 30, 4, Article 14 (Nov. 2012), 26 pages. https://doi.org/10.1145/2382553.2382556

[17] Gartner Reveals Top Predictions for IT Organizations and Users in 2017 and Beyond. 2017. http://www.gartner.com/newsroom/id/3482117.

[18] Matthew P. Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert N. M. Watson, Andrew W. Moore, Steven Hand, and Jon Crowcroft. 2015. Queues Don'T Matter when You Can JUMP Them!. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15)*. USENIX Association, Berkeley, CA, USA, 1–14. http://dl.acm.org/citation.cfm?id=2789770.2789771

[19] Ajay Gulati, Chethan Kumar, Irfan Ahmad, and Karan Kumar. 2010. BASIL: Automated IO Load Balancing Across Storage Devices. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*. USENIX Association, Berkeley, CA, USA, 13–13. http://dl.acm.org/citation.cfm?id=1855511.1855524

[20] Ajay Gulati, Ganesha Shanmuganathan, Irfan Ahmad, Carl Waldspurger, and Mustafa Uysal. 2011. Pesto: Online Storage Performance Management in Virtualized Datacenters. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing (SOCC '11)*. ACM, New York, NY, USA, Article 19, 14 pages. https://doi.org/10.1145/2038916.2038935

[21] Handling of signaling storms in mobile networks. 2017. https://goo.gl/fzkSGH.

[22] V. Heorhiadi, S. Rajagopalan, H. Jamjoom, M. K. Reiter, and V. Sekar. 2016. Gremlin: Systematic Resilience Testing of Microservices. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. 57–66. https://doi.org/10.1109/ICDCS.2016.11

[23] High Availability is more than five nines. 2017. https://goo.gl/o4dV3E.

[24] Intel Data Plane Development Kit. 2017. http://dpdk.org/.

[25] Internet of Things: Network Data Traffic Collection. 2018. http://iotanalytics.unsw.edu.au/.

[26] Jenkins hash function. 2017. https://en.wikipedia.org/wiki/Jenkins_hash_function.

[27] Roger Piqueras Jover. 2015. *Security and impact of the IoT on LTE mobile networks.*

[28] Murad Kablan, Azzam Alsudais, Eric Keller, and Franck Le. 2017. Stateless Network Functions: Breaking the Tight Coupling of State and Processing. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 97–112. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/kablan

[29] M. Laner, P. Svoboda, N. Nikaein, and M. Rupp. 2013. Traffic Models for Machine Type Communications. In *ISWCS 2013; The Tenth International Symposium on Wireless Communication Systems*. 1–5.

[30] Yuanjie Li, Zengwen Yuan, and Chunyi Peng. 2017. A Control-Plane Perspective on Reducing Data Access Latency in LTE Networks. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom '17)*. ACM, New York, NY, USA, 56–69. https://doi.org/10.1145/3117811.3117838

[31] Yuanjie Li, Zengwen Yuan, and Chunyi Peng. 2017. A Control-Plane Perspective on Reducing Data Access Latency in LTE Networks. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom '17)*. ACM, New York, NY, USA, 56–69. https://doi.org/10.1145/3117811.3117838

[32] Load Balance MME in Pool. 2017. https://goo.gl/61CqWz.

[33] LTE SUBSCRIBER SERVICE RESTORATION. 2017. https://goo.gl/nfmLv6.

[34] Managing LTE Core Network Signaling Traffic. 2017. https://insight.nokia.com/managing-lte-core-network-signaling-traffic.

[35] Matteo Pozza et al., Solving Signaling Storms in LTE Networks: a Software-Defined Cellular Architecture. 2017. http://tesi.cab.unipd.it/53297/1/tesi_Pozza.pdf.

[36] Diomidis S Michalopoulos, Mark Doll, Vincenzo Sciancalepore, Dario Bega, Peter Schneider, and Peter Rost. 2017. Network Slicing via Function Decomposition and Flexible Network Design. (2017).

[37] MMLite: LTE UE Emulator to generate LTE control messages. 2018. https://github.com/vasu018/LTE-UE.

[38] MobileInsight: Data Sharing. 2018. . http://www.mobileinsight.net/data.html.

