Cloudified Mobility and Bandwidth Prediction in Virtualized LTE Networks

Zhongliang Zhao[1], Morteza Karimzadeh[2], Torsten Braun[1], Aiko Pras[2], Hans van den Berg[2,3]

[1]*University of Bern, Switzerland,* [2]*University of Twente, The Netherlands*
[3]*Netherlands Organization for Applied Scientific Research (TNO)*
Email:{zhao, braun}@inf.unibe.ch, {m.karimzadeh, a.pras, j.l.vandenberg}@utwente.nl

***Abstract*—Network Function Virtualization involves implementing network functions (*e.g.,* virtualized LTE component) in software that can run on a range of industry standard server hardware, and can be migrated or instantiated on demand. A prediction service hosted on cloud infrastructures enables consumers to request the prediction information on-demand and respond accordingly. In this paper we introduce MOBaaS, which is a network function of Mobility and Bandwidth prediction cloudified over the cloud computing infrastructure. We implemented the service orchestration framework of MOBaaS, which can easily be setup and integrated with any other cloud-based LTE entities to provide prediction information about the future location of mobile user(s) as well as the network link(s) bandwidth availability. This information can be used to generate required triggers for on-demand deployment or scaling-up/down of virtualized network components as well as for the self-adaptation procedures and optimal network function configuration. We also describe the performance evaluation of the MOBaaS cloudification procedures and present an example of the benefit of such a prediction service.**

I. Introduction

Long Term Evolution (LTE) as the fourth generation (4G) cellular system is capable of providing high data rates as well as support of high-speed mobility. Even though LTE promises a faster and more efficient mobile data network, its core network architecture is still highly hierarchical, leading to high bandwidth requirement and processing load on core network nodes. The straightforward and short-term solution to cope with these issues, may consist of operators investment to upgrade the resources and build a scalable architecture to handle the bandwidth-intensive mobile applications more efficiently. This approach is technically and technologically feasible. However, the network operators mostly desire to stand for the cost-effective solutions.

The huge appreciation received by cloud computing technologies in latest years motivated mobile network operators to plan the adoption of Network Function Virtualization (NFV) in their network to meet the increasing mobile traffic demands. In this regard, cloudification of the LTE [1] aims to integrate the use of cloud computing concepts in the LTE network with the objective of increasing LTE's performance by building a shared distributed mobile network while optimizing the utilization of computation, storage and networking resources. This can be realized by implementing and running the LTE's components – *i.e.,* E-UTRAN (Evolved UMTS Terrestrial Radio Access Network) and the EPC (Evolved Packet Core) – over distributed cloud computing data centers (Fig. 1). The most important benefits of this approach are the support of on-demand deployment, provisioning, disposing of virtualized LTE system components, the support of resource allocation both on-demand and dynamically.

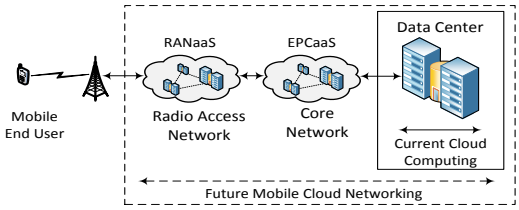

Fig. 1: A view of cloudified LTE network system.

Generally, the duration of deploying, scaling, or migration of Virtual Machine (VM) are higher than the acceptable delay for the communication of a subscriber (*e.g.,* booting up of a VM may take 10-20s, but a hand over is lost after 30-50ms in LTE). Therefore, to give enough time for the system to adapt to guarantee the desirable quality of service, an appropriate mechanism other than reacting to metered values by the monitoring system is required. Prediction approach in order to estimate the system change state in the future, is one of the key mechanism that can be used to trigger the decision making process.

In [2] we proposed a software architecture entitled as MOBaaS (Mobility and Bandwidth Prediction as a Service), encompassing two algorithms that can provide prediction information about the user(s) mobility and link bandwidth availability in a particular future moment. Estimation of both the algorithms rely on a significant amount of user's mobility or network link's bandwidth usage historical information. The prediction information provided by MOBaaS can be interpreted as an estimation of geographic and temporal traffic distribution in a network, which are key parameters for realising the Virtualized Network Functions (VNF) and smart city applications.

