Scaling the LTE Control-Plane for Future Mobile Access

Arijit Banerjee
University of Utah
arijit@cs.utah.edu

Rajesh Mahindra
NEC Labs America Inc.
rajesh@nec-labs.com

Karthik Sundaresan
NEC Labs America Inc.
karthiks@nec-labs.com

Sneha Kasera
University of Utah
kasera@cs.utah.edu

Kobus Van der Merwe
University of Utah
kobus@cs.utah.edu

Sampath Rangarajan
NEC Labs America Inc.
sampath@nec-labs.com

Abstract

In addition to growth of data traffic, mobile networks are bracing for a significant rise in the control-plane signaling. While a complete re-design of the network to overcome inefficiencies may help alleviate the effects of signaling, our goal is to improve the design of the current platform to better manage the signaling. To meet our goal, we combine two key trends. Firstly, mobile operators are keen to transform their networks with the adoption of Network Function Virtualization (NFV) to ensure economies of scales. Secondly, growing popularity of cloud computing has led to advances in distributed systems. In bringing these trends together, we solve several challenges specific to the context of telecom networks. We present SCALE - A framework for effectively virtualizing the MME (Mobility Management Entity), a key control-plane element in LTE. SCALE is fully compatible with the 3GPP protocols, ensuring that it can be readily deployed in today's networks. SCALE enables (i) computational scaling with load and number of devices, and (ii) computational multiplexing across data centers, thereby reducing both, the latencies for control-plane processing, and the VM provisioning costs. Using an LTE prototype implementation and large-scale simulations, we show the efficacy of SCALE.

CCS Concepts

•**Networks** → **Network services;** *Wireless access points, base stations and infrastructure; Network control algorithms;*

Keywords

NFV, EPC, MME, scaling, elasticity, Geo-multiplexing

1. INTRODUCTION

Factors such as always-on connectivity, the rise of cloud-computing [1] and the growth of IoT devices projected at 26 billion by 2020 [2] have implications on the amount of control signaling in mobile networks. Nokia [3] estimates the growth in the signaling traffic would be 50% faster than the growth in data traffic. In some US markets, the peak network usage recorded was as high as 45 connection requests per device per hour [4]. In a European network, about 2500 signals per hour were generated by a single application [5], causing network outages. In LTE, this increased signaling traffic will have a significant impact on the performance of the MME, the main control-plane entity that handles 5 times more signaling, than any other entity [6].

As the first step towards evolving their mobile networks, operators are keen to consider NFV [7]. Recent trends strongly suggest that virtualized clouds [8] can provide high reliability at lower cost. With NFV, the virtualized MME entities would execute as VMs over general-purpose hardware. However, to ensure quick adoption of NFV in LTE, it is critical to effectively manage the virtual MME resources to handle the expected signaling growth. Efficient management of the MME resources encompasses 2 key aspects: *performance* and *lower operating costs*. Firstly, overloaded MME VMs can cause significant delays for connectivity and handovers, directly affecting the user experience. Secondly, control signaling does not generate any direct revenue for the mobile operators. To ensure cost effectiveness, it is important to dimension the VM resources according to current load. With the expected growth of certain IoT services that exhibit signaling-centric behavior, conservative provisioning may lead to large number of under-utilized VMs.

The transition from dedicated hardware to virtualized software based platforms will itself prove beneficial. However, MME implementations are designed for specialized hardware and cannot be directly virtualized since they are inelastic: hard to manage compute resources and rigid: involve static configurations. It is critical to apply experiences and concepts from the distributed systems [9] to virtual MMEs to ensure maximum gains. However, the concepts leveraged by distributed systems in cloud systems cannot be directly applied to a telecom service as the latter has unique characteristics that must be considered. Typical telecom services deal with well defined *interfaces* and *protocols*, *persistent sessions*, resulting in coupled storage and compute management and *highly distributed deployments*. A well-defined interface exists between the MMEs and the basestations.A

CoNEXT '15, Dec 1–4, 2015, Heidelberg, Germany

http://dx.doi.org/10.1145/2716281.2836104

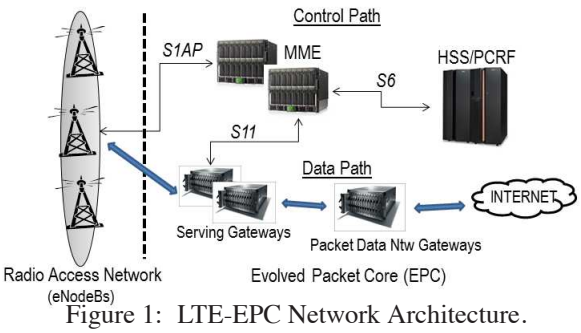


Figure 1: LTE-EPC Network Architecture.

device is persistently managed by the same MME for time-periods much longer than the duration of individual application flows. MME deployments are well distributed given the widespread presence of telecom data-centers [10]. On the other hand, cloud services have higher flexibility to leverage custom protocols, decouple storage and compute management as most sessions are short-lived and have relatively more centralized deployments (For instance, 3 DCs in the US in the case of EC2).

We present the design of SCALE, that systematically overcomes the afore-mentioned challenges in bridging the divide between virtualization and MME implementations. Our contributions are multi-fold: (1) SCALE re-architects the MME functionality into 2 parts–a front-end load balancer that maintains standard interfaces and a back-end elastic MME processing cluster. (2) SCALE instruments consistent hashing [11] for MME implementations to ensure scalability with large number of devices. The instrumentation includes a replication strategy that intelligently replicates states of devices and places the replicas across VMs. To devise this strategy, SCALE adopts a stochastic analysis driven approach that accounts for both compute and memory resources. (3) SCALE gives flexibility to operators to reduce costs by lowering the VM provisioning based on two key features: (i) leverage access patterns of devices, if available, to intelligently reduce memory usage due to replication and (ii) pro-actively replicate selective device states externally across data centers to leverage spatial multiplexing of compute resources. (4) SCALE is implemented on an end-to-end LTE Release-9 compatible testbed using the OpenEPC [12] platform. We perform extensive experiments to illustrate the inefficiencies with the current systems and show the feasibility of SCALE to work with existing protocols. In a particular instantiation, SCALE reduces the 99th percentile delay of processing the control-plane messages from more than 1s with current implementations to about 250ms. (5) We also perform system simulations to show the efficacy of SCALE at larger scales.

2. BACKGROUND

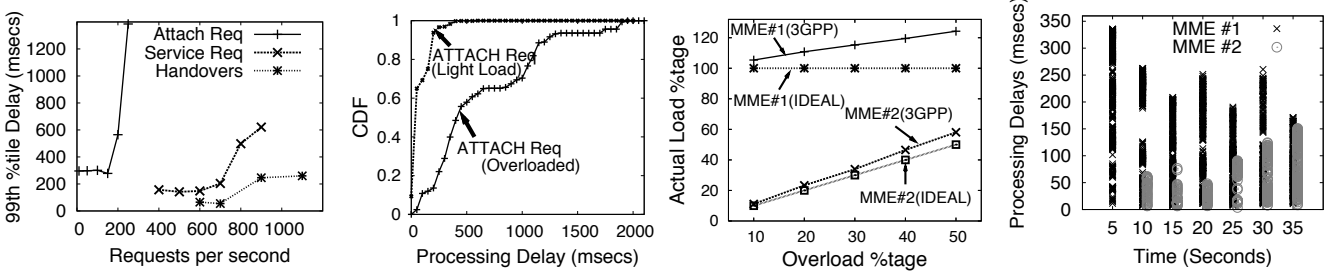
In this section, we provide a brief background of the LTE architecture and the MME functionality. LTE networks consists of the Radio Access Network (RAN) and the Evolved Packet Core (EPC) as shown in Figure 1. The RAN includes eNodeBs that wirelessly serve the users; the EPC consists of network entities that both *manage* the devices and *route* the data traffic. The control plane elements consists of the MME (Mobility Management Entity), HSS (Home Subscriber Server) and the PCRF (Policy and Charging Rules Function). The S-GW (Serving Gateway) and the P-GW (Packet Data Network Gateway) are routers that provide connectivity to the devices. HSS and PCRF are the database servers for user subscription information and QoS/billing policies respectively. MME is the key control node in the EPC network since it manages both device connectivity and mobility. In addition to being the entry point for control plane messages from the devices, it manages other control-plane entities using 3GPP standard [13] specific well-defined interfaces: (a) The S1AP interface with the eNodeBs carries the control protocols exchanged between the MMEs and the eNodeBs and the MME and the devices; (b) The S11 interface with the S-GW carries the protocols to create and destroy the data-path for each device and (c) The S6 interface with the HSS is used for protocol exchange to retrieve user information from the HSS.

MME Procedures: When a device is powered-on, it registers with the network. Henceforth, the device operates in two modes: *Active* and *Idle* mode. There are several procedures involving signaling between the MME nodes, eNodeBs and the devices. We briefly describe the key procedures: *(a) Attach/Re-Attach:* When a device powers-on or needs to transmit a packet while in Idle mode, it sends either an "attach request" or a "service request" message to the MME over the S1. The MME (re)-authenticates the device and (re)-establishes the data plane at the S-GW; the device moves back to Active mode. *(b) Tracking Area (TA) updates:* A device makes a transition into Idle mode after an inactivity timeout. The MME releases the data plane at the S-GW. While in Idle mode, the device sends periodic TA or location updates to the MME. *(c) Paging:* If a packet in the downlink is received for a device in Idle mode, the MME initiates the Paging procedure to all the eNodeBs in the device's TA and the device responds with a re-attach procedure. *(d) Handovers:* The MME establishes the data-path between the S-GW and the new eNodeB and tears down the data-path with the old eNodeB.

