Design and Implementation of a Distributed Mobility Management Entity on OpenStack

Gopika Premsankar, Kimmo Ahokas and Sakari Luukkainen
Department of Computer Science
Aalto University, Finland
Email: {*gopika.premsankar, kimmo.ahokas, sakari.luukkainen*}*@aalto.fi*

***Abstract*—Network Functions Virtualization (NFV) consists of implementing network functions as software applications that can run on general-purpose servers. This paper discusses the application of NFV to the Mobility Management Entity (MME), a control plane entity in the Evolved Packet Core (EPC). With the convergence of cloud computing and mobile networks, conventional architectures of network elements need to be re-designed in order to fully harness benefits such as scalability and elasticity. To this end, we design and implement a distributed MME with the three-tier architecture common to web applications. We deploy the components of the distributed MME on two separate OpenStack clouds and evaluate the latency of the attach procedure. We find that the placement of the components within the data center significantly affects the attach latency.**

I. Introduction

The mobile core network today comprises proprietary hardware equipment that is designed to meet high performance requirements. Upgrading or expanding the network demands investment in expensive hardware and the deployment process is slow and cumbersome [1]. Network Functions Virtualization (NFV) employs standard IT virtualization technologies to enable mobile network operators to deploy network functions on a cloud computing infrastructure. Operators can then rapidly scale their networks to meet the growing demand for mobile data [2] and simultaneously reduce their expenditure in proprietary hardware. This paper focuses on the virtualization of the Mobility Management Entity (MME), a key control plane element in the Evolved Packet Core (EPC).

The MME is an ideal candidate for virtualization. It is purely a control plane element and, hence, it does not need any specialized packet forwarding hardware. Furthermore, in present-day networks, signaling traffic is growing rapidly due to continuous keep-alive messages generated by smartphones and emerging machine-to-machine applications [3]. An MME deployed on the cloud can employ virtually infinite computing resources to handle this load. Moving the MME software as-is to the cloud, however, does not allow the MME to fully leverage the benefits of cloud computing [4]. When standalone virtual MMEs (vMME) are deployed to handle increased network load, each newly created vMME has to be configured with EPC-specific parameters. Also, other network elements (such as eNodeBs) have to be informed about the existence of this new MME. Furthermore, once the vMMEs are serving subscribers, they maintain information about active subscribers in their local storage. In case a vMME is no longer required, for instance, due to reduced subscribers, the vMME cannot be simply shut down as active sessions will be affected.

Taking these issues into consideration, we design and implement a three-tier architecture for the vMME. This corresponds to the 1:N mapping defined in [4]. This architecture consists of three components: a *front end* (FE), one or more *workers* and a *state database*. The FE behaves as an intelligent proxy and maintains interfaces to other EPC elements. The workers are responsible for the actual functional processing in handling user mobility and sessions. User state information (referred to as UE context) is stored in the state database, thereby making the workers stateless. With this design, the workers and state database are transparent to external network entities. Thus, the workers can be easily scaled out or scaled in depending on network load without the need to inform other network elements. Furthermore, the vMME is resilient to the failure of workers as the UE context is saved on the state database. Any worker with access to this information can take over the processing in case of failures.

However, de-constructing the MME architecture results in increased latency for EPC procedures. All messages between external network elements and the workers incur an extra hop as they have to be forwarded through the FE. Storing the UE context in the state database requires additional messaging which also contributes to the increase in latency. This paper quantifies this increase in latency for the E-UTRAN initial attach procedure by comparing the distributed design to that of a standalone MME. We also investigate how placement of the distributed MME components within the data center affects the attach latency.

The rest of the paper is organized as follows. Section II discusses the related research literature. Section III details the functionality of the components of the distributed vMME. Section IV describes the testbed and presents the results of the experiments performed. Finally, Section V provides some concluding remarks.

II. Related Work

Virtualization of network elements in the EPC and IP Multimedia Subsystems (IMS) are active areas of research. Distributed architectures for the MME have been presented in [5], [6]. Our three-tier architecture for the MME is similar to the one presented in [5]. However, while the authors of [5] have focused on scaling operations, we rather evaluate the latency during attach procedures with a distributed vMME. Furthermore, we compare the performance of the distributed vMME to that of a standalone vMME [5]. The distributed MME in [6] consists of stateless message processors organized in a distributed hash table with an external user state storage system. In this case, migration to another MME is possible only when the UE is in the idle state. When the UE is in the active state and attached to one MME, if a network event for that UE reaches another MME instance, the request has to be forwarded to the correct MME. In our design, instead, these additional redirections are not required and the state migration can also occur while the user is active.

