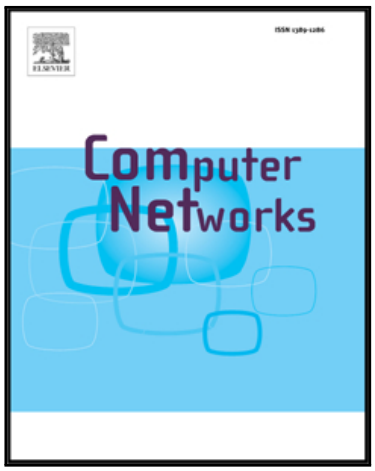

CDS-MEC: NFV/SDN-based application management for MEC in 5G Systems

E. Schiller, N. Nikaein, E. Kalogeiton, M. Gasparyan, T. Braun

PII: S1389-1286(18)30080-X
DOI: 10.1016/j.comnet.2018.02.013
Reference: COMPNW 6411

To appear in: *Computer Networks*

Received date: 6 August 2017
Revised date: 9 January 2018
Accepted date: 14 February 2018

Please cite this article as: E. Schiller, N. Nikaein, E. Kalogeiton, M. Gasparyan, T. Braun, CDS-MEC: NFV/SDN-based application management for MEC in 5G Systems, *Computer Networks* (2018), doi: 10.1016/j.comnet.2018.02.013

CDS-MEC: NFV/SDN-based application management for MEC in 5G Systems

E. Schiller[a,*], N. Nikaein[b], E. Kalogeiton[a], M. Gasparyan[a], T. Braun[a]

[a] *Communication and Distributed Systems (CDS), University of Bern, Neubrückstrasse 10, 3012 Bern, Switzerland*
[b] *Communication Systems Department, EURECOM, Campus SophiaTech, 450 Route des Chappes, 06410 Biot Sophia Antipolis, France*

Abstract

This paper presents and evaluates the first open-source Network Function Virtualization (NFV)/Software Defined Networking (SDN)-based Mobile Edge Computing (MEC) platform. Our platform solves the Mobile Edge (ME) management issues with respect to Application (App) provisioning and traffic management. First, the ME Apps are managed as Virtual Network Functions (VNFs) on top of the virtual environment through the Juju VNF Manager (VNFM). Second, we develop an SDN controller to manage traffic on the ME System. Third, unlike other relevant architectures of ME systems, we use the control plane (i.e., S1 interface) to derive appropriate states for traffic management. Finally, we evaluate our solution in two use-cases: ME caching and Information Centric (ICN)/Delay Tolerant (DTN) Public Safety communication (PS). The MEC caching framework displays improved user Quality of Experience, e.g., latency, in comparison to direct communication, while the PS solution provides a residual mean of communication for rescue teams, when the network core (EPC) and a Public Data Network (PDN) are unavailable.

1. Introduction

There is an on-going effort that will change the ecosystem of future mobile networks providing intelligence at the network edge. Mobile Edge Computing (MEC) [1, 2] will be used to provide computing and storage directly at or close to an evolved Node B (eNB). Due to MEC, content, services, and applications will greatly benefit from reduced delay of the network edge. MEC is also foreseen in 3GPP 5G networks as an important technological enabler towards new genres of applications that intelligently combine location, network conditions, and radio information to provide enriched services to end-users. Therefore, MEC will widely spread in the ecosystem of future 5G networks. MEC could be implemented using older management techniques (i.e., not Network Function Virtualization (NFV)/Software Defined Networking (SDN)-based), however, NFV/SDN will greatly improve flexibility and rapid building of services at the edge.

This work presents the architecture as well as implements and evaluates the performance of the CDS-MEC System[1]. The paper is organized in the following way. Sec. 2 discusses the related work on 3GPP networks and MEC. In Sec. 3, we describe architecture and implementation details of our MEC platform. Sec. 4 describes the architecture of the SDN controller. The performance of our architecture running selected ME Apps is illustrated in Sec. 5. Finally, we conclude in Sec. 6.

[1] CDS refers to the Communication and Distributed Systems Group of the University of Bern.

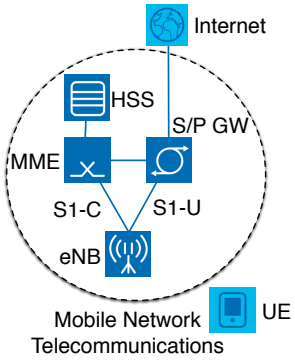

Figure 1: A simplified architecture of the LTE system.

2. Related Work

2.1. Mobile Network Telecommunications

In Fig. 1, we depict a simplified schematic of 4G Mobile Network Telecommunications. The LTE network is divided into the Evolved Packet Core (EPC) and the Radio Access Network (RAN). An eNB is a base station that provides a RAN towards end-users operating User Equipment (UE). The EPC contains a Home Subscriber Server (HSS), a Mobility Management Entity (MME), a Serving Gateway (SGW), and a Packet Data Network Gateway (PGW) [3]. The HSS is responsible for maintaining the user subscription information. The MME is a critical network function, which deals with the control plane. The SGW is responsible for handling user plane packets between the eNB and the PGW. The PGW is a user plane component, which forwards packets between the LTE network and packet networks (e.g., the Internet). In the remaining part of this paper, we refer to both PGW and SGW as Serving Packet Gateway (SPGW). Moreover, we will put a particular focus on the traffic management of the S1 (c.f., Fig. 1) interface between the eNB and MME in the control plane (S1-C) and the eNB and SPGW in the data plane, i.e., GPRS Tunnelling Protocol (S1-U).

2.2. SDN/NFV in Mobile Networks

In the EPC, the NFV concept solves flexibility and cost-efficiency problems through the on-demand instantiation of Virtual Network Functions (VNFs) [4], while SDN is mainly proposed for traffic optimizations [5, 6, 7, 8] in the core focusing on benefits including performance, scalability, interoperability, and flexibility.

There are several projects using the concept of NFV/SDN in Mobile Networks. Claudia[2] can instantiate services in private (i.e., OpenNebula, Eucalyptus, vSphere) and public clouds (Amazon, Flexiscale, etc.). The EU FP7 T-NOVA project [9] implements an orchestration platform for provisioning, configuration, monitoring, and optimization of Network Function-as-a-Service over virtualized infrastructures. The orchestration aspects covered by T-NOVA primarily include service chain mapping, service chaining and provisioning. In terms of service chaining, it employs SDN to install the forwarding state into the switches for traffic steering through the VNF chain. The EU H2020 SONATA project [10] also implements an orchestration and management framework, which allows both the service operator and the service developers to influence the deployment and placement of service chains on the physical infrastructure. SONATA supports a Development and Operations (DevOps) work-flow, which allows both developers and service operators to collaborate during the orchestration to optimize the design and deployment of the service. The EU FP7 UNIFY project [11] proposes an orchestration layer, which aims to achieve optimal placement of service chains on a physical infrastructure across different domains. The orchestration layer also provides an abstract and unified view of physical resources across different infrastructure providers to a service layer, through which customers can request a service. The EU FP7 Mobile Cloud Networking (MCN) project [12] provides a distributed orchestration layer consisting of a service manager (e.g., a RAN provider) and multiple service orchestrators per domain. The service manager provides an interface to the end customer to request a service from the corresponding domain. For each requested service chain, the service manager creates a service orchestrator, which configures, creates and deploys the service on the domain in-

[2] http://occi-wg.org/tag/claudia/

frastructure through its controller. At the standardization level, the ETSI NFV Industry Specification Group (ISG) is defining concepts, architectures, and interfaces for delivery and management of VNFs and their service chains. In ETSI NFV MANagement and Orchestration (MANO) [13], the NFV Orchestrator, in combination with the VNF Manager, is in charge of deploying the network services over the physical infrastructure as well as configuring and operating (scale-in/scale-out) the VNFs covering all the VNFs' life-cycles. In terms of software, several open source projects are addressing platforms for NFV Infrastructures and NFV MANO tools. Virtual Infrastructure Manager (VIM) and NFV Infrastructure (NFVI) are the current focus of the OPNFV[3] initiative, which has the goal to provide NFVI and VIM components, as well as their open Application Programming Interfaces (APIs). Other projects focus more on the management and orchestration functions of the NFV MANO architecture: OpenBaton[4], Open-O[5] and OpenSourceMANO (OSM)[6] provide open source software for NFV-Orchestration and generic Virtual Network Function Managers (VNFMs).

2.3. ME Systems

Roman et al. [14] compared MEC, fog computing, and cloudlet systems. A derivation of a conceptual architecture, spanning functionalities and interfaces for provisioning applications on MEC systems is derived in [15]. The ETSI MEC ISG provides an open standardization environment for the development of architectures for ME Systems. Initially, ETSI defined six application use-cases [1] for mobile edge systems. The work of ETSI concentrates on a top-down approach starting with MEC applications. The derivation of the ME Host architecture currently focuses on the management of application life-cycle through virtualization and appropriate management of the data plane [1, 2]. However, the reference points of the lowest level are not defined. A road-map for ME systems focusing on (power consumption, delay, bandwidth utilization, and scalability) with a careful study on application categorization is presented in [16]. For example, MEC services can help with MEC task offloading (e.g., the video encoding process), hence improving power consumption in mobile devices [17]. Moreover, as a complementary functionality in current and future networks, MEC may become an enabler for real-time context-aware applications combining MEC and RAN [18]. A tree-like mobile edge organization of a multi-tier cloud architecture was proposed in [19]. It allows an aggregation of the load across different tiers of cloud servers to maximize the mobile workloads being served. The work of [20] proposes and implements a MEC framework of ETSI and 3GPP compliance and focuses on the integration of LTE/LTE-A, MEC, and SDN. SDN is emerging as a natural solution for next generation cellular networks as it enables further network function virtualization opportunities and network programmability [21, 22]. In MEC, NFV and SDN will allow extreme flexibility, when it comes to the specification of extended logics of micro-service architectures at the network edge. The MEC function chain will be managed by the VNFM responsible for the instantiation of Virtual Network Functions (VNFs) and the SDN controller (e.g., OpenDayLight[7] with SDN-Switches) connecting elements all together [23]. Such a solution hides all the control-plane complexities of underlying resources from an end-user, requires the definition of appropriate hardware abstractions and communication protocols such as OpenFlow with OpenFlow eXtensible Match (OXM) on the south-bound interface [23], which automates rapid building of SDN/NFV-based function chains [24]. A top-level orchestrator providing an appropriate level of Quality of Service (QoS) will manage the SDN/NFV controllers through the north-bound API. To the best of our knowledge, however, an open-source NFV/SDN-based MEC platform has not been implemented yet. Moreover, the main idea behind this paper is a solution that allows for App provisioning at the edge that does not require additional signalling between

[3] https://www.opnfv.org
[4] https://openbaton.github.io
[5] https://www.open-o.org
[6] https://osm.etsi.org
[7] https://www.opendaylight.org/

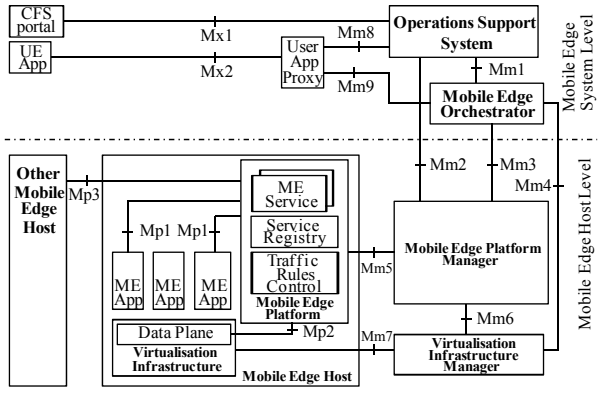

Figure 2: The architecture of the MEC system (from [2]).

the EPC and the ME System. We therefore rely on the existing S1 protocol to derive necessary states on the ME cloud and do not require any changes to the existing LTE architecture. This distinguishes our solution from other state of the art MEC architectures proposed to date [15, 20, 25].

