Cominardi, L., Bernardos, C. J., Serrano, P., Banchs, A. & Oliva, A. D. L. (2018). Experimental evaluation of SDN-based service provisioning in mobile networks. *Computer Standards & Interfaces, vol. 58*, pp. 158–166.

DOI: 10.1016/j.csi.2018.01.004

Experimental evaluation of SDN-based service provisioning in mobile networks

Luca Cominardi*, Carlos J. Bernardos, Pablo Serrano, Albert Banchs, Antonio de la Oliva

***Abstract*—5G networks will be characterized by their diversity in terms of traffic patterns, multi-tenancy and heterogeneous and stringent traffic requirements. Network *softwarization* is a key enabler to cope with such management burden, as it provides the ability to control all networking functions through (re)programming, thus providing higher flexibility to meet heterogeneous requirements while keeping deployment and operational costs low. In this article, we aim at experimentally validating how Software Defined Networking (SDN) concepts can greatly simplify network operation in future 5G operator networks. This simplification is achieved by allowing to very easily create and modify network services and thus customize network operation based on the operator's requirements. The main contribution of this article is to present a prototype of an SDN-based architecture in a real-life test-bed, where we evaluate the associated implementation costs and we confirm through experimentation that novel complex services can be created with relatively low effort.**

I. Introduction

Nowadays network technologies are experiencing a shift towards *softwarization*. The key idea is to bring the flexibility and reduced cost of software development to network deployment. This idea is materialized by the Software Defined Networking (SDN) paradigm, which moves the intelligence residing in the network elements to a central controller, which implements the network functionality through software. In traditional approaches, the network's control plane is distributed throughout all network devices, while SDN logically centralizes the control plane. This removes the need of complex and costly changes in equipment or firmware updates in order to introduce new characteristics in the network, as only a change in the software running at the controller is required. The main advantage of this approach is that operators can benefit from an increased *flexibility* to manage their networks and implement new services.

Initial work on SDN focused on wired networks, though its advantages are even more important for mobile wireless networks. Indeed, in mobile networks users may change their location over time, and therefore the flexibility provided by SDN is not only beneficial for optimizing the traffic distribution inside the network, but it is also useful to adapt the way traffic is steered after users' movement. In particular, some of the benefits of adopting SDN in mobile networks include the following:

* Corresponding author. Email: luca.cominardi@uc3m.es. All authors are with Universidad Carlos III de Madrid, Spain. A. Banchs is also with Institute IMDEA Networks, Spain.

The research leading to these results has been partially performed within the framework of the H2020-ICT-2014-2 projects 5G NORMA and 5G-Crosshaul, and spanish DRONEXT project.

- *Modification of traffic engineering policies*: With SDN, an operator can easily change the traffic engineering policies implemented in the network. This can be useful for many reasons, such as for example: (i) to select a new gateway for outgoing traffic, (ii) route all traffic through a given firewall, or (iii) route certain traffic differently.
- *Online traffic optimization*: SDN does not only allow to flexibly change the traffic engineering policies but it also allows to execute them a finer granularity in terms of (i) the timing involved in taking routing decisions, and (ii) the traffic flows affected by such decisions. This allows optimizing the way traffic is distributed, adapting it as users move to new locations and load changes.
- *Creation of novel services*: SDN also allows treating packets differently based on the user or the application. This can be used to create novel services; for instance, the gateway providing connectivity to a user can be selected (among a pool of available ones) based on the location privacy preferences/requirements of the user. If SDN is combined with Network Function Virtualization, very agile services creation can be achieved, as network functions can be dynamically started and logically chained to compose customized end-to-end network services. Mobile network services can thus be deployed on the fly, allowing network operators to gain control over their network.

Note that the above includes a fairly wide wide range of services which are enabled by SDN (which we refer to as 'SDN-based services'). The main goal of this paper is to provide proofs in a real-life environment of the benefits of applying SDN concepts to deploy such services in terms of easiness, flexibility and agility when creating them. To do so, we quantify the effort involved for such service creation when using an SDN-based architecture in mid-size test-bed, confirming one of the key advantages of the proposed framework.