In addition to the MME, research literature describes similar distributed design for the IMS. An elastic core architecture is described in [7], which separates state processing for virtualized network functions from the state information stored in a database. Software-defined networking is used to flexibly allocate new resources for virtualized elements. The authors further analyze the application of this architecture to a virtualized IMS. Clearwater [8] is an open-source implementation of IMS designed to run on the cloud. In this design, long term state information is moved out of the processing nodes to an external data store. The authors of [9] discuss different software architectures for efficient virtualization of the IMS.

III. Design and Implementation of the vMME

We re-design the MME software developed in [10] to realize a 1:N mapping. Fig. 1 shows an overview of the system with the vMME's new architecture. The three tiers (i.e., FE, workers and state database) collectively represent a single MME to external elements such as the eNodeB, Serving Gateway (SGW) and PDN Gateway (PGW). Our system also includes an *OpenStack load balancer* which is responsible for creating and deleting worker VMs on OpenStack. We now describe the functionality of each tier.

A. Front End

The main functions of the front end (FE) are the following:

1) Maintain 3GPP standardized interfaces towards other EPC network elements. In our system, the FE only maintains an S1-MME interface towards eNodeBs and an S11 interface towards SGWs.
2) Balance requests between worker nodes. Our FE design employs a simple round-robin load balancing scheme to distribute new requests to worker nodes.
3) Inform the OpenStack load balancer when new workers are required to be created or deleted.

To realize the first two functions, the FE needs to correctly forward S1 Application Protocol (S1AP) messages between workers and eNodeBs, and GPRS Tunneling Protocol (GTP) messages between workers and SGWs. The FE maintains a mapping for each UE, which associates the UE to the worker, eNodeB and SGW responsible for handling the user session. The FE determines the UE identity based on information elements present in the message to be forwarded. To realize the third function, the FE maintains a long-lived TCP connection to the HTTP server of the OpenStack load balancer. When the number of incoming attach requests per worker goes above or below a certain threshold, the FE sends an HTTP request to the load balancer for the creation or deletion of a worker.

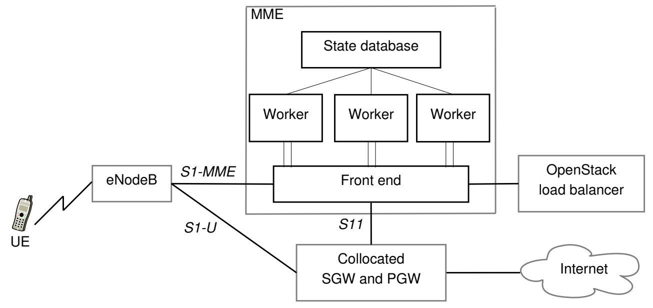

Fig. 1. System architecture

B. Worker

The worker represents the actual functionality of the MME and handles the processing of call flows. Each worker maintains two separate interfaces towards the FE, one for S1AP messaging and the other for GTP messaging. The Home Subscriber Server is implemented as a MySQL database and is co-located with the worker. Thus, the S6a interface is not realised. We implement and test only the E-UTRAN initial attach and explicit UE-initiated detach procedures specified in [11]. In order to make the workers stateless, we introduce additional messages to store the UE context on the state database. Communication between a worker and the state database occurs over a TCP connection and the UE context is saved at the end of a call flow. Storing the UE context at each step in the call flow can help to increase the resilience of the MME. However, this leads to increased CPU utilisation and network overhead with external database queries. By choosing to store UE context at the end of a call flow, there is a trade-off between low latency (with fewer database operations) and decreased resilience.

C. State database

The state database stores state information for each UE attached to the MME. We choose Redis [12] to implement the state database. The in-memory feature of Redis ensures that it operates with very low latency. We use the Redis cluster feature to shard the data over three VMs. The UE context is stored as a key-value pair. The key is the MME UE S1AP ID and the value is the UE context in binary format. Furthermore, Redis offers various persistence policies to enable recovery of the stored data in case of a Redis server failure.

IV. Experimental Evaluation

This section describes the testbed used and presents the results of the experiments performed.

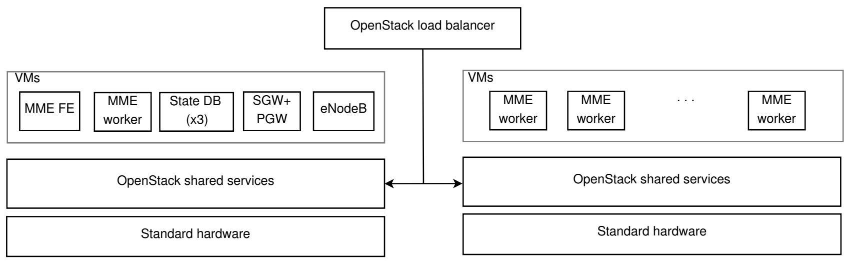

Fig. 2. Overview of the testbed

A. Testbed setup

Our testbed consists of two OpenStack [13] clouds running the Icehouse release 2014.1.3. Each OpenStack cloud consists of four identical blade servers. Two servers are used as compute hosts on which VMs are run. The compute hosts use the Kernel-based Virtual Machine (KVM) hypervisor. The OpenStack controller runs on the third server. The fourth server is used for deploying networking services with an Open vSwitch virtual switch. Our test environment consists of VLAN-based virtual networks which allows us to separate the traffic within the VMs deployed in the testbed. Fig. 2 shows the architecture of the testbed. We deploy the following VMs on the testbed:

- **FE**. The FE of the distributed vMME is deployed on one VM and contains EPC-specific configurations for the MME.
- **Worker**. Each MME worker is deployed on a separate VM and configured to connect to the FE. Workers can be deployed on either OpenStack cloud.
- **State database**. The state database consists of three Redis servers, each on a different VM, configured as a cluster of three master nodes. Redis 3.0.1 is used on all three VMs.
- **eNodeB**. The eNodeB is a simple C program which sequentially sends the messages needed to test the attach and detach procedures.
- **SGW and PGW**. The nwEPC, an open source implementation of the gateways [14], is used to run a collocated SGW and PGW on a VM.
- **Standalone MME**. The MME developed in [10] represents the original standalone MME and is deployed on a VM (not shown in the figure).

B. Experimental Results

We compare the performance of the distributed vMME to that of the original standalone vMME. The experiments performed and the results obtained are presented below:

1) Attach latency: The attach latency is measured as the time elapsed between the eNodeB sending an Attach Request and receiving an Attach Accept. We run this experiment for both the distributed and standalone vMMEs to compare their average latency. Each experiment consists of attaching 50 UEs sequentially. The latency calculated is the average of results obtained from running five sets of experiments. Table I shows the average value of the measured attach latency. The increase in attach latency for the distributed vMME is 4.4 milliseconds on average. This occurs because each message exchanged between the worker and an external network element has to be forwarded through the FE.

We further measure the attach latency for different placements of FE and workers. The following placement configurations are used:

a) Worker and FE on different OpenStack clouds;
b) Worker and FE on the same compute host in the same OpenStack cloud;
c) Worker and FE on different compute hosts in the same OpenStack cloud

Fig. 3 shows the cumulative distribution function (CDF) of the measured attach latency and Table II shows the average attach latency for the three different placement configurations. The measured latency for case (b) wherein the FE and worker are on the same compute host shows the lowest average value of 12.368 milliseconds. In this configuration the lowest latency measured is 6.394 milliseconds, whereas the maximum latency shows a ten-fold increase to 65.553 milliseconds. Similar outliers are seen in further experiments with this placement configuration (not reported in this paper). The lower average value for this configuration can be attributed to the networking setup on OpenStack, wherein communication between VMs co-located on the same compute host occur locally and need not be sent to the networking host [15]. However, these local communications can increase CPU and memory overhead making the runtime effects hard to predict and possibly leading to performance degradation [15]. For VMs on different compute hosts, the latency increases with a more predictable trend. The distributions of attach latency for cases (a) and (c), wherein the FE and worker are on different compute hosts, are quite similar. The maximum measured attach latency does not exceed 25 milliseconds in both cases.

2) UE context retrieval time: We compare the time taken to retrieve UE context in the distributed vMME to that taken

TABLE I. AVERAGE OF ATTACH LATENCY FOR STANDALONE AND DISTRIBUTED VMME

vMME type	Average latency	95% confidence interval
Standalone	8.399 ms	±0.563
Distributed	12.782 ms	±0.208

TABLE II. Average of attach latency for placement configurations of FE and worker

Placement	Average latency	95% confidence interval
(a)	12.914 ms	±0.222
(b)	12.368 ms	±0.505
(c)	13.065 ms	±0.288

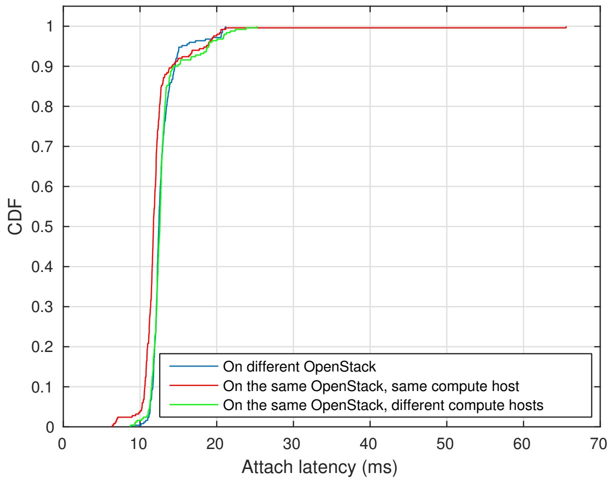

Fig. 3. CDF of attach latency for three different placement configurations of FE and worker

by the standalone vMME. For the distributed vMME, the measured value includes the time taken to send a request from a worker over the network, execute the query on the Redis server and receive the response from the same server. On the standalone vMME, the UE context is stored in memory as a C structure in a hash table. The time measured is simply the retrieval time from the hash table. Table III presents the average value of 250 retrievals. For the distributed vMME, the main contributor to the measured time is the network latency between the worker and Redis server. The time taken for running the commands on the Redis server (monitored using *redis slowlog*) is in the order of a few microseconds.