In the paper we extend our previous works by threefold: (i) we present in detail the MOBaaS's implementation architecture that can be readily integrated with any other virtualized LTE component to provide prediction information; (ii) we develop an framework to implement a fully cloudified MOBaaS, which can provide prediction services on-demand such that it can easily be instantiated, deployed and disposed in a cloud infrastructure; (iii) we verify the MOBaaS cloudification operations and analyze its functional and non-functional performance.

978-3-901882-89-0 @2017 IFIP

The remainder of the paper is organized as follows. Section II describes briefly a few applications of prediction information and presents the motivation for the MOBaaS. The internal components of the MOBaaS implementation architecture is described in Section III. Section IV details the technical implementation components of the MOBaaS cloudification reference architecture. Section V presents the cloudification performance evaluation results. Finally, the paper ends up with conclusion in Section VI.

II. Related Work

Predicting mobile users locations at any time moment in the future is essential for a wide range of mobile applications, such as location-based services, mobile access control, mobile multimedia QoS provision, as well as the resource management for mobile network operators and cloud storage.

A variety of mobility prediction systems have been proposed. However, most of them focus on using cloud computing platform to provide storage and computing resources [3]. Liang et al. have been developed a mechanism to address a prediction problem in a personal communication service networks based on user's location, movement history, movement pattern, velocity, etc [4]. However, most of the works focus on providing an isolated prediction framework, which can not be utilized by other network services. Few efforts have been made to provide mobility prediction as a virtualized network function (VNF) for the virtualized LTE network. In a cloudified LTE system, different virtualized services might also need the mobile users location information to optimize network performance. As an example, Follow-Me Cloud concept and Information Centric Network (ICN) could benefit from the mobility prediction information to (re)place the content closer to locations that users will visit in the future [5].

In this work, we present *Mobility and Bandwidth prediction as a service* - a new service model of telecom operator cloud that enhances the telecom cloud with mobility prediction capacity. We explain MOBaaS via describing the service lifecycle of an on-demand, elastic, and pay-as-you-go mobility prediction service instantiated on top of the cloud infrastructure. MOBaaS service life-cycle management is a process of network design, deployment, run-time management, and disposal that allows to rapidly architect, instantiate, and reconfigure the network components and their associated services.

III. MOBaaS Architecture

Given the above description, a MOBaaS cloudification implementation architecture needs to fulfill the following requirements:

- An efficient algorithm to predict the user(s) mobility availability and network link(s) bandwidth availability.
- A reference architecture that the MOBaaS can be easily integrated with other virtualized LTE components to retrieve the input data for the prediction algorithms or to provide the prediction results.
- An orchestration framework that manages the MOBaaS service instantiation, deployment, and disposal on-demand on a cloud infrastructure.

In [2], we discussed and evaluated the user(s) mobility and network link(s) bandwidth availability prediction algorithms comprehensively. In this section we detail the proposed reference architecture for the MOBaaS and present the composing entities and their functionalities. In Section IV we describe the orchestration framework and present its components for the MOBaaS cloudification process.

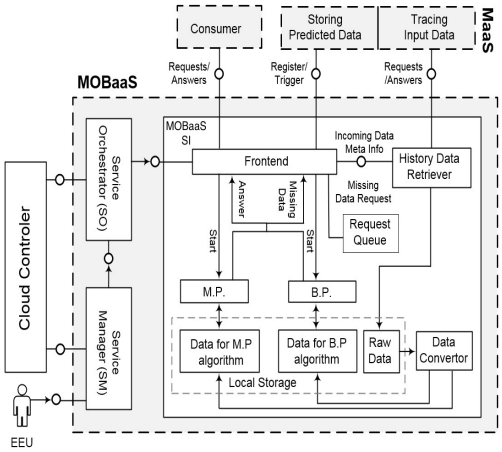

Fig. 2: MOBaaS implementation architecture.