II. Network services: exemplary use cases

We next describe some representative examples of use cases/services for mobile networks, covering a wide spectrum: from functionality-oriented, to more complex and service-oriented ones. These exemplary use cases will be later used to evaluate the benefits, in terms of service creation time, of properly applying SDN concepts to mobile network architectures. The list of identified use cases is not meant to be exhaustive. Nevertheless, we believe that these use cases described next are relevant to the SDN context where enhanced flexibility is needed [1].

A. Smart and flexible mobility management

Future mobility management solutions will require increased *flexibility* and shall be capable of adapting to the particular characteristics of the different traffic flows as well as to the heterogeneous nature of future Radio Access Networks (RANs). Indeed, both the IETF and the 3GPP are currently working on enhanced mobility architectures and mechanisms providing this additional flexibility, e.g., enabling selective offload of selected traffic to local breakout points when possible [2], [3]. This is a clear example of the need for mechanisms enabling powerful and dynamic traffic engineering policies.

Therefore, the mobility solutions should be capable of (i) choosing the right access technology(ies) used by the connected terminals, (ii) selecting the gateways and IP addresses assigned to each flow, based on its characteristics and requirements, and, (iii) computing the forwarding paths between the radio access points and the used gateways. To achieve this goal, mobility mechanisms require an up-to-date and enriched information regarding the status of the network (e.g., gateway load). SDN is a key technology for gathering, combining, and enriching the information regarding the status of the underlying network.

To that end, [4] leverages SDN to offer connectivity management as a service (CMaaS) to application developers and over-the-top service providers to support different types of user mobility at different price levels. On another end, [5] introduces an SDN-based mobility management for integrating heterogeneous network technologies and optimizing the data transmission costs. Similarly, [6] increases network flexibility and efficiency by integrating through SDN resource, traffic, and mobility management methods of mobile network services. [7] summarizes the approach developed by the Mobile Packet Core project within the Open Networking Foundation (ONF) to integrate an SDN architecture into the mobile packet core of an operator, assuming the existence of an NFV context and proposing a unified control architecture considering both SDN and NFV. Finally, 3GPP considers SDN and cloud-based architectures since R14 to improve network management [8].

B. Location privacy

Tracking the users' location has become very common in recent years as a way to provide customized services. However, this poses several privacy issues [9] which led to the proposal of many counter-measures to preserve user location [10] [11]. Notwithstanding, latest distributed mobility management proposals,[1] which envision a flat network architecture with multiple distributed gateways, could unveil more information about the user location than desired. Indeed, such mechanisms route the traffic of a mobile user through the geographically-closest gateway for traffic optimization reason. One of the security issues raised by such solutions is that they allow tracking the (approximate) location of a mobile user, by monitoring the source IP address of her packets (which reveals the user's gateway), or the service consumed in case it is provided close to the gateway [12]. An obvious way of preserving users' location privacy is to always use the same gateway for a given user, independently of her location; however, this has a very high cost for the operator as traffic would be frequently routed through non-optimal paths.

To address the above problem, one could think of a new privacy service that works as follows: (i) for those users that want to preserve their privacy, and contract the corresponding service, a fixed gateway could be used, and (ii) for the other users, we could simply use the best gateway from a traffic engineering perspective. Note that this approach is in line with the recent developments at the IETF, where solutions are being discussed to allow taking into consideration application/user needs when selecting the right anchor/IP address [13]. Such a solution has a number of advantages: (i) it preserves privacy of those users that require it, (ii) it has a low cost for the operator in terms of traffic engineering, as efficient routes are used for most traffic, and (iii) it provides the means to the operator to receive a revenue from those users willing to contract this service. This is a good example of a new service provided by using modified traffic engineering policies.

