A Prelude to the 5G Core Network Architecture

Ishan Vaishnavi, David Perez-Caparros, Xun Xiao, Riccardo Trivisonno, Riccardo Guerzoni, Artur Hecker
Huawei European Research Center, Munich, Germany

***Abstract*—The next generation of telecom networks will support a number of diverse vertical industries. Given the economic pressures of the competitive telecom environment, we regard it as unlikely or at least very difficult that a single telecom operator could acquire expertise in all the different vertical domains to ensure an efficient service operation for each and every multitude of those verticals. This means that the 5G network shall appear different to different users, molding itself and its architecture based on the requirements of the vertical it is addressing. In previous work, we proposed a *plastic* architecture for 5G as a step to a fully programmable infrastructure that can natively support different network architectures. The most important aspect of the plastic 5G network is therefore going to be its mold-ability. This mold-ability means that each user, device and/or applications sees the architecture specifically designed for itself in the network. In this demo we show our implementation of this plastic architecture assuming the use of SDN in the future 5G network.**

I. INTRODUCTION

Various use cases, typically, in the machine to machine domain are driving the burst of small cheap devices and sensors that can connect to the telecommunication network. In the near future there are estimated to be millions and millions of such devices in the market each with its own set of requirements on the network. In such a case, the 4G design of a *one for all* standardized network architecture with tunneling and fixed data/control anchoring points irrespective of the device or the application becomes too rigid. Each set of such devices imposing glaringly different requirements on the network shall render the *one for all* architecture inefficient. Instead, our vision proposes a dynamic architecture for each device that is instantiated when the device or set of devices join the network. Not only is this architecture dynamically instantiate-able, it should also be programmable by the users who own those devices.

Our architecture, shown in Figures 1 and 2, presented partly earlier in [1] is divided into two layers: a) the basic minimalistic standardized architecture that all devices must follow and b) the architecture that is then specifically instantiated for the device (or set of devices/application) depending upon its registered identity by the minimalistic architecture. The basic architecture is designed around the combined SDN and NFV paradigms, and it assumes a clean slate forwarding plane, spanned by SDN switches and a logically unique yet distributed SDN controller, upon which both Control plane and Data plane are built.

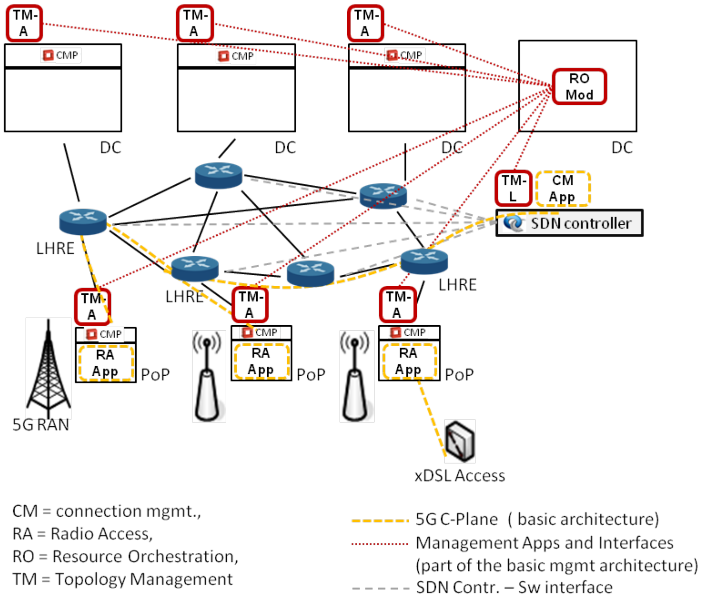


Fig. 1. 5G Plastic Architecture: Basic Architecture

Besides the applications that do background topology management and resource management tasks the basic control plane architecture includes access applications, a general purpose connectivity management application and last hop routing elements. Access applications, implemented on point of presence close to access nodes, are responsible to provide the last hop physical connectivity. In Figure 1, for sake of generality, general access nodes are shown, including 5G RAN nodes, femto nodes or xDSL boxes. Each access app is connected to a Last Hop Routing Element (LHRE), which represents a soft anchor point for all devices connecting via that access app. The LHRE is an SDN switch capable of forwarding early control messages, sent at the attachment phase by any device, to the general purpose connectivity management app. The general purpose connectivity management app is capable of interpreting the attachment request, and to initiate a service/device/application specific control and data plane architecture, suiting the attachment request. Figure 2 illustrates an example of service specific control architecture. In the example, it is assumed the services and applications, the attaching device might be initiating, could require mobility and flow management applications. For this reason, additional control apps are indicated in the figure, some of them implemented as virtual functions on data centers, others directly embedded on the SDN controller. The complete definition of the 5G plastic architecture as described in [1] includes a list of basic control functions. It should be noted the definition of a service specific architecture (i.e. the definition of the proper set of interconnected control applications) does not dictate by itself how such architecture is instantiated on