Fig. 2 shows the MOBaaS Service Instance (SI) and the MOBaaS internal components. The internal components are included in the Service Instance Components (SIC). The roles of each components are listed below:

- ***Frontend:*** It works as the interface towards service consumers. If there is a prediction request whose required input information is currently not available, the frontend stores the request in the *Request Queue*, and that request will be postponed until the missing data becomes available. If the request can be met, it will starts the prediction algorithm threads. It also handles the trigger-based prediction. In this case, it periodically calculates future states of users' movements and network links bandwidth and stores the prediction information in MaaS (Monitoring as a Service). This data is used later to trigger a consumer for a particular decision making procedure.

- ***M.P.:*** It presents the mobility prediction algorithm and predicts the location of an individual/a group of end-users in a future moment of time. The improved version of Dynamical Bayesian Network (DBN) [6] was used as the mobility prediction mechanism in this architecture. The rationale behind using DBN is that, the next location (cell) visited by a user only depends on its current location, the current time, and the day of the week that the user is in the movement. This approach utilizes the trace of mobile user trajectories to predict the next location that may be visited, and more details can be found in [2].

- ***B.P.:*** It presents the bandwidth prediction algorithm and estimates the bandwidth used/available at a certain network link in a future moment of time. The B.P. relies on the dimensioning approach, which is able

to take into account the impact of possible traffic bursts on the required link capacity. The algorithm applied in bandwidth prediction approach relies on the information obtained from Flow data (e.g., NetFlow, IPFIX, J-Flow). More details can be found in [2].

- ***History Data Retriever:*** This module continually retrieves end users' historical movement traces from MaaS. This data later is processed and used as the input data for the prediction algorithms.

- ***Data Converter:*** It processes the raw monitored data and converts it to the format usable by the algorithms.

- ***Request Queue :*** It queues the requests when either a specific data to perform a prediction is missing in the input trace data, or more than one prediction is requested by consumers.

- ***Data for M.P./B.P. algorithms:*** These are the local databases storing the data processed by the *Data Converter*. To perform estimations on the user(s) mobility or the available bandwidth in a certain network link, a significant amount of user(s) movement or link usage history information is required. This information can be acquired from MaaS.

MOBaaS consists of two types of prediction notification mechanisms: *request-based* and *trigger-based*.

In the request-based approach, a web server running at the Frontend constantly waits for prediction requests from the consumers. Whenever a consumer requires a prediction, it sends the request information to MOBaaS using a JSON (Java Script Object Notation) message. The request message includes the User ID (for single user), the current time and date, the current Cell ID, and the time period of the required prediction. Given this information, M.B. makes the mobility prediction of which cell the specified user(s) will be located at a certain future time. Next, Frontend returns the prediction results in the JSON message to the consumers, including multiple pairs of *<Cell ID, Probability>*. The bandwidth prediction is similar to the mobility prediction, except that the request message includes the ID of a link, and its aiming time for bandwidth prediction. In this case, the results message sent by B.P. includes the *<Link ID, Available Capacity>*.

In the trigger-based approach, the web server running at the Frontend gets the prediction requests from consumers including two parameters: the maximum number of users (N) in each cell (for mobility prediction) or the minimum available capacity (C) in a specific network link (for bandwidth prediction) as a threshold, and the prediction interval (δ). Knowing this information, the M.B. and B.P. periodically in every δ intervals perform mobility and bandwidth predictions, respectively. If the number of users per cell exceeds (for mobility prediction), or the amount of available capacity in the network link reduces (for bandwidth prediction) the threshold, a trigger message is generated by the *Frontend* and results are pushed into MaaS. This information could later be used by consumers for a specific action (*e.g.,* scaling the resources, instantiating the VMs) [2].