MME Provisioning: In performing the afore-mentioned procedures, an MME stores an associated *state* in memory for each device assigned to it, and executes *computational tasks* on the *state* to respond to the control requests from the device. Some of the *tasks* include protocol parsing, authentication, authorization, inter-eNodeB mobility management, roaming, WiFi mobility, paging and TA-updates, S-GW load-balancing, generation of Call-Data Records, billing, and lawful intercepts [14]. The *state* typically consists of timers, cryptography keys, S-GW, PGW and other data-path parameters, eNodeB radio resource management configurations, CDRs and location. Hence, MMEs have to be provisioned *jointly* for memory resources (to store state of all devices) and compute resources (to process requests of *Active* devices).

3. FUTURE NETWORK EVOLUTION

As the demand for mobile data grows at a significant pace, current networks need to evolve continuously to keep up



(a) Static Assignment (b) Overload Protection (c) Signaling Overhead (d) Scaling-out

Figure 2: Limitations of the current MME platform.

with the growth. The 5G vision for the radio access network (RAN) is to explore higher frequencies (e.g., mmWave) to increase bandwidth and reduce delays. However, in the case of the EPC network, the focus is to evolve the current components to meet future requirements of flexibility and cost reduction [15]. While the data-plane entities are critical, it is equally important to evolve the control-plane of the EPC network. As a first step to evolve the control-plane, operators are keen to adopt Network Function Virtualization (NFV) based MME deployments [7]. To maximize the effectiveness of virtualization in managing the expected growth of control-plane signaling, we list some requirements and opportunities for future virtual MME deployments:

1**. Scale of Operation:** MMEs face a unique challenge – while the number of active devices that generate signaling may be relatively lower, typically, the number of registered devices are significantly higher. This trend is going to dominate in future with the proliferation of IoT devices. Hence, as devices evolve from smartphones to wearables, sensors, vehicles etc., it is important that the design of the MME is light-weight to handle the growing number of devices.

2**. Elasticity:** Historically, scaling MMEs has been a hardware process involving upgrading to faster network processors while keeping the same architecture and software. To leverage virtualization in ensuring lower operating costs, it is critical to provision and scale-up and down the virtual MME resources proportional to the current signaling load.

3**. Distributed deployments:** Although operators have highly distributed DCs (Data Centers), most MME deployments are centralized to reduce provisioning costs. Within the NFV paradigm [10], there is an opportunity for operators to leverage virtualization to distribute the MME computation. Doing this efficiently would reduce provisioning costs, by multiplexing the resources across DCs and increase availability.

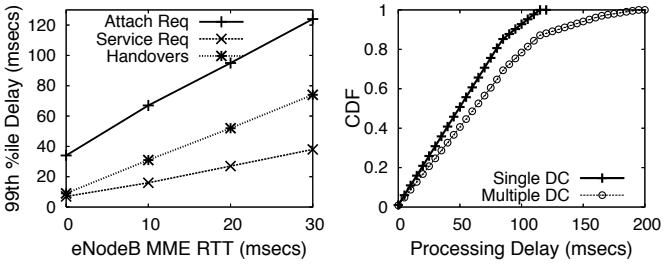
4**. Lowering Processing Delays:** Higher processing delays for critical requests like handovers have negative impacts on TCP [16]. Devices make frequent transitions to *Idle* mode to reduce battery usage [17]; delays in transitioning back to *Active* mode affect the web page download times. Thus, vendors are focussing on defining strict delays for control-plane in future systems [18].

3.1 State of the Art: Limitations

To meet future requirements, the idea of evolving the current MMEs by simply porting the code to a virtualized platform will be highly inefficient. The fundamental problem is that the MME design and implementations strictly adhere to 3GPP-based standards based on archaic assumptions: (i) over-provisioned MMEs deployed on specialized hardware, (ii) limited number of devices and (iii) infrequent capacity expansions. Field studies have shown that over-provisioned MME systems are also subject to persistent overloads [4]. Future application characteristics such as synchronous mass-access [19], where multiple event-triggered devices become active simultaneously , will aggravate the inefficiencies further due to spatio-temporal overloads and load-skewness.

Experimental Illustration: The OpenEPC platform [12] is a software-based implementation of 3GPP standard-compliant EPC components that executes over general purpose servers. We use experiments on a OpenEPC testbed (details in Section 5) to elaborate inefficiencies with current implementations. We leverage OpenEPC, since it is the only EPC platform that is readily available. Moreover, it is primarily a reference design that is implemented similar to current hardware-based MMEs. Although the absolute performance numbers from the OpenEPC experiments may not be valid for other commercial MME platforms, the trends shown in the experiments will generally hold to virtual MME platforms. The experiments primarily focus on the inefficiencies of elasticity of current MMEs. However, we break it down into 4 experiments for the ease of understanding individual procedures that contribute to the inefficiencies.

1**. Static Assignment:** The fundamental problem is that an eNodeB statically assigns devices to a particular MME and the requests from a device are always forwarded to the same MME. Specifically, a group of MME servers are clustered to form a pool and directly connect to all the eNodeBs in specific geographical areas as shown in Figure 1.When a device attaches to the network, the corresponding eNodeB selects an appropriate MME from the pool. After successful attachment, the device is assigned a temporary ID, known as the `GUTI` (*Globally Unique Temp id*), that contains the unique ID of the MME. Subsequent requests from that device are routed to the same MME by the eNodeBs using the `GUTI`. With static assignment, a sudden surge in the number of active devices can cause significant overloads in a MME. To illustrate the adverse effect of overloads, we measured the processing delays for different types of procedures processed by the same MME in Figure 2(a). As the load is increased by increasing the number of requests per second, the delays increase significantly after a particular threshold for each case. Once the compute capacity is reached, the



(a) Propagation Delays (b) Avg Load Conditions

Figure 3: MME pooling across multiple DCs.

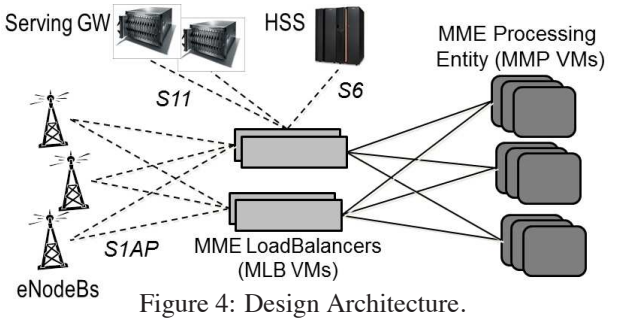


Figure 4: Design Architecture.

requests have to be queued, resulting in high and unpredictable delays. As done in current deployments, overloaded VMs could be avoided by conservatively assigning devices to each VM in order to handle the worst-case scenario where all the devices become active at the same time. Hence, static assignment would lead to provisioning significantly higher number of VMs.

2**. Overload Protection:** When overloaded, an MME has the provision to reassign devices to another MME in the same pool. The reassignment procedure, however, involves high overheads. Each device is sent messages asking it to re-initiate a control connection to ensure that the eNodeB would assign it to another MME server. In addition, messages are exchanged between the MMEs to transfer the state of the devices. The additional signaling causes high delays and further increase in load. In an experiment with 2 MMEs, Figure 2(b) illustrates this by comparing the delays for two cases: (i) the devices of MME1 attach to the network when MME1 is lightly loaded and (ii) devices of MME1 attach to the network when MME1 is overloaded, and MME1 reassigns them to MME2. To show the effect of extra signaling on increase in load, we plot the average load on both the MMEs as the percentage of overload on MME1 is increased in Figure 2(c). The ideal scenario represents the case where MME2 is able to absorb the additional load from MME1 without any overhead. Clearly, the average load due to the additional signaling increases on both MMEs compared to the ideal case as the amount of overload increases. In virtual MMEs, fine-grained load balancing among co-located VMs must be: (i) transparent to devices to ensure scalability and (ii) proactive instead of reactive to ensure low delays.

3**. Scaling-out:** Although it is possible to add MME servers to a pool, the procedure is cumbersome and designed for infrequently planned capacity expansions. For instance, when an MME is added, irrespective of its processing capability, it is initially configured with a lower weight. This ensures it is aggressively assigned devices by the eNodeBs compared to other MMEs. Moreover, only unregistered devices can be assigned to the newly added MMEs, limiting the ability to re-balance the existing load. In an experiment with an overloaded MME (MME1), MME2 is instantiated at about 10 seconds. The average load in the system is around 50 requests per second and 10% of the total requests are from unregistered devices. Figure 2(d) plots the connectivity delays perceived by the devices every 5 seconds in both the MMEs. Since the system is unable to rebalance the load from existing devices, it takes approximately 35 seconds for the delays at both the MMEs to equalize. The effect will be worse in large-scale deployments. Dynamic partitioning of device state needs careful thought since the primary driver for virtualization is the ability to adapt resources proportional to the load.