3. MEC Architecture Specification

As illustrated in Fig. 2 [2], the ME system consists of the (upper) ME system level and the (lower) ME host level. The Customer Facing Service (CFS) for third parties and UE application portals are entry points towards the ME System. Roughly speaking, the portal allows third parties such as vertical providers or mobile users (UEs) to install Mobile Edge (ME) Apps on the ME Host (i.e., small cloud). The ME App receives traffic directly from the data plane from nearby eNBs by an appropriate traffic configuration. The platform is divided into separate inter-connected entities, which communicate through reference points defined between them (Mm1-9, Mp1-3, Mx1-2). The ME Host provides a ME platform and a virtualization infrastructure, which run and control ME Apps. From the perspective of ME Apps, the ME Platform uses the Mp1,2 reference points to provide:

- service discovery, registration, and communication, i.e., offering and consuming services (Mp1),

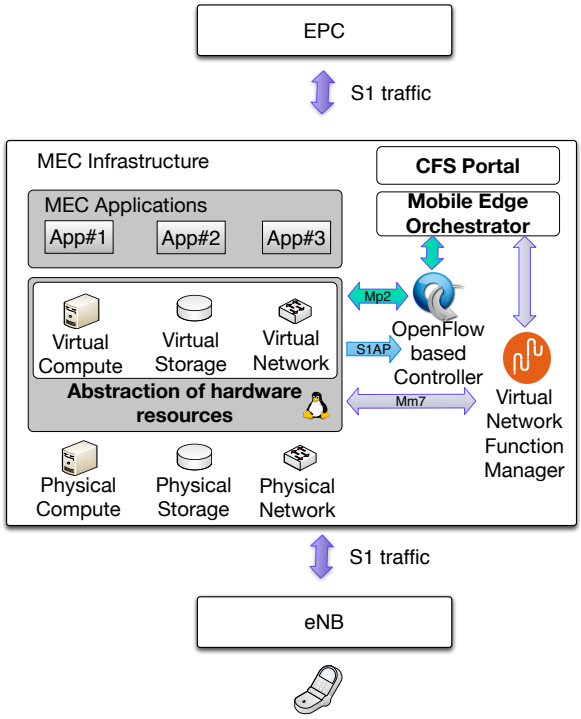

Figure 3: The architecture of the CDS-MEC system.

- data plane into the virtualized infrastructure of ME Apps (Mp2)

A user requests a new App through the portal (CFS, UE App). First, the request arrives at the Operations Support System (OSS). In turn, the OSS communicates with the Mobile Edge Orchestrator to manage the life-cycle of Apps. The orchestrator uses the Mobile Edge Platform Manager and VIM to appropriately configure the Mobile Edge Platform and Virtualization Infrastructure on the ME Host respectively. On the way from the CFS portal, the life-cycle management of Apps on ME Host is controlled by the Mx1 - Mm1 - Mm3 - Mm6 - Mm7 reference points, while the traffic rules providing the data plane to ME Apps are provided by the Mx1 - Mm1 - Mm3 - Mm5 - Mp2 reference reference points. For more details, please consult [2].

In Fig. 3, we present the CDS-MEC architecture integrated with the LTE infrastructure. We enrich the LTE ecosystem with a ME cloud residing close to the eNB. The idea behind this infrastructure is to allow for i) the instantiation of arbitrary ME Apps and ii) the response to UE requests from a close vicinity of the eNB. Our architecture of the MEC platform is closely related to ETSI MEC white-papers (c.f., Fig. 2 [1, 2]).

The ME cloud builds upon hardware resources composed of computing units equipped with CPUs, RAM, disks, and network adapters. In the case of sparse resources, one cloud server can build the entire ME micro-cloud, e.g., having an i7/Xeon CPU, RAM, disk, and one Intel dual-port 10 GbE-T card, on board. In such a configuration, ME micro-cloud is connected to the EPC through the first port and to the eNBs through the second port of the network interface. Hardware resources will be abstracted towards a VNFM, which automatically deploys ME Apps (i.e., VNFs) on the hardware infrastructure equipping VNFs with virtual compute, storage, and networking resources (the ETSI MEC Mm7 reference point). As the ETSI MEC Mp2 reference point providing data plane within the virtual resources of ME Apps, we develop an OpenFlow [26]-based controller and use the Open Virtual Switch (OVS) [27]. However, external SDN switches (i.e., not integrated with the ME cloud) can be used as well. The VNFM and controller will be managed by the CFS through a Mobile Edge Orchestrator. In this work, we did not focus on the development of the CFS and orchestrator, i.e., the VNFM and SDN controller are directly provided with information that should be derived by the CFS/orchestrator.

3.1. Virtual Network Function Manager

The main building block of our system is Juju developed by Canonical[8]. Juju provides a generic VNFM that can be adopted to heterogeneous environments such as Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) clouds (e.g., abstracted towards Juju through Ubuntu[9], OpenStack[10], etc.). It natively supports service provisioning and scaling functions for scale-in/scale-out scenarios. Therefore, it dynamically handles workloads by properly adjusting resources to momentary situations. Juju provisions various services provided as software on-demand. Services are described by charms, i.e., service manifests allowing for appropriate service configurations. Juju allows for "gluing" or "bundling" services all together by implementing logic allowing for automatic associations between services (i.e., service chaining). In the ETSI Management and Orchestration (MANO)[11] architecture, Juju should be classified as a VNFM of extended capabilities, helping MANO vendors to implement advanced business logic in the service orchestration part to support an enhanced Quality of Service (QoS) through contracting appropriate Service Level Agreements (SLAs). The charm store (i.e., a Juju charm repository) and Juju controller play the role of the VNFM, which allows us to spawn VNF bundles on the MEC infrastructure. The Juju service bundle could be connected with the help of the virtual switch [27] providing a virtual network.

In our architecture, the hardware resources are abstracted towards Juju through an Ubuntu Xenial system[12]. Juju VNFM automatically deploys ME Apps (i.e., VNFs) on the hardware infrastructure equipping VNFs with virtual compute, storage, and networking resources (implementing the ETSI MEC Mm7 reference point). As an example, Juju can automatically deploy a KVM[13]/LXD[14]-based caching service (e.g., squid[15]) that responds to user requests directly from the network edge (c.f., Fig. 4).

3.2. Management of the Traffic at the Network Edge

In the ME Platform, the data plane traffic management for ME function chaining should leverage

[8] https://www.ubuntu.com/cloud/juju

[9] https://www.ubuntu.com
[10] https://www.openstack.org
[11] http://osm.etsi.org
[12] http://releases.ubuntu.com/16.04
[13] https://www.linux-kvm.org
[14] https://linuxcontainers.org/lxd
[15] http://www.squid-cache.org

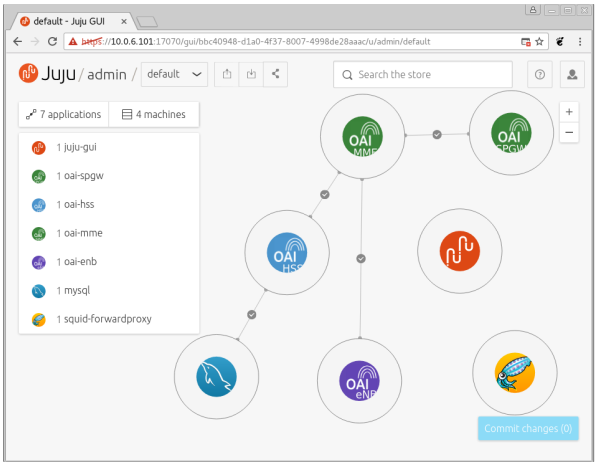

Figure 4: A screen-shot of the Juju Graphical User Interface (GUI), in which virtual elements, i.e., OpenAirInterface (OAI) [28] (LTE Network) and Squid Forward Proxy (App) are instantiated on the MEC-Cloud Infrastructure. This allows the UE attached to an OAI eNB to access the external web-server through a dynamically instantiated Squid forward proxy.

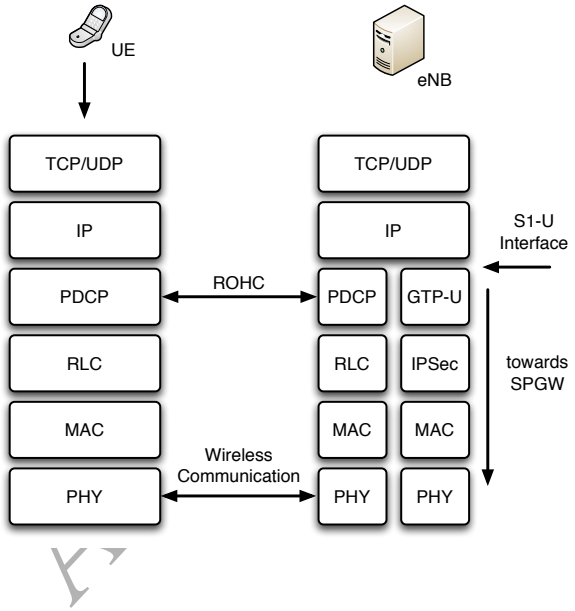

Figure 5: eNB Communication Diagram

SDN, which provides increased scalability and enhanced flexibility already demonstrated in LTE core networks (c.f., Sec. 2.2). An SDN switch providing networking in ME systems will manage the data plane according to a flow table. The flow table matches traffic and uses actions to redirect packets towards necessary Apps (i.e., VNFs).