IV. MOBaaS Cloudification

In this section, we describe how mobility and bandwidth prediction services have been cloudified, achieving a MOBaaS. The cloudification of MOBaaS allows it to run in the cloud platform with all the benefits brought by cloud principles. Namely, the EEU (Enterprise End User) only needs to provide the network topology for the deployment among other a few settings. Afterwards, it will get a running service instance that manages itself for a typical operation. The key components for MOBaaS cloudification are the service manager (SM), service orchestrator (SO), cloud controller (CC), and service development kit (SDK). These logic entities have been designed and developed within the context of the Mobile Cloud Networking (MCN) project [1]. In the following subsections, these components along with other cloudification-related components are described in detail.

A. Technical Reference Architecture

In principle, the architectural elements are influenced by Start of Authority (SOA) principles, and constructs are in place to facilitate the service lifecycle management. The detailed

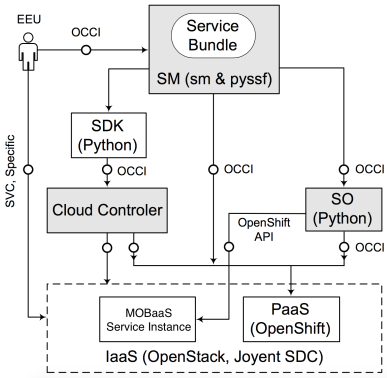

Fig. 3: MOBaaS orchestration framework architecture.

representation of the architecture element is shown in Fig. 3, and the roles of different elements are listed below:

- ***Service Orchestrator (SO):*** It is responsible for the orchestration of the MOBaaS service, consisting of two internal modules: the Service Orchestrator Decision (SOD) and Service Orchestrator Execution (SOE). The decision module is the component responsible for handling inputs from other services, mainly monitoring information (from *e.g.,* MaaS) that makes decisions on whether a scaling decision is needed. The execution module is responsible for the interaction with the Cloud Controller (CC) using a Service Development Kit (SDK), ensuring that service instances are mapped to cloud resources as well as are deployed, scaled, and disposed whenever requested. It is also responsible for handling requests from SM to instant or dispose the MOBaaS service instance through the SDK.

- ***Service Manager (SM):*** It provides an external interface to the EEU and a list of available services (Service Catalog). It is responsible for deploying the SO, and forwarding requests to SO for MOBaaS service instance deployment and disposals. Upon receipt of a service creation request, the SM creates a Docker

container in OpenShift [7] and when the container is ready, pushes the service bundle into the container and deploys it. Once the SO is ready, the various life-cycles of the services are activated (Fig. 3).

- ***Open Cloud Computing Interface (OCCI):*** OCCI [8] enables to maintain interoperability and a common interface and model between different services (*e.g.,* MOBaaS with MaaS, RaaS, etc). It forms the core of the interfaces exposed by the SM, CC, and the SO. It is also the specification used by the northbound interface of the Cloud Controller.
- ***Cloud Controller (CC):*** CC is implemented through a set of modules to support service deployment, provisioning, and disposal. Each of the modules is modeled as a service itself.
- ***Service Development Kit (SDK):*** It is designed to support service developers to manage different services. It servers two main purposes: (i) it supports the basic functionalities for the life-cycle management of any service by allowing the SO to interact with the various internal services of the CC. Therefore, any changes to the internal implementation of the CC will have no impact on the code written, as the SDK abstracts the technologies used; (ii) it provides access to the supporting services, which allows for the developer of a certain service to interact with another service in a certain way, regardless how the other service is implemented [9],[10].
- ***Service Bundle:*** It contains the Service Manifest details, which comprises of Service Template Graph (STG) where dependencies on other external services are encoded, and Infrastructure Template Graph (ITG), which is realized as a Heat Template (HT) that encodes all the details for the deployment of the service itself. The service bundle also contains the SO logic (the application code).
- ***OpenStack Infrastructure:*** OpenStack is the cloud management framework that allows the life cycle management of virtual machines. It provides infrastructure cloud services, namely compute, storage, and networking. OpenStack is chosen as the reference implementation framework. The OpenStack Heat [11] orchestration module allows service deployment as one logical unit instead of a collection of a number of virtual machines.
- ***OpenShift:*** A PaaS (Platform as a service) solution called OpenShift [7] is adopted to enable the CC deployment module to host the SO instance. OpenShift provides quick and effective means to run SO workloads, and OpenShift v3 (Docker container based) builds the application container for every SO instantiation efficiently.

