DevOps for IoT Applications using Cellular Networks and Cloud

Athanasios Karapantelakis*†, Hongxin Liang*, Keven Wang*, Konstantinos Vandikas*,
Rafia Inam*, Elena Fersman*, Ignacio Mulas-Viela*, Nicolas Seyvet*, Vasileios Giannokostas†
*Ericsson Research, Stockholm, Sweden
{firstname.lastname}@ericsson.com
†Royal Institute of Technology (KTH), Stockholm, Sweden
{athkar, vasgia}@kth.se

***Abstract*—The Internet of Things (IoT) is a vision of a future society where an ever-increasing number of heterogeneous physical devices ("things") obtain Internet connectivity, thus enabling a large number of applications for a broad range of industries and society at large. Mobile network operators, expected to provide the network infrastructure for many of these applications, face an unprecedented level of complexity. This complexity not only relates to the number of applications that share the network infrastructure, but also to the different network Quality of Service (QoS) requirements these applications have. To achieve economies of scale, automation in management of those applications throughout their lifecycle is essential. In this paper, we propose an architecture that automates allocation, monitoring and deallocation of both cloud and cellular network resources to ensure QoS for applications that involve connected devices communicating with cloud-hosted software. We describe an implementation of this architecture using a combination of open source, commercial, and custom components and evaluate it through a series of measurements. Results show that our implementation can simultaneously support mobile broadband and low latency, high availability mission-critical applications.**

***Index Terms*—5G, Cloud, Internet of Things, Network Slicing, DevOps, Resource Orchestration, Automation.**

I. Introduction

Internet of Things (IoT), envisions billions of Internet-addressable, heterogeneous devices which will enable development of a new generation of applications for a wide range of industries including transportation, healthcare but also residential and workspace environments [1]. The upcoming, fifth generation of mobile networks (5G) is set to play an important role for supporting these applications. While mobile-broadband applications continue to drive demand for higher data traffic capacity and higher end-user data rates, a new generation of "mission critical" applications that require reliable connectivity and high availability emerges [2].

As "applications" in the context of this paper, we consider *the subgroup of all IoT applications that require cellular network connectivity, and consist of software (hosted in a cloud), communicating with devices through a mobile network*[1]. Examples of mission-critical applications are those that require remote operation of equipment such as vehicles, surgical equipment, cranes etc. and applications monitoring the surrounding environment for unexpected events (e.g. monitoring for forest fires, road accidents, etc.).

In previous work, we have identified Quality of Service (QoS) differentiation, and complexity in management of heterogeneous applications that share the same network infrastructure, as key challenges that 5G operators will face in the IoT market [4]. We have proposed a high-level, logical architecture of a system for automated application lifecycle management. This architecture is based on automated allocation and deallocation of network resources to and from applications. These network resources include both cloud and mobile network (consisting of Core Network and Radio Access Network, or simply "Core and RAN"). We consider the aggregate of network resources allocated to support an application as a "network slice" and formally define the term as *a logical network serving a defined business purpose or customer, consisting of all required network resources configured together* [4].

In this paper, the authors present a realisation of the system envisioned in previous work. The contributions are as follows:

- An evolution of the previous logical architecture into a more detailed functional architecture. The architecture supports current technologies, but can also be used with 5G technologies as they become available.
- Implementation of a system for automated lifecycle management of applications, designed to support Development and Operations (DevOps) environments. The system automatically creates, monitors and eventually decomissions network slices for applications.
- Evaluation of the implemented system. Results show that the system can concurrently manage best-effort and mission-critical, QoS-demanding applications throughout their lifetime and without human supervision.

Paper Outline: Section II presents related work. Section III describes the architecture of the system and our implementation. Evaluation results of the implemented system are presented in Section IV, and finally, Section V concludes the paper with an overview of the key points presented and a

[1]Note that for some IoT applications, cellular connectivity may not be applicable or needed, e.g. for those deployed in remote locations and covering small geographical areas with low-power, short-range networks [3].

description of future work.

II. Related Work

A. DevOps Introduction and Relevance to IoT

DevOps promotes an agile relationship between software developers and mobile network operators [5]. In contradiction to traditional software development, where software is developed in isolation from its operational environment, DevOps includes operational requirements in the development process. DevOps in context of IoT application development is challenging, due to the complex operational requirements that include interoperability between cloud infrastructure, mobile network and devices. To achieve effective DevOps in IoT, a platform that can flexibly allocate, deallocate network resources for repeated, automated application redeployment is essential. Central to DevOps concept is application lifecycle management. As DevOps is an iterative process, it is likely that applications will be redeployed repeatedly both externally for use and feedback from users but also internally for testing by the development team. To achieve low lead times, shorten internal and external feedback loops and ensure quality testing and user feedback, automation and QoS in application lifecycle management are essential [6].

B. DevOps and Resource Allocation

Systems supporting DevOps software development have been well-explored, particularly through use of cloud computing and "as-a-service" models [7]. However, work on use of cloud computing for Core and RAN resource allocation has only recently started. In Core network, Network Functions Virtualisation (NFV), which virtualises core network functions so they can run on top of third-party infrastructure, has been defined by the European Standards Telecommunication Institute (ETSI) [8], and prototypes are currently being developed [9]. In RAN, approaches such as Cloud-RAN (C-RAN) [10] and RAN as a Service (RANaaS) [11], are based on virtualising part of, or complete baseband signal processing now locally done at base stations with dedicated hardware. In both cases, high data transfer rates from the antenna to the cluster where the processing takes place are required.

