A Cloud Native Solution for Dynamic Auto Scaling of MME in LTE

Amogh PC, Goutham Veeramachaneni, Anil Kumar Rangisetti,
Bheemarjuna Reddy Tamma, and Antony Franklin A
Department of Computer Science and Engineering, Indian Institute of Technology Hyderabad, India
Email:[cs15mtech01002, cs14btech11014, cs12p1001, tbr, antony.franklin]@iith.ac.in

***Abstract*—Due to rapid growth in the use of mobile devices and as a vital carrier of IoT traffic, mobile networks need to undergo infrastructure wide revisions to meet explosive traffic demand. In addition to data traffic, there has been a significant rise in the control signaling overhead due to dense deployment of small cells and IoT devices. Adoption of technologies like cloud computing, Software Defined Networking (SDN) and Network Functions Virtualization (NFV) is impressively successful in mitigating the existing challenges and driving the path towards 5G evolution. However, issues pertaining to scalability, ease of use, service resiliency, and high availability need considerable study for successful roll out of production grade 5G solutions in cloud. In this work, we propose a scalable Cloud Native Solution for Mobility Management Entity (CNS-MME) of mobile core in a production data center based on micro service architecture. The micro services are lightweight MME functionalities, in contrast to monolithic MME in Long Term Evolution (LTE). The proposed architecture is highly available and supports auto-scaling to dynamically scale-up and scale-down required micro services for load balancing. The performance of proposed CNS-MME architecture is evaluated against monolithic MME in terms of scalability, auto scaling of the service, resource utilization of MME, and efficient load balancing features. We observed that, compared to monolithic MME architecture, CNS-MME provides 7% higher MME throughput and also reduces the processing resource consumption by 26%.**

I. Introduction

The increase in traffic diversity and accelerated capacity demand in mobile networks have pushed design of innovative architectural solutions and cost-effective paradigms for 5G evolution. 5G is expected to offer extremely high data rates, high spectral efficiency, improved coverage, low latency, and enhanced signaling efficiency. Apart from data traffic intensification, the signaling overhead due to a large number of mobile devices and different types of IoT devices is expected to overwhelm the LTE control plane nodes (MME) leading to network outages and performance penalties.

In order to meet this traffic explosion and growing signaling overhead at MME, technologies like cloud computing, Software Defined Networking (SDN) [1], [2], and Network Functions Virtualization (NFV) offer competitive architectural solutions and ensure cost-effectiveness. Architecting the network functions as a deliverable service package on the cloud paradigm is commonly referred to as virtualized network functions or VNFs. Through VNFs, a wide range of benefits including operational simplification, reduction in Operational expenditure (OPEX) and Capital expenditure (CAPEX), and efficient life cycle management can be realized by mobile operators. However, the generic, general purpose hardware infrastructure in data center environments poses many research challenges as it is prone to failures and may not act in accordance with desired QoS requirements of 5G networks. The flexibility of cloud computing paradigm in offering VNFs must accommodate additional considerations to improve service resilience by incorporating auto scaling of services, high availability, and load balancing.

Cloud native solutions build on top of cloud computing service model where the applications fully leverage advantages in service deployment and orchestration [3], [4]. Also, it implies horizontal scalability, no single points of failure, survivability, automatable operations, and no platform-specific encumbrances. Cloud Native Computing Foundation (CNCF) [5] highlights three major properties of cloud-native applications:

1) *Containerized*: Each part (applications, processes, *etc.*) is packaged in its own container. This facilitates reproducibility, transparency, and resource isolation.
2) *Dynamically Orchestrated*: Containers are actively scheduled and managed to optimize resource utilization.
3) *Microservices oriented*: Applications are segmented into microservices. This significantly increases the overall agility and maintainability of applications.

This model works very well in production environments and excels at leveraging the cloud if the application is architected correctly.

II. Motivation and Related Work

Next generation wireless networks could face an explosion of control signaling overhead due to IoT applications, vehicular networks, and dense deployment of small cells. Specifically in IoT applications compared to data traffic, control signaling overhead is much higher because it is necessary to handle explosion of device attach/registration and detach messages frequently. Besides, to conserve energy IoT devices frequently go to idle or sleep state which could lead to large number of location updates. Hence, operators are desperately looking for scalable solutions for deploying LTE and 5G networks.