B. Sequence Diagrams of MOBaaS Service Management

The sequence diagrams are interaction diagrams that show the different steps that have to be done by the system to realize a certain operation requested from an external actor, either a person or an external system. It shows the different classes that are involved in the operation and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario (*e.g.,* MOBaaS instantiation).

In the following, we describe the general MOBaaS cloudification operations, including the sequence diagrams of service deployment, provisioning, and disposal. Through the SM and the SO, the MOBaaS system can request the CC to deploy, dispose, provision, and management operations of the MOBaaS instance. It allows the EEU to get a MOBaaS instance, deploy and dispose it in an on-demand fashion.

Fig. 4 shows the deployment of the MOBaaS service. The main users of the system (Service Provider) can log into the administrator panel and execute a command for deployment of an instance. As a start, the EEU requests through MOBaaS's SM the deployment and provisioning of a MOBaaS instance. The SM module then send the request including the OpenStack Heat template graph of the service to the CC to start an instance of the SO. Heat is the main project in the OpenStack Orchestration program.

It implements an orchestration engine to launch multiple

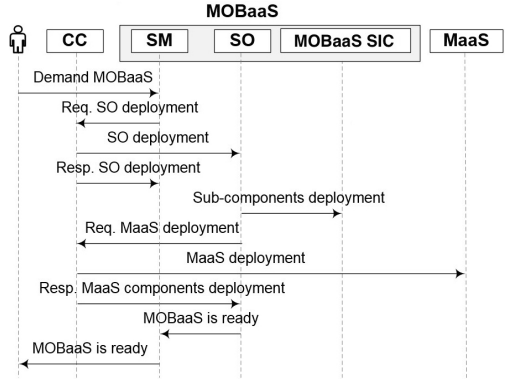

Fig. 4: MOBaaS service instance deployment procedure.

composite cloud applications based on templates in the form of text files that can be treated as a code. In general, a Heat template describes the infrastructure for a cloud application in a text file that is readable and writable by humans. Once the SO is instantiated, the SM requests the SO to deploy the MOBaaS. Afterwards, SO requests the CC to instantiate the ITG of the MOBaaS. Next, the CC instantiates the internal components of the MOBaaS instance (Frontend, M.P., B.P., Historical Data Retriever, Data Converter, and Request Queue). After this, if everything is correct, as a response, an endpoint to the created instance will be returned, which makes us able to configure and manage the created instance.

Fig. 5 shows the provisioning phase of the service. After deployment, provisioning should have the actual virtual machines configured and running. To achieve this, after getting the SO instance from the CC, SM commands SO to provision required VMs according to the provided configuration. In this way, we have an MOBaaS instance deployed and ready to use. Fig. 6 shows the disposal phase of the service. Disposal means disposing all the virtual machines created in the provisioning phase. Whenever the EEU does not need the service instance anymore, it may ask SO for the disposal. When receiving the disposal request, the SO will ask the CC to dispose all the stacks associated with the service instance. After disposing all the resources by the SO, it sends a message to the SM to

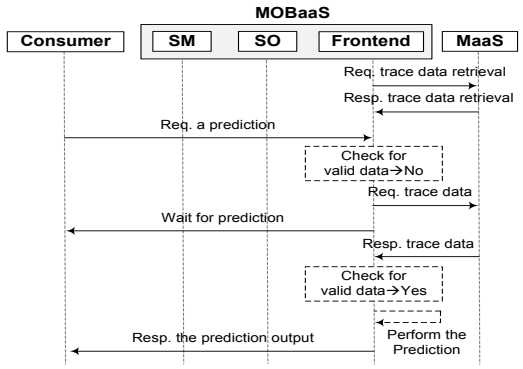

Fig. 5: MOBaaS service instance provisioning procedure.

inform it about the situation. Afterwards, the service manager sends the destroy command to the SO. When all is done, a conformation message will be shown to the user.