V. Conclusion and Future Work

We have developed a working proof-of-concept implementation of a three-tier architecture for the MME. We have identified that the network latency between the components of the distributed vMME affects the attach latency. Furthermore, the attach latency varies depending on the placement of the VMs on the physical compute hosts. With an intelligent orchestrator and placement algorithm, it is possible to deploy the components of a distributed MME by taking into account different considerations such as latency, resilience, networking bandwidth and CPU interference. From the results presented in this paper, we believe that the increase in latency is offset by the benefits of scalability and resilience of the vMME. However, the increased complexity of the architecture demands careful software design and intelligent placement of the components so as not to degrade performance of the overall system. In the future, we plan to investigate the effects of the Redis persistence policies on the attach latency. Furthermore, in our current implementation, compute hosts of both OpenStack clouds reside on the same rack server. It will be interesting to evaluate the distributed vMME with geographically distributed data centers or public clouds. Additionally, we aim to investigate container-based virtualization to reduce the current worker boot up times.

TABLE III. Average retrieval time for UE context on standalone and distributed MME

vMME type	Average time	95% confidence interval
Standalone	20.7 μs	±0.675
Distributed	1256.724 μs	±18.028

Acknowledgment

This work has been partially supported by the CELTIC-Plus project C2012/2-5 SIGMONA. The information reflects the consortium's view, but the consortium is not liable for any use that may be made of any of the information contained therein.

References

[1] T. Taleb, M. Corici, C. Parada, A. Jamakovic, S. Ruffino, G. Karagiannis, and T. Magedanz, "EASE: EPC as a service to ease mobile core network deployment over cloud," *Network, IEEE*, vol. 29, no. 2, pp. 78–88, 2015.
[2] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014-2019," Cisco, White Paper, February 2015.
[3] T. Taleb and A. Kunz, "Machine type communications in 3gpp networks: potential, challenges, and solutions," *Communications Magazine, IEEE*, vol. 50, no. 3, pp. 178–184, 2012.
[4] "MCN D4.1, Mobile Network Cloud Component Design, European Commission, EU FP7 Mobile Cloud Networking public deliverable," November 2013.
[5] Y. Takano, A. Khan, M. Tamura, S. Iwashina, and T. Shimizu, "Virtualization-Based Scaling Methods for Stateful Cellular Network Nodes using Elastic Core Architecture," in *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on.* IEEE, 2014, pp. 204–209.
[6] X. An, F. Pianese, I. Widjaja, and U. G. Acer, "dMME: Virtualizing LTE mobility management," in *Local Computer Networks (LCN), 2011 IEEE 36th Conference on.* IEEE, 2011, pp. 528–536.
[7] M. Tamura, T. Nakamura, T. Yamazaki, and Y. Moritani, "A study to achieve high reliability and availability on core networks with network virtualization," NTT Docomo, Technical Journal, July 2013, online; Accessed on 02.07.2015.
[8] "Project Clearwater," http://www.projectclearwater.org/, online; Accessed on 02.07.2015.
[9] G. Carella, M. Corici, P. Crosta, P. Comi, T. M. Bohnert, A. A. Corici, D. Vingarzan, and T. Magedanz, "Cloudified IP Multimedia Subsystem (IMS) for Network Function Virtualization (NFV)-based architectures," in *Computers and Communication (ISCC), 2014 IEEE Symposium on.* IEEE, 2014, pp. 1–6.
[10] V. F. Guasch, "LTE network Virtualisation," Master's thesis, Aalto University School of Electrical Engineering, October 2013.
[11] 3GPP, "General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access (Release 12)," 3rd Generation Partnership Project (3GPP), TS 23.401, December 2014.
[12] "Redis," http://redis.io/.
[13] "OpenStack," http://www.openstack.org/.
[14] "nwEPC - EPC SAE Gateway ," http://sourceforge.net/projects/nwepc/, online; Accessed on 02.06.2015.
[15] A. Corradi, M. Fanelli, and L. Foschini, "VM consolidation: A real case based on OpenStack Cloud," *Future Generation Computer Systems*, vol. 32, pp. 118–127, 2014.