MME is the major control plane processing entity in the LTE architecture. MME is a heavy processing entity and is responsible for handling various control signaling functions like attach, detach, authentication, authorization, UE context management, bearers management for handling QoS of UEs traffic, handovers, location update, billing, and logging of call data. It uses well-defined interfaces for communicating with various entities (eNodeB, HSS, and S-GW) to handle various control tasks. We can classify the MME state into static details and dynamic details. Static state details include timers, security keys, globally unique identifiers, temporary mobile subscriber identities, and profiles of UEs for QoS and policy enforcement. Dynamic state details include bearer details, location, call records, billing, and S-GW and P-GW addresses. Because the MME needs to maintain various state information related to UEs for handling aforementioned tasks, auto scaling and seamless resilient implementations are complex. For example, the MME holds the context and state of every UE connected to it. If MME crashes, it will disrupt every UE connected to it. One way to mitigate it would be to run two or more instances of MME which would be used if one goes down. However, as the MME has information about the UE contexts, we cannot simply swap it with another instance of MME.

This stateful, monolithic MME architecture also causes issues while scaling, where sometimes the only option is vertical-scaling (adding more processing resources like CPU, RAM, *etc*.) as horizontal-scaling (adding more VNF instances) would require significant operational overhead due to complex state management. The frequency and commonality of network partitions and other failures in the cloud make it unacceptable to run the monolithic EPC in the cloud. We need a new architecture that enables the EPC to be highly available so that it could be hosted in the cloud.

Cloud Native Applications (CNAs) are usually stateless micro services which use a centralized highly-available datastore to gain context and state *i.e.,* in the case of MME, all the UEs context will be stored inside a datastore and whenever a UE interacts with the MME its context is accessed from the datastore. Besides, multiple instances of MME can access the shared datastore and serve the control signal requests in a simple round-robin way. Here, scaling would mean just adding more MME instances and the system is tolerant to failures of individual MMEs.

There are few existing scalable solutions for handling control signaling overhead in LTE-EPC architecture using NFV and distributed data center platforms. In [6], the authors proposed a Distributed MME (DMME) architecture. The main objective of DMME architecture is to split the task of processing control plane events among a large number of servers that manage the UE mobility state as independently as possible. In DMME architecture, the processing of UE control plane messages is done in DMME nodes and UE's state is managed using reliable object storage (ROS) subsystem. But, as the MME is a monolithic process it cannot be auto scaled seamlessly. In [7], the authors proposed a scalable implementation of MME (SCALE) with two components: MME Load Balancer (MLB) and State Management. However, this architecture mainly suffers from flexible auto scaling in terms of handling monolithic MME implementation and complex state management. In [8], the authors proposed container-driven fine-grained resource flexing. They benchmarked the performance of the MME, Suricata, and Snort VNFs on bare metal, containers, and VMs under varying workloads.

Our proposed work is different from above works in terms of design of scalable MME and usage of CNA platform for providing auto scaling, resilient features, and provisioning of processing and storage resources. In our proposed scalable MME architecture, state of the MME is separated from its processing tasks. Besides, to design lightweight MME functions, scalable VNFs are proposed for various MME tasks as micro services. In following section, we give more details of CNA systems Kubernetes [9], Docker, and Prometheus that are used to deploy and manage micro services with minimal OPEX.

We present four main contributions in developing a scalable Cloud Native Solution for LTE MME (CNS-MME) which are summarized as follows:

- We propose a CNS-MME for LTE based on micro service architecture aligned with ETSI-NFV reference architecture [10]. We extended NFV-LTE-EPC framework [11], [12] for designing our proposed architecture. The implementation utilizes an open-source orchestrator, Kubernetes along with Docker container platform for virtualizing network functions. Prometheus is used as a monitoring tool to capture various statistics of the network.
- We implement a stateless network function for MME with clear separation of MME states into a centralized data store.
- We evaluate our CNS-MME microservice based implementation with a monolithic MME in terms of achievable MME throughput, auto scaling capability, load balancing, and computational resource usage.
- We build an L7-loadbalancer to enable CNS-MME architecture and contrast it to a traditional L4-loadbalancer implemented in monolithic MME.