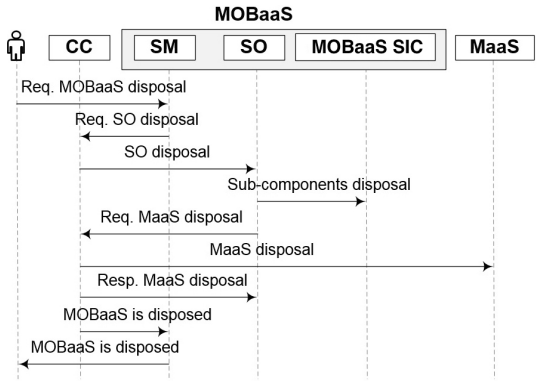

Fig. 6: MOBaaS service instance disposal procedure.

V. Evaluation

This section describes the performance evaluation of the proposed service, which includes description of the evaluation scenario and methodology, as well as an analysis of the results.

A. Scenarios

In order to present privilege of the proposed MOBaaS service and to validate its functionality and performance, we target at a content migration scenario where a mobile user moves from one place (eNodeB) to another. The user requests the same content at different places, and we analyze the content retrieval time when the user is moving with and without the help of mobility prediction provided by the MOBaaS.

B. Methodology

The evaluation procedure has two aspects: *non-functional* performance evaluation and *functional* performance evaluation.

The non-functional evaluation refers to the time measurements of deployment, provisioning and disposal the MOBaaS instance in a cloud infrastructure. To perform this, a monitoring system is needed. We used the Graylog tool [12], integrated in the SM library, as a log aggregator to collect the related timing information for the MOBaaS cloudification operations. The data collected by this tool is used to measures the timing of deployment, provisioning, and disposal procedures of the MOBaaS instance.

The functional evaluation presents the accuracy of the MOBaaS's prediction algorithms on estimating the mobile user(s) location and network link(s) bandwidth availability in a particular moment of time at future, as well as shows the benefit of using the MOBaaS to a consumer service (*e.g.,* in terms of reducing content retrieval time for the scenario described in Section V-A)

C. Performance Evaluation Results

1) Non-Functional Evaluation: The results of non-functional evaluation are illustrated in Fig. 7, including the MOBaaS instance deployment, provisioning (for both the mobility and bandwidth predictions) and disposal latency. To study the impact of hardware and platform of the cloud computing infrastructure on the MOBaaS implementation, we used two different testbeds with different allocated resources for the MOBaaS instance as follows:

- ***Testbed 1:*** It offers all essential services, including OpenStack Nova with KVM, Glance, Cinder, Neutron, and Heat. The OpenStack version is Juno running on Ubuntu 12.04. The testbed consists of a controller node, computing node (Dell PowerEdge R520), and a storage node (Dell PowerVault MD3800i). It operates 32 cores, Interl(R) Xeon(R) CPU E5-2450 v2@2.50GHz, 192 GB RAM, 2.3 TB HDD for nova-computing filesystem, 9.0 TB HDD for the cinder block storage, and 120 floating IPs.
  We have committed the following resources for MOBaaS implementation: 100 instances, 100 VCPU, 50GB RAM, 50 public IPs, and 1 TB of storage.
- ***Testbed 2:*** It offers the basic OpenStack services as Testbed 1. The OpenStack version is Kilo. It consists of multiple servers and has a total of 64 cores, 377 GB RAM, 5.5 TB disk and 44 floating IPs available. Additionally, Testbed 2 is an Identity Provider, a service that allows users to authenticate and get a token to interact with other services. The Identity Provider is implemented with OpenStack Keystone. The OpenShift V3 [7], is hosted in Bart also and is used by the orchestration.
  Resources available to implement the MOBaaS are: 10 instances, 20 VCPU, 20GB RAM, 8 public IPs, and 1TB of storage.