While our architecture as presented in section III does not exclude the use of any of the aforementioned technologies for Core and RAN resource allocation, our suggested implementation currently focuses on end-to-end QoS. As such the novelty lies in orchestrating cloud resources (in the form of Virtual Machines), as well as RAN and Core resources (in the form of policy functions to guarantee QoS). We expect to integrate Core and RAN virtualisation in subsequent years, as technologies become available. The rest of this section presents related work in QoS for Core and RAN, that forms the foundation of our work.

C. Quality of Service for Mobile Networks

5G standardisation work in third generation partnership project started in early 2016, and first results are expected to be part of "Release 14", with 5G standards not expected before the end of this release, currently tentatively set to June 2017 [12]. However, even if 5G standards are not yet available, QoS specifications already exist as part of current, fourth generation mobile networks (4G)[2] standards.

4G is designed as as an All-IP Network (AIPN) in order to provide mobile subscribers with a range of IP-based services, many of which have QoS requirements [13]. *Bearers* are logical transmission paths on top of physical network, which channel data traffic through the Core and RAN and are used for QoS control[14]. In this study we focus on the Evolved Packet System (EPS) bearer, which tunnels traffic from the mobile device (also known as "User Equipment", or "UE" in LTE terminology), through the RAN to the core network's Packed Data Network (PDN) Gateway (PGW). Bearers are parametrised by Quality of Service Class Identifiers (QCIs), which contain a set of QoS characteristics [15]. Table I shows the 3 QCI classes we have used in our paper[3].

TABLE I
QCIs used in this paper

QCI	Resource Type	Priority Level	Packet Delay Budget	Packet Error Loss Rate
4	GBR	5	300 ms	10^{-6}
5	Non-GBR	1	100 ms	10^{-2}
9	Non-GBR	9	300 ms	10^{-6}

According to the specification, applications using a QCI class with Guaranteed Bit Rate (GBR) can assume that congestion-related packet drops will not occur as long as the data transmission rate is smaller or equal to the GBR [15]. Priority indicates the sequence in which the Core and RAN data traffic schedulers will meet the Packed Delay Budget (PDB) of an application. Highest priority is inversely related to the number assigned, therefore QCI 9 has the lowest priority of all classes and is the one used by bearers of "best effort" applications.

III. System Overview

A. Introduction: Core Concepts and Terminology

We have implemented a prototype system for automated lifecycle management of applications. This process of lifecycle management includes allocation and deallocation of network resources based on an application description document. An "application description document" contains a *machine-readable description of the QoS requirements of the application as well as the software to be deployed on the cloud infrastructure and the mobile devices* (also known as User Equipment or UE). The concept of "network resources" encapsulates *network infrastructure resources, including cloud, core and RAN and transport resources*, as a cellular radio

[2]Note that "4G" includes the "Long Term Evolution" (LTE) and LTE-Advanced (LTE-A) technologies.

[3]Note that as of Release 13, there exist 13 QCI classes in total (see table 6.1.7 of [15]). Even though for our experiments we chose 3 QCIs to highlight the difference between applications requiring QoS and those that use best-effort traffic, supporting more QCI classes in our implementation is straightforward.

network would be required for communication between UEs and the part of the application instance running in the cloud and vice versa (see also application description in section I).

"Applications" in the context of this paper, include both software deployed in the cloud infrastructure, but also software deployed locally in mobile devices. We therefore view mobile devices as part of the application. This concept bodes well with IoT, where many applications are expected to use both centrally-deployed cloud software, but also have software on the devices (i.e. on the edge). An example of this class of application can be found in Intelligent Transport Systems (ITS), where applications include vehicles making autonomous operational decisions and centralised software making strategic decisions and sending this information to vehicles through cellular network infrastructure (e.g. fleet management). We also consider "application instances" as applications that are deployed in the network infrastructure, following allocation of network resources, according to the requirements set by application description documents.

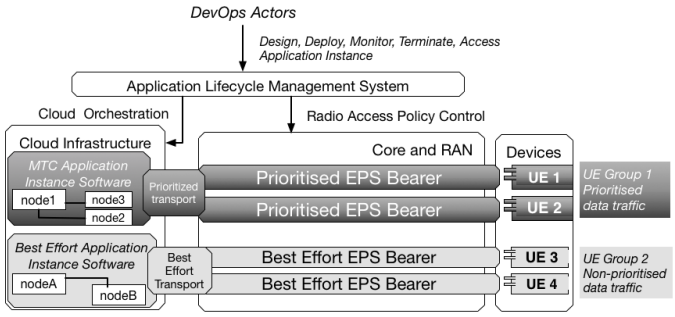

Fig. 1. This figure shows the concept of the application lifecycle management function, which creates new network slice instances using transport, radio access and cloud network resources. The flexibility of this function to manage application lifecycle, makes it ideal for use in DevOps environments, where short feedback loops between development and operation of applications is a requirement. This example shows two deployed applications, a mission-critical "Machine Type Communication" (MTC) application instance and a best-effort mobile broadband application instance. Data traffic between the software part of the mission-critical application instance which is hosted in the cloud (e.g. as a topology of Virtual Machines or containers) and UE1, UE2 is prioritized over data traffic between UE3, UE4 and the cloud-hosted software of the best-effort "Mobile Broadband" (MBB) application. More applications can be deployed, application instances can be monitored accessed and eventually terminated by using APIs offered by the system.