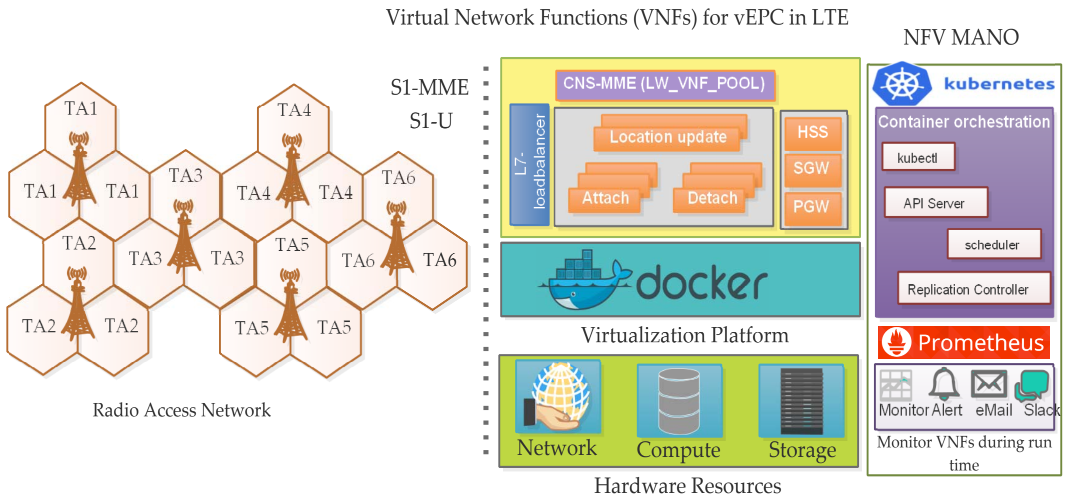

Fig. 1: Cloud Native NFV Architecture for CNS-MME

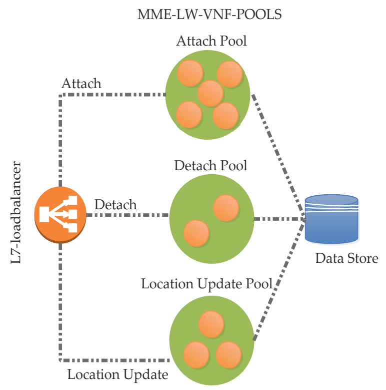

Fig. 2: L7-loadbalancer in CNS-MME

III. Cloud Native Scalable MME Architecture

A. System Architecture

The basic principle of the proposed architecture, CNS-MME, is to build auto scalable lightweight MME micro services with simplified state management. CNS-MME architecture shown in Fig. 1 is designed to handle a RAN which spans a geographical area which is divided logically into a large number of Tracking Areas (TAs). Besides, in CNS-MME architecture, S-GW/P-GW, and HSS are also deployed as VNFs. To handle huge control signaling overhead from the RAN with a large number of TAs, we build a scalable MME as a pool of attach, detach and location update lightweight VNFs. In order to simplify VNF deployment, auto scaling, and monitoring for resilient operations, we employed cloud native frameworks called Kubernetes and Prometheus for realizing NFV MANO (NFV management and orchestration) platform. To loadbalance between different VNFs, we also built a loadbalancer which inspects the packet type and sends it to the respective VNF pool (attach, detach, location update).

L4-loadbalancer vs L7-Loadbalancer: In existing NFV-LTE-EPC [11], [12] and SCALE [7] implementations, which are based on monolithic MME architecture, L4-loadbalancer is used. L4-loadbalancer forwards packets based on IP tables to MME pool. But L4-loadbalancer cannot be used for classification of packets and forwarding to the respective entity in the VNF pool. Hence, in contrast to existing works, to implement CNS-MME with MME-LW-VNF-POOL, we developed an L7-loadbalancer (refer Fig. 2), which classifies MME control signaling and forwards it to the respective entity in the VNF pool. The L7-loadbalancer also needs to distribute the control signal overhead to respective MME lightweight VNF pool (MME-LW-VNF-POOL).

As MME-LW-VNF-POOL needs to maintain and update the states of a large number of UEs, the state management and updates are done using storage component of the platform. In this architecture, state management can be simplified by carefully handling the static and dynamic states of UEs.