As the 1st innovation, we provide a rationale for the necessary SDN functions that have to be standardized in OpenFlow and implemented by SDN switches to allow for SDN-based traffic management in ME Systems exchanging traffic between UEs and IP-based ME Apps not supporting the LTE stack.

3.2.1. Basic eNB Operation

An eNB provides radio access for UEs using the core network (c.f., Fig. 5). When a user generates traffic, the IP messages are being forwarded through an S1-U tunnel towards the SPGW. An S1-U message encapsulates a 32 bit Tunnel Endpoint Identifier (TEID), (e.g., 0x00000001) so that the traffic can be appropriately recognized at the EPC on a per-user level. Please notice that the downlink packets from the EPC to the UE traverse the stack in the opposite direction. In Fig. 6, we present S1-U messages exchanged between the eNB and the EPC. We also refer to the eNB through the Base-Band-Unit (BBU), which is the signal processing entity of the eNB. The TEIDs on the upstream and downstream differ. However, they are related as they belong to the same bearer with the same IP_{UE}. (c.f., Sec. 4).

3.2.2. Required SDN Actions

To redirect traffic towards Apps, the SDN switch has to be appropriately programmed by a Controller (c.f., Fig. 7). First, the flow tables have to intercept S1-U traffic from the eNB towards EPC. The encapsulated packets (c.f., Fig. 6, Inner IP Packet) should be provided towards a ME App. Second, the IP traffic from Apps towards a UEs should be provided as S1-U traffic (with an appropriate TEID) towards the eNB. In order to accomplish this goal, the SDN switch has to implement the following functions (please notice that the last two actions are already standardized in OpenFlow):

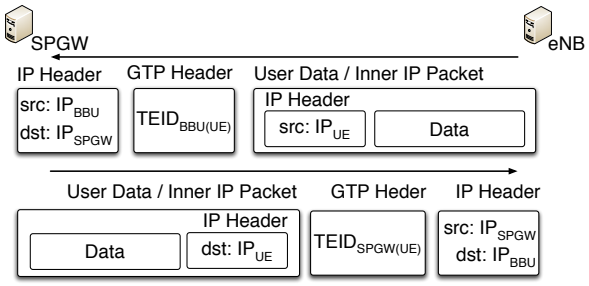

Figure 6: S1-U GTP packet description.

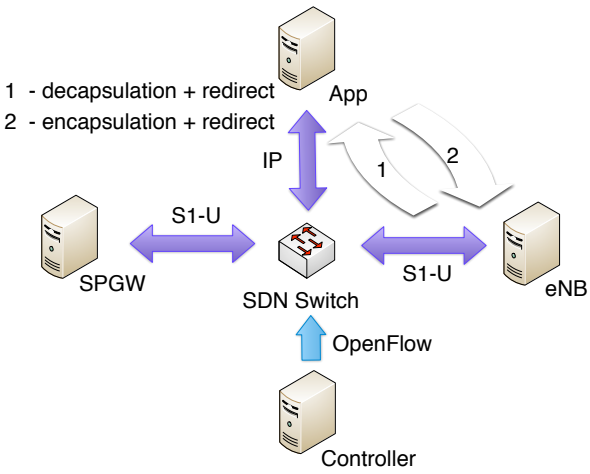

Figure 7: Basic SDN operations to redirect traffic between UEs and Apps.

- GTP decapsulation of S1-U, which strips off the IP/UDP/GTP header (c.f., Fig. 6) leaving the inner IP packet,
- GTP encapsulation of S1-U, which equips an IP packet with a S1-U GTP tunnel of an arbitrary TEID,
- MAC-based/IP-based packet modifications, which alter the destination addresses at the MAC and IP level,
- SDN port action, which sends the packet to a given output port.

Now, let us demonstrate the completeness of these SDN actions allowing for successful communication between UEs and ME Apps. Let us assume that a user is attached to an eNB. The eNB and EPC recognize the traffic of a given UE by a bearer (be it a default or dedicated bearer) consisting of an S1-U GTP tunnel using $\text{TEID}_{\text{SPGW(UE)}}$ on the downstream and $\text{TEID}_{\text{ENB(UE)}}$ on the upstream. On every bearer, the UE receives a different IP_{UE}. The UE, eNB, and SPGW are equipped with IP addresses IP_{UE}, IP_{eNB}, and IP_{SPGW}, respectively. When the UE originates an IP packet p of source IP_{UE}, it is encapsulated by the eNB as an inner data packet into a GTP tunnel, i.e., packet $P_{\text{TEID}_{\text{ENB(UE)}}}(p)$ with $\text{TEID}_{\text{ENB(UE)}}$. The packet is pushed from the eNB (i.e., IP_{eNB}) towards the SPGW (i.e., IP_{SPGW}) as illustrated in Fig 6 and Fig 7. On the way towards the SPGW, the SDN switch captures the packet, strips off the GTP header $p = \texttt{GTP decapsulation}(P_{\text{TEID}_{\text{ENB(UE)}}}(p))$, and provides p using a `Redirect(p)` function towards the ME App (c.f., Sec. 3.3.2, Sec. 4.2, Sec. 5.1, and Sec. 5.2). Note that a typical function redirecting packets towards a ME App could be materialized by modifying the link layer destination (i.e., the destination MAC address) and providing the packet towards an appropriate outgoing port on the switch (i.e., the SDN port action). The ME App recognizes the network layer source of the transmission as IP_{UE}. It then issues a downstream packet p' using the received IP_{UE} as the destination (c.f., Fig. 7). The packet goes through the SDN switch, which in turn intercepts packet p' from the ME App going towards the IP_{UE}. Such an IP packet p' cannot be delivered to the eNB directly. It has to be first tunnelled towards the eNB with an appropriate $\text{TEID}_{\text{SPGW(UE)}}$ by issuing a GTP packet $P'_{\text{TEID}_{\text{ENB(UE)}}} = \texttt{GTP Encapsulate}(p', \text{IP}_{\text{eNB}}, \text{TEID}_{\text{SPGW(UE)}})$. The `GTP Encapsulate` function is provided with the target endpoint, i.e., the eNB IP address and the SPGW TEID to appropriately encapsulate the traffic. Finally, the S1-U GTP packet arrives at the eNB, which recognizes the user using the SPGW TEID from the GTP header and delivers p' through the air interface to the UE.

3.2.3. SDN Matching

Packet matching is also an important property. Usually, SDN switches match packets based on various policies such as addresses at the various level of the IP stack and other important fields in packets. Due to the fact that the provider has to have flexibility in terms of allocating services to UEs, the SDN switch has to allow us to match packets based on the GTP-TEID. Moreover, after decapsulation, the SDN switch will allow us to inspect the fields of inner data packets to redirect various protocols towards appropriate MEC applications. As an example, a UE HTTP request (towards TCP port 80) could be redirected to the MEC HTTP cache server running squid[16], while an SSH packet (towards TCP port 22) can go directly to the SPGW.

3.3. SDN Switches & SDN Switch Modifications

The SDN switch is in the core of our architecture. As an example, OpenvSwitch (OVS) is a software-based, OpenFlow compatible switch developed by Nicira [27]. It allows for the on-demand establishment of virtual switches among Windows or Linux operating systems. On the north-bound interface, the switch uses OpenFlow to communicate with the controller. It supports various matching rules at different levels of the IP stack as well as many actions (e.g., modification of addresses; tunneling, encapsulation, decapsulation in GRE, VxLAN, etc.) that allow for advanced traffic engineering. OVS supports OpenFlow 1.1-1.4 protocols with OVS Extensible Flow Match (NXM) / OpenFlow eXtensible Match (OXM). We worked with OpenFlow 1.4 using OXM/NXM extensions to provide GTP matching rules (c.f., Sec. 3.2.3).

3.3.1. Necessary modifications to OVS

The 2nd innovation is the implementation of a micro-flow [27] GTP matcher[17] in the user and Linux kernel spaces. The OVS optimized packet forwarding consists in general of three techniques, i.e., user space packet matching, kernel space matching, and so called kernel space mega-flow [27], out of which we implement the first two. Moreover, we used the S1-U tunnelling patch allowing for the S1-U GTP decapsulation/encapsulation[18]. This provides enough functionality to support services running on MEC infrastructures with appropriate traffic handling as shown in Sec. 3.2.2.

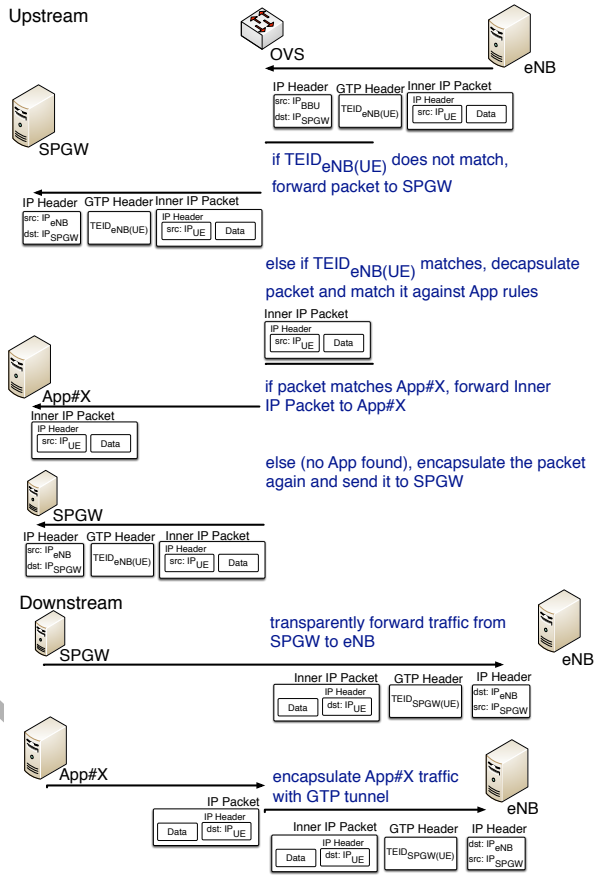

Figure 8: Processing of GTP packets by OVS.