[39] Ali Mohammadkhan, K.K. Ramakrishnan, Ashok Sunder Rajan, and Christian Maciocco. 2016. CleanG: A Clean-Slate EPC Architecture and ControlPlane Protocol for Next Generation Cellular Networks. In *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking (CAN 16)*. ACM, New York, NY, USA, 31–36. https://doi.org/10.1145/3010079.3010084

[40] M. Moradi, Y. Lin, Z. M. Mao, S. Sen, and O. Spatscheck. 2018. SoftBox: A Customizable, Low-Latency, and Scalable 5G Core Network Architecture. *IEEE Journal on Selected Areas in Communications* 36, 3 (March 2018), 438–456. https://doi.org/10.1109/JSAC.2018.2815429

[41] Vasudevan Nagendra, Himanshu Sharma, Ayon Chakraborty, and Samir R. Das. 2016. LTE-Xtend: Scalable Support of M2M Devices in Cellular Packet Core. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges (MobiCom Workshop, ATC '16)*. ACM, New York, NY, USA, 43–48. https://doi.org/10.1145/2980055.2980062

[42] Binh Nguyen, Tian Zhang, Bozidar Radunovic, Ryan Stutsman, Thomas Karagiannis, Jakub Kocur, and Jacobus Van der Merwe. 2018. ECHO: A Reliable Distributed Cellular Core Network for Hyper-scale Public Clouds. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18)*. ACM, New York, NY, USA, 163–178. https://doi.org/10.1145/3241539.3241564

[43] Nokia Siemens Networks: Signaling is growing 50% faster than data traffic. 2017. https://goo.gl/oTbTmM.

[44] Mohammad Noormohammadpour and Cauligi S Raghavendra. 2017. Datacenter Traffic Control: Understanding Techniques and Tradeoffs. *IEEE Communications Surveys & Tutorials* 20, 2 (2017), 1492–1525.

[45] Sangchul Oh, Byunghan Ryu, and Yeonseung Shin. 2013. EPC signaling load impact over S1 and X2 handover on LTE-Advanced system. In *2013 Third World Congress on Information and Communication Technologies (WICT 2013)*. 183–188. https://doi.org/10.1109/WICT.2013.7113132

[46] Vladimir Olteanu, Alexandru Agache, Andrei Voinescu, and Costin Raiciu. [n. d.]. Stateless datacenter load-balancing with beamer.

[47] openair-cn: Evolved Core Network Implementation of OpenAirInterface. 2017. https://gitlab.eurecom.fr/oai/openair-cn.

[48] OpenNetVM. 2017. http://sdnfv.github.io/onvm/.

[49] Pareto Distribution. 2017. https://en.wikipedia.org/wiki/Pareto_distribution.

[50] Nohhyun Park, Irfan Ahmad, and David J. Lilja. 2012. Romano: Autonomous Storage Management Using Performance Prediction in Multi-tenant Datacenters. In *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12)*. ACM, New York, NY, USA, Article 21, 14 pages. https://doi.org/10.1145/2391229.2391250

[51] Ilia Pietri, Gideon Juve, Ewa Deelman, and Rizos Sakellariou. 2014. A Performance Model to Estimate Execution Time of Scientific Workflows on the Cloud. In *Proceedings of the 9th Workshop on Workflows in Support of Large-Scale Science (WORKS '14)*. IEEE Press, Piscataway, NJ, USA, 11–19. https://doi.org/10.1109/WORKS.2014.12

[52] T. Potsch, S. N. K. Khan Marwat, Y. Zaki, and C. Gorg. 2013. Influence of future M2M communication on the LTE system. In *6th Joint IFIP Wireless and Mobile Networking Conference (WMNC)*. 1–4. https://doi.org/10.1109/WMNC.2013.6549000

[53] G. Premsankar, K. Ahokas, and S. Luukkainen. 2015. Design and Implementation of a Distributed Mobility Management Entity on OpenStack. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. 487–490. https://doi.org/10.1109/CloudCom.2015.54

[54] Zafar Ayyub Qazi, Phani Krishna Penumarthi, Vyas Sekar, Vijay Gopalakrishnan, Kaustubh Joshi, and Samir R Das. 2016. KLEIN: A minimally disruptive design for an elastic cellular core. In *Proceedings of the Symposium on SDN Research*. ACM, 2.

[55] Zafar Ayyub Qazi, Melvin Walls, Aurojit Panda, Vyas Sekar, Sylvia Ratnasamy, and Scott Shenker. 2017. A High Performance Packet Core for Next Generation Cellular Networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. ACM, New York, NY, USA, 348–361. https://doi.org/10.1145/3098822.3098848

[56] Shriram Rajagopalan, Dan Williams, Hani Jamjoom, and Andrew Warfield. [n. d.]. Split/Merge: System Support for Elastic Execution in Virtual Middleboxes.