B. MME-LW-VNF-POOL

In typical MME, there are many functions such as connection management, mobility management, bearers management, and updating UEs static and dynamic state information. In a monolithic MME instance, auto scaling requires provisioning of high processing and memory resources. In order to load balance across MMEs, complex state management is involved to consistently distribute state across MMEs. For example in IoT applications, there will be a lot of control signaling related to attach, detach and location updates compared to bearers management and handovers. Hence, these applications do not require complete monolithic MME scaling as it leads to unnecessary higher processing and memory resource provisioning for virtual machines which host the monolithic MME. In contrast to monolithic MME pools, in proposed MME-LW-VNF-POOL only necessary VNFs (e.g., attach or location updates) can be scaled out. This is also known as horizontal scaling of micro services. But in case of monolithic MME, horizontal scaling is inflexible and costly.

To enable horizontal scaling of MME-LW-VNF-POOL, MME functions are divided into VNF pools with each pool implementing a specific functionality: attach, detach and location update. An attach procedure consists of: authentication, Evolved Packet System (EPS) session creation and default bearer creation. A location update procedure consists of Tracking Area Update (TAU) and authentication. A detach procedure consists of bearer deletion and UEs state/context deletion. Moreover, to ensure interoperability with other systems the proposed L7-loadbalancer in-front would be acting as the interface between UEs and MME and transparently routes each message to the right VNF pool. For example: all attach requests will be forwarded to the attach VNF-pool only. Further, if one of the downstream VNFs is down or in an inconsistent state, the request is retried on another VNF from the same pool before propagating the error to the UE. This adds additional resiliency to crashed VNFs.

C. Advantages

1) Stateless and Scalable VNFs: In proposed architecture, the VNF state is separated and managed in an external datastore. At the beginning of each request from a UE, VNFs restore the UE context from the datastore and update the context at the end of the request. To reduce the number of control signaling requests hitting the datastore, we use the same VNF instance for each procedure, for example, all control signaling in a single attach procedure will be served by the same VNF instance. This is realized by using a single connection to the MME (L7-loadbalancer) for each of the procedures. This makes the L7-loadbalancer highly-stateless and scalable as the only state being stored is the mapping of each connection to a VNF and this state exists only as long as the connection exists. This means we can horizontally scale the loadbalancer with no disruption. The VNF, on the otherhand, will make only two calls to the datastore for each UE it serves, one for getting the UE context, and other for updating it at the end. This significantly reduces the overhead of the external datastore and makes it viable to be used in the time-critical LTE stack. Since all the VNFs are stateless, we can add VNFs at will and delete them when they are not serving any requests. For delete, we take the VNF instance out of the loadbalancer pool so that it will receive no new requests and once it finishes any pending requests, we delete the instance. This allows for seamless scale-up and scale-down of the VNF pools with no disruptions. It should further be noted that because we leverage VNFs and containers, we can scale each of the attach, detach and location update pools based on their respective loads irrespective of the MME as a whole. While this might seem ideal, there is significant operational cost to manage such a highly-distributed infrastructure.

2) Simplified Deployment of VNFs: We mitigate OPEX by adopting the ETSI NFV architecture, where we leverage containers and Kubernetes. Kubernetes is an open-source container orchestrator that has gained a lot of traction and is being used in production by several companies. Kuberenetes, infact, has been built to handle the dynamic and distributed workloads of the proposed architecture. It also fits right into the ETSI NFV architecture by acting as the MANO with Docker as the virtualization platform. The proposed architecture when containerized and deployed using Kubernetes, will have significantly less OPEX with Kubernetes managing the deployments and states. We also instrumented the VNFs and L7-loadbalancer to export metrics to Prometheus, an open-source monitoring system. We use these metrics to alert on anomalies or issues that arise. Further, we can also link Kubernetes and Prometheus to implement auto scaling based on the metrics exported by the application.

IV. Cloud Native Solutions for Simplified Orchestration of VNFs

Some of the main issues around deploying VNFs in production environments are the issues around operating and managing them. By leveraging Kubernetes and containers, we bring production ready operations to VNFs. The proposed architecture could get unprecedented ease in doing the operations that follow. In the following, we discuss different operational scenarios seen in the real-world and look at how Kubernetes helps in those.