[16] http://www.squid-cache.org
[17] https://github.com/ejschiller/FLEX/tree/master/ovs

3.3.2. Traffic Rules on the OVS Switch

The OVS switch distributes traffic among Apps using traffic characteristics (network protocols, transport protocols), the corresponding addresses (e.g., TCP Port 80), and flags (e.g., TCP_SYN). An example OVS rule will resemble:

- a GTP tunnel of $\text{TEID}_{\text{BBU(UE)}}$ carrying a TCP request from IP_{UE} towards port 80 goes through $\text{App\#2/IP}_{\text{App\#2}}$,
- every GTP tunnel (from all UEs) carrying a request towards port 80 goes through $\text{App\#3/IP}_{\text{App\#3}}$.

To distribute traffic among selected Apps and by default send traffic through the EPC, we have derived the following forwarding strategy (c.f., Fig. 8). On the upstream, a packet originated by a UE transparently goes to the SPGW if it does not match any of GTP rules on the OVS (e.g., TEID=0x00000001). However, when the packet TEID matches a GTP rule, the packet should target ME Apps on the ME Host. Therefore, the switch removes the GTP header and inserts the inner packet into the switch App flow tables for processing. If the UE inner packet matches the App rule#X (e.g., IP packet, TCP destination port 80 (HTTP)), it is redirected to a given App#X (e.g., a squid forward proxy). If no App rule matches, the packet is again encapsulated with the initial upstream TEID and goes towards the SPGW. On the downstream, the SPGW packets targeting the eNB go transparently through the OVS switch. However, packets returning from Apps (targeting a given UE based on the IP address) get encapsulated with a GTP header using an appropriate downstream $\text{TEID}_{\text{SPGW(UE)}}$.

Currently, a limited version of service function chaining is supported, i.e., UE $\rightarrow$ App $\rightarrow$ UE and UE $\rightarrow$ EPC $\rightarrow$ UE chain types are implemented. To support more complex chains having many Apps inside the chain, the forwarding Apps will have to carry the original source IP_{UE} in the header of forwarded packets. The flow rules will use IP_{UE} as a matching condition for traffic forwarding. For example, if traffic for a given IP_{UE}, has to go from App_{A} to App_{B}, a switch has to forward a packet with IP_{UE} from App_{A} towards the input port of App_{B} according to flow rules installed on the output port of App_{A}. Such a method will allow for multi App chains. We do not study, however, multi App chains in this paper.

[18] https://patchwork.ozlabs.org/patch/579431/

4. SDN-based Controller

This section describes an SDN-based controller that allows for the S1-U traffic distribution among ME Apps on both a per UE and App using the packet forwarding technique demonstrated in Sec. 3.3.2. The OpenFlow-based controller responds to user requirements through the CFS portal (c.f., Fig. 3). The CFS portal allows the user for the preparation of templates for traffic management. The user assigns a given App to all users attached to an eNB (provides the IP_{UE} wildcard) or only selected users based on specific IP_{UE}. The role of the controller is to derive traffic rules using appropriate $\text{TEID}_{\text{BBU(UE)}}$ on the upstream and $\text{TEID}_{\text{SPGW(UE)}}$ on the downstream matching the IP_{UE} of the UE. The relation is derived by a tracking module capturing the S1-C control plane between the EPC and eNB on the OVS switch. The module derives the bearer relation combining IP_{UE}, IP_{SPGW}, $\text{TEID}_{\text{eNB(UE)}}$, and $\text{TEID}_{\text{SPGW(UE)}}$. This information is eventually used for constructing SDN rules on the OVS switch (on the downstream and upstream) from templates provided by the user.

4.1. S1-C Tracking Module

The 3rd innovation is the passive monitoring of the control plane to derive the necessary parameters upon the UE attachment. We introduce a *tracker*, which is an auxiliary module of our architecture. Its role is to recognize the transmission bearer and associate the IP_{eNB}, IP_{SPGW}, upstream $\text{TEID}_{\text{eNB(UE)}}$, downstream $\text{TEID}_{\text{SPGW(UE)}}$ with the IP address of the user IP_{UE} (c.f., Fig. 9). Upon the UE attachment procedure, the tracker first processes the *Initial Context Setup Request* of the S1-C protocol exchanged between the MME and the eNB. The

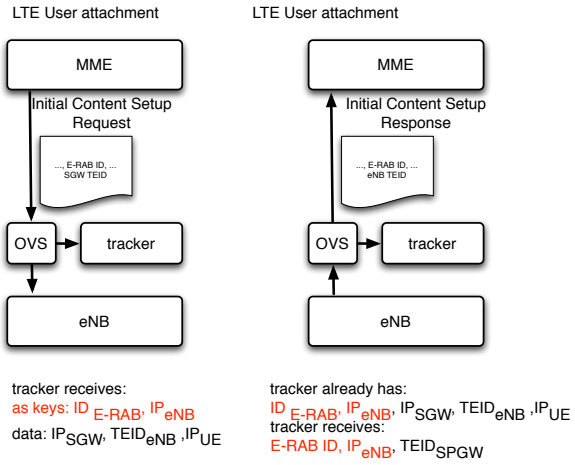

Figure 9: Tracking procedure.

request contains a per-user unique E-UTRAN Radio Access Bearer (E-RAB) that can serve as an information key distinguishing bearers in the database (db). The tracker retrieves, the E-RAB ID, IP_{SPGW}, IP_{eNB} (destination of the IP packet), TEID_{eNB}, and the user Packet Data Network (PDN) IP_{UE} from the message to keep in the database. Second, when the LTE *Initial Context Setup Response* arrives for the same bearer, the tracker can extend the cached data with the $\text{TEID}_{\text{SPGW}}$ from the message, as the E-RAB ID from the S1-C response is equal to the E-RAB ID from the S1-C request, which already serves as the database key. Therefore, the tracker needs to receive the Initial Content Setup Request and Initial Content Setup Response to derive the complete user information.

We present our tracking procedure in Algorithm 1. The procedure requires two input parameters, i.e., the port for traffic monitoring and the switch for SDN-based management. The procedure collects traffic on a given port and fills out variables required for appropriate traffic management in the db structure. When the data is ready, meaning that $\text{db}[\text{p.IP}_{\text{eNB}}][\text{p.ID}_{\text{E-RAB}}]$ is fully populated with IP_{eNB}, IP_{SPGW}, TEID_{eNB}, $\text{TEID}_{\text{SPGW}}$, IP_{UE}, the

Algorithm 1 Tracking algorithm

1: $\text{db}[\,][\,] = \emptyset$
2: **procedure** TRACKING(interface, switch)
3: **while** $p \leftarrow$ getPkt(interface) **do**
4: **if** p is Initial Context Setup Req. **then**
5: delete-ovs-rules(switch, $\text{db}[\text{p.IP}_{\text{dst}}][\text{p.ID}_{\text{E-RAB}}]$)
6: delete $\text{db}[\text{p.IP}_{\text{dst}}][\text{p.ID}_{\text{E-RAB}}]$
7: $\text{db}[\text{p.IP}_{\text{dst}}][\text{p.ID}_{\text{E-RAB}}].\text{IP}_{\text{eNB}} \leftarrow \text{p.IP}_{\text{dst}}$
8: $\text{db}[\text{p.IP}_{\text{dst}}][\text{p.ID}_{\text{E-RAB}}].\text{IP}_{\text{SPGW}} \leftarrow \text{p.IP}_{\text{SPGW}}$
9: $\text{db}[\text{p.IP}_{\text{dst}}][\text{p.ID}_{\text{E-RAB}}].\text{TEID}_{\text{eNB}} \leftarrow \text{p.TEID}_{\text{eNB}}$
10: $\text{db}[\text{p.IP}_{\text{dst}}][\text{p.ID}_{\text{E-RAB}}].\text{IP}_{\text{UE}} \leftarrow \text{p.PDN.IP}_{\text{UE}}$
11: **else if** p is Initial Context Setup Resp. **then**
12: $\text{db}[\text{p.IP}_{\text{src}}][\text{p.ID}_{\text{E-RAB}}].\text{TEID}_{\text{SPGW}} \leftarrow \text{p.TEID}_{\text{SPGW}}$
13: **if** $\text{db}[\text{p.IP}_{\text{dst}}][\text{p.ID}_{\text{E-RAB}}]$ is full **then**
14: install-ovs-rules(switch, $\text{db}[\text{p.IP}_{\text{dst}}][\text{p.ID}_{\text{E-RAB}}]$)
15: **end if**
16: **else if** p is S1-U **then**
17: no operation
18: **end if**
19: **end while**
20: **end procedure**

procedure executes the install-ovs-rules function to translate user templates into actual rules. Currently, the rule (i.e., state) is stored on the switch, until the EPC creates a bearer with the same E-RAB/IP_{UE}. When such a new bearer appears in the system, the old rules are recycled. Notice that we assume that distinct bearers carry different IP_{UE} addresses. MME or X2 handovers in the case of UE mobility are left for future work.

This is a novel method of reusing the S1-C to manage traffic rules on the ME Platform. Previous works [2] only acknowledge the distribution of the data plane among Apps, but the exact method is not discussed in the literature. User space capturing of the S1-C protocol between the MME and eNB is computationally inexpensive, as it resembles an ordinary S1-C protocol handling of a typical eNB. The expensive part is the user space capturing of S1-U packets

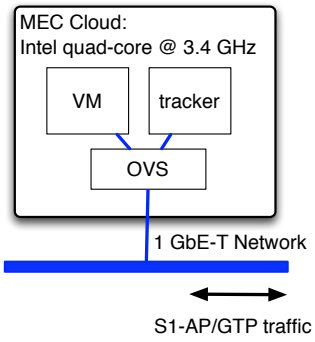

Figure 10: Traffic switching on the ME cloud.