Figure 1 illustrates the aforementioned concept in greater detail. We introduce an "Application Lifecycle Management System", which deploys, monitors, provides access to and decommissions application instances. The "southbound" interface towards the network infrastructure, orchestrates cloud resources and includes a policy control function in the Core and RAN to trigger creation of EPS bearers between the UEs and the Core network endpoint. The "northbound" interface exposes a set of Application Program Interfaces (APIs) to DevOps actors for application lifecycle management operations. Different APIs are accessed by different actors. For example, developers may use the application design API to create a new application description document, whereas Quality Assurance (QA) "acceptance" testers and IT operators may use APIs for deploying applications and monitoring their performance. Finally customers can use access APIs to use and/or evaluate an application instance.

B. Architecture

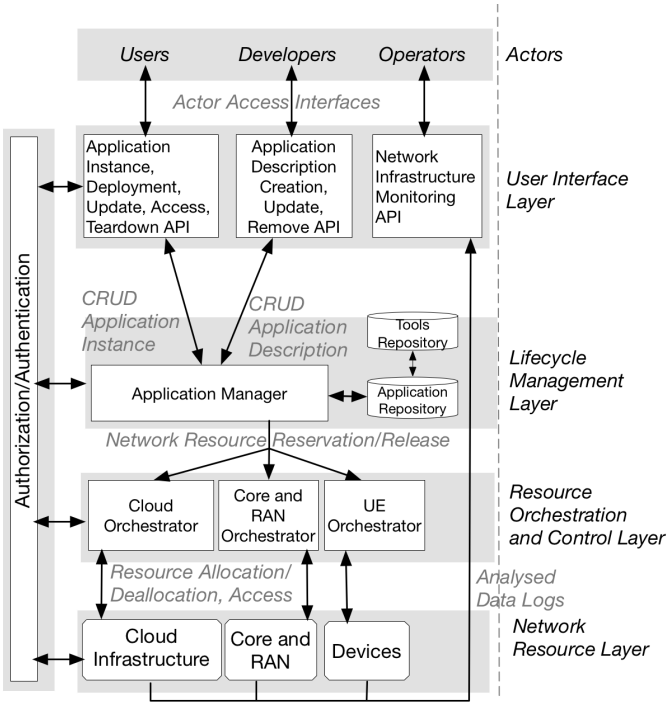

Fig. 2. Functional architecture of the proposed system, see section III-B for a detailed description.

Figure 2 illustrates the block components of the system architecture. Based on authors' previous work, the components are logically separated in five layers [4].

The "User Interface Layer" contains a set of APIs that are used by external actors and expose functionality for application management (deployment of applications and monitoring, decommissioning and access of application instances), application design and monitoring of system operation. They can interface with actors directly, or through external systems such as Graphical User Interfaces (GUIs).

We define three actors for our system. The *Developers* role refers to application developers. Note that the system does not offer integrated development environments (IDEs) for the developers to write their applications on, but instead, developers are required to provide an "application description document", which contains information about how to build an application[4], as well as this application's QoS requirements from the network infrastructure (see section III-A).

[4]Note that the building process of an application during its deployment may reuse software components already present in the system, for example databases or application servers. A reusable repository of software, or "tools" in the context of this paper, can save considerable amounts of time during deployment, as tools that are part of the application are retrieved from local repositories, instead of developers having to provide this software from external sources.

The "Application Description, Creation, Update and Remove" component offers CRUD (*Create*, *Read*, *Update* and *Delete*) functionality to developers to create their own application descriptions.

The *User* role uses the "Application Instance Access, Deployment and Teardown" API to CRUD application instances. The API also offers access functionality to running application instances. Finally, the *Operator* role monitors the operation of each application instance in real time, and issues alerts when a violation of the QoS requirements of an application service is detected[5].

Note that the distinction between actor roles is logical, and multiple roles may be assumed by the same physical entity. For example DevOps teams have both developer and operator roles. A customer may have the user role, and can create application instances of the latest build developers release through the developer role.

The "Lifecycle Management" layer consists of the "Tools Repository" component, which stores reusable software components. These components provide generic functionality such as storage (e.g. relational, document, graph databases), networking (e.g. virtual routers, firewalls, traffic shapers), billing, etc. and are referenced from *application description documents* (see section III-A) stored in the "Application Repository". The role of the "Application Manager" component is threefold.

- Application instance access: Provides secure access to running application instances.
- Storage of applications and tools: Provides a secure interface for Developers to register new application description documents or update/remove existing ones. In addition, developers can add new reusable tools that can be referenced from other application description documents.
- Application instance lifecycle management: Given an application description document, a user action (create, update, delete) and user action parameters, it creates a resource allocation plan and distributes it to the orchestrator components in the network resource orchestration layer below.