[57] Ashok Sunder Rajan, Sameh Gobriel, Christian Maciocco, Kannan Babu Ramia, Sachin Kapur, Ajaypal Singh, Jeffrey Erman, Vijay Gopalakrishnan, and Rittwik Jana. 2015. Understanding the bottlenecks in virtualizing cellular core network functions. *The 21st IEEE International Workshop on Local and Metropolitan Area Networks* (2015), 1–6.

[58] RAMCloud. 2017. http://dpdk.org/doc/guides-16.04/linux_gsg/nic_perf_intel_platform.html.

[59] RAMCloud. 2017. https://ramcloud.stanford.edu/docs/doxygen/md_README.html.

[60] The OpenAirInterface repository. 2017. https://gitlab.eurecom.fr/oai/openairinterface5g.

[61] M. R. Sama, X. An, Q. Wei, and S. Beker. 2016. Reshaping the mobile core network via function decomposition and network slicing for the 5G Era. In *2016 IEEE Wireless Communications and Networking Conference*. 1–7. https://doi.org/10.1109/WCNC.2016.7564652

[62] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K. Reiter, and Guangyu Shi. 2012. Design and Implementation of a Consolidated Middlebox Architecture. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*. USENIX Association, Berkeley, CA, USA, 24–24. http://dl.acm.org/citation.cfm?id=2228298.2228331

[63] Muhammad Zubair Shafiq, Lusheng Ji, Alex X. Liu, Jeffrey Pang, and Jia Wang. 2012. A First Look at Cellular Machine-to-machine Traffic: Large Scale Measurement and Characterization. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '12)*. ACM, New York, NY, USA, 65–76. https://doi.org/10.1145/2254756.2254767

[64] A. Singh, M. Korupolu, and D. Mohapatra. 2008. Server-storage virtualization: Integration and load balancing in data centers. In *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12. https://doi.org/10.1109/SC.2008.5222625

[65] Study on provision of low-cost Machine-Type Communications (MTC) User Equipments (UEs) based on LTE, 3GPP spec: 36.888. 2017. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2578.

[66] T. Taleb, A. Ksentini, and B. Sericola. 2016. On Service Resilience in Cloud-Native 5G Mobile Systems. *IEEE Journal on Selected Areas in Communications* 34, 3 (March 2016), 483–496. https://doi.org/10.1109/JSAC.2016.2525342

[67] T. Taleb and K. Samdanis. 2011. Ensuring Service Resilience in the EPS: MME Failure Restoration Case. In *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*. 1–5. https://doi.org/10.1109/GLOCOM.2011.6133654

[68] Traffic models for machine-to-machine (M2M) communications: types and applications. 2014. http://www.eurecom.fr/publication/4265.

[69] OpenEPC Tutorial using the classic PhantomNet portal. 2017. . https://wiki.emulab.net/wiki/phantomnet/oepc-protected/openepc-tutorial.

[70] I. Widjaja, P. Bosch, and H. La Roche. 2009. Comparison of MME Signaling Loads for Long-Term-Evolution Architectures. In *2009 IEEE 70th Vehicular Technology Conference Fall*. 1–5. https://doi.org/10.1109/VETECF.2009.5378833

[71] Heejung Yu, Howon Lee, and Hongbeom Jeon. 2017. What is 5G? Emerging 5G Mobile Services and Network Requirements. *Sustainability* 9, 10 (2017), 1848.

[72] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. 2017. Resilient datacenter load balancing in the wild. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 253–266.

[73] Wei Zhang, Jinho Hwang, Shriram Rajagopalan, K.K. Ramakrishnan, and Timothy Wood. 2016. Flurries: Countless Fine-Grained NFs for Flexible Per-Flow Customization. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies (CoNEXT '16)*. ACM, New York, NY, USA, 3–17. https://doi.org/10.1145/2999572.2999602

[74] Timothy Zhu, Daniel S. Berger, and Mor Harchol-Balter. 2016. SNC-Meister: Admitting More Tenants with Tail Latency SLOs. In *Proceedings of the Seventh ACM Symposium on Cloud Computing (SoCC '16)*. New York, NY, USA, 374–387.

[75] Timothy Zhu, Michael A. Kozuch, and Mor Harchol-Balter. 2017. WorkloadCompactor: reducing datacenter cost while providing tail latency SLO guarantees. In *SoCC*.

[76] Timothy Zhu, Alexey Tumanov, Michael A. Kozuch, Mor Harchol-Balter, and Gregory R. Ganger. 2014. PriorityMeister: Tail Latency QoS for Shared Networked Storage. In *Proceedings of the ACM Symposium on Cloud Computing (SOCC '14)*. New York, NY, USA, Article 29, 29:1–29:14 pages.