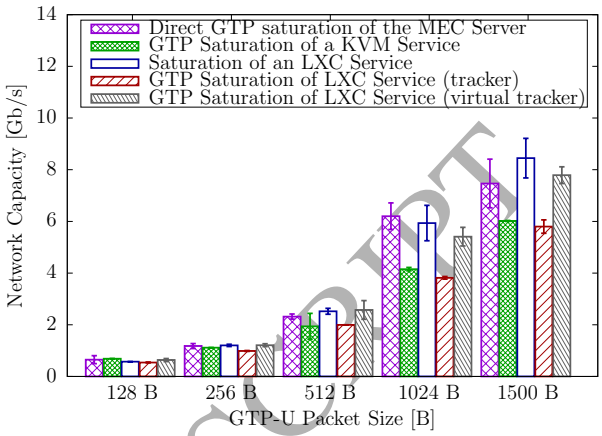

Figure 11: Performance of traffic switching on ME clouds.

going between the eNB and the EPC that can be received together with S1-C (i.e, no operation for heavy traffic) by the tracking module. We, however, do not require any S1-U packets in our monitoring operation. The further optimization of the tracking procedure uses the SDN capabilities of the switch. We install appropriate rules on the switch to filter out the S1-U communication from the traffic captured by the user space tracker. Please notice that the S1-C packet copying (i.e., to the monitoring interface) is easily implementable in SDN/OVS by the application of the *output* action providing the tracker with traffic. Generally, the S1-C traffic can be provided towards the tracker through the Out-Of-Band (OOB) or regular interfaces.

We performed the experiment illustrated in Fig. 10. We connect the ME cloud to a S1-C/S1-U traffic source (i.e., eNB) through a 10 Gigabit Ethernet port (i.e., the Intel 10 GbE-T X540 NIC). The most significant traffic is S1-U. First, we saturate the network from the eNB with S1-U traffic of different packet sizes between 128 and 1500 bytes. There are 5 situations considered: i) The S1 traffic targets the IP address of the MEC Cloud directly, ii) the S1 traffic targets the IP address of the LXC-based ME Service instantiated on the MEC Cloud, iii) the S1 traffic targets the IP address of the KVM-based ME Service instantiated on the MEC Cloud, iv) the S1 traffic targets the LXC-based ME Service and the tracker captures the whole S1 traffic on the switch, and v) the S1 traffic targets the LXC-based ME Service, but the (virtual) tracker is only provided with S1-C traffic due to an appropriate configuration of the OVS, i.e., the separation of the control and data plane on the ME cloud.

S1-U traffic is handled by the underlying UDP protocol. The UDP implementation on Linux is slow and does not saturate the link as easily as a regular TCP connection[19,20]. TCP displays a regular throughput of 9.3 Gbps when targeting the cloud server directly, while we reach 7-9 Gbps with huge performance variations with S1-U/UDP in an similar situation (c.f., Fig. 11). This is due to the fact that the RX queue processing of the 10 GbE-T network adapter fully saturates the CPU thread in our experiment. For protecting UDP packets against out of order delivery, Linux assigns a single CPU thread for processing a given UDP stream based only on network and transport protocol addresses. Therefore, S1-U traffic between eNB and ME cloud is recognized as a single UDP stream, even if it carries traffic from a large number of distinct UEs. Moreover, we notice that LXC ME Service displays better throughput

[19] https://blog.cloudflare.com/how-to-receive-a-million-packets

[20] https://fasterdata.es.net/network-tuning/udp-tuning

than KVM-based Services (even with multi-queue TUN/TAP interfaces) due to virtualization overhead. Performance-wise, LXC behaves nearly as good as the physical infrastructure. Furthermore, a new problem arises, when the tracker (implemented using the python pcapy[21] capture library) starts capturing an interface with both the S1-C and S1-U traffic between the EPC and eNB. Even though, S1-U is not required by the tracker, it can severely degrade performance as it quickly saturates the CPU thread (c.f., Fig. 11) and in consequence the forwarding capacity of the whole system. Notice that OVS is a software switch that also competes for computing resources (e.g., against the tracker) on the ME cloud. We, therefore, separated the control and data plane on the OVS switch, and provided the tracker with the control plane only. When the tracker does not receive S1-U packets, the original forwarding capacity restores.

4.2. User Template Handling

Listing 1: The OVS template for traffic redirections.

```
# MATCHING THE UPSTREAM PACKET OF A GIVEN UE (TEID
# BASED) AND REDIRECTING IT TO THE LOCAL GTP TUNNEL
# END-POINT USING THE mod_dl_dst MAC ADDRESS
# MODIFICATION AND output ACTIONS
ovs-ofctl add-flow OVS-SWITCH "in_port=$PORT_ENB,
ip,udp,tp_dst=2152,gtp_teid=$TEID_ENB,nw_dst=
$IP_SPGW,action=mod_dl_dst:$MAC_LOCAL,
mod_nw_dst=$IP_LOCAL,NORMAL"

# IMPLICIT DECAPSULATION OF THE GTP HEADER ON THE
# UPSTREAM RESUBMITTING PACKETS TO $PORT_ENB
ovs-ofctl add-flow OVS-SWITCH "tun_src=$IP_ENB,
tun_id=$TEID_ENB,action=mod_dl_dst:$MAC_APP,
resubmit:$PORT_ENB"

# REDIRECT A TCP PORT 80 (HTTP) PACKET TO A LOCAL CACHE
# USING THE mod_dl_dst MAC ADDRESS MODIFICATION AND
# output ACTIONS (APP#1)
ovs-ofctl add-flow ovs-br "in_port=$PORT_ENB,ip,tcp,
tp_dst=80,nw_src=$IP_UE,action=mod_dl_dst:$MAC_APP,
output:$PORT_APP"

# SEND REMAINING PACKETS (NOT MATCHING APP#1) TO THE
# SPGW
ovs-ofctl add-flow ovs-br "in_port=$PORT_ENB,nw_src=
$IP_UE,action=load:0->NXM_OF_IN_PORT[],set_tunnel:
$TEID_ENB,set_field:$IP_SPGW->tun_dst,output:
$GTP_TUN_PORT"

# ENCAPSULATING THE DOWNSTREAM PACKET FROM CACHE
# WITH THE APPROPRIATE SPGW TEID SENDING THE
# ENCAPSULATED PACKET TO THE ENB
ovs-ofctl add-flow OVS-SWITCH "in_port=$PORT_APP,
ip,nw_dst=$IP_UE,nw_src=$IP_APP,action=set_tunnel:
$TEID_SPGW,set_field:$IP_ENB->tun_dst,
output:$GTP_TUN_PORT"
```

In Listing 1, we provide an example Unix shell-based template to distribute traffic among the EPC and ME Apps in OVS. Please notice that the gtp_teid matcher (implemented by CDS) and GTP tunneling end-point (installed patch) are used in the script, but are not available in the default version of OVS (c.f., Sec. 3.3.1).

`OVS-SWITCH` is the OVS instance deployed on the ME Cloud, `$GTP_TUN_PORT` is the OpenFlow Port number of the local GTP tunneling end-point, `$PORT_APP` is the number of the OpenFlow Port connecting the switch with the App, `$PORT_ENB` is the number of the OpenFlow Port connecting the switch with the eNB, `$TEID_ENB` is the GTP tunnel endpoint identifier of the user on the eNB side, `$TEID_SPGW` is the GTP tunnel endpoint identifier of the user on the SPGW side, `$IP_APP` is the IP address of the App (e.g., web cache), `$IP_ENB` is the IP address of the eNB, `$IP_LOCAL` is the local IP address of the switch on the MEC Cloud, `$IP_UE` is the IP address of the UE, `$IP_SPGW` is the IP address of the SPGW, `$MAC_APP` is the MAC address of the App (e.g., web cache), and `$MAC_LOCAL` is the local MAC address of the switch on the MEC Cloud. Moreover, we assumed that the GTP tunnel between the eNB and SPGW is a UDP stream of source and destination port 2152.

Most of the variables are static (with respect to the UE attachment), i.e., `$GTP_TUN_PORT`, `$PORT_APP`, `$PORT_ENB`, `$IP_APP`, `$IP_ENB`, `$IP_LOCAL`, `$MAC_APP`, `$MAC_LOCAL`. As mentioned in Sec. 4.1 there are dynamic parameters established upon a UE attachment, i.e., `$TEID_ENB`, `$TEID_SPGW`, `$IP_UE`, and `$IP_SPGW`.

A user derives traffic redirection rules for App#X and fills out App related variables: `$PORT_APP`, `$IP_APP`, `$MAC_APP`. Then the template is submitted to the CFS portal. The portal provides the template to the ME Orchestrator, which in turn fills out all the cloud-deployment related parameters `$GTP_TUN_PORT`, `$PORT_ENB`, `$IP_ENB`, `$IP_LOCAL`, `$MAC_LOCAL` and sends the template down to the controller. The controller (tracker) monitors the S1-C traffic, and for all discovered UEs attached (IP_{UE} matching IP_{SPGW}, $TEID_{eNB}$, $TEID_{SPGW}$) substitutes corresponding parameters in the template (`$IP_UE`, `$IP_SPGW`, `$TEID_ENB`, `$TEID_SPGW`) and executes the derived OVS rules on the OVS switch. When the user specifies `$IP_UE` in the template, the controller will execute the rule for the specified

[21] https://pypi.python.org/pypi/pcapy

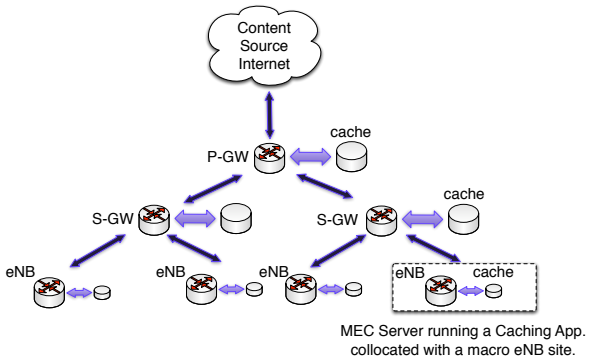

Figure 12: The LTE network architecture with caches at different levels.

`$IP_UE` only.

The current implementation integrates our tracking module (c.f., Sec. 4.1) with the shell-based controller that both automatically deploy necessary rules for discovered UEs on the underlying OVS switch. We do not, however, study scaling of the shell-based controller as the number of UEs attached with a given eNB/ME cloud will remain at a relatively small level of 1000 users per cell. Moreover, as we do not implement mega-flow optimizations in OVS [27], we cannot currently elaborate on scaling of switching performance with the increasing number of flow-rules installed on the switch. Finally, we do not consider any security measures in the installation of flow rules.

5. Use-cases

In this section, we will evaluate two important use-cases.

5.1. SDN/NFV based MEC Caching

As the 4th innovation, with the help of SDN/NFV, we enrich a hierarchical network architecture of LTE with persistent caching at the network edge as shown in Fig. 12 [29]. In a typical LTE network, traffic originated by users is forwarded in a hierarchical manner through eNBs, SGWs, PGWs that provide access to an external network (e.g., the Internet). Generally, every level of the LTE network can be equipped with a cache, which stores a fraction of cached content. This enables popular content to be stored at the network edge very close to the user.