4**. Geo-multiplexing:** It is possible to provision resources across data centers (DCs) in current systems by deploying certain MME servers of the same pool at remote DCs. However, it is highly inflexible and inefficient due to static configurations: (i) The eNodeBs do not consider the propagation delays to the MMEs while assigning devices to an MME server. As shown in Figure 3(a), the propagation delays can adversely affect the overall control plane delays for different types of messages. (ii) Once a device is assigned to an MME placed in a remote DC, the requests from that device are always forwarded to the remote DC by the eNodeBs, even when the local DC is not overloaded. Figure 3(b) shows that such assignment leads to inflated control-plane delays even under average load conditions. Hence, there is a need to devise ways to *opportunistically* offload processing to remote DCs when the local DC resources face persistent loads.

4. SCALE: MME VIRTUALIZATION

Towards addressing the above limitations, we design and implement SCALE, a framework for efficient managing the resources of virtual MMEs. The first steps towards the design of an efficient virtual MME is to address the rigidity in current MME platforms. We begin with a description of SCALE's architecture followed by the detailed system design.

4.1 Architecture

In architecting a virtual MME framework, SCALE has to enable elasticity of MME resources while ensuring standards compatibility. To achieve elasticity, architecture must ensure that (i) the device assignment decisions taken by the eNodeBs can be over-ridden by the MMEs and (ii) the MME resource management procedures, such as VM creation and deletion should be transparent to the eNodeBs. Additionally, support for standard interfaces is critical for several reasons: *(a)* Incremental deployment of virtual MME alongside legacy EPC nodes, *(b)* eNodeBs are commoditized and expensive to replace or upgrade and *(c)* The presence of Middleboxes [20, 21] for traffic analytics and optimizations.

In order to meet these two requirements, SCALE decouples standard interfaces and eNodeB based device assignment from

the MME process implementation. SCALE achieves such decoupling by architecting the MME cluster as two separat logical functions as shown in Figure 4. The figure show a single instance of an MME pool realized by SCALE tha would be deployed at a particular DC.

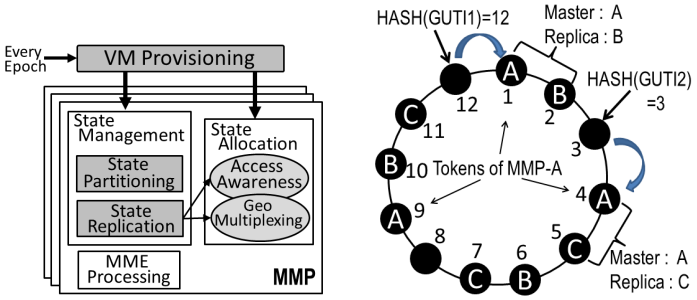
1. MLB (MME LoadBalancer): The design of the MLB is es sentially similar to that of HTTP load-balancers in IT cloud The MLB acts as a MME proxy: each MLB entity represen a single MME entity by exposing standard interfaces to th external entities. For instance, it establishes the S1AP an S11 interfaces with the eNodeBs and S-GWs respectively. The MLB essentially negates the effect of device assignment and request routing decisions taken by the eNodeBs. The eNodeBs simply choose the MLB to route a device request and the MLB forwards these requests to the appropriate MME processing VM. Hence, the MLB ensures that the device (re)-assignment decisions within the MMP processing cluster can be performed without affecting the eNodeBs or the S-GWs.

2. MMP (MME Processing Entity): The MMP VMs collectively form a MME pool to process requests from all devices belonging to the geographic area managed by that pool. Essentially each MMP VM of a certain pool can process requests from devices assigned to different MMEs in that pool. This requires device-to-MME mapping information to be stored for each device at the MMP VMs. SCALE adds this information to the existing state information that the MMP VMs already store for each device. Such a design improves the utilization of the cluster as the devices belonging to a particular DC can be flexibly assigned across the MMP VMs.

4.2 System Design Principles

While the architecture effectively decouples the MME processing from the eNodeB interfaces, the MMP VMs of a pool need to collectively *manage* all the registered devices. Figure 5(a) depicts the system components of SCALE. Every epoch (time granularity of several minutes), SCALE provisions the appropriate number of MMP VMs independently at each DC, based on the expected load for the current epoch. The VM provisioning procedure triggers the State Management routine, that effectively *assigns* current devices across the active MMP VMs. Subsequently, the State Allocation routine strives to refine the device allocations to reduce VM usage and assign selected devices to MMP VMs across DCs to avoid DC overload scenarios. In designing these components, following considerations are key in the context of MMEs:

1**. Coupling between state management and computation:** In addition to managing (storing) the state of all the devices, the MMEs perform intensive processing on the state of the active devices. While it may be trivial to assign the device states uniformly across the VMs, performing it dynamically: (a) to provision and account for the compute resources and (b) in presence of VM addition or removal, is non-trivial. Hence, SCALE breaks the problem into two steps, namely state partitioning and state replication as shown in Figure 5(a). At a high-level, the problem of state partitioning is similar to that faced by distributed key-value storage systems, such as dynamoDB and Cassandra, that have been deployed successfully at scale [9, 22]. Thus, SCALE applies the same basic



(a) MMP Design (b) Consistent Hashing

Figure 5: SCALE's key design components

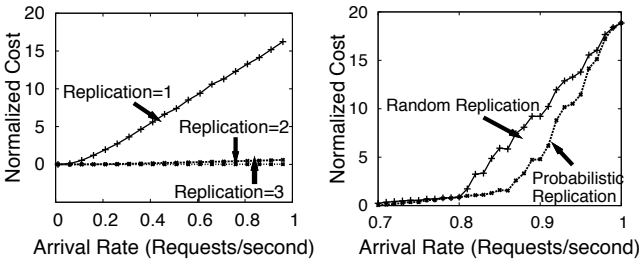
technique leveraged by such systems, i.e., consistent hashing and tailors it to the needs of the MME. Additionally, SCALE relies on state replication to manage the compute resources at fine time-scales and avoid overload of MMP VMs. However, several factors need to be considered when devising the ideal replication strategy: (i) Compute provisioning for processing the requests from active devices, (ii) Consistency across the multiple replicas is critical for proper operation of MME protocols. To this end, SCALE leverages an analytical framework, that accounts for the above factors, to determine the right number and placement of the replicas of device states at appropriate MMP VMs.

2**. Resource Provisioning Costs:** As opposed to IT clouds that have the ability to scale *infinitely*, there is expected to be an associated cost with every additional VM in an operator's DC. Typically, operator clouds have higher presence and hence, it is harder for operators to over-provision resources at individual DCs. Moreover, multiple network services besides the EPC, typically share the resources within a single DC. Also, the total number of registered devices for which state has to be stored can be significantly higher than the number of active devices during most time instances. Consequently, storing multiple replicas of all devices with limited memory resources becomes challenging, requiring SCALE to be prudent about *which devices' state need to be replicated*. To address these challenges, SCALE couples VM Provisioning with intelligent state allocation as shown in Figure 5(a). SCALE leverages *access awareness* or knowledge of past device connectivity patterns to (i) reduce the number of replicas for selective devices; and (ii) accommodate replicas of external (from remote DCs) device states at a local DC - this enables computational multiplexing across DCs at fine time-scales without incurring additional VM provisioning.

4.3 State Management

We now describe each component in detail.

4.3.1 State Partitioning: SCALE leverages consistent hashing [11] to uniformly distribute device states across the active MMP VMs. In consistent hashing, the output range of a hash function is treated as a "fixed circular ring". Figure 5(b) shows a simple instantiation of a consistent hash ring with output range [1-12]. Each MMP VM is represented by a number of *tokens* (random numbers), and the tokens are hashed to the ring so that each VM gets assigned to multiple points on the ring. For example, MMP-A gets assigned to points 1, 4



(a) Effect of Replication (b) Access Awareness

Figure 6: State Allocation

& 9 on the ring. Then, each incoming device is assigned to an MMP VM by first hashing its GUTI to yield its position on the ring. The ring is then traversed in clockwise direction to determine the first MMP VM that has a position larger than the device's position on the ring; this VM becomes the master MMP for that device. For instance, in Figure 5(b), the MMP-A is selected as the master MMP for the device with GUTI1. Partitioning the device states using consistent hashing ensures that (a) MMP VMs scale incrementally in a scalable, decentralized way: addition or removal of VM only affects state re-assignment among neighboring VMs, (b) MLB VMs do not need to maintain routing tables for device to MMP mapping, making them efficient in terms of both lower memory usage as well as scalable with faster lookup speeds. Hence, when a control message or *request* from a device is received by the MLB for processing, it simply forwards the request to the appropriate MMP using the hash value of the GUTI. In case of a request from an unregistered device, the MLB first assigns it a GUTI before routing its request.