A. Deployments

When we want to deploy a particular VNF, we can either do it via the Command Line Interface (CLI) or the provided dashboard UI (User Interface). Everything deployed by Kubernetes is a container and we will first need to package the VNF as a container image. When deploying, we can limit the CPU and memory that the VNF can utilize and that combined with containers and their isolation mechanisms, provide effective boundaries for most VNF requirements. For VNFs that require higher security and isolation guarantees, Kubernetes also supports a Hypervisor backed container runtime which provides VM level isolation. While deploying we can also specify the number of instances in a pool for redundancy and fault-tolerance and Kubernetes makes sure that any crashed instances are restarted and distributes the VNFs onto the different nodes available.

B. Auto Scaling

The promise of VNFs is resiliency and effective resource utilization. Key to that is replicated VNFs where the number of VNFs is increased and decreased dynamically with no human intervention. Kubernetes enables this via its Horizontal Pod Autoscaling (HPA). It allows to scale on any parameters including CPU/Memory utilization, Requests per Second, and percentile latency. We both scale-up and scale-down based on the parameters and it is a regression based algorithm which means sudden spikes in usage do not cause heavy scale-up activity. Scaling happens with instances evenly distributed across nodes and any failed nodes/instances are immediately brought back. All of this is automatic and involves no manual input and has been proven in production environments with large cluster sizes.

Scaling model for CNS-MME: In CNS-MME, we used the Autoscaling algorithm of Kubernetes. We can scale on any parameters like CPU, requests per second and 90th percentile latency. We demonstrate the scaling on CPU utilization. Kubernetes periodically evaluates the target CPU utilization and average CPU utilization of VNFs (containers), and scaling will be done as follows: To avoid rapid scale-up and scale-down of VNFs due to fluctuating load, scale-up happens only 3 minutes after the last scaling event and scale-down only 5 minutes after the last scaling event. This difference is because scaling down is a non-critical activity and can be slower.

C. Monitoring and Repair

After deployment, Kubernetes constantly monitors each VNF instance and node, and if there is any failed VNF instance or node, it is either retried (restarted) or replaced. This combined with using a pool of VNFs offers low maintenance cost. Further, because the pool is distributed across nodes, we can also tolerate node failures. Any instances on a failed-node will be restarted on the other nodes. If the existing nodes do not satisfy the resource requirements, we can use cluster auto scaling to request new nodes from the infrastructure provider.

D. Updates and Rollbacks

Another key operation in CNS-MME is the application updates: How to deploy a new version without any disruption? This is enabled again by Kubernetes rolling updates. When a new version is deployed, it brings up some instances on the new version and once they are active, it scales down the corresponding number of old instances. The scaling down is again seamless and the instances are not removed until they finish processing any pending requests. It also handles roll-over updates (issuing new updates while a previous update is in progress) by scaling up the newest versions and scaling down any old versions that are running. Further, in case of a bad update we can automatically roll-back to any of the previous versions. This enables the operator to deploy updates with more confidence.

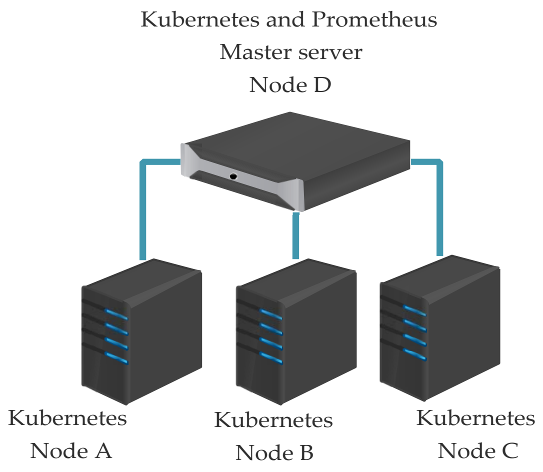

Fig. 3: Topology Setup for CNS-MME testbed

V. Experimental Setup and Performance Results

The experimental setup is made by extending NFV-LTE-EPC [12], [11] simulator. In [12], [11], the authors developed and evaluated SDN based LTE-EPC implementation and NFV based LTE-EPC implementation. In this work, we used NFV-LTE-EPC architecture and thoroughly extended for ETSI-NFV aligned CNS-MME implementation. We leverage Docker, Kubernetes and Prometheus to build ETSI aligned NFV platform. Our experimental setup runs on virtualized platform Docker and is provisioned on top of four commodity hardware servers having resources as given in Table I.