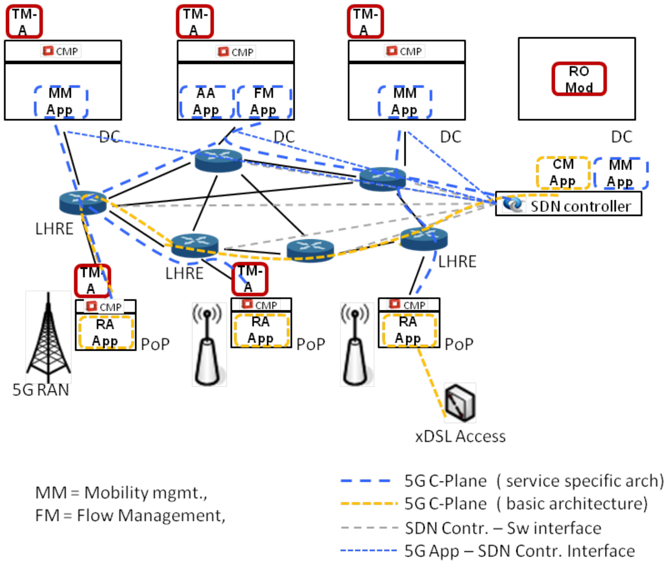
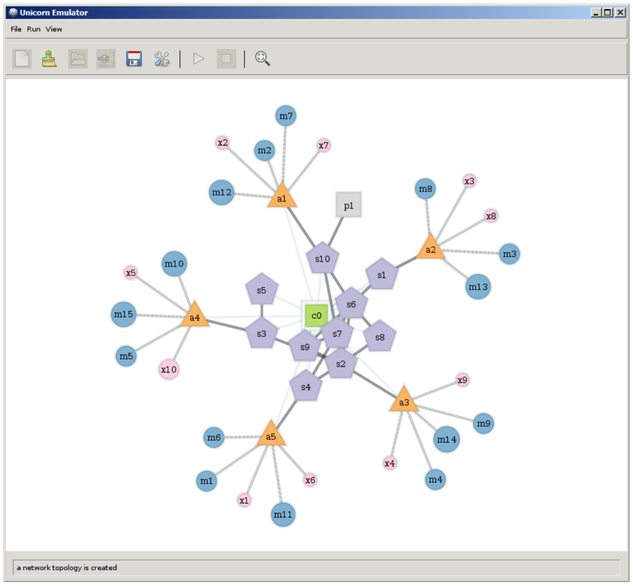


Fig. 2. 5G Plastic Architecture: Service Specific Architecture



a network topology is created

Fig. 3. The extended mininet network with access points (a1-5), mobile (m1-15) and sensor nodes (x1-10)

the physical infrastructures. The suitable instantiation will depend upon on implementation specifics as well as scalability, reliability and expected load considerations.

An example, that will be demonstrated during the demo is shown in Figure 3. There are devices in the mobile network that are authorized to be mobile and others that aren't, for example static weather sensors. When a static weather sensor attaches and communicates over the network it is only authorized to send data over and attach to a specific access point. Accessing the network via any other access point as well as mobility of the device is not configured. Then in a ideally programmable architecture the mobility management application should not be bothered with the existence of this device. However, this behaviour needs to be programmed in the network. This is explained in the next section.

II. SDN Basics

The SDN approach to network design and management:

- Separates the control from the forwarding plane of the network
- Centralizes the control plane to a logically single controller

SDN network is composed of simple switches (or forwarding elements) in the forwarding plane and an intelligent SDN controller that configures how those switches behave by installing flow (or forwarding) rules on the switches. In the simplest view, the flow rules can be thought match-action pairs. The context information (such as the incoming switch port), header and/or other parts of an incoming data flow, frame, packet, datagram or segment (in the following called packet for simplicity) may be matched to the contents of the flow table of the switch, and, in case of a match, the switch may trigger actions, such as forwarding to a certain port, dropping the packet altogether, redirecting the packet to the controller and so on. In case an incoming packet does not match any of the flow rules defined on the switch then the switch can send what is called a PACKET_IN consisting of the incoming packet as well as some switch context information (such as incoming port) to the controller. An application on top of the controller may analyze this PACKET_IN typically resulting in either i) installation of new flow rules on some switches, e.g. to handle future packets somehow corresponding to the initial one and/or ii) sending out of some packets, e.g. relaying the originally received one. A complete telecom network can be designed as a combination some basic applications and some additional vertical specific implementations. This is our proposal for the 5G architecture [1].