4.3.2 State Replication: To counter overloads, SCALE devices a strategy for the number and placement of replicas.

1.Number of Replicas, R: While increasing the number of replicas leads to better load balancing, it contributes to higher memory requirements and synchronization costs to ensure consistency. To strike a balance between these conflicting objectives, we employ stochastic analysis to model the impact of replication in consistent hashing on load balancing. The analysis helps us understand the impact of replication frequency R on the average delay ($\bar{C}$) experienced by a device's control request within a DC during a time epoch of duration T. We defer the proof to the Appendix and instead focus on the implications. The closed-form expression for the average delay, $\bar{C}$ as a function of R given in the Equation 10 in Appendix-$A1$. Using the analysis, we now plot the average cost or delay in processing the requests as a function of the arrival rate of requests. As seen from Figure 6(a), as the arrival rate increases, the load on the VMs increases causing higher normalized cost (processing delays) for requests when no replication is used. However, by replicating the state of a device in just one other VM drastically reduces the delays perceived by the devices. Hence, the utility of increasing the replication count (R) beyond 2 does not have significant value. This is promising as it indicates that most of the load balancing benefits can be realized with just 2 copies of the state. In SCALE, replication is performed by the master MMP asynchronously as and when a request for a device is received. After processing the request for a device, the master MMP may choose to replicate the state of that device to its neighboring MMP on the hash ring.

2.Placement of Replicas: Although SCALE replicates the state of each device only once, the notion of "tokens" in consistent hashing aids load balancing. Each node has multiple neighbor nodes on the ring since each node is represented by multiple tokens on the ring. Hence, the states of devices assigned to a particular VM end up getting replicated uniformly across multiple other VMs, thereby avoiding hot-spots during replication. For instance, in Figure 5(b), while the master MMP for both devices- GUTI1 & GUTI2 is MMP-A, their states are replicated at MMPs-B&C resp. Although using higher number of tokens increases the number of VMs involved in exchanging states during VM addition or removal, most of the benefit is achieved even with a relatively low number of tokens.

4.4 VM Provisioning

Every epoch, the MMP VMs are provisioned by considering the maximum of the processing and memory requirements. While SCALE estimates the processing requirement for the current epoch based on the load from the past epochs, the memory requirements are dictated by the storage space required for the state of the currently registered devices. Let $K(t)$ be the number of registered devices, and $\bar{L}(t)$ be the average expected signaling load from the existing devices at the DC in an upcoming epoch t. Let N is the number of requests that each MMP VM can process in every epoch, based on its compute power and S represents the maximum number of devices whose state can be stored in it, based on its memory capacity. The number of MMP VMs required to meet the processing requirements for an epoch t is computed by dividing the expected signaling load $\bar{L}(t)$ by the compute capacity N of each MMP VM; the number of MMP VMs required to meet the memory requirements is computed by dividing the aggregate storage required for all the replicas of the existing devices $K(t)$ by the memory capacity S of a MMP VM:

$$\begin{aligned} V_C(t) &= \left\lceil \left(\frac{\bar{L}(t)}{N} \right) \right\rceil, V_S(t) = \left\lceil \beta \cdot \left(\frac{R \cdot K(t)}{S} \right) \right\rceil \quad (1) \\ \bar{L}(t) &\leftarrow \alpha L(t-1) + (1-\alpha)\bar{L}(t-1) \end{aligned}$$

where β is a parameter ($(0, 1]$) used to control the VM provisioning, and R is the number of replicas ($R = 2$) needed for each device. The average load for an upcoming epoch ($\bar{L}(t)$) is estimated as a moving average of actual ($L(t-1)$) and average loads from the prior epoch. Now, the required number of MMP VMs is $V(t) = \max\{V_C(t), V_S(t)\}$.

The choice of β plays a critical role in VM provisioning. Recall that the total number of registered devices is significantly higher than the active ones in an epoch. Further, a large fraction of these devices have low probability of access in a given epoch. Simply storing R copies of the state of each device would result in the memory component (V_S) dominating the VM provisioning costs, unnecessarily driving it high. While β can be used as a control parameter to restrict the VM provision costs, this will amount to some devices

not being replicated and could lead to increased processing delays for those devices. Hence, it is important to appropriately select both, the value of β and the devices that will get replicated, which we address next.

4.5 State Allocation

SCALE keeps track of the average access frequency of a device in an epoch (as a moving average) and includes it with the rest of the state that is already stored for the device. Future networks are expected to comprise of a large number of IoT devices; some studies have shown that certain IoT devices exhibit predictable connectivity patterns [19]. For instance, smart meters upload information to the cloud periodically. Such predictable access patterns, when available, contribute to a more accurate profiling of device access frequency. SCALE leverages such access frequency information of devices to intelligently determine if their state will get replicated. This can lead to reduced VM provisioning costs on two fronts: (i) within a DC without an appreciable impact on load balancing; and (ii) across DCs by making room in each DC to store state of devices from remote DCs, thereby allowing for multiplexing of resources across DCs.

4.5.1 Access-aware Replication: Let w_i be the access frequency of a device i; highly active devices will have a high value for w_i while devices that are mostly dormant will have lower values for w_i. At every epoch in a DC, SCALE strives to select a subset of devices for whom a single replica (i.e $R = 1$) of the state might suffice without effecting the efficacy of load-balancing. SCALE selects the number of such devices appropriately to achieve the required reduction in the number of VMs. Such reduction is valid in the case when the memory capacity of the MMP VMs is the main constraint ($V_S > V_C$ in Equation 1). Specifically, SCALE estimates the number of devices:$\hat{K}(x)$, with low access probability, such that $w_i \leq x$ (eg. $x = 0.1$). The state of such devices is only maintained in their master MMP VM. Part of this reclaimed memory can be used to accommodate new devices (S_n, e.g., 5% of K) that might register with this DC in the epoch as well as for state of devices from remote DCs for the purpose of multiplexing (S_m, explained in the next subsection).Thus, only $\hat{K}(x) - S_n - S_m$ effectively contributes to reduction in memory, resulting in

$$\beta(x) = 1 - \frac{(\hat{K}(x) - S_n - S_m)}{RK} \tag{2}$$

By increasing the fraction of devices whose state is not replicated (increasing x), we reduce $\beta(x)$ and hence the VM provisioning cost. Thus, based on the distribution of access probabilities of devices, an appropriate $\beta(x)$ can be used to determine the VM provisioning (V) using Equation 1. Once the provisioning is done, the actual replication of device states are executed in an access-aware manner as follows: (i) Each device state is stored in its master MMP VM (ii) The replica of the state is stored in the neighboring MMP VM on the hash ring, based on the remaining memory and proportional to its access probability as,

$$\mathbb{P}_i(\text{rep}) = \left(\frac{w_i}{\sum_j w_j}\right)(VS - S_n - S_m - K) \tag{3}$$

To understand the impact of such access-aware replication, we extend our stochastic model to incorporate access-awareness. We defer the proof of the analysis to the Appendix and focus on the implications. The closed-form expression for the average delay for a device i in a memory-constrained environment, $\bar{C}_i$ gets modified to Equation 13 in Appendix-$A2$. Figure 6(b) shows the normalized cost (processing delay) comparison of SCALE with a system that randomly picks devices whose states get a replica (i.e. access unaware). Clearly, leveraging access patterns helps SCALE reduce the impact of insufficient memory. For instance, in the plot, to support a given load of say 0.85, SCALE reduces control-plane latency by $5x$ as compared to the baseline system that performs access unaware replication. Alternately, access-aware replication allows SCALE to provide a comparable performance at a reduced VM provisioning cost. For instance, in a particular case with 25% of the devices having low frequency of access, SCALE reduces VMs by 10% (as shown in Section 5.1).

4.5.2 Geo-Multiplexing: By provisioning resources and maintaining separate hash rings for MMP VMs at each individual DC, SCALE ensures that the master MMP VM for every device is located in the local DC. This allows for minimizing delays by processing as many requests as possible at the local DC. However, to load balance the processing across DCs during periods of local DC overloads, SCALE needs to (i) make room (S_m^i) in each DC i for state of devices from other DCs ($j \neq i$), and (ii) decide which devices in a DC will have their state replicated remotely and in which remote DC. While the former is handled by the DC, the latter is handled by the MMP VMs independently for scalability. The sequence of steps are as follows.

1**. DC-level operation:** Each DC i (i) independently chooses S_m^i (*state budget*) to capture potential under-utilization in processing in an epoch (e.g., 10% of net processing $V \cdot N$). S_m^i is the maximum amount of state, DC i will accept from devices belonging to external DCs; (ii) maintains and updates a variable $\hat{S}_m^i$, that represents the amount of external device state from the total S_m^i, that is unused; (iii) periodically broadcasts the value of $\hat{S}_m^i$ to the neighboring DCs. (iv) periodically updates the value of $S_m^i, \hat{S}_m^i$ to track the average processing load and hence room for processing external state; and (v) if at any stage $\hat{S}_m^i \geq S_m^i$ or $\hat{S}_m^i = S_m^i = 0$ (over-load), it requests the other DCs to appropriately reduce their share of device states stored in DC i to reflect the reduction in S_m^i.