5.1.1. Caching Benefits

The hierarchical organization allows us to provide popular content directly from edge caches, while less popular objects shall be forwarded further to the next level (i.e, SGW cache, PGW cache) of growing capacity (however, the placement of cached content is out of the scope of this paper). When the content is unpopular (i.e., not available from the caches), it will be directly accessed from the content source. However, caching of unpopular objects does not provide significant benefits.

This caching approach brings a multitude of advantages for end users and Mobile Network Operators (MNOs). For users, the QoE significantly increases due to lower access latency (also lower fluctuations of the object access time) and increased throughput. Moreover, the backhaul traffic is significantly reduced, allowing quicker access from distant content providers. It is envisioned to reduce the Operational Expenditures (OPEX) by 36% [30] due to the lower load of the core infrastructure. In 5G networks, the deployment of caches will be managed by NFV, i.e., when a new cache is required it will be dynamically instantiated as a Virtual Machine (VM). An SDN controller shall manage traffic on-demand to make use of appropriate caching instances at every level of the LTE network.

5.1.2. Software & Hardware Architecture

The fully open-source caching solution is presented in Fig. 13. We enrich a typical LTE architecture with the cloud node residing in between the eNB and the network core (EPC). The cloud node is controlled with the Juju VNFM. Moreover, we provide the cloud node with the OVS switch equipped with the tunnelling and matching patches. The web cache is instantiated as a VNF from the Juju controller. The remaining elements of the LTE infrastructure are OAI-MME, OAI-SPGW, and OAI-ENB implemented by EURECOM [31] providing the UE with access to the Internet. The eNB is configured to use

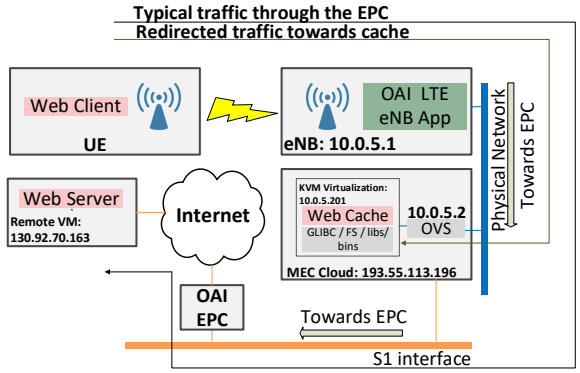

Figure 13: The implementation of the caching solution.

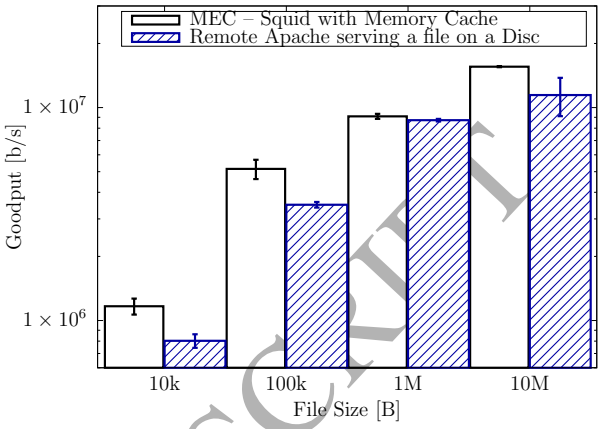

Figure 14: MEC Caching improves the QoS of content delivery in comparison to generic HTTP access.

5 MHz channels in Band7 of the LTE spectrum range in the Single Input Single Output (SISO) mode. The hardware used in the experiment is the following. The eNB is an Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz quad-core, 32 GB RAM, 1 TB HDD computer equipped with a USRP B210[22] Software Defined Radio (SDR). The MEC Cloud is an Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz quad-core, 16 GB RAM, 200 GB SDD computer. Both computers run the Ubuntu 16.04 (Xenial) operating system. They are also equipped with Intel X540T2 10 GbE-T interfaces. The UE uses a Huawei E392 dongle[23].

5.1.3. Results

The results are presented in Fig. 14. First, we run the experiment in which the cache is inactive. We therefore access the remote server of address 130.92.70.163 at the University of Bern being 15 hops away from EURECOM to directly retrieve the content from this location. Second, we activate the SDN controller, add necessary traffic templates, and actively store the content in the squid memory (i.e., we allow for cache memory hits, by allowing large objects of up to 10 MB to be stored in the memory). Since, the controller adds appropriate traffic rules for a given UE (transmission towards TCP port 80 goes through the squid App for all attached UEs), the objects can be directly retrieved from the cache. We measure the time required to download the content (from the cache and remote server), i.e., the time between issuing the HTTP request and retrieving the whole object. We then convert it into the UE experienced throughput by dividing the file size by the time required to download the object. The experiments are performed several times for every file size considered, i.e., 10 kB to 10 MB. The error bars are calculated as the file size divided by the average transmission time for a given file size squared multiplied by the standard deviation of the transmission time for that size (the following differential relation holds for the calculation of the error bar $\text{Goodput} = \frac{\text{filesize}}{t} \rightarrow |\sigma\text{Goodput}| = |\frac{\text{filesize}}{t^2}\sigma t|$, where t is the transmission time (c.f., Fig. 14)).

For all the file sizes, we noticed a significant improvement in the user QoE. The files are downloaded quicker when directly served from the cache. The improvement in the experienced capacity is about 30% for both big and small files (10 kB, 100 kB, 10 MB). However, sometimes due to momentary network conditions, the performance gain might not be so significant (e.g., 1MB).

[22] https://www.ettus.com/product/details/UB210-KIT
[23] http://m.huawei.com/enmobile/consumer20150301/press/news/hw-256115.htm

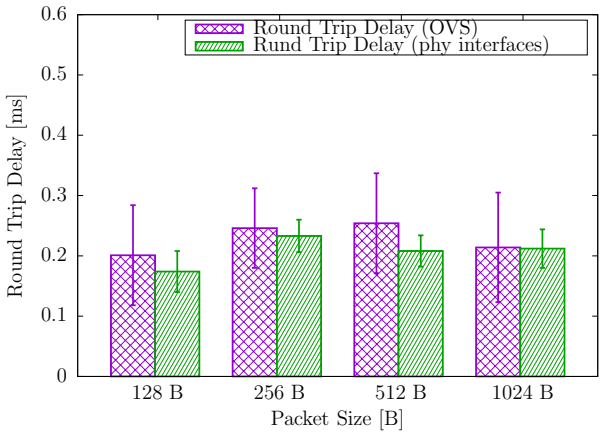

Figure 15: OVS delay for other applications.

We therefore clearly see the benefit of edge caches. However, other UE traffic (e.g., targeting the EPC) will have to go through an additional element: the OVS instantiated on the ME cloud. In Fig. 15, we measure the round trip delay of S1-U/UDP packets exchanged between the eNB and the MEC Cloud for different packet sizes. We observe that OVS does not provide significant overhead in terms of delay in comparison to the situation, when only physical interfaces (i.e., physical network adapters) are used. Therefore, other traffic will not experience significant performance degradation due to the deployment of OVS between the eNB and EPC.

5.2. SDN/ICN Public Safety Solutions

The 5th innovation is the Public Safety (PS) application for networks with a disconnected core.

5.2.1. DTN/ICN Benefits in Critical Situations

The utilization of Delay Tolerant Networks (DTNs) in disaster scenarios for Public Safety (PS) applications has been studied in literature. In [32], the authors propose a terrain discovery system using a DTN environment in a disaster scenario. Specifically, they use civilians that carry sensor nodes in their mobile devices and collect Data. Nodes use DTN to transfer data reaching computing nodes that perform a discovery of the affected area. In [33] the combination of DTN with the Cognitive Wireless Network (CWN) for disaster networks is proposed. Furthermore, Fajardo *et al.* [34] implemented a data collection method that uses people and their mobile phones as sensor nodes.

Tyson *et al.* [35] study the utilization of Information Centric Networks (ICNs) in disaster scenarios. The authors argue that ICN could improve connectivity resilience. This is due to the fact that in an ICN architecture, nodes can explore multiple interfaces at the same time. ICN does not have to maintain short connection timeouts as in classical networks. ICN requires no particular underlying network-layer, as it creates its own ad-hoc network. Deploying ICN in a network could improve QoS, as different requests could be treated differently. ICN supports the store, carry, and forward mechanism, as each node could be equipped with a cache, which is important in disaster scenarios, where connectivity may momentarily disappear. We therefore argue that the integration of DTN/ICN with LTE is an important research topic.

5.2.2. Software & Hardware Architecture

In previous work [36], we provided an orchestration framework for PS applications. We worked out a ME architecture that provides ICN/DTN network Apps in the case, when a still functional eNB can provide a RAN towards end-users (UEs), but it does not have a valid connection to the network core. Typically, when a fully functional eNB looses its connection to the EPC, it stops providing RAN service as a result of a failed S1 or EPC.

A macro MEC-enabled eNB [1] is the main architectural element of this system. In the ordinary situation, when the network core is reachable, an eNB site runs a BBU that provides E-UTRAN and communicates with the operator network core to provide mobile access. When the core is unavailable, the ME-enabled base station can actively cooperate in the DTN/ICN information dissemination by instantiating DTN/ICN based Apps as VNFs. The primary purpose of this section is to provide an evaluation of our DTN/ICN architecture in a disaster situation, when a bundle of a micro LTE core is provided to

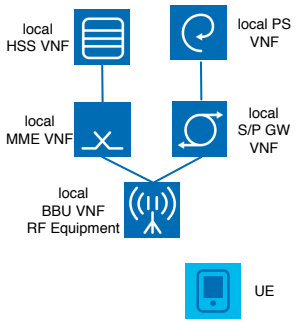

Figure 16: Service bundle for PS applications

run Radio Access Network (RAN) integrated with DTN/ICN (c.f., Fig. 16). To accomplish this goal, we integrate the PS orchestration solution with SDN traffic management derived in this paper to obtain a fully functional PS solution (i.e., with traffic). Traffic measurements were out of scope of the work presented in [36].

Whenever a local Application Management Unit on an eNB discovers that it runs disconnected from the core network, it starts the recovery procedure to provision a new communication service. Such a service is deployed as a bundle of VNFs that defines the required network services, namely eNB, local SPGW, MME, HSS, and PS. User information will be maintained either by replicating the HSS database if possible, or by provisioning the IMSI range for rescue teams with necessary key and sequence numbers. Note that the authentication procedure can also be relaxed to accept all the attach procedures.