III. Technical Description of the Demonstration

The mobile network is emulated over an self built extension of an open source network emulation software, Mininet [2]. Mininet is well known in the SDN research community and it is widely used for network prototyping. The original Mininet software uses lightweight, OS-level virtualization to emulate hosts, switches, and network links. For instantiating virtual hosts, it leverages on cgroups (Linux Control Groups), which are later attached to a network namespace. Virtual interfaces in the network namespace are connected to software switches, such as Open vSwitch [3], using virtual Ethernet links.

Our platform extends Mininet capabilities in order to emulate mobile SDN networks. New types of network elements, such as Access Points (AP), Mobiles, and Sensors were added to the original node library. Mobiles and sensors, unlike Hosts, can be moved during runtime, and virtual links are dynamically created between them and the closest access point at any given point of time. The experimental cluster support in Mininet was also extended so that large scale mobile networks can be emulated across several physical machines.

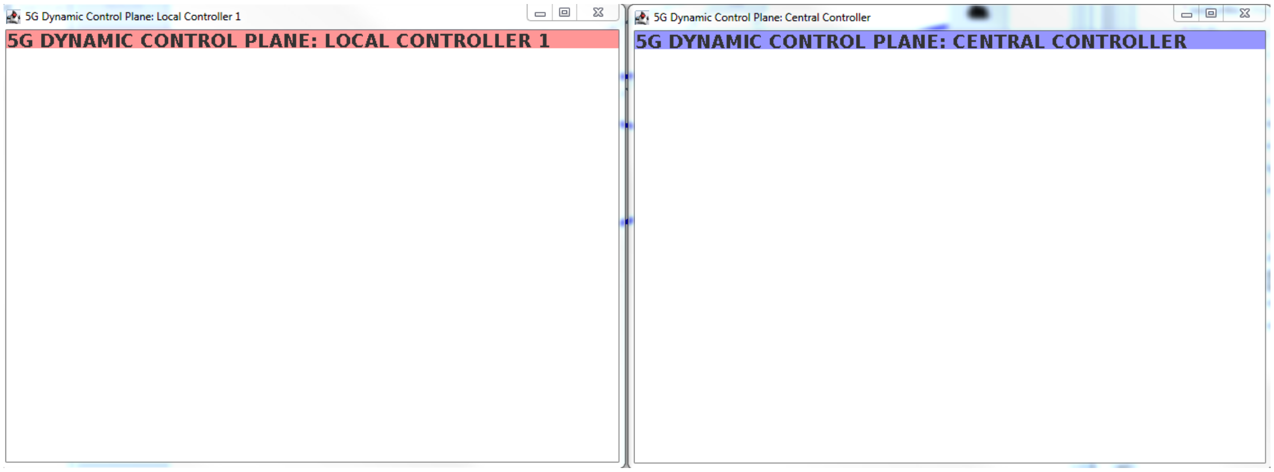
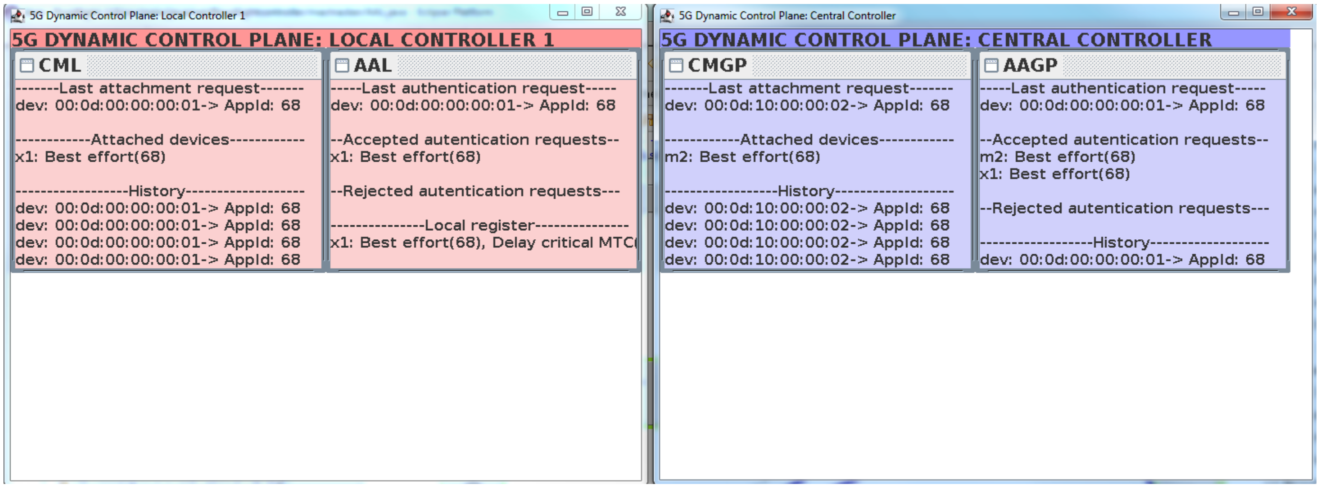