2**. MMP-level operation:** (1) *Choice of devices:* With each DC i making a room of S_m^i for external state, it has an equivalent room for S_m^i of its devices to have their state replicated remotely (to ensure conservation of external state resources across DCs). However, deciding which devices' state need to be stored remotely is not straight-forward. Note that each DC would like to process most of its high probability devices locally so as to keep the processing delays low. At the same time, storing low probability device states remotely will not help multiplex resources from remote DCs, since the probability of those devices appearing is low to

begin with. To balance between processing delays and resource multiplexing, each MMP VM v_k (at a DC) in SCALE selects its share of $\frac{S_m^i}{V}$ devices of high access probability ($w_i \geq 0.5$) in an epoch, to be replicated once in the external space ($S_m^j, j \neq i$) reserved by one of the remote DCs. However, this replication is in addition to the two copies that are stored locally for the high probability devices so as to not affect their processing delays appreciably. Further, SCALE replicates the state of a device with $w_i \geq 0.5$ externally, proportional to its access probability as $\left(\frac{w_i}{\sum_{j \in v_k : w_j \geq 0.5} w_j}\right)\left(\frac{S_m^i}{V}\right)$.

(2) *Choice of remote DCs:* Once a device's state is chosen by a MMP VM for external replication, it determines the appropriate destination (remote DC) for the state based on two factors: the remote DC's current occupancy by external state ($\hat{S}_m^j$) and inter-DC propagation delay. Specifically, the MMP VM executes the following steps: (i) it checks if at least one DC j has available budget for external state i.e., $\hat{S}_m^j \geq 0$; (ii) if multiple remote DCs have a non-zero budget, it probabilistically selects the appropriate one proportional to the following metric p:

$$p = \frac{\frac{1}{D_{ik}}}{\sum_{i=1}^{C} \frac{1}{D_{ij}}}$$

where D_{ij} is the propagation delay between DC i and j; C is the total number of remote DCs with $\hat{S}_m^j \geq 0$. SCALE performs such probabilistic replication to ensure that hot spots are avoided in cases where certain DCs have low delays to multiple adjacent DCs. (iv) if requested by a DC j to reduce its replications by $y\%$, the MMP VM deletes its share of external state replication ($\frac{y}{V}\%$) at DCj by starting with those states having a relatively low access probability.

(3) *Execution:* Similar to local replication, geo-replication is performed asynchronously by the master MMP. After processing the request of a device, the master MMP may choose to replicate its state to a remote DC. The replication is done using a MLB VM of the remote DC, which selects the MMP VM based on the hash ring of that DC. Additionally, the master MMP attaches the location of the external state of a device (i.e., the remote DC id) to its current state. This ensures that the requests for the devices can be routed to the appropriate DC by the local MMP VMs.

4.6 Fine-grained Load-balancing

The state management and allocation appropriately maintain the replicas of the state of the devices to ensure the ability of the MLB VMs to perform efficient load balancing. We elaborate on key design considerations for load-balancing:

(1) Low-overhead: The online load balancing is designed with minimal overhead on the MLB, to ensure faster look-up speeds when routing requests to the MMPs. Specifically, the MLB is unaware of the (a) number and (b) placement of the replicas of the state of a device to avoid memory and exchange of per-device information at the MLB VMs. Hence, the only meta-data information needed by the MLB VMs are the (i) updated consistent hash ring as MMP VMs are added or removed, and (ii) current load (moving average of CPU utilization) on each MMP VM.

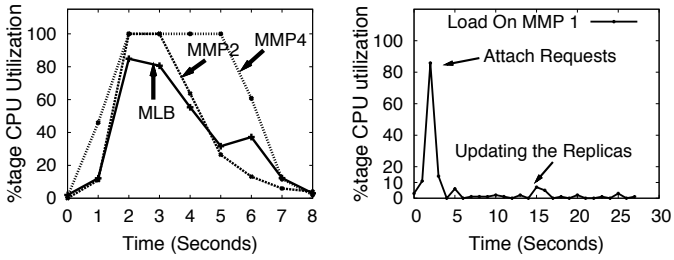
(2) Granularity: As elaborated in Section 2, the (a) processing requirements for a device is higher while it is in the *Active* mode and (b) the processing delays are more critical when the device makes a transition from the *Idle* to *Active* mode. Keeping this in mind, the MLB assigns the least loaded MMP VM among the choices for a device request when it makes a transition to the *Active* mode. Subsequent requests are sent to the same MMP VM until the device make a transition back to the *Idle* mode. This design choice was based on a few key observations: (i) While the device is *Active*, the state of the device is larger and the MMP VM maintains several timers. Hence, maintaining consistent state across VMs while the device is *Active* makes the system complicated. By load balancing only when the device enters the *Active* mode, SCALE performs updates of the replicas once the device goes back to *Idle* mode; (ii) The routing implementation requires more meta-information since, once the device is in the *Active* mode, the subsequent requests do not contain its GUTI.

When a request is received from an existing device, the MLB extracts the GUTI of the device from the request and hashes it to obtain both the master MMP and the replica MMP VMs. The request is forwarded to the least loaded MMP VM. When a MMP VM receives a request, it performs one of the following tasks:(1) it processes the request if it has the state of the device;(2) it forwards the request to the master MMP if it does not have the state of the device. This may happen if the device has been replicated only once; (3) it forwards the processing request to the MLB of the appropriate remote DC, if its load is above a threshold, and the device's state has been replicated externally.

5. PROTOTYPE AND EVALUATION

We implement a prototype of SCALE on the OpenEPC platform [12], since we licensed it's source-code. The OpenEPC testbed in our lab is a LTE Release 9 compatible network, consisting of standard EPC entities and an eNodeB emulator. Each component is written in C, and deployed over Intel based servers running Linux. Since the original OpenEPC implementation does not support the split architecture for the MME from Figure 4, we implemented the MLB and MMP components by modifying the source code for the MME. Once the architecture was implemented and tested, we implemented all the components of SCALE from Figure 5(a) by modifying the source code for the MLB and MMP components. Although, our implementation is mainly focussed on the MLB and MMP components, we spent considerable effort on modifying and stabilizing the code for entities like the eNodeB emulator, S-and P-GWs to ensure that a reasonable amount of load, in terms of device requests per second could be supported.

eNodeB: The eNodeB emulator supports the higher-layer protocols of the eNodeB. The eNodeB includes an interface that accepts device connections. We implemented a python based load generator that emulates device connections by generating attach/reattach/handover requests with different connectivity patterns.



(a) MLB Processing (b) State Replication

Figure 7: E1,E2: Feasibility of SCALE

MLB: Most of our implementation effort was spent on the MLB. To implement the MLB, we modified and added code to the original MME function of the openEPC. The MLB has three subcomponents: *(i) Standard Interfaces*: The MLB supports the interfaces S1AP, S11 and S6 with the eNodeBs, S-GWs and the HSS resp. as shown in Figure 1. Hence, the MLB maintains standard compliant interactions with the other components (eNodeBs, S-GWs etc.) and hence acts as an MME to them. As devices make transitions between *Active* and *Idle* modes, the MLB is implemented such that it ensures routing of requests to the appropriate MMP VMs. The requests may belong to any of the above mentioned interfaces. *(ii) Load Balancing*: We implemented the Consistent Hashing functionality using the MD5 hash libraries and linked it to the S1AP protocol parsing function in the MLB. When the MLB receives a (re)-attach request from a device, it extracts its GUTI and obtains the corresponding master and replica MMP VMs. The initial request is then forwarded to the least loaded MMP VM. However, as per the 3GPP standard, while in the *Active* mode, the subsequent requests for the same device from the eNodeB and the S-GW contain other unique identifiers: *S1AP-id* and *S11-tunnel-id* respectively, each assigned by the associated MMP. Hence, in SCALE, each MMP embeds its unique ID in both the *S1AP-id* & *S11-tunnel-id*, thus enabling the MLB to route the subsequent requests to the appropriate active MMP. *(iii) MMP Interface*: The requests from the eNodeBs are forwarded by the MLB to the MMP VMs using SCTP connections using an interface similar to S1AP. To receive meta-information such as updates on the consistent hash ring and load information from the MMP VMs, an existing TCP connection used for the purpose of management is leveraged.

MMP: The following functionalities were implemented in the MMP: *(i) State Partitioning*: The Consistent Hashing functionality was added to (re-)partition the state of devices across the current MMP VMs. *(ii) State Replication*: The master MMP VM is responsible for replicating the state of a device both within a DC and across DCs. We implemented the replication framework, such that the master MMP VM replicates the state of a device after it processes its initial attach request. In case of within a DC, the master MMP and the replica VM uses a direct TCP connection for state replication and synchronization to ensure consistency. For inter-DC, a similar protocol is used except that the master MMP communicates with the remote replica through the remote DC's MLB.

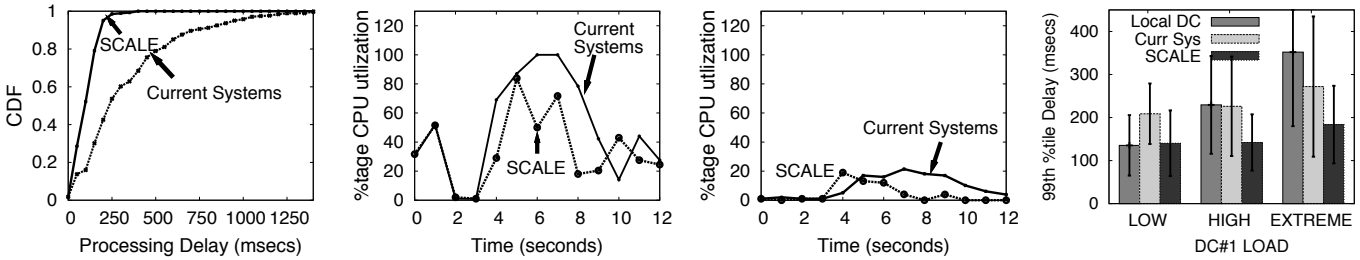
5.1 Evaluation

We perform extensive experiments and simulations to show the efficacy of SCALE based on different metrics. We measure (a) the end-to-end delay of the control-plane requests as perceived by the devices; (b) the number of VMs required to process requests to achieve a target control-plane latency requirement and (c) the average CPU load of VMs to show the efficacy in managing overloads in VMs.