The "Resource Orchestration and Control Layer" contains a number of network resource orchestrators. Network resources include cloud, Core and RAN as well as UE. Each orchestrator can allocate, provide access to and deallocate network resources. The nature of resource allocation/deallocation is different depending on the type of network resource[6]. The "Network Resource Layer" contains the physical network resources.

Finally, the "Authorisation and Authentication" (A/A) layer provides cross-layer security for all 4 aforementioned layers. This layer interfaces with a user directory provides for user role management, user authentication, as well as user and software component action authentication. Note that the user directory may be internal to the component, or it can also be external (for example company-wide corporate directories are commonplace and already present in large organizations).

[5]Depending on the type of the error, recovery may be automated, e.g. by use of "Application Update API" to allocate more resources to the application instance, or it may require manual intervention.

[6]As an example, as discussed in section II, in our implementation allocation/deallocation of resources in the cloud means creation/deletion of virtual machines or containers to host application software. On the contrary, allocation/deallocation of resources in core and access network currently translates to establishment/teardown of EPS bearers with different priority.

C. Implementation

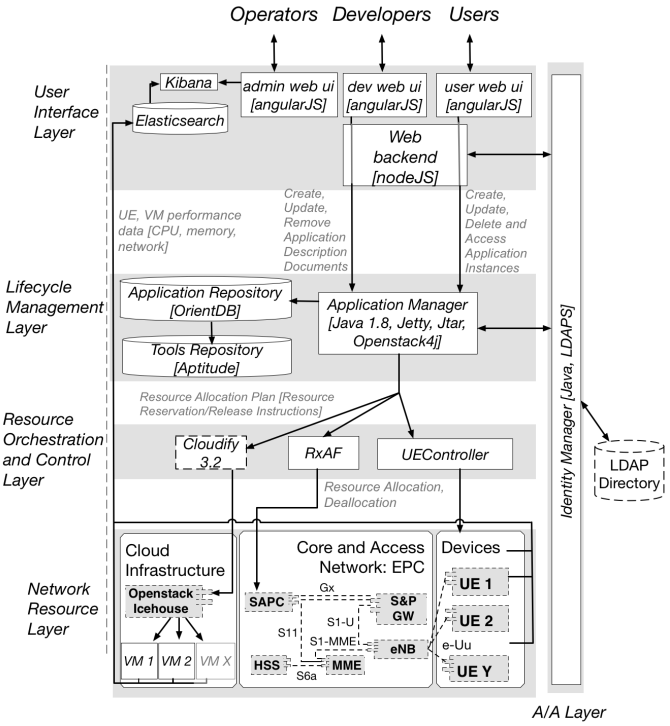

Fig. 3. Implementation of the proposed system. Note the the dashed interfaces (lines) and software components with dashed border illustrate third-party network resources that were not implemented by the authors, but were setup for testing the system. Bracketed text in component description shows the software used as basis for implementing the component.

Figure 3 shows the implemented system, deployed in Kista suburb of Stockholm, Sweden. The system, based on the architecture presented in section III-B, is a combination of open-source and commercial components. The rest of this section describes the implemented components from bottom to top layer.

The Core network is based on LTE standard and contains a set of nodes known as Evolved Packet Core (EPC)[7]. In addition to standard EPC nodes, the "Service-Aware Policy Controller" (SAPC) node is used for policy control of EPS bearers. The SAPC is Ericsson's implementation of the "Policy

[7]EPC has four basic nodes. The Home Subscriber Server (HSS), a database containing subscriber information, the Serving Gateway (S-GW), which transports data traffic between UEs and external networks, the Packed Data Network Gateway (PDN-GW), which connects the Core network to external networks and the Mobility Management Entity (MME), which deals with control-plane signalling (note that in figure 3 PDN-GW and S-GW are aggregated together as a S&P GW component). In the best interest of space, this paper will not describe functionality of these nodes in detail, but will instead refer interested readers to the corresponding 3GPP webpage [16].

and Charging Rules Function" (PCRF), which enforces QoS policies in LTE Core and RAN [17]. RAN is currently single-cell, operating in LTE Frequency Band 40 (2.3 - 2.4 GHz).

The cloud infrastructure, managed by Openstack version "IceHouse" [18], consists of 32 servers (348 cores and 3072GB of RAM in total), over a 10 GBps network infrastructure. This network connects the cloud infrastructure to the PDN-GW of the LTE core network. UEs are Raspberry PI computers with USB modems operating in LTE Frequency Band 40 [19]. In the current implementation, USB modems are pre-provisioned with SIM cards and can attach to the LTE network directly, however we have plans for remote provisioning of devices in the future (see section V).

Cloudify is used as the resource orchestrator for the Openstack-managed cloud infrastructure [20]. Cloudify interfaces with Openstack to deploy or decommission "topologies", i.e. formal descriptions specifying software nodes and their interconnections. Cloud topology descriptions, widely known as "blueprints" are expressed in TOSCA language[21]. For Core and RAN resource orchestration, we use a software component developed in-house, called Rx Application Function (RxAF). This component exposes a REST API to the application manager for triggering creation of EPS bearers, and uses the Rx interface of the SAPC node of the Core network for policy control of EPS bearers [17]. Finally, UEController uploads any software required from the application description document to the UEs. Similar to RxAF, UEController also exposes a REST API to the application manager.