Fig. 4. Initial controller status window: no devices attached to the network, therefore, no applications are running.



```
-------Last attachment request-------
dev: 00:0d:00:00:00:01-> AppId: 68

-----------Attached devices-----------
x1: Best effort(68)

-----------------History------------------
dev: 00:0d:00:00:00:01-> AppId: 68
dev: 00:0d:00:00:00:01-> AppId: 68
dev: 00:0d:00:00:00:01-> AppId: 68
dev: 00:0d:00:00:00:01-> AppId: 68
```

```
-----Last authentication request-----
dev: 00:0d:00:00:00:01-> AppId: 68

--Accepted autentication requests--
x1: Best effort(68)

--Rejected autentication requests---

--------------Local register--------------
x1: Best effort(68), Delay critical MTC
```

```
-------Last attachment request-------
dev: 00:0d:10:00:00:02-> AppId: 68

-----------Attached devices-----------
m2: Best effort(68)

-----------------History------------------
dev: 00:0d:10:00:00:02-> AppId: 68
dev: 00:0d:10:00:00:02-> AppId: 68
dev: 00:0d:10:00:00:02-> AppId: 68
dev: 00:0d:10:00:00:02-> AppId: 68
```

```
-----Last authentication request-----
dev: 00:0d:00:00:00:01-> AppId: 68

--Accepted autentication requests--
m2: Best effort(68)
x1: Best effort(68)

--Rejected autentication requests---

-----------------History------------------
dev: 00:0d:00:00:00:01-> AppId: 68
```

Fig. 5. The applications running in the controller when both the sensor (x2) and the mobile phone(m2) attach. These besides other management applications form the basic set of applications: authorization and basic connection management

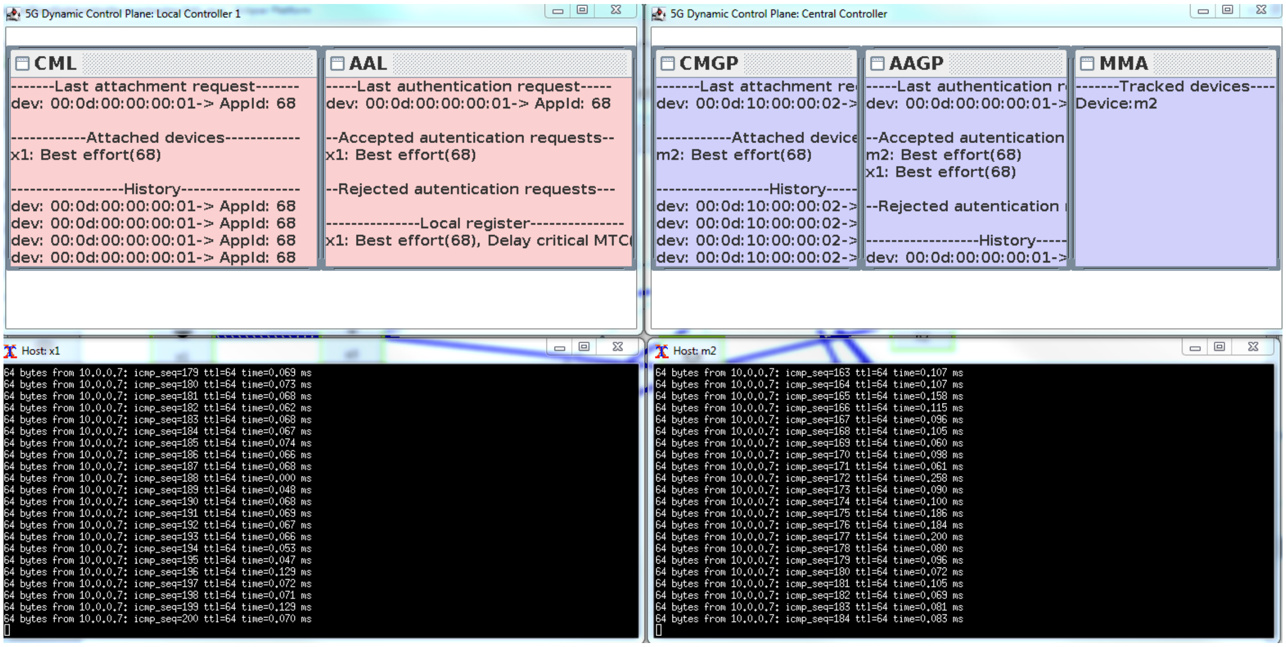
In order to visualize such large scale mobile networks, a new Graphical User Interface (GUI) was developed. This new GUI allows the user to design its own network topology, drag mobile nodes around during runtime, and see how data plane flows are dynamically set up by a centralized openflow controller.