All the VNF functions are instantiated on the same edge cloud. The BBU has to be re-instantiated to acknowledge local copies of the MME, SPGW, HSS providing core network services. MME, SPGW, and HSS are minimal services of a small footprint. They provide basic LTE functions and connect UEs attached with the macro eNB. Due to the basic core function, the UEs attached to the same eNB can communicate directly with the help of the PS VNF. The PS VNF is based upon DTN and/or ICN applications such as CCNx[24] or DTN2[25]. In our previous work, we implemented a DTN2 charm for Juju VNFM [36]. The PS VNF is a communication end-point and a relay between other clients instantiated on UEs. The established setup allows end users to attach to macro eNB. The rescue teams can now freely attach to the open eNB instantiated and exchange data using DTN and/or ICN relay points. If a macro eNB shares a functional X2 interface with another nearby base station, the X2 interface can be used as an interface to share data among nearby cells. Otherwise, the cell-cell communication can be based upon data mules.

5.2.3. Results

In Fig. 17, we gather instantiation times for the PS service bundle provisioned with Juju VNFM (MySql is a supporting system for HSS) [36]. We tested two scenarios, when a DTN App (i.e., PS instance), MySQL, EPC, and HSS were instantiated on KVM and LXC respectively. We use a single host machine with Intel 3.20 GHz quad core CPU and 16 GB RAM. The services use 1 thread, 1 GB RAM; 1 thread 1 GB RAM; 4 threads 8 GB RAM; 1 thread, 2 GB RAM; 1 thread, 1 GB RAM for MySQL, HSS, eNB, EPC, and DTN resp. Therefore, the PS bundle can be instantiated within around 600 seconds after the failure in the EPC was discovered.

The architecture of the developed Public Safety (PS) Solution is presented in Fig. 18. It is similar to the solution studied in 5.1. However, the EPC runs on the same ME cloud as VNFs and the protocol considered is different. The SDN traffic management scheme remains the same.

The main core network of the provider is disconnected. We therefore instantiate a complete small core on the MEC Cloud (i.e., MME, SPGW, HSS) as Virtual Network Functions (VNFs) through the Juju VNFM. The eNB connects to the newly instantiated core. We use OpenAirInterface [28] from the OAI development branch as the EPC and eNB. The eNB is configured to use 5 MHz channels in Band7 of the LTE spectrum with SISO mode. We also instantiate

[24] http://blogs.parc.com/ccnx/ccnx-downloads/
[25] https://sourceforge.net/projects/dtn/files/DTN2/

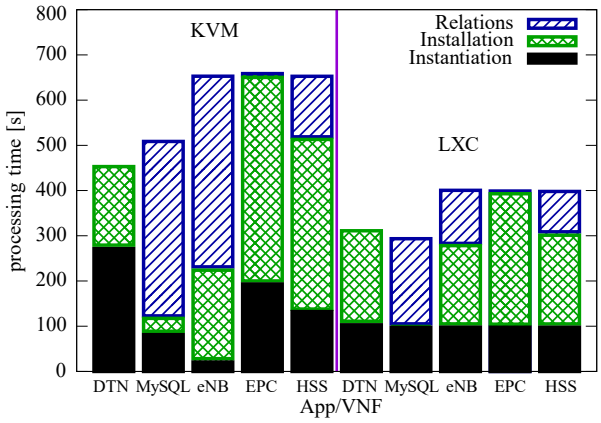

Figure 17: Provisioning time.

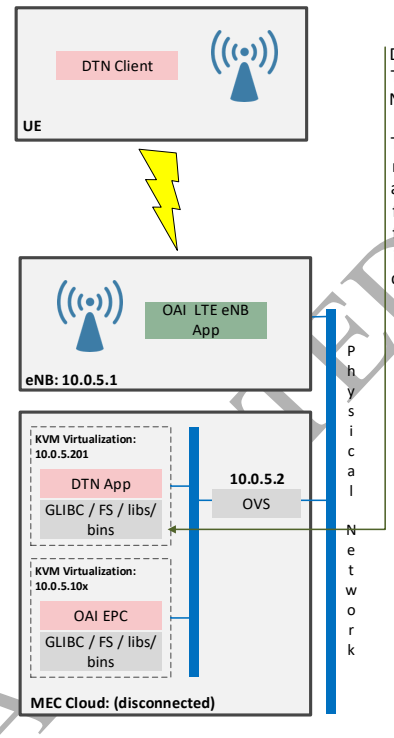

Figure 18: Architecture of the PS solution.

a DTN PS service and the SDN controller to manage traffic between the UE and the DTN App. The UE using a Huawei E392 dongle is equipped with a DTN PS client. Hence, the UE and DTN PS can directly communicate when the UE is successfully attached to the eNB.

We verified that our SDN/NFV-based PS solution works well. We were able to successfully dtnping (use the dtn ping function) between the UE of (dynamically assigned) IP address 172.16.0.2, eNB TEID 0x0000001, SPGW TEID 0xca6fe0dd, and DTN address ue.dtn towards the DTN App on the ME cloud of address 10.0.5.203 and DTN address enb.dtn. The dtnping confirms connectivity in a small DTN network of UE and eNB.

It takes around 237 $\pm$ 38 ms to discover the connected eNB App (enb.dtn) from the UE. When the connectivity was tested successfully, we started sending files between the UE (ue.dtn) and ENB (enb.dtn). It is worth noting that the dtnping times are much larger than the classical ICMP ping times (20.9 $\pm$ 2.6 ms), as the DTN service has to first discover the destination (if connected). The example transmission of six 1 MB files (the time period between consecutive transmissions is 40 s) is provided in Fig. 19. The throughout of 1.06 $\pm$ 0.08 MB/s was established on average in a single transmission (c.f., Fig. 20). The figures present the amount of data sent from the UE to the eNB through a DTN2 App.

6. Conclusions

Our paper consists of five innovations in prototyping of the MEC environment. The first innovation is the organization of the traffic management at the network edge. We show how to manage GTP-tunnels and traffic redirections to successfully exchange traffic between IP-based MEC applications and the LTE-based UE. The second innovation is the implementation of the OVS GTP matcher that together with OVS GTP tunneling service provides traffic to MEC services. The third innovation is in the domain of SDN rule management at the ME cloud with an SDN controller equipped with an S1-C tracking module. The fourth innovation is the first fully open-source implementation of the ME cloud based on Juju

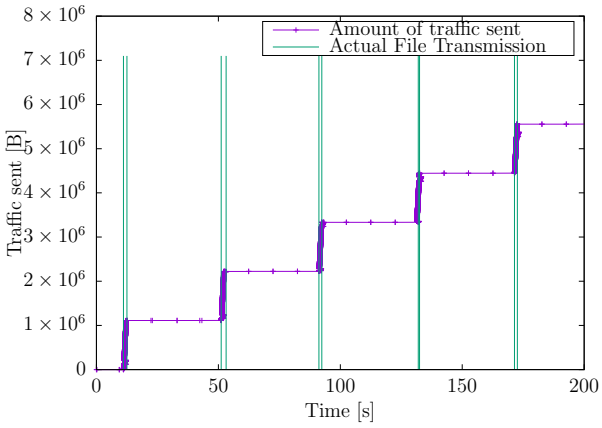

Figure 19: Six consecutive file transmissions.

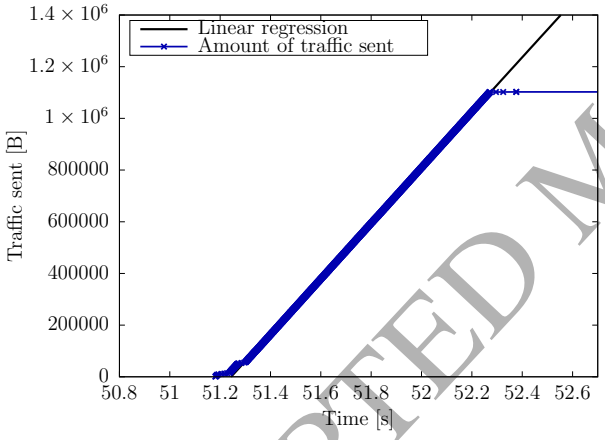

Figure 20: Single file transmissions.

VNFM, SDN controller, OVS, and OAI. It improves the perceived throughput of a UE by 30%. Finally, our fifth innovation is the evaluation of the MEC-enabled public safety solution. It is a fully open-source solution that spawns a minimal fully operating core (EPC) in the disconnected core critical situation. As mobility is a crucial aspect of mobile systems, in future work, we will study mobility and handover in our MEC architecture.

Acknowledgement

This work was partially supported by the EU FP7 Project FLEX (612050) and COST STSM grants (CA15127). We would like to also thank Dr. Peppo Brambilla from the Institute of Computer Science of the University of Bern for helping us out with the 10 GbE-T connectivity tests.

References

[1] M. Patel, J. Joubert, J. R. Ramos, N. Sprecher, S. Abeta, A. Neal, Mobile-Edge Computing, ETSI, white paper: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf (2014).

[2] ETSI GS MEC 003: Mobile Edge Computing (MEC); Framework and Reference Architecture V1.1.1, http://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/gs_MEC003v010101p.pdf (Mar. 2016).

[3] F. Lobillo, Z. Becvar, M. A. Puente, P. Mach, F. L. Presti, F. Gambetti, M. Goldhamer, J. Vidal, A. K. Widiawan, E. Calvanesse, An architecture for mobile computation offloading on cloud-enabled lte small cells, in: 2014 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), 2014, pp. 1–6. doi:10.1109/WCNCW.2014.6934851.

[4] H. Hawilo, A. Shami, M. Mirahmadi, R. Asal, NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC), IEEE Network 28 (6) (2014) 18–26. doi:10.1109/MNET.2014.6963800.

[5] A. Ksentini, M. Bagaa, T. Taleb, On Using SDN in 5G: The Controller Placement Problem, in: 2016 IEEE Global Communications Conference (GLOBECOM), 2016, pp. 1–6. doi:10.1109/GLOCOM.2016.7842066.

[6] M. Martinello, M. R. N. Ribeiro, R. E. Z. de Oliveira, R. de Angelis Vitoi, Keyflow: a prototype for evolving SDN toward core network fabrics, IEEE Network 28 (2) (2014) 12–19. doi:10.1109/MNET.2014.6786608.