TABLE I: Configuration of testbed

Entity	# Core	RAM	OS
[A,B,C] Node	40	64GB	Ubuntu 16.10, 64 bit
[D] Node+Master	56	64GB	Ubuntu 16.10, 64 bit

TABLE II: Simulation Parameters

Parameter	Value
Number of UEs	0 to 300
Simulation time	120 Minutes
Virtualization platform	Docker
NFV orchestrator	Kubernetes v1.6.3
Live status monitor	Prometheus 1.6.2

In order to perform simulations, we have deployed ETSI based CNS-MME on distributed nodes as shown in Fig. 3 and conducted two sets of experiments to benchmark performance benefits. They are summarized as follows.

- Scenario-1: Evaluation of overhead in L7-loadbalancer vs L4-loadbalancer
- Scenario-2: Evaluation of scalable monolithic MME POOL vs scalable MME-LW-VNF-POOL

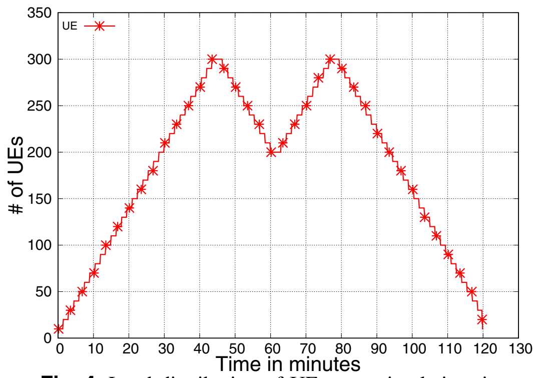

Fig. 4: Load distribution of UEs over simulation time

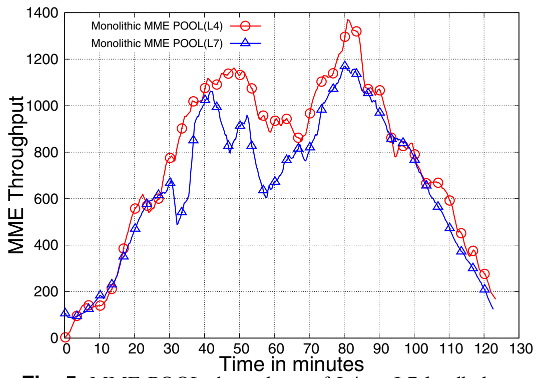

Fig. 5: MME-POOL throughput of L4 vs L7-loadbalancer

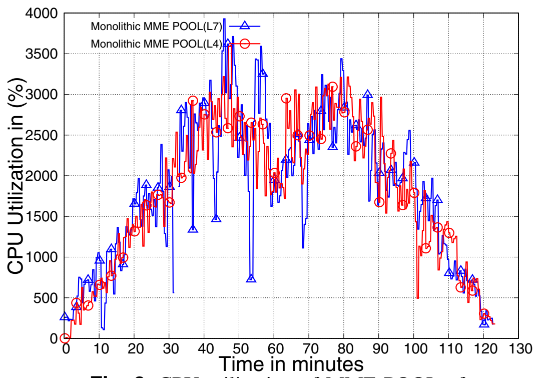

Fig. 6: CPU utilization of MME-POOL of L4 vs L7-loadbalancer

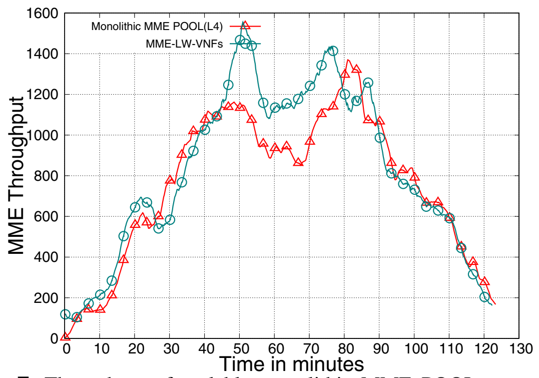

Fig. 7: Throughput of scalable monolithic MME POOL vs scalable MME-LW-VNF-POOL

In order to evaluate the above test scenarios, the simulation parameters given in Table II are considered. Using RAN simulator of NFV-LTE-EPC [11], continuous control signaling traffic is generated to the EPC. We simulate concurrent UE threads on the RAN simulator and make the UEs continuously perform attach, detach, and location update from the LTE network. We measure the performance of test scenarios using following three key performance metrics:

1) MME Throughput: Number of control signals processed per second by MME
2) # of VNF instances: Number of VNF instances of MME/LW-VNFs that are required to handle the control signals
3) CPU utilization: The total CPU utilization by a particular VNF-POOL.