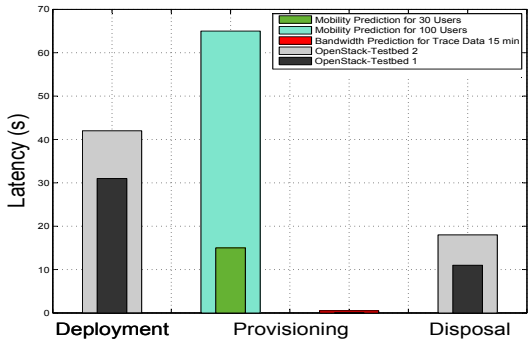

Fig. 7: Evaluation of the MOBaaS cloudification operations.

The deployment phase includes the creation of a VM and allocation of related resources to it. We can observe that the deployment phase takes more time than provisioning and disposal phases. This is due the fact that the deployment phase, includes also the allocation floating IP addresses to VMs, the installation of the SO as a docker container in the OpenShift platform. The provisioning phase refers to properly configure the MOBaaS's VM and the internal components that has been setup in the deployment phase. It is also include the operations for providing the mobility and bandwidth results. The disposal phase includes the deletion of the VM and releasing the allocated resources and IP addresses that are no longer required.

Obtained results show that, in Testbed 1 the deployment time (31s) and disposal time (11s) are shorter than in Testbed 2 (42s and 18s, respectively). This is because more resources are allocated at Testbed 1 during the evaluation procedure. However in both cases, the related latencies are in the order of seconds and short enough, and could meet the on-demand instantiation requirements.

In the provisioning phase, the size of the trace files significantly influences the prediction calculation time. In the mobility prediction the number of users also directly affects the prediction latency (more detail can be find in [2]). Therefore, for the time-critical applications *e.g.,* smart city the prediction requests with large computation overhead, which might lead to a longer response delay, need to be split into multiple parallel lightweight computation procedures.

2) Functional Evaluation: Fig. 8 shows the accuracy of the mobility prediction algorithm for a few users (as examples) in different weekdays. As it is observed, a high prediction accuracy of around 85% could be reached for some users (*e.g.,* user with ID 6030) on Friday, while some users can only get an accuracy of 10 % (*e.g.,* user with ID 6016) on Sunday. This variation of prediction accuracy is due to the different qualities of users' traces.

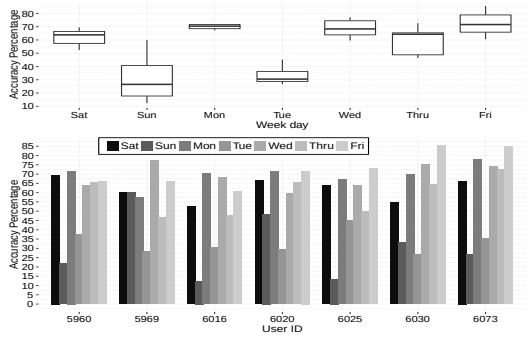

Fig. 8: Mobility prediction accuracy for some users during weekdays.

In the used trace files, for some users there is continuous recording of the location information, which leads a very good trace quality. While for some others the mobile-phone might always is not ON or is not taken with the users such that their movement trace files are discrete, which makes it very difficult to generate accurate prediction. In addition to the heavy dependence on the trace files quality, we can also see that prediction accuracy varies for different weekdays. The prediction is more accurate in weekdays than in weekend, this is due to the fact that people have regular movement pattern during the weekdays, such as fix movements from office to home, etc. Details about the explanation of the prediction algorithms and its evaluation can be found in [2].

In order to illustrate the benefit of MOBaaS, we measure the content retrieval time for a user when it requests to access a content at the destination place for both the case with and without the help of mobility prediction information. The benefit of using mobility prediction result is clear from a user's perspective, since the content retrieval time is 33% lower for all content file sizes (Fig. 9). This is due the fact that with the help of mobility prediction, the interested content is already available at (or started to be migrated to) the next location before user's actual movement. In scenarios without mobility prediction, after reaching user to the destination cell, it has to request the content, which takes more time to access the requested consent compared to the scenario with using the mobility prediction information. The performance improvements are more obvious when the content size is bigger.