The "Application Manager" is implemented in Java and is used to coordinate deployment of application instances, based on application description documents, which are also described in TOSCA [21]. Upon a request from a user to create an application instance with a given application description document, the "Application Manager" deconstructs this document into a set of EPS bearer creation requests towards the RxAF, a set of software upload requests towards the UEController, as well as a single request to deploy a blueprint describing the cloud software topology of the application, to the Cloudify orchestrator. Actions other than create (e.g. update or remove) also contain similar requests sent to the affected orchestrators. The application manager also accepts requests from developers for registering new application description documents, relaying those requests together with the descriptions to the "Application Repository", which is implemented using OrientDB graph database [22].

Information are collected by probes installed in the network infrastructure: Upon deployment of a new application instance, probes measuring performance of the cloud infrastructure as well as associated UEs are installed by their respective orchestrator. The probes consist of collectd [23] and logstash [24] daemons that measure the network, memory and CPU status and send these measurements for storage and visulization by the ELK stack [25].

Finally, the identity module is implemented in Java and uses Lightweight Directory Access Protocol (LDAP) [26] over Secure Sockets Layer (SSL)/Transport Layer Security(TLS) [27] to securely connect to an external Active Directory (AD), which is a directory service that contains user information such as username, password, user role and contact details. Based on the information contained in the AD, the identity module can authenticate users logging in from one of the user interfaces but also authorise application manager actions (see section III-B).

IV. Evaluation

A. Introduction

This section describes an evaluation of the implemented architecture, as described in section III. The goals of this section are twofold:

- To measure the impact of background load in the network infrastructure during deployment of a new application, identify potential bottlenecks in the deployment process, and suggest improvements. Additionally, to identify the time needed for a typical application to be deployed and decommissioned.
- To identify whether network QoS guarantees can be maintained for mission-critical application instances sharing network resources with mobile broadband application instances.

In the context of this study, we consider a typical application to consist of a number of connected devices, sending data to software deployed in the cloud. This software consists of a data aggregation entity, which collects data from all devices and stores them to a database. Concurrently, data from these devices are used to make a decision which is sent back to the device.

Based on the above description and as part of the evaluation process, we have implemented an "automotive application" for remote vehicle fleet management. This application involves vehicles with LTE connectivity sending status data to software in the cloud over the LTE RAN and EPC core network (see section III-C). This software aggregates received data, stores and uses this data in order to make decisions affecting the vehicles. These decisions are subsequently sent back to affected vehicles. A typical scenario for this application would be remote fleet management for a goods transport company. Vehicle status data such as location, speed, direction of travel and current vehicle capacity can be used in conjunction with data from goods available for pickup from various locations, to task vehicles to pickup goods from locations that are close to the direction of travel. The decision to pickup goods from a location could be done by cloud-hosted software, thus negating the need for manual vehicle dispatch from a human operator.

The application description document consists of two parts. The first part is a "blueprint", i.e. topology of software nodes and their interconnections, which is deployed in the cloud infrastructure. This topology consists of software components as well as a description of interfaces between these software components, and is described in OASIS TOSCA [21]. The second part of the document contains a description of the QoS required from this application, which in technical terms, is a

description of type of EPS bearers to be created in the core and access network (see section II-C). We have developed two different configurations of the application topology. The first configuration is "single-node", with cloud-hosted software that is deployed in one virtual machine (VM), and the second is "multi-node", consisting of four VMs (see figures 4 and 5).

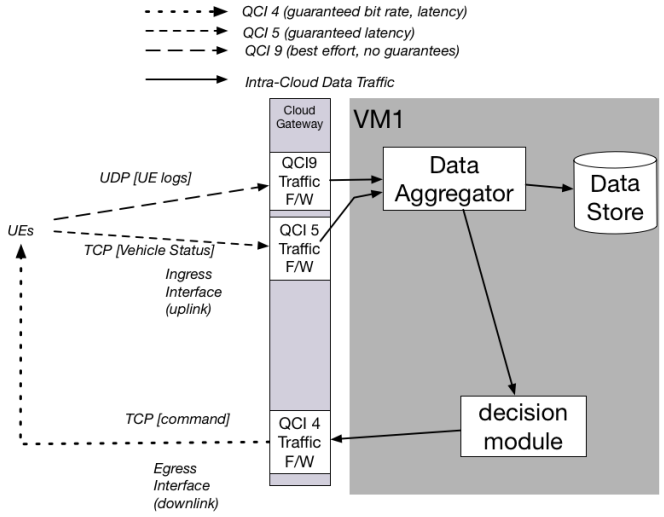

Fig. 4. Deployed automotive application instance, single-node configuration.