A. Scenario-1: Evaluation of overhead in L7-loadbalancer vs L4-loadbalancer

Aim of this test scenario is to evaluate the overhead of L7-loadbalancer compared to L4-loadbalancer, since L7-loadbalancer could affect throughput of CNS-MME architecture. This experiment is based upon stateless scalable monolithic MME pool having all functions of a MME in one VNF. It reckons the effect of an L7-loadbalancer over L4-loadbalancer. The simulation scenario shown in Fig. 4, starts with 10-UE threads and 10 more UE threads are added for every 90 secs. Load is varied until limit reaches to 300 UEs. Thereafter, number of UE threads are decreased to 200 to check auto scaling of VNFs. Again, number of UE threads are increased until 300. And again this load distribution process is repeated throughout the simulation time. Fig. 5 shows comparison of MME throughputs of L4-loadbalancer and L7-loadbalancer. It is observed that MME throughput for L7-loadbalancer is about 13.19% lower compared to L4-loadbalancer. L4-loadbalancer forwards packets based on IP tables to a MME pool, without inspecting the contents of the packets. Hence its processing overhead is lower. Where as L7-loadbalancer deals with the actual packet inspection and processing in order to forward it to the right LW-VNF-POOL. Hence, it leads to slower control signal processing and lesser MME throughput. Now, while the packet inspection is done by the L7-loadbalancer, in L4-loadbalancer, it is offset to the application. As a result, the CPU utilization is almost equal in both (refer Fig. 6) cases. The throughput gain is mainly due to using IP tables instead of using DNS lookups and cache. However, to run LW-VNF-POOLs L7-loadbalancer is indispensable.

B. Scenario-2: Evaluation of Scalable Monolithic MME POOL vs Scalable MME-LW-VNF-POOL

Aim of this test scenario is to show how stateless MME-LW-VNF-POOL can offset the processing overhead and uses less resources compared to monolithic MME pool. Similar to the test scenario-1, UEs load distribution process is repeated throughout the simulation

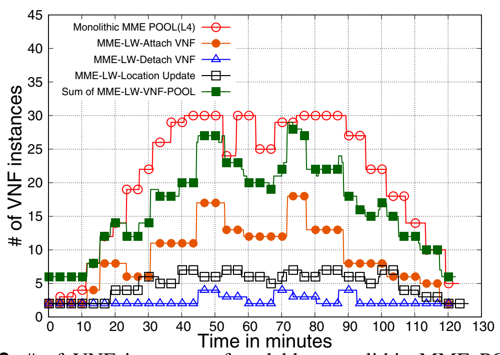

Fig. 8: # of VNF instances of scalable monolithic MME POOL vs MME-LW-VNF-POOL

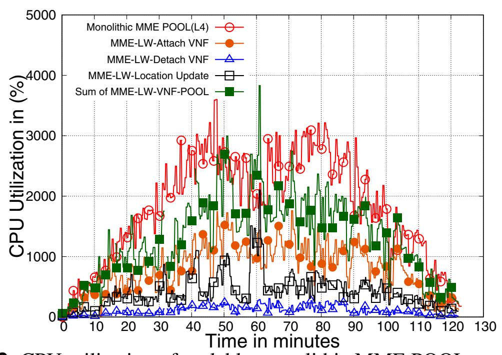

Fig. 9: CPU utilization of scalable monolithic MME POOL vs scalable MME-LW-VNF-POOL