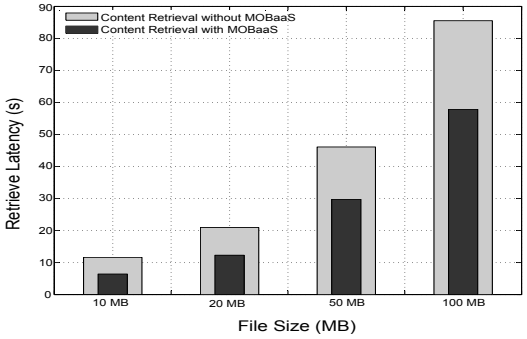

Fig. 9: Content retrieval with/without using the MOBaaS.

VI. Conclusion

Virtualization of the LTE network requires on-demand deployment, provisioning, and disposing of the cloudified LTE components. In this paper, we have developed an implementation architecture for the MOBaaS (Mobility and Bandwidth prediction as a Service), which can easily be setup on a OpenStack cloud computing infrastructure, and can be readily integrated with any other virtualized LTE component to provide prediction information. The MOBaaS prediction information can be used to generate the required triggers for on-demand deployment of virtualized network components as well as for the self-adaptation procedures and optimal network function configuration during run-time operation. These are also a key enabling concepts to optimize the smart city operations, such as data traffic planning and network links bandwidth re-allocation. We detailed the service management lifecycles of the MOBaaS cloudification operations, and shown that its instance can be deployed and disposed in enough short time with an on-demand fashion. To present the benefit of using MOBaaS, we evaluated the content retrial time from a user's perspective in a ICN network, with and without using the mobility prediction information. The obtained results shown that the access time for the requested content can be enhanced almost 33% using the MOBaaS prediction information.

ACKNOWLEDGEMENTS

The research work presented in this paper is conducted as part of the Mobile Cloud Networking project, and has been funded by the European Union Seventh Framework Program under grant agreement [# 318109].

REFERENCES

[1] EU FP7 Mobile Cloud Networking project, February 2015. http://www.mobile-cloud-networking.eu/site/.

[2] Morteza Karimzadeh, Zhongliang Zhao, Luuk Hendriks, Ricardo de O. Schmidt, Sebastiaan la Fleur, Hans van den Berg, Aiko Pras, Torsten Braun, and Marius Julian Corici. Mobility and bandwidth prediction as a service in virtualized LTE systems. In *IEEE 4th International Conference on Cloud Networking (CloudNet)*, 2015.

[3] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless com. and mobile computing*, 13(18):1587–1611, 2013.

[4] B. Liang and Z. J. Haas. Predictive distance-based mobility management for pcs networks. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1377–1384 vol.3, Mar 1999.

[5] Bruno Sousa, Zhongliang Zhao, Morteza Karimzadeh, David Palma, Vitor Fonseca, Paulo Simoes, Torsten Braun, Hans van den Berg, Aiko Pras and Luis Cordeiro. Enabling a Mobility Prediction-aware Follow-Me Cloud Model. In *41st IEEE Conference on Local Computer Networks (LCN)*, 2016.

[6] Vincent Etter, Mohamed Kafsi, Ehsan Kazemi, Matthias Grossglauser, and Patrick Thiran. Where to go from here? Mobility prediction from instantaneous information. *Elsevier Pervasive and Mobile Computing*, 2013.

[7] OpenShift. https://www.openshift.com. Online, accessed Feb. 2016.

[8] OCCI-Open Cloud Computing Interface. http://occi-wg.org. Online, accessed Feb. 2016.

[9] Infrastructure Management Foundations–Final Report on Component Design and Implementation, MCN D3.4. In *European Commission, EU FP7 Mobile Cloud Networking public deliverable, available at[1].*

[10] Final Overall Architecture Definition, MCN Dd2.5. In *European Commission, EU FP7 Mobile Cloud Networking public deliverable, available at[1].* 2015.

[11] OpenStack Heat Orchestration. https://wiki.openstack.org/wiki/Heat. Online, accessed Feb. 2016.

[12] Graylog. https://www.graylog.org. Online, accessed Feb. 2016.