[7] V. Nguyen, Y. Kim, Proposal and evaluation of SDNbased mobile packet core networks, EURASIP Journal on Wireless Communications and Networking 2015 (1) (2015) 18–26. doi:10.1186/s13638-015-0395-1.

[8] K. Pentikousis, Y. Wang, W. Hu, Mobileflow: Toward software-defined mobile networks, IEEE Communications Magazine 51 (7) (2013) 44–53. doi:10.1109/MCOM.2013.6553677.

[9] I. Giannoulakis, E. Kafetzakis, G. Xylouris, G. Gardikis, A. Kourtis, On the applications of efficient NFV management towards 5G networking, in: 1st International Conference on 5G for Ubiquitous Connectivity, 2014, pp. 1–5. doi:10.4108/icst.5gu.2014.258133.

[10] H. Karl, S. Drxler, M. Peuster, A. Galis, M. Bredel, A. Ramos, J. Martrat, M. S. Siddiqui, S. van Rossem, W. Tavernier, G. Xilouris, DevOps for network function virtualisation: an architectural approach, Transactions on Emerging Telecommunications Technologies 27 (9) (2016) 1206–1215. doi:10.1002/ett.3084.

[11] S. Sahhaf, W. Tavernier, J. Czentye, B. Sonkoly, P. Skldstrm, D. Jocha, J. Garay, Scalable Architecture for Service Function Chain Orchestration, in: 2015 Fourth European Workshop on Software Defined Networks, 2015, pp. 19–24. doi:10.1109/EWSDN.2015.55.

[12] B. Sousa, L. Cordeiro, P. Simes, A. Edmonds, S. Ruiz, G. A. Carella, M. Corici, N. Nikaein, A. S. Gomes, E. Schiller, T. Braun, T. M. Bohnert, Toward a Fully Cloudified Mobile Network Infrastructure, IEEE Transactions on Network and Service Management 13 (3) (2016) 547–563. doi:10.1109/TNSM.2016.2598354.

[13] ETSI GS NFV-MAN 001: Network Functions Virtualisation (NFV), Network Functions Virtualisation (NFV); Management and Orchestration V1.1.1, http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf (Dec. 2014).

[14] R. Roman, J. Lopez, M. Mambo, Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges, Future Generation Computer Systemsdoi:http://dx.doi.org/10.1016/j.future.2016.11.009.

[15] C.-Y. Chang, K. Alexandris, N. Nikaein, K. Katsalis, T. Spyropoulos, MEC Architectural Implications for LTE/LTE-A Networks, in: Proceedings of the Workshop on Mobility in the Evolving Internet Architecture, MobiArch '16, ACM, New York, NY, USA, 2016, pp. 13–18. doi:10.1145/2980137.2980139.

[16] M. T. Beck, M. Werner, S. Feld, T. Schimper, Mobile Edge Computing: A Taxonomy, in: The Sixth International Conference on Advances in Future Internet, AFIN '14, IARIA, 2014, pp. 48–54.

[17] M. T. Beck, S. Feld, A. Fichtner, C. Linnhoff-Popien, T. Schimper, ME-VoLTE: Network functions for energy-efficient video transcoding at the mobile edge, in: 2015 18th International Conference on Intelligence in Next Generation Networks, 2015, pp. 38–44. doi:10.1109/ICIN.2015.7073804.

[18] S. Nunna, A. Kousaridas, M. Ibrahim, M. Dillinger, C. Thuemmler, H. Feussner, A. Schneider, Enabling Real-Time Context-Aware Collaboration through 5G and Mobile Edge Computing, in: 2015 12th International Conference on Information Technology - New Generations, 2015, pp. 601–605. doi:10.1109/ITNG.2015.155.

[19] L. Tong, Y. Li, W. Gao, A hierarchical edge cloud architecture for mobile computing, in: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, 2016, pp. 1–9. doi:10.1109/INFOCOM.2016.7524340.

[20] A. Huang, N. Nikaein, T. Stenbock, A. Ksentini, C. Bonnet, Low Latency MEC Framework for SDN-based LTE/LTE-A Networks, in: IEEE International Conference on Communications, ICC '17, 2017, pp. 1–6.

[21] K. Wang, M. Shen, J. Cho, A. Banerjee, J. Van der Merwe, K. Webb, Mobiscud: A fast moving personal cloud in the mobile network, in: Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges, AllThingsCellular '15, ACM, New York, NY, USA, 2015, pp. 19–24. doi:10.1145/2785971.2785979.

[22] A. Manzalini, et al., Towards 5g software-defined ecosystems: Technical challenges, business sustainability and policy issues, white paper.

[23] J. Kempf, B. Johansson, S. Pettersson, H. Lning, T. Nilsson, Moving the mobile evolved packet core to the cloud, in: 2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2012, pp. 784–791. doi:10.1109/WiMOB.2012.6379165.

[24] N. Nikaein, E. Schiller, R. Favraud, K. Katsalis, D. Stavropoulos, I. Alyafawi, Z. Zhao, T. Braun, T. Korakis, Network store: Exploring slicing in future 5g networks, in: Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture, MobiArch '15, ACM, New York, NY, USA, 2015, pp. 8–13. doi:10.1145/2795381.2795390.

[25] B. Nguyen, N. Choi, M. Thottan, J. V. der Merwe, SIMECA: SDN-based IoT Mobile Edge Cloud Architecture, in: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2017, pp. 503–509. doi:10.23919/INM.2017.7987319.

[26] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: Enabling Innovation in Campus Networks, SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74. doi:10.1145/1355734.1355746.
URL http://doi.acm.org/10.1145/1355734.1355746

[27] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, M. Casado, The design and implementation of open vswitch, in: Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation, NSDI'15, USENIX Association, Berkeley, CA, USA, 2015, pp. 117–130.
URL http://dl.acm.org/citation.cfm?id=2789770.2789779

[28] N. Nikaein, R. Knopp, L. Gauthier, E. Schiller, T. Braun, D. Pichon, C. Bonnet, F. Kaltenberger, D. Nussbaum, Demo: Closer to cloud-ran: Ran as a service, in: Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, MobiCom '15, ACM, New York, NY, USA, 2015, pp. 193–195. doi:10.1145/2789168.2789178.

[29] C. Anastasiades, A. Gomes, R. Gadow, T. Braun, Persistent caching in information-centric networks, in: 2015 IEEE 40th Conference on Local Computer Networks (LCN), 2015, pp. 64–72. doi:10.1109/LCN.2015.7366284.

[30] H. Sarkissian, The business case for caching in 4g lte networks, LSI White Paper (2013).

[31] N. Nikaein, R. Knopp, F. Kaltenberger, L. Gauthier, C. Bonnet, D. Nussbaum, R. Ghaddab, Demo: OpenAirInterface: An Open LTE Network in a PC, in: Proceedings of the 20th Annual International Conference on Mobile Computing and Networking, MobiCom '14, ACM, New York, NY, USA, 2014, pp. 305–308.

[32] E. M. Trono, M. Fujimoto, H. Suwa, Y. Arakawa, M. Takai, K. Yasumoto, Disaster area mapping using spatially-distributed computing nodes across a DTN, in: 2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops), IEEE, 2016, pp. 1–6.

[33] N. Uchida, N. Kawamura, N. Williams, K. Takahata, Y. Shibata, Proposal of delay tolerant network with cognitive wireless network for disaster information network system, in: Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on, IEEE, 2013, pp. 249–254.

[34] J. T. B. Fajardo, K. Yasumoto, N. Shibata, W. Sun, M. Ito, Disaster information collection with opportunistic communication and message aggregation, Journal of information processing 22 (2) (2014) 106–117.

[35] G. Tyson, E. Bodanese, J. Bigham, A. Mauthe, Beyond content delivery: Can icns help emergency scenarios?, IEEE Network 28 (3) (2014) 44–49.

[36] E. Schiller, E. Kalogeiton, T. Braun, A. Gomes, N. Nikaein, 11 - icn/dtn for public safety in mobile networks, in: D. Camara, N. Nikaein (Eds.), Wireless Public Safety Networks 3, Elsevier, 2017, pp. 231 – 247. doi:https://doi.org/10.1016/B978-1-78548-053-9.50011-1.
URL http://www.sciencedirect.com/science/article/pii/B9781785480539500111

Biography

Eryk Schiller (schiller@inf.unibe.ch) received two M.Sc. diplomas: in electronics and telecommunications from the University of Science and Technology, and in theoretical physics from the Jagiellonian University, Cracow, Poland, in 2006 and 2007, respectively. He got a Ph.D. in computer science from the University of Grenoble, France, in 2010. He was a postdoctoral scholar at the University of Neuchatel, Switzerland. Since 2014, he has been with the University of Bern, Switzerland, as a senior researcher.

Navid Nikaein (navid.nikaein@eurecom.fr) has been an assistant professor in the Communication Systems Department at EURECOM since 2009. He received his Ph.D. degree in communication systems from the Swiss Federal Institute of Technology (EPFL) in 2003. Currently, he is leading a research group focusing on experimental system research related to wireless systems and networking. Broadly, his research interests include wireless access and networking protocols (4G/5G), cloud-native and programmable mobile networking (SDN, NFV, MEC), and real-time RF prototyping and emulation/simulation.

Eirini Kalogeiton (kalogeiton@inf.unibe.ch) re-

ceived Dipl.Eng. and M.Sc. degrees in electrical and computer engineering from the Democritus University of Thrace, Xanthi, Greece, in 2014 and 2016, respectively. Since 2016, she is a PhD student at the University of Bern, Bern, Switzerland. Her main research interests include vehicular ad-hoc networks, routing in named data networking networks and software defined networking.

Mikael Gasparyan (gasparyan@inf.unibe.ch) is a Ph.D. student at the University of Bern, Switzerland. He obtained his Master degree with a specialization in distributed systems at the University of Fribourg in Switzerland. He earned his Bachelor degree at the University of Applied Science in Sierre, Switzerland. His Ph.D. research topic is Service-Centric Networking. In his Ph.D. work, he aims to develop and evaluate a novel Service-Centric Networking architecture, which integrates service request requirements like load-balancing and session support.

Torsten Braun (braun@inf.unibe.ch) got his Ph.D. degree from the University of Karlsruhe, Germany, in 1993. From 1994 to 1995 he was a guest scientist at INRIA Sophia-Antipolis (France). From 1995 to 1997 he worked at the IBM European Networking Centre Heidelberg, Germany. He has been a full professor of computer science at the University of Bern, Switzerland, since 1998 and a member of the SWITCH (Swiss education and research network) Board of Ttrustees since 2001.