The main focus of this demo lies in the control plane which comprises of a local fast controller and a centralized controller. The controller appears logically as one to the end devices. Initially, the control plane is empty with no applications running, see Figure 4. In the first step of the demonstration a sensor and a mobile node both attach themselves to the network. The sensor is authorized over a local controller since it is meant to always attach to a same controller and requires fast control plane response times. The mobile is however authorized by the centralized authorization, since it may attach anywhere in the network, see Figure 5. Both the mobile and the sensor node then start to send data over the network to gateways p1 and p2 respectively. Figure 6 shows that as soon as the mobile sends data the mobility application is enabled for the mobile. The configuration of when an application is started can be left to the programmer. However, no such thing happens when the sensor starts transmitting data.

In the next step this is verified by moving the mobile and the sensor as shown in Figure 7. Our platform enables both the senor and the mobile to physical reattach themselves to the closest new access point. However, the data transmission of the mobile continues and the mobility management agent detects this mobility and correspondingly sets up new paths for the communication. For the sensor, since the mobility management app is not enabled the new paths required to continue its communication are not setup.

IV. CONCLUSION

This demonstration with the mobility management app demonstrates the power of programmability in the platform. Based on each user device a completely different architecture can be constructed in the network. Thus, a car company could itself program how it wants the network to behave with respect



CML
```
-------Last attachment request-------
dev: 00:0d:00:00:00:01-> AppId: 68

------------Attached devices------------
x1: Best effort(68)

------------------History-------------------
dev: 00:0d:00:00:00:01-> AppId: 68
dev: 00:0d:00:00:00:01-> AppId: 68
dev: 00:0d:00:00:00:01-> AppId: 68
dev: 00:0d:00:00:00:01-> AppId: 68
```

AAL
```
-----Last authentication request-----
dev: 00:0d:00:00:00:01-> AppId: 68

--Accepted autentication requests--
x1: Best effort(68)

--Rejected autentication requests---

---------------Local register---------------
x1: Best effort(68), Delay critical MTC
```

CMGP
```
-------Last attachment re
dev: 00:0d:10:00:00:02->

------------Attached device
m2: Best effort(68)

------------------History-----
dev: 00:0d:10:00:00:02->
dev: 00:0d:10:00:00:02->
dev: 00:0d:10:00:00:02->
dev: 00:0d:10:00:00:02->
```

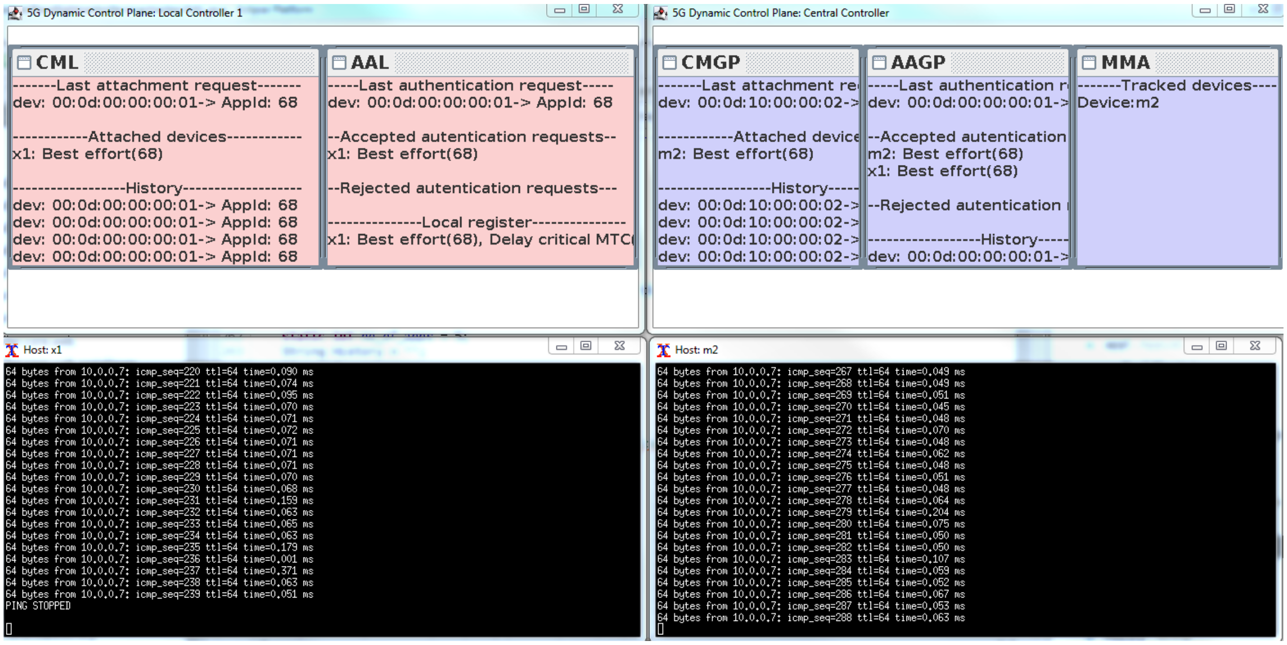
AAGP
```
-----Last authentication r
dev: 00:0d:00:00:00:01->

--Accepted autentication
m2: Best effort(68)
x1: Best effort(68)

--Rejected autentication

------------------History-----
dev: 00:0d:00:00:00:01->
```