1**. Prototype Evaluation:** We first describe the results of experiments from our prototype implementation. We were unable to perform experiments with large number of VMs on public clouds, such as EC2 due to licensing restrictions. Additionally, the original OpenEPC implementations do not support the generation and handling of significantly high loads. Note that we posses the source-code and did spend considerable effort to enable higher loads that aided our evaluation. Nonetheless, our small-scale prototype provided key insights that helped the design and a real implementation proves the feasibility of its components in practice.

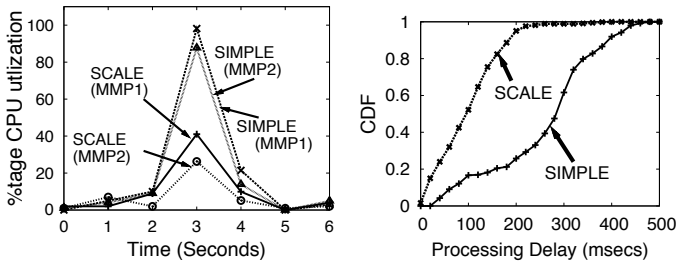
E1: Overhead of MLB: It is not intuitive whether the overhead introduced by the MLB in SCALE's architecture outweighs its benefit in providing elasticity and fine-grained load balancing in the MMP cluster. To prove the feasibility of the MLB, we perform an experiment with a single MLB VM and a MMP VMs. At each step, we add a MMP VMs and also attach corresponding devices to saturate the CPU load on the MMP VMs. We stopped the experiment at 4 MMP VMs, since we observed reasonable CPU load on the MLB VM. We plot the CPU load of the MLB VM and a couple of the MMP VMs in Figure 7(a). Clearly, while the CPUs of the MMP VMs are completely utilized, the maximum CPU load on the MLB VM is slightly below 80% utilization. This is a promising result given that MLB code was modified from an existing MME code base and not optimized to function as a MME proxy.

E2: Replication Overhead: Another key difference of SCALE from current status quo MME systems is the use of proactive replication of device states to ensure efficient load balancing. To study the overhead of replication on the CPU cycles spent by the MMP VMs, we setup the prototype with 4 MMP VMs. The eNodeB emulator is configured to generate around 200 active devices whose state is stored at MMP1 and replicated across other MMP VMs. We plot the CPU load of MMP1 in Figure 7(b). We force the MLB to forward all the requests to MMP1. In order to process the requests, the load on MMP1 reaches almost 90% at around 2-4 seconds into the experiment. After about 10 seconds of inactivity, all the devices make a transition into *Idle* mode (at about 15s). At this moment, MMP1 updates the copy of the state of all the devices in the respective replica MMPs. As seen from the figure, at about 15 seconds, the CPU load due to replication at MMP1 is less than 8%. Although this result shows the low overhead of proactive replication, note that the replication framework on the prototype was built over existing code base of OpenEPC. With further optimizations for state synchronization, e.g., differential replication and using technologies like remote direct memory access (RDMA) [23], the overhead can be significantly reduced.



(a) Processing delays (b) Load on MMP1 (c) Load on MMP2 (d) Geo-multiplexing

Figure 8: E4: Efficacy of SCALE over Current (3GPP standard-based) Systems



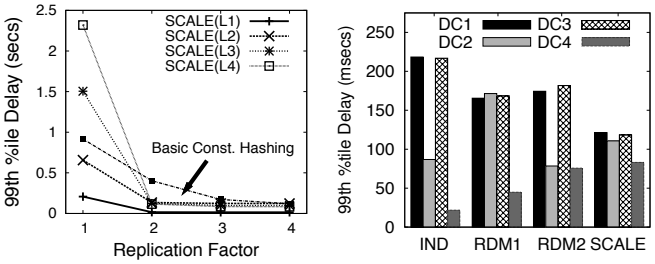
(a) CPU Usage (b) Delay

Figure 9: E3: Placement of Replicas

E3: Placement of Replicas: We compare SCALE with a system, namely SIMPLE that uniformly distributes the state of the devices across existing VMs and additionally replicates the states of each VM to another VM. To the best of our knowledge, the design of SIMPLE is representative of a few commercially available virtual MME systems. While such an approach keeps the implementation simple, it has several drawbacks compared to SCALE: (i) The MLB has to maintain per-device routing table, hindering scalability to the future device growth (ii) The load balancing is inefficient in handling VM overloads. To drive our point, we conduct an experiment with 5 MMP VMs where we generate high signaling traffic on MMP1, about twice its processing capacity. As shown in Figure 9(b), the 99th %tile delays perceived with SIMPLE replication are more than 400ms, while the same for SCALE are below 200ms. The high delay in the former is because the state of the devices assigned to MMP1 are only replicated to MMP2. Hence, when the processing load on MMP1 is sufficiently high, it causes overloads on both the MMPs as seen in Figure 9(a). However, SCALE replicates the state of disjoint subsets of devices assigned to MMP1 onto MMP2, MMP3, and MMP4, causing relatively low loads on both MMP1 and MMP2. The above results highlights the importance of SCALE's choice of utilizing consistent hashing with tokens for distributed placement of the replicas (Section 4.3).

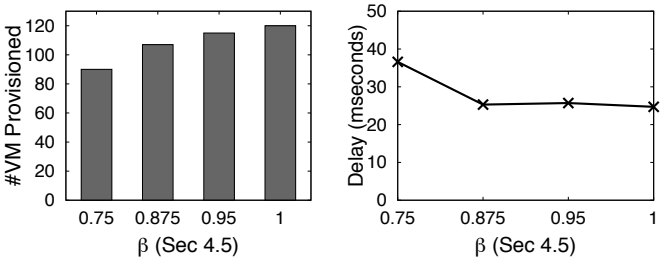
E4: The Status Quo: We compare the efficacy of SCALE over with standard-based MME systems that were described earlier in Section 3. We conduct experiments to show the benefits of SCALE for two types of scenarios:
(i) *VM Overloads Within a DC*: While current systems are reactive to overloads by transferring devices to other MMEs, SCALE performs proactive replication of device state across VMs to ensure that fine-grained load balancing. To quantify the benefit, we conduct an experiment with the same setup consisting of 2 MMP VMs and configure the testbed such that MMP1 receives a high number of device requests above its processing capacity. To emulate current systems with our setup, the MMPs are statically assigned devices with no replication of state. When MMP1 gets overloaded, it selects several active devices and sends messages to the devices to reconnect again, to ensure they are assigned to the other MMP by the eNodeBs. In addition, MMP1 transfers the state of those devices to MMP2 to ensure that MMP2 can continue the processing. However, in the case of SCALE, the state of the device assigned to MMP1 are proactively replicated to MMP2. While in a real deployment of SCALE, the replicas of the state of devices stored in MMP1 would get replicated across multiple other MMP VMs, we use only 2 VMs in this experiment for simplicity. In Figure 8(a), we plot the CDF of control-plane delays of the devices assigned to MMP1 for both the cases. Clearly, the 99th %tile delay is more than 1 second with current systems that take a reactive approach as opposed less than 250 ms with SCALE. Such high delays affect connectivity times for devices causing degraded QoE.

We also plot the CPU load on both the MMPs in Figures 8(b) and 8(c) respectively. SCALE ensures that the load on the MMP1 does not reach 100%, thus avoiding high queueing delays for the requests. SCALE effectively offloads processing to MMP2 at fine time-scales. While the current system also strives to offload processing to MMP2, the overhead of the signaling per device to transfer state of the devices between the MMPs leads to a higher CPU load on both the MMPs compared to the case with SCALE. Hence, SCALE's state management is effective in managing the load across MMP VMs to ensure low control-plane delays.
(ii) *Persistent DC overloads*: To show the ability of SCALE to perform fine-grained load balancing across DCs, we conduct an experiment with 3 DCs. While the DCs 2&3 are fixed to be lightly loaded, the load of DC1 is varied. We also emulate inter-DC propagation delays using netem [24]. We setup the experiment for 3 different cases: *(a) Local DC*: Resources are not pooled across DCs, hence the devices are always processed at the local DCs. *(b) Current Systems*: In current deployments, resources could be multiplexed across DCs by deploying a MME pool across DCs as explained in Section 3. *(c)* SCALE: It proactively replicates selected state across DCs. In Figure 8(d), we plot the mean and standard deviation of the 99th %tile delays perceived by the devices registered with the DC1 under different load conditions of DC1. In current systems, requests from devices that are



(a) Impact of Replication (b) Inter-DC

Figure 10: S1,S2: State Management and Allocation



(a) #VM Provisioned (b) Control-plane Delays

Figure 11: S3: Access Aware Replication

assigned to a remote DC, are always processed at the remote DC irrespective of the load conditions at the local DC. Hence, in the case of low loads at DC1, the delays are higher for current systems due to the propagation delays to the remote DCs. However, SCALE processes all requests locally at the DC1 since the master MMP for each device is located at the local DC, resulting in low delays. Even for higher loads at DC1, SCALE achieves better control-plane delays than current systems as shown in Figure 8(d). Unlike current systems that have to be statically configured, SCALE's replication strategy dynamically accounts for both (1) load conditions at DCs and (2) inter-DC propagation delays. This replication strategy give SCALE the ability to perform more efficient and fine-grained load balancing.