time as shown in Fig. 4. Fig. 7 shows the comparison of MME throughputs of monolithic MME POOL and MME-LW-VNF-POOL. We can clearly see that VNF pool has a slightly higher throughput, about 7.19%, but it also uses 26.64% lower CPU (refer Fig. 9). Major reason for this higher throughput and lesser CPU utilization in MME-LW-VNFs is because LW-VNFs are handling only their specific control signals and hence they are consuming lesser processing resources compared to monolithic MME instances. Also, the load is lower on each VNF, and hence the response is faster, there by increasing the throughput. We can also observe that (refer Fig. 8) during entire simulation time total # of VNF-instances are 20.53% lesser compared to monolithic MME instances. This clearly shows that a micro-service based architecture can cause a significant increase in throughput and therefore offsets the drawbacks of using an L7-loadbalancer. From the results, we can conclude that CNS-MME architecture using MME-LW-VNF-POOL can provide a scalable MME solution using lesser resources consumption over existing monolithic MME architectures. For the detailed Prometheus monitoring results of our experimental setup, please refer [13].

VI. Conclusions and Future Work

In order to enable 5G deployments for next generation mobile networks, cloud native solutions can offer unprecedented scalability and resilience with minimal OPEX. In this work, we proposed a CNS-MME architecture to handle huge control signaling overhead in mobile networks. And the composability offered by this architecture where the operators can only deploy specific functions on demand as opposed to a monolithic solution, will enable new use-cases and solutions that are hard to achieve with a monolithic solution. Finally, we evaluated monolithic MME implementation and MME-LW-VNF-POOLs, and observed that our proposed solution provides 26.64% lesser processing resources besides 7.19% higher MME throughput.

As a part of future work, we would like to evaluate our CNS-MME solution using real-time LTE-EPC implementation with different scaling algorithms. Besides, we are also planning to propose a scalable and flexible VNF placement strategy using cloud native solutions.

Acknowledgment

This work is supported by the project ”Converged Cloud Communication Technologies”, MeitY, Govt. of India.

References

[1] A. Brunstrom, K.-J. Grinnemo, J. Taheri, *et al.*, “SDN/NFV-based mobile packet core network architectures: A survey,” *IEEE Communications Surveys & Tutorials*, 2017.
[2] C. Bouras, A. Kollia, and A. Papazois, “SDN & NFV in 5G: Advancements and challenges,” in *Proc. of Innovations in Clouds, Internet and Networks (ICIN)*, pp. 107–111, IEEE, 2017.
[3] N. Kratzke and R. Peinl, “ClouNS-a Cloud-Native Application Reference Model for Enterprise Architects,” in *Proc. of Enterprise Distributed Object Computing Workshop (EDOCW)*, pp. 1–10, IEEE, 2016.
[4] G. Toffetti, S. Brunner, M. Blöchlinger, J. Spillner, and T. M. Bohnert, “Self-managing cloud-native applications: Design, implementation, and experience,” *Future Generation Computer Systems*, vol. 72, pp. 165–179, 2017.
[5] “Cloud Native Computing Foundation (CNCF).” https://www.cncf.io/.
[6] X. An, F. Pianese, I. Widjaja, and U. Günay Acer, “DMME: A distributed LTE mobility management entity,” *Bell Labs Technical Journal*, vol. 17, no. 2, pp. 97–120, 2012.
[7] A. Banerjee, R. Mahindra, K. Sundaresan, S. Kasera, K. Van der Merwe, and S. Rangarajan, “Scaling the LTE control-plane for future mobile access,” in *Proc. of ACM Conference on Emerging Networking Experiments and Technologies*, p. 19, ACM, 2015.
[8] A. Sheoran, X. Bu, L. Cao, P. Sharma, and S. Fahmy, “An empirical case for container-driven fine-grained vnf resource flexing,” in *Proc. of IEEE NFV-SDN*, pp. 121–127, IEEE, 2016.
[9] “Kubernetes.” https://kubernetes.io/.
[10] “ETSINFV.” http://www.etsi.org/technologies-clusters/technologies/nfv.
[11] A. Jain, N. Sadagopan, S. K. Lohani, and M. Vutukuru, “A comparison of SDN and NFV for re-designing the LTE packet core,” in *Proc. of IEEE NFV-SDN*, pp. 74–80, IEEE, 2016.
[12] “NFV-LTE-EPC.” https://github.com/networkedsystemsIITB.
[13] “Prometheus monitoring results.” http://newslab.iith.ac.in/cnsmme/.