MMA
```
-------Tracked devices----
Device:m2
```

```
64 bytes from 10.0.0.7: icmp_seq=179 ttl=64 time=0.069 ms
64 bytes from 10.0.0.7: icmp_seq=180 ttl=64 time=0.073 ms
64 bytes from 10.0.0.7: icmp_seq=181 ttl=64 time=0.068 ms
64 bytes from 10.0.0.7: icmp_seq=182 ttl=64 time=0.062 ms
64 bytes from 10.0.0.7: icmp_seq=183 ttl=64 time=0.068 ms
64 bytes from 10.0.0.7: icmp_seq=184 ttl=64 time=0.067 ms
64 bytes from 10.0.0.7: icmp_seq=185 ttl=64 time=0.074 ms
64 bytes from 10.0.0.7: icmp_seq=186 ttl=64 time=0.066 ms
64 bytes from 10.0.0.7: icmp_seq=187 ttl=64 time=0.068 ms
64 bytes from 10.0.0.7: icmp_seq=188 ttl=64 time=0.000 ms
64 bytes from 10.0.0.7: icmp_seq=189 ttl=64 time=0.048 ms
64 bytes from 10.0.0.7: icmp_seq=190 ttl=64 time=0.068 ms
64 bytes from 10.0.0.7: icmp_seq=191 ttl=64 time=0.069 ms
64 bytes from 10.0.0.7: icmp_seq=192 ttl=64 time=0.067 ms
64 bytes from 10.0.0.7: icmp_seq=193 ttl=64 time=0.066 ms
64 bytes from 10.0.0.7: icmp_seq=194 ttl=64 time=0.053 ms
64 bytes from 10.0.0.7: icmp_seq=195 ttl=64 time=0.047 ms
64 bytes from 10.0.0.7: icmp_seq=196 ttl=64 time=0.129 ms
64 bytes from 10.0.0.7: icmp_seq=197 ttl=64 time=0.072 ms
64 bytes from 10.0.0.7: icmp_seq=198 ttl=64 time=0.071 ms
64 bytes from 10.0.0.7: icmp_seq=199 ttl=64 time=0.129 ms
64 bytes from 10.0.0.7: icmp_seq=200 ttl=64 time=0.070 ms
```

```
64 bytes from 10.0.0.7: icmp_seq=163 ttl=64 time=0.107 ms
64 bytes from 10.0.0.7: icmp_seq=164 ttl=64 time=0.107 ms
64 bytes from 10.0.0.7: icmp_seq=165 ttl=64 time=0.158 ms
64 bytes from 10.0.0.7: icmp_seq=166 ttl=64 time=0.115 ms
64 bytes from 10.0.0.7: icmp_seq=167 ttl=64 time=0.096 ms
64 bytes from 10.0.0.7: icmp_seq=168 ttl=64 time=0.105 ms
64 bytes from 10.0.0.7: icmp_seq=169 ttl=64 time=0.060 ms
64 bytes from 10.0.0.7: icmp_seq=170 ttl=64 time=0.098 ms
64 bytes from 10.0.0.7: icmp_seq=171 ttl=64 time=0.061 ms
64 bytes from 10.0.0.7: icmp_seq=172 ttl=64 time=0.258 ms
64 bytes from 10.0.0.7: icmp_seq=173 ttl=64 time=0.090 ms
64 bytes from 10.0.0.7: icmp_seq=174 ttl=64 time=0.100 ms
64 bytes from 10.0.0.7: icmp_seq=175 ttl=64 time=0.186 ms
64 bytes from 10.0.0.7: icmp_seq=176 ttl=64 time=0.184 ms
64 bytes from 10.0.0.7: icmp_seq=177 ttl=64 time=0.200 ms
64 bytes from 10.0.0.7: icmp_seq=178 ttl=64 time=0.080 ms
64 bytes from 10.0.0.7: icmp_seq=179 ttl=64 time=0.096 ms
64 bytes from 10.0.0.7: icmp_seq=180 ttl=64 time=0.072 ms
64 bytes from 10.0.0.7: icmp_seq=181 ttl=64 time=0.105 ms
64 bytes from 10.0.0.7: icmp_seq=182 ttl=64 time=0.069 ms
64 bytes from 10.0.0.7: icmp_seq=183 ttl=64 time=0.081 ms
64 bytes from 10.0.0.7: icmp_seq=184 ttl=64 time=0.083 ms
```