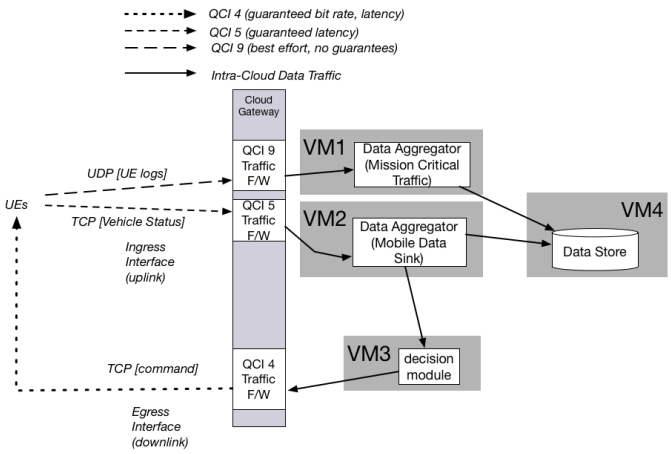

Fig. 5. Deployed automotive application instance, multi-node configurationration.

The advantage of the multi-node deployment is its scalability, as data aggregation, storage and decision functions can be isolated in dedicated VMs and even distributed across multiple VMs using load balancers. The disadvantage of such a deployment is the consumption of more cloud resources. The rest of this section presents the aforementioned two sets of measurements for both multi-node and single-node application configurations in greater detail.

In our experiments, we used two sets of data streams originating from a UE, one transmitting mission-critical TCP traffic, and one transmitting background UDP traffic. Both data streams were transmitted through the Core and RAN to the data aggregation component of the cloud service and subsequently stored in a database. A gateway forwarding data received from and sent to the UEs towards and from the cloud-hosted part of the application instance, was setup at the edge of the cloud.

Data from the TCP stream was used by a decision module to make a decision and send this decision back to the UE originating the TCP stream. Data traffic from the UEs, through the RAN and Core to the cloud used different bearers, depending whether the traffic was mission critical or not. For mission-critical data traffic we chose QCI 4, whereas for background traffic QCI 9. For sending decisions back to the UE originating mission-critical traffic we chose QCI 5. In this way, we prioritised mission-critical uplink and downlink traffic over background traffic (see also section II-C).

B. Application instance deployment and teardown measurements

In this section, we present measurements on application deployment and teardown time. In "continuous delivery" environments such as DevOps, automation and short lead-time between application deployment, use, teardown are of significant importance. As part of the goal for doing the measurements was to investigate application deployment and teardown times for different background load, we used "stress-ng" tool [28] to enforce artificial CPU use. We perform two sets of measurements using the single-node and multi-node application configuration, under different levels of CPU load.

The deployment of an application, includes deployment of software in the cloud, communication with SAPC and setup of EPS bearers in the Core and RAN, as well as deployment of software in the devices. In the case of our application as presented in section IV-A there was no software deployed on the device other than the traffic generators used for QoS measurements (see IV-C). Teardown of an application includes deletion of the virtual machines in the cloud, reset of all bearers to default (QCI 9) for all UEs, as well as removal of any software in UEs themselves. Table II shows application deployment and teardown measurements under different CPU load, for single-node and multi-node configurations. For every application deployment or teardown case, we did 5 experiments and computed the average. Standard deviation (σ) indicates small variations of time for all experiments.

In general, we have observed small fluctuations in both deployment and teardown times when comparing between cases of different background CPU load for both single-host and multi-host configurations. We traced the cause for these fluctuations to various external software package dependencies required during cloud software deployment. Those packages are downloaded from repositories hosted on the Internet, where there can be no network delay guarantees. Local software repositories contained within the cloud should reduce the observed time deltas, and we plan to investigate this in the future.

TABLE II
APPLICATION INSTANCE DEPLOYMENT AND TEARDOWN TIME

Scenario 1: Application Instance Deployment Measurements				
CPU Load	Single-node Scenario		Multi-node Scenario	
	Time (mm:ss)	σ (sec)	Time (mm:ss)	σ (sec)
No CPU Load	06:40	7.778	09:53	11.213
25% CPU Load	06:41	6.753	09:46	10.043
50% CPU Load	06:44	5.543	10:55	11.443
75% CPU Load	07:22	7.321	11:13	12.419
Scenario 2: Application Instance Teardown Measurements				
CPU Load	Single-node Scenario		Multi-node Scenario	
	Time (mm:ss)	σ (sec)	Time (mm:ss)	σ (sec)
No CPU Load	00:44	0.707	01:09	2.828
25% CPU Load	00:44	0.711	01:05	1,41
50% CPU Load	00:43	1.081	01:07	2.206
75% CPU Load	00:45	2.121	01:14	0.707

When comparing single-host and multi-host application configurations, we observe an increase in the amount of time both for application deployment and teardown. This is generally attributed to the additional time required by Openstack cloud resource management to reserve and release additional cloud resources in the case of multi-host configuration. On the other hand, the time for EPS bearer creation remains relatively same across all experiments, with small impact on the total deployment or teardown time (around 2 seconds).

C. QoS Measurements During Runtime

In this scenario we measure two basic network QoS parameters. The first, "end-to-end latency", measures latency from the moment a mission-critical request is transmitted to the cloud-hosted software, until the moment the UE receives a response. When the request is received in the cloud, a "decision" is made based on current and historical UEs status, and this decision is sent back to the UE that generated the request. The second measurement concerns throughput. Here, we investigate whether the guaranteed throughput of mission-critical traffic is unaffected from the background throughput generated from the best effort datastream.