2**. System Simulations:** To show the efficacy of SCALE in larger setups with higher number of VMs and devices, we built a custom event-driven simulator in Python. The simulator is split into a load generator, that generates requests with different access patterns and a cluster emulator that emulates the processing at the MMP VMs. We now highlight the key components of SCALE, with focus not just on their efficacy but also stress on a few key design considerations.

S1: State Management: Recall from Section 4.3 that SCALE leverages consistent hashing for state partitioning across active MMP VMs and maintains 2 replicas of each device state to ensure efficient load balancing. To verify these design choices, we setup a cluster of 30 MMP VMs and initialize 80K devices that are assigned to the VMs based on consistent hashing. Each VM is represented by 5 tokens on the hash ring.We repeat the experiments for 4 different load scenarios and measure the 99th percentile of connectivity delays of the devices as shown in Figure 10(a). In each run, certain VMs are selected to have higher number of active devices than their processing capacity such that the load will be skewed across VMs. The different scenarios L1-L4 represent increasing skewness with L1 having lowest and L4 having the highest skewness across the VMs. As shown in the Figure, irrespective of the degree of load skewness, most of the benefit is obtained by replicating twice and replication beyond that does not decrease the connectivity delays significantly.

To show the effect of mapping each VM as multiple tokens on the ring, we repeat the experiment with the basic consistent hashing without tokens. This algorithm simply maps each VM directly on the ring, causing (i) uneven distribution of state, (ii) all the states of a VM are replicated onto the neighboring VM. Hence, when a VM is overloaded, its load can only be offloaded to another VM. Hence, further replication is required to balance the load (Figure 10(a)).

S2: Geo-Multiplexing: SCALE proactively replicates state across DCs to ensure higher resource utilization while reducing the end-to-end control-plane delays. Referring to Section 4.5, SCALE effectively considers two key parameters: (1) Current resource *Utilization* at each DC and (2) *Propagation delays* between the DCs. To focus on the importance of these factors, we setup a simulation with 4 DCs such that DCs 1, 3 are overloaded while DCs 2, 4 are lightly loaded. We plot the 99th %tile processing delays perceived by the devices belonging to all 4 DCs in Figure 10(b) for 4 different cases. (i) In the first case (IND), the devices are always processed by their respective local DCs, causing high delays for the two overloaded DCs. This represents most deployments today. In the next two cases, RDM1 and RDM2, each DC uniformly replicates a fixed percentage of its total device states across the other 3 DCs. (ii) *Current Load*: In RDM1, we configure the experiment such that the current load of DC2 is higher than that of DC4. As seen in Figure 10(b), although the delays of devices at DCs 1,3 were reduced compared to those with IND, the additional load from DCs 1, 3 on DC 2 causes much higher delays at DC2 while the delays at DC4 are lower. (iii) *Propagation Delays*: In RDM2, the experiment is configured such that the propagation delays from DCs 1,3 to DC 2 are higher than the delays from DCs 1,3 to DC 4. As seen in Figure 10(b), the delays of devices at DCs 1 and 3 are not significantly lower than those with IND. The additional propagation delay to DC 2 causes higher control plane delays for the devices belonging to DCs 1, 3. (iv) By considering both the current utilization and propagation delays, SCALE achieves better load balancing across DCs and causes lower delays for all the devices as shown in Figure 10(b).

S3: Access-Awareness: This experiment shows the flexibility of VM provisioning with SCALE. Specifically, SCALE contains a parameter β to control the number of VMs in scenarios when the state requirement is large due to high number of registered devices (Refer to Equation 1 in Section 4.5). We set x=0.2, i.e., SCALE maintains only a single replica for all devices that have access frequency below 0.2 ($w_i \leq x$). We repeat the experiment by varying the number of low probability devices, while keeping the number of total devices constant at 100K. This setup gives us a sense of VM savings that SCALE can achieve depending on the fraction of devices that have low frequencies of access, a trend that is

visible with the advent of M2M or IOT. As shown in Figure 11(a), β=1 represents the case where the VMs are provisioned to store 2 replicas of the state of all devices. As the number of low probability connections increases, the β value decreases and the VMs provisioned by SCALE decrease as seen in the figure. Figure 11(b) depicts the delays perceived in different scenarios. Even in the case with β=0.75, where almost 50% of device states are not replicated more than once, SCALE reduces the number of VMs by 25% without affecting the average delays significantly.

6. RELATED WORK

Distributed Systems: Distributed key-value store systems [9, 22, 25] face requirements similar to SCALE in their respective domains. Although they effectively manage data at scale, there are few key distinctions: (i) they do not consider the compute provisioning since processing requirements are not intensive. (ii) such systems are over-provisioned to meet the stringent SLA requirements. Low latencies leads to direct revenue, hence over-provisioning is sustainable. (iii) data is accessed simultaneously by multiple users or applications. Hence, it benefits to replicate high-access keys in order of magnitude more than regular keys (upto 10) [26]. Hence, although SCALE does borrow techniques from such systems, the key is that SCALE had to additionally solve challenges involving both the MME protocols and compute workloads.
NFV Trends: Recently, NFV has been a major focus of the telecom industry leading them to transform their network functions to execute as software over generalized intel-based hardware. In the mobile context, it has led to the emergence of commercial virtual EPC platforms [27, 28]. However, the primary focus has been on software improvements, such as hypervisor enhancements to improve the performance of the functions running as VMs, while using simple, reactive mechanisms for elasticity. SCALE is complementary to these approaches since it adopts a system-wide approach to design MME implementations with better elasticity and scalability.
EPC research: Recent works [29, 30] have proposed new architectures and protocols for an efficient EPC design. Although effective, such approaches are costly to deploy and they ignore many aspects of device management prevalent in current networks. Nonetheless, SCALE is a complementary solution that can be leveraged to virtualize such architectures if realized in future. The dMME system [31] proposes a split MME design with distributed processing nodes accessing a centralized storage. The dMME design is an alternate design choice to SCALE. However, the dMME systems has not been implemented and evaluated with reasonable loads. It is a good avenue for future work to compare the two design choices in the scope of virtual MMEs.

7. CONCLUSION

SCALE is a virtualized MME framework designed to meet the future requirements of mobile access. SCALE introduces a decoupled MME architecture to enable elasticity. To meet the goal of performance in terms of lowering signaling delays, SCALE adapts the consistent hashing scheme with analytically driven replication strategy to achieve efficient load balancing. To reduce VM footprint, SCALE performs device-aware replication and leverages geo-multiplexing. With a prototype using a LTE testbed and large-scale simulations, we show the feasibility and efficacy of SCALE.
Acknowledgments: We thank the reviewers and and Ajay Mahimkar, our shepherd, for the insightful comments that improved the paper significantly.

8. REFERENCES

[1] Qian et al. Periodic transfers in mobile applications: origin, impact, and optimization. In *ACM WWW*, 2012.
[2] Forecast: The Internet of Things. http://www.gartner.com/newsroom/id/2636073.
[3] Signaling is growing 50% faster than data traffic. http://goo.gl/89RtYs.
[4] Managing LTE Core Network Signaling Traffic. https://goo.gl/2nxq0X.
[5] Angry Birds Ruffle Signaling Feathers. http://goo.gl/Xqj6ti.
[6] Charting the Signaling Storms. http://tinyurl.com/k7qqeeb.
[7] AT&T Domain 2.0 Vision White Paper, 2013. http://tinyurl.com/p4uv3s3.
[8] Amazon Elastic Cloud. http://aws.amazon.com/ec2/.
[9] G. DeCandia et. al. Amazon DynamoDB: A Seamlessly Scalable Database Service. In *ACM SIGMOD*, 2012.
[10] Why distribution is important in NFV? http://tinyurl.com/kyewvqq.
[11] Karger et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. STOC '97. ACM.
[12] OpenEPC. http://www.openepc.com/.
[13] GPRS enhancements for E-UTRAN access. http://www.3gpp.org/DynaReport/23401.htm.
[14] MME Functions. http://goo.gl/1BuKpB.
[15] Path to 5G. http://tinyurl.com/kgqe76t.
[16] Nguyen et al. Towards Understanding TCP Performance on LTE/EPC Mobile Networks. In *All Things Cellular: Operations, Applications, & Challenges*. ACM, 2014.
[17] Huang et al. Close Examination of Performance and Power Characteristics of 4G Networks. In *MobiSys*. ACM, 2012.
[18] Zeljko Savic. Lte design and deployment strategies. http://tinyurl.com/lj2erpg.
[19] Shafiq et al. A first look at M2M traffic: Measurement and Characterization. In *ACM SIGMETRICS*, 2012.
[20] RAN Congestion Analytics. http://tinyurl.com/nlhqg96.
[21] VidScale Mediwarp RAN. http://www.vidscale.com.
[22] Lakshman et al. Cassandra: A Decentralized Structured Storage System. *ACM SIGOP*, 2010.
[23] Remote Direct Memory Access (RDMA) over IP Problem Statement. http://www.ietf.org/rfc/rfc4297.txt/.
[24] Netem. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.
[25] R. Nishtala. Scaling Memcache at Facebook. In *NSDI 2013*.
[26] Zhang et al. Load balancing of heterogeneous workloads in memcached clusters. In *Feedback Computing*, 2014.
[27] NEC Virtual EPC. http://tinyurl.com/nuefq28.
[28] The Journey to Packet Core Virtualization. http://resources.alcatel-lucent.com/asset/174234.
[29] Jin et al. Softcell: Scalable and Flexible Cellular Core Network Architecture. In *ACM CoNext*, 2013.
[30] Moradi et al. SoftMoW: Recursive and Reconfigurable Cellular WAN Architecture. In *ACM CoNext*, 2014.