Fig. 6. The applications running in the controller when the sensor (x2) and the mobile (m2) are transmitting data (ping requests). The MMA is a specialised app for the mobile, it is unaware of the sensor device in the network



CML
```
-------Last attachment request-------
dev: 00:0d:00:00:00:01-> AppId: 68

------------Attached devices------------
x1: Best effort(68)

------------------History-------------------
dev: 00:0d:00:00:00:01-> AppId: 68
dev: 00:0d:00:00:00:01-> AppId: 68
dev: 00:0d:00:00:00:01-> AppId: 68
dev: 00:0d:00:00:00:01-> AppId: 68
```

AAL
```
-----Last authentication request-----
dev: 00:0d:00:00:00:01-> AppId: 68

--Accepted autentication requests--
x1: Best effort(68)

--Rejected autentication requests---

---------------Local register---------------
x1: Best effort(68), Delay critical MTC
```

CMGP
```
-------Last attachment re
dev: 00:0d:10:00:00:02->

------------Attached device
m2: Best effort(68)

------------------History-----
dev: 00:0d:10:00:00:02->
dev: 00:0d:10:00:00:02->
dev: 00:0d:10:00:00:02->
dev: 00:0d:10:00:00:02->
```

AAGP
```
-----Last authentication r
dev: 00:0d:00:00:00:01->

--Accepted autentication
m2: Best effort(68)
x1: Best effort(68)

--Rejected autentication

------------------History-----
dev: 00:0d:00:00:00:01->
```

MMA
```
-------Tracked devices----
Device:m2
```

```
64 bytes from 10.0.0.7: icmp_seq=220 ttl=64 time=0.090 ms
64 bytes from 10.0.0.7: icmp_seq=221 ttl=64 time=0.074 ms
64 bytes from 10.0.0.7: icmp_seq=222 ttl=64 time=0.095 ms
64 bytes from 10.0.0.7: icmp_seq=223 ttl=64 time=0.070 ms
64 bytes from 10.0.0.7: icmp_seq=224 ttl=64 time=0.071 ms
64 bytes from 10.0.0.7: icmp_seq=225 ttl=64 time=0.072 ms
64 bytes from 10.0.0.7: icmp_seq=226 ttl=64 time=0.071 ms
64 bytes from 10.0.0.7: icmp_seq=227 ttl=64 time=0.071 ms
64 bytes from 10.0.0.7: icmp_seq=228 ttl=64 time=0.071 ms
64 bytes from 10.0.0.7: icmp_seq=229 ttl=64 time=0.070 ms
64 bytes from 10.0.0.7: icmp_seq=230 ttl=64 time=0.068 ms
64 bytes from 10.0.0.7: icmp_seq=231 ttl=64 time=0.159 ms
64 bytes from 10.0.0.7: icmp_seq=232 ttl=64 time=0.063 ms
64 bytes from 10.0.0.7: icmp_seq=233 ttl=64 time=0.065 ms
64 bytes from 10.0.0.7: icmp_seq=234 ttl=64 time=0.063 ms
64 bytes from 10.0.0.7: icmp_seq=235 ttl=64 time=0.179 ms
64 bytes from 10.0.0.7: icmp_seq=236 ttl=64 time=0.001 ms
64 bytes from 10.0.0.7: icmp_seq=237 ttl=64 time=0.371 ms
64 bytes from 10.0.0.7: icmp_seq=238 ttl=64 time=0.063 ms
64 bytes from 10.0.0.7: icmp_seq=239 ttl=64 time=0.051 ms
PING STOPPED
```

```
64 bytes from 10.0.0.7: icmp_seq=267 ttl=64 time=0.049 ms
64 bytes from 10.0.0.7: icmp_seq=268 ttl=64 time=0.049 ms
64 bytes from 10.0.0.7: icmp_seq=269 ttl=64 time=0.051 ms
64 bytes from 10.0.0.7: icmp_seq=270 ttl=64 time=0.045 ms
64 bytes from 10.0.0.7: icmp_seq=271 ttl=64 time=0.048 ms
64 bytes from 10.0.0.7: icmp_seq=272 ttl=64 time=0.070 ms
64 bytes from 10.0.0.7: icmp_seq=273 ttl=64 time=0.048 ms
64 bytes from 10.0.0.7: icmp_seq=274 ttl=64 time=0.062 ms
64 bytes from 10.0.0.7: icmp_seq=275 ttl=64 time=0.048 ms
64 bytes from 10.0.0.7: icmp_seq=276 ttl=64 time=0.051 ms
64 bytes from 10.0.0.7: icmp_seq=277 ttl=64 time=0.048 ms
64 bytes from 10.0.0.7: icmp_seq=278 ttl=64 time=0.064 ms
64 bytes from 10.0.0.7: icmp_seq=279 ttl=64 time=0.204 ms
64 bytes from 10.0.0.7: icmp_seq=280 ttl=64 time=0.075 ms
64 bytes from 10.0.0.7: icmp_seq=281 ttl=64 time=0.050 ms
64 bytes from 10.0.0.7: icmp_seq=282 ttl=64 time=0.050 ms
64 bytes from 10.0.0.7: icmp_seq=283 ttl=64 time=0.107 ms
64 bytes from 10.0.0.7: icmp_seq=284 ttl=64 time=0.059 ms
64 bytes from 10.0.0.7: icmp_seq=285 ttl=64 time=0.052 ms
64 bytes from 10.0.0.7: icmp_seq=286 ttl=64 time=0.067 ms
64 bytes from 10.0.0.7: icmp_seq=287 ttl=64 time=0.053 ms
64 bytes from 10.0.0.7: icmp_seq=288 ttl=64 time=0.063 ms
```