For the purposes of these measurements, we created a series of scenarios where we increasingly loaded the "UE to cloud" uplink interface with background traffic. Given that we had limited number of UEs in our disposal, we limited the bandwidth of the RAN to 5 MHz, which gave us a total capacity of approximately 3.55 Mbps on the uplink interface. Therefore, we could easily load the network to the full of its capacity just by using one UE transmitting at 3.55 Mbps. For both end-to-end latency and throughput scenarios we used the single-node configuration of the application, as we focused on investigating QoS of the network links and the intra-cloud latency was less then 1ms for both configurations.

Figure 6 shows end-to-end latency measurements for different levels of background traffic. We observe that when the network is not overloaded, both QCIs can give good result; however when the network is overloaded (from 4 Mbps background data traffic), the data stream with best-effort bearer (QCI 9) shows significant increase of end-to-end delay while QCI 4 still gives good result. The reason is due to, as previously described, guaranteed delay budget for QCI 4 (see section II).

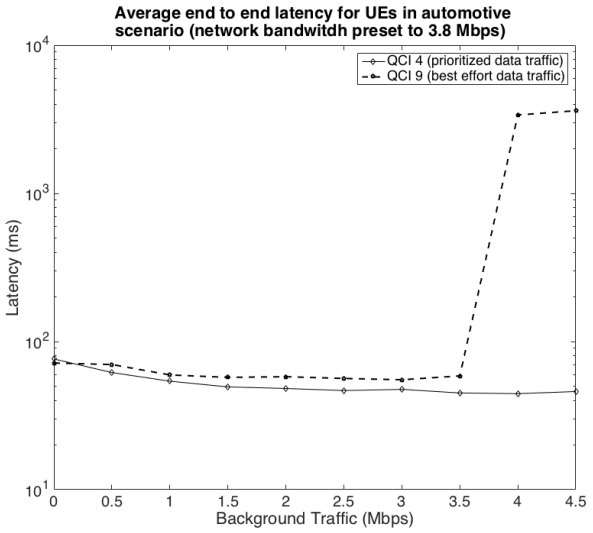

Fig. 6. End-to-end delay (latency) measurements.

Figure 7 shows throughput measurements conducted on the uplink interface of the single-node configuration (see figure 4). In this scenario, we used traffic generators to generate prioritised (QCI 4) and best-effort (QCI 9) traffic. In the "Data Aggregator" module, we installed traffic sinks and observed the data reception rate. We preset the transmission rate of prioritised data traffic to 1 Mbps, and experimented with different levels of background data traffic transmission rate (from 0 to 4.5 Mbps).

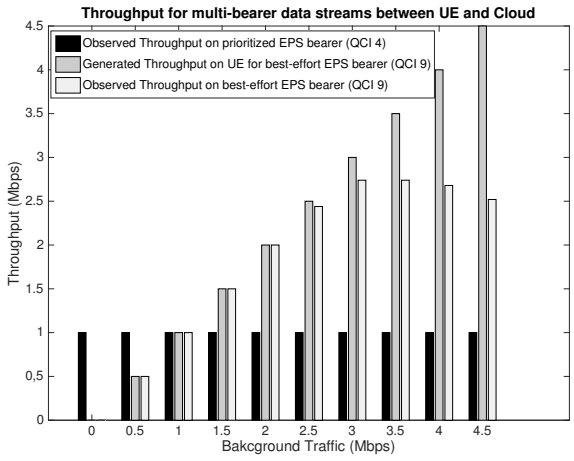

Fig. 7. Throughput measurements for multiple bearers on the uplink interface between UE and cloud gateway.

Given a network capacity of 3.55 Mbps, we observed that when the aggregate of generated throughput for both types of traffic exceeded this amount, Core and RAN would drop packets from best-effort data traffic, in order to still guarantee the 1Mbps of mission-critical traffic. This is evident in figure

7 from the point where background traffic is 2.5 Mbps or greater, wherein we see a mismatch between the generated background traffic at the source (UE) and the observable rate at the sink (cloud VM). In conclusion, we observed that for both cases of end-to-end latency and throughput, guarantees of network QoS for mission-critical application instances are kept, even to the expense of best-effort application instances.

V. Conclusions

In this paper, we have developed a system for supporting development and operation of IoT applications requiring cellular network access. The system automatically deploys, monitors and decommissions applications that include cloud software interacting with mobile devices over cellular network. We integrate a policy control mechanism on the RAN together with cloud orchestration, in order to provide end-to-end network QoS guarantees to mission-critical applications. In addition to presenting a functional architecture, we implement a prototype system and measure its performance.

We observed that the time for deployment and teardown of applications remained stable irrespective of network load. Additionally, during application operation, prioritised EPS bearers were able to guarantee network QoS for mission-critical applications. Finally, we identified dependencies of cloud software on external repositories as one of the points contributing the most to deployment time and is something we plan to address in the next iteration of the system.

Future plans include implementation of transport resource orchestration between the cloud and the core network, using the 3GPP standard for internetworking between Core/RAN and Fixed Broadband Access [29]. We also plan to measure application scalability to thousands of UEs, as the current set of measurements were conducted with the few UEs we had at our disposal. Finally, we intend to deploy and test more applications from automotive and other industry domains, that may have more diverse network resource and QoS requirements.