[31] An et al. DMME: A Distributed LTE Mobility Management Entity. *Bell Labs TR*, 2012.

APPENDIX

Model: Let the number of replications of a device's state be R, with N being the limit on the number of devices a VM can process, and T being the size of the epoch over which decisions are made. With Poisson processes serving as a representative tool for modeling web requests, call arrivals, etc., we assume that the devices arriving at VM j follow a Poisson process with incoming rate λ (Varied arrival rates across VMs can be easily incorporated).

For the sake of analysis, we assume the following simple strategy for serving an incoming device: when a device arrives it is randomly assigned to one of the R VMs that containts its state - in other words, it is equally likely for the device to be assigned to one of the R VMs. Although simplistic, such a strategy is helpful to gain useful insights and hence guide the design of the replication process. Under this strategy, it can be seen that each stream of incoming devices at a VM j gets split into R sub-streams, which are sent to one of the R VMs, where the state of the devices in VM j are replication (based on consistent hashing). Each of these sub-streams are also Poisson (Poisson splitting), but with rate $\frac{\lambda}{R}$. Thus, from the perspective of each VM j, it gets a combination of R sub-streams (from R VMs including itself), each being a Poisson process with rate $\frac{\lambda}{R}$, resulting in an aggregate process that is also Poisson (Poisson combining) with rate λ.

Let us assume that a device incurs a cost C only when it cannot be served/processed (0 otherwise). Further, let the probability of a device arriving in an epoch be w_i.

A1.Impact of Replication Frequency: We represent the counting process associated with the Poisson arrival at VM j as $N_j(t)$ indicating the number of devices that have arrived until time t. Now, we characterize the probility that a device i cannot be served (represented by $\notin_S$) in a VM j (V_j).

$$\begin{aligned}
&\mathbb{P}(i \notin_S V_j \text{ at } t)\\
&= \mathbb{P}\left(\{i \in_A (t,T]\} \cap \{i \notin_A (0,t]\} \cap \{N_j(t) \geq N\}\right)\\
&= \sum_{k=N}^{\infty} \mathbb{P}\left(\{i \in_A (t,T]\} \cap \{i \notin_A (0,t]\} \cap \{N_j(t) = k\}\right)\\
&= \sum_{k=N}^{\infty} \mathbb{P}\left(\{i \in_A (t,T]\} | \{i \notin_A (0,t] \cap N_j(t) = k\}\right)\\
&\quad \cdot \mathbb{P}\left(\{i \notin_A (0,t]\} \cap \{N_j(t) = k\}\right)\\
&= \sum_{k=N}^{\infty} \mathbb{P}\left(\{i \in_A (t,T]\} | \{i \notin_A (0,t] \cap N_j(t) = k\}\right)\\
&\quad \cdot \mathbb{P}\left(\{i \notin_A (0,t]\} | \{N_j(t) = k\}\right) \cdot \mathbb{P}\left(N_j(t) = k\right) \qquad (4)
\end{aligned}$$

where $\in_A$ ($\notin_A$) indicates if a device arrives (otherwise). Given the arrival process being Poisson, $\mathbb{P}(i \in_A (0,t])$ depends on the number of devices that have arrived until t. If $N_j(t) = k$, then we have,

$$\mathbb{P}(i \notin_A (0,t] \mid N_j(t) = k) = \left(1 - \frac{w_i}{\lambda T}\right)^k$$

$$\text{Further, } \mathbb{P}(N_j(t) = k) = \frac{(\lambda t)^k e^{-\lambda t}}{k!}$$

$$\text{and } \mathbb{P}\left(\{i \in_A (t,T]\} | \{i \in_A (t,T] \cap N_j(t) = k\}\right) = \left\{1 - e^{-\lambda(T-t)}\right\} \cdot w_i$$

Applying back into Equation 4, we have

$$\mathbb{P}(i \notin_S V_j \text{ at } t) = \left\{1 - e^{-\lambda(T-t)}\right\} \cdot w_i \cdot \sum_{k=N}^{\infty} \frac{(\lambda t)^k e^{-\lambda t}}{k!} \cdot \left(1 - \frac{w_i}{\lambda T}\right)^k \qquad (5)$$

The probablity that device i cannot be served at any of the R VMs where its state resides can be obtained as,

$$\mathbb{P}(i \notin_S \text{ at } t) = \left(\mathbb{P}(i \notin_S V_j \text{ at } t)\right)^R \qquad (6)$$

Now, the expected cost incured by device i reduces to,

$$\bar{C}_i = C \int_0^T \left(\mathbb{P}(i \notin_S V_j \text{ at } t)\right)^R \, dt \qquad (7)$$

Applying Equation 5 in 6 and simplifying, we obtain the following closed-form expression for the expected cost of a device for large T as,

$$\bar{C}_i = \left(\frac{C}{\lambda}\right) w_i^R \sum_{k=N}^{\infty} \left(1 - \frac{w_i}{\lambda T}\right)^{kR} \left(\frac{\Gamma(kR+1)}{(\Gamma(k+1))^R R^{kR+1}}\right) \qquad (8)$$

where $\Gamma(n) = (n-1)!$ is the Gamma function. For large values of k, the numerator and denominator of the above equation would be hard to compute. Hence, we provide the following simplification for easier computaiton, wherein,

$$\left(\frac{\Gamma(kR+1)}{(\Gamma(k+1))^R R^{kR+1}}\right) = \left(\frac{1}{R}\right) \Pi_{p=0}^{k-1} \Pi_{q=0}^{R-1} \left(1 - \frac{q}{(k-p)R}\right) \qquad (9)$$

Now, the average cost (delay) experienced by a device in an epoch is given by,

$$\bar{C} = \frac{\sum_i w_i \bar{C}_i}{\sum_i w_i} \qquad (10)$$

Using the above, the impact of replication frequency (R) on the average delay experienced by a device can be easily studied.

A2.Impact of Access-awareness: Here, we aim to understand the importance of replicating the state of a device in proportion to its access probability w_i. This is especially useful, when the total state in the VMs does not allow all the devices to be replicated R times. Hence, it becomes important to understand which set of devices should be replicated more relative to others.

Let S be the total state capacity at each VM, with K be the total state of all devices that need to be stored. Let V be the number of VMs in the datacenter. Further, let S' be the actual state capacity that is available for devices in local DC after accounting for new device states (S_n) and external state of devices from remote DCs (S_m), i..e $S' = S - \frac{S_n + S_m}{V}$. To replicate each device's state R times ($R = 2$ in `SCALE`), we need a total state capacity of RK. We need $VS' \geq K$ to allow for each device's state to be stored at least once. If $VS' < RK$, then not all devices can be replicated R times. In this case, each device's state can be replicated $R' = \left\lfloor \frac{VS'}{K} \right\rfloor$ times ($R' = 1$ in `SCALE`). With the remaining $\left(\frac{VS'}{K} - \left\lfloor \frac{VS'}{K} \right\rfloor\right) K$ state capacity not being sufficient to accommodate $(R - R')$ replications of all the devices, it becomes important to decide which of the devices will get an additional replication.

Here, we consider two strategies: (i) access-unaware: each of the device has an equal probability of

$$\mathbb{P}_i(\text{rep}) = \frac{VS'}{K} - \left\lfloor \frac{VS'}{K} \right\rfloor, \ \forall i \qquad (11)$$

to get an additional replication; and (ii) access-aware: the probability of replication is proportional to the device's access probability:

$$\mathbb{P}_i(\text{rep}) = \min\left\{1, \left(\frac{w_i}{\sum_i w_i}\right)\left(\frac{VS'}{K} - \left\lfloor \frac{VS'}{K} \right\rfloor\right) K\right\} \qquad (12)$$

Now, let us represent the average cost of a device in Equation 8 as a function of R as $\bar{C}_i(R)$. Then, incase R replicas are not possible for each device, the average cost of a device gets modified as,

$$\bar{C}_i = (1 - \mathbb{P}_i(\text{rep}))\bar{C}_i(R') + \mathbb{P}_i(\text{rep})\bar{C}_i(R'+1) \qquad (13)$$

where, $\mathbb{P}_i(\text{rep})$ will depend on the strategy used for replicating devices in the space remaining after R' replications of each device (i.e. Equations 11,12). With the above analysis, we can now easily study the impact of access-aware replication on the average delay experienced by a device in a datacenter.