Fig. 7. Programmable behavior: Sensor (X2) cannot ping when moved since the MMA app is not tracking it as during attach it registers as a static (non-mobile) sensor

to its cars. A energy company could itself decide in a completely different architecture over the same physical network how it wants to collect data from its sensors. During the demonstration other applications such as those for verticals, or those achieving a specialized network function shall also be demonstrated.

V. FUTURE WORK

The presented platform is still under development and new features will be soon available. For instance, an enhanced mobility engine that enables mobiles to autonomously move in the network at a user-defined rate (e.g. number of mobility events per second). Scenarios with multiple logical controllers are also being tested. On the controller end newer vertical/service specific controller applications are continuously being developed and experimented with by our team. In the near future, we plan to release the software as open-source for other researchers to use and extend.

VI. AUTHOR BIOGRAPHIES

Dr. Ishan Vaishnavi received his MTech. from IIT Delhi (2003) and after a 3 year stint with Sun Microsystems in the Solaris Team, did his PhD in multimedia networking from Vrije Universiteit Amsterdam while working at Centrum Wiskunde en Informatica in Amsterdam from 2006-2010. He then worked for DoCoMo euro-Labs as a researcher in the core networks department focusing on SDN and orchestration architectures. For the past three years he has been working at Huawei European Research Center at Munich as a Senior Solutions Architect on orchestration, SDN and their application in designing the 5G network architecture.

David Perez Caparros received his MSc. degree in Telecommunications Engineering from the University Miguel Hernndez of Elche. Since 2014, he is part of the Huawei European Research Center in Munich. His main topics of interest include future mobile network architectures, network function virtualization and software defined networking.

Dr. Xun Xiao is a researcher at Huawei European Research Center in Munich. He gained his B.E. and M.E. both from South-Central University for Nationalities in Wuhan China in 2006 and 2009 respectively, and Ph.D from City University of Hong Kong in 2012. He was a postdoctoral fellow at Max-Planck Institute Molecular Cell Biology and Genetics in Dresden, Germany from 2012 to 2014. His research interests are computer networking optimizations and system engineering.

Dr. Riccardo Trivisonno received his MSc and PhD in Telecommunications Engineering from University of Bologna and has worked for 15 years in R&D for mobile network industry. He has been with Huawei European Research Centre since 2011, working on next generation network architecture and technologies.

Mr. Riccardo Guerzoni Riccardo Guerzoni received a M.Sc. degree with maximum score cum laude in electronic engineering from the University of Ferrara, Italy, in 1997. After that, he has been in the ICT industry for more than 15 years. Since 2011 he is Senior Solution Architect at Huawei European Research Center in Munich, exploring future carrier networks technologies and business models. In his current role, he is working on network function virtualization (NFV) and application of SDN techniques to 5G.

Dr. Artur Hecker received his MSc from Universitt Karlsruhe and PhD from the ENST, Paris. He is currently Director of Future Networks Technologies at the European Research Center of Huawei Technologies. From 2006 to 2013, Artur was Associate Professor at Tlcom ParisTech, where he was leader of Security and Networking group. Artur looks back at more than 15 years of entrepreneurial, academic and industry experience in networks, systems and system security research.

REFERENCES

[1] R. Trivisonno, R. Guerzoni, I. Vaishnavi, and D. Soldani. Towards zero latency software defined 5g networks. In *Communications (ICC), HCN Workshop, 2015 IEEE International Conference on*, July 2015.

[2] Mininet: An Instant Virtual Network on your Laptop. www.mininet.org.

[3] Open vSwitch: An open virtual switch. www.openvswitch.org.