Moreover, it is interesting to challenge resource orchestration one step further from an edge computing perspective to benefit from computational availability and data proximity as previously studied in [30].

References

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A Survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[2] E. Dahlman, G. Mildh, S. Parkvall, J. Peisa, J. Sachs, Y. Selén, and J. Sköld, "5G Wireless Access: Requirements and Realization," *Communications Magazine, IEEE*, vol. 52, no. 12, pp. 42–47, December 2014.

[3] P. Pereira, J. Eliasson, R. Kyusakov, J. Delsing, A. Raayatinezhad, and M. Johansson, "Enabling Cloud Connectivity for Mobile Internet of Things Applications," in *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*, March 2013, pp. 518–526.

[4] R. Inam, A. Karapantelakis, K. Vandikas, L. Mokrushin, A. Feljan, and E. Fersman, "Towards automated service-oriented lifecycle management for 5G networks," in *Emerging Technologies Factory Automation (ETFA), 2015 IEEE 20th Conference on*, Sept 2015, pp. 1–8.

[5] DevOps Definition. Ernest Mueller, James Wickett, Karthik Gaekwad, and Peco Karayanev. [Online]. Available: http://theagileadmin.com/what-is-devops/

[6] J. Humble and M. Molesky, "Why Enterprises Must Adopt Devops to Enable Continuous Delivery," *Cutter IT Journal*, vol. 24, no. 8, pp. 6–12, August 2011.

[7] M. J. Kavis, *Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS)*. Hoboken, New Jersey: Wiley, 2014.

[8] ETSI. ETSI GS NFV 002 V1.2.1 (2014-12): Network Functions Virtualisation (NFV);Architectural Framework. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf

[9] ——. ETSI NFV Proofs of Concept. [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv/nfv-poc

[10] M. Hadzialic, B. Dosenovic, M. Dzaferagic, and J. Musovic, "Cloud-RAN: Innovative radio access network architecture," in *ELMAR, 2013 55th International Symposium*, Sept 2013, pp. 115–120.

[11] D. Sabella, P. Rost, Y. Sheng, E. Pateromichelakis, U. Salim, P. Guitton-Ouhamou, M. Di Girolamo, and G. Giuliani, "Ran as a service: Challenges of designing a flexible RAN architecture in a cloud-based heterogeneous mobile network," in *Future Network and Mobile Summit (FutureNetworkSummit), 2013*, July 2013, pp. 1–8.

[12] 3GPP. Release Timeline. [Online]. Available: http://www.3gpp.org/specifications/67-releases

[13] ——. Specification series 36: LTE (Evolved UTRA), LTE-Advanced, LTE-Advanced Pro radio technology. [Online]. Available: http://www.3gpp.org/DynaReport/36-series.htm

[14] ——. 3GPP TS 23.107 V13.0.0 (2015-12): 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Quality of Service (QoS) concept and architecture (release 13). [Online]. Available: http://www.3gpp.org/DynaReport/23107.htm

[15] ——. 3GPP TS 23.203 V13.6.0 (2015-12): Technical specification group services and system aspects; policy and charging control architecture (release 13). [Online]. Available: http://www.3gpp.org/DynaReport/23203.htm

[16] F. Firmin. The Evolved Packet Core. [Online]. Available: http://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core

[17] 3GPP. 3GPP TS 29.214 V13.4.0 (2015-12): Universal Mobile Telecommunications System (UMTS); Policy and charging control over Rx reference point. [Online]. Available: http://www.3gpp.org/DynaReport/29214.htm

[18] Openstack. Openstack. [Online]. Available: https://www.openstack.org

[19] Raspberry Pi homepage. Raspberry Pi Foundation. [Online]. Available: https://www.raspberrypi.org

[20] Cloudify. GigaSpaces Technologies. [Online]. Available: http://getcloudify.org

[21] (2013, November) Topology and orchestration specification for cloud applications version 1.0. OASIS. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html

[22] OrientDB homepage. OrientDB. [Online]. Available: http://orientdb.com/orientdb/

[23] Collectd - The system statistics collection daemon. Collectd. [Online]. Available: https://collectd.org

[24] Logstash homepage. Elastic. [Online]. Available: https://www.elastic.co/products/logstash

[25] The elastic stack — make sense of your data. elastic.co. [Online]. Available: https://www.elastic.co/products

[26] J. Sermersheim, "Lightweight Directory Access Protocol (LDAP): The Protocol," RFC 4511 (Proposed Standard), Internet Engineering Task Force, Jun. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4511.txt

[27] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), Internet Engineering Task Force, Aug. 2008, updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685. [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt

[28] Stress-ng utility. [Online]. Available: http://kernel.ubuntu.com/~cking/stress-ng/

[29] 3GPP. 3GPP TS 29.139 V13.0.0 (2015-12): Universal Mobile Telecommunications System (UMTS);LTE; 3GPP System - Fixed Broadband Access Network Interworking; Home (e)Node B - Security Gateway Interface (Release 13). [Online]. Available: http://www.3gpp.org/DynaReport/29139.htm

[30] A. M. Haubenwaller and K. Vandikas, "Computations on the edge in the internet of things," *Procedia Computer Science*, vol. 52, pp. 29–34, 2015.