C. Dynamic Service Composition with Network Function Virtualization

Network Function Virtualization (NFV) is a new trend that aims at transforming the way telecommunications operators build, manage and exploit their networks, relying on software virtualization techniques. NFV involves the implementation of network functions in software and its execution on non-specialized and shared hardware. Thus, CAPEX and OPEX are reduced [14], as maintenance and updating-related tasks are simplified, and new functions can be introduced via software. SDN is typically seen as complementary with NFV as: (i) NFV can support SDN by providing the infrastructure upon which the SDN software can be run, and (ii) SDN capacity to create network abstractions can help NFV achieve its goals by enhancing performance. More recently, both the NFV and SDN communities have analyzed in more detail how they can co-exist and mutually benefit [15], [16]. In an NFV context, SDN is often viewed as a tool to (*i*) enable a flexible and fast interconnection of resources at the NFV Infrastructure (NFVI) level, and (*ii*) facilitate fast configuration of connectivity of Virtual Network Functions (VNFs) at service level. We elaborate a bit more on this next.

A key feature of NFV is that it enables faster innovation by supporting dynamic, adaptive and quick service deployment. Since network functions can be run on general purpose hardware hosted on data centers, the operator can dynamically react to network and user demand changes by launching services as required and where required. This usually requires the *chaining* of simpler, independent network functions to compose a more complex service. In order to chain these (virtual) network functions, we need to be able to dynamically adjust routing in order to forward traffic to the location where the corresponding function is being executed. At this end, SDN can

[1]Examples of these approaches are the ones being developed by the IETF Distributed Mobility Management WG: https://datatracker.ietf.org/wg/dmm/

enable a much easier service function chaining/composition by automatically creating the required forwarding paths. Both ONF and ETSI NFV have acknowledged this in their latest architecture framework updates [15], [16].

D. Multi-tenancy

In order to meet the growing demands of users, network deployments are increasing their density of access points/base stations. At the same time, the cost of deploying and maintaining these new dense deployments is also increasing, reaching a point where operators are looking for innovative ways of reducing their costs. An analysis of how the adoption of NFV and SDN will impact on the reduction of operators' costs is provided in [14], where authors analyze if the intuitive statement often made about the cost reduction originated from the flexibility and simplification enabled by SDN/NFV actually holds. Authors provide a view into the operational costs of a typical service provider and then discuss how the NFV/SDN attributes can be expected to influence the business equation. One of the possible mechanisms identified to reduce costs is the sharing of the network infrastructure, moving into a world where network deployments are multi-tenant by default. In this way, several operators use the resources of a network simultaneously. This requires of complex interactions between the operators and network controllers ensuring the correct utilization, isolation and sharing of resources.

Online traffic optimization mechanisms may be used to ensure the correct isolation and sharing of network resources, so enforcing multi-tenancy in next generation networks. These mechanisms require an up-to-date view of the status of the network. Such requirement can be fulfilled by SDN which enables the continual monitoring of the network. Moreover, SDN allows the abstraction of different virtual network infrastructures, which control can be delegated to each tenant. Each operator sharing the infrastructure can operate and manage their virtual view of the network. This concept has been already explored by the Open Network Foundation in [17]. In this document, initial thoughts on how SDN, and OpenFlow in particular, can be used to provide recursion of controllers, enabling the sharing of the infrastructure. Complementary virtualization technologies (i.e., cloud sharing) are envisioned to support multi-tenancy in NFV along with SDN. Indeed, for a full multi-tenancy environment all the components in the network (switches, data centers, etc.) must support multi-tenancy. Architectures and Proof-of concept demonstrations in the literature, such as [18]–[20], further explore the integration of SDN and NFV in a multi-tenant scenario.

III. A FUNCTIONAL ONF-BASED ARCHITECTURE FOR QUICK SERVICE PROVISIONING

In this section we first provide some background on SDN related technologies, by presenting the architecture framework defined by the Open Network Foundation (ONF). Then we present and ONF-based architecture of SDN controller plane that would be implemented to experimentally assess its feasibility to significantly improve flexible and fast service creation in mobile networks.

A. ONF architecture framework

The ONF architecture specifies, at a high level, the reference points and open interfaces that enable the development of software that can control the connectivity provided by a set of network resources and the flow of network traffic though them, along with possible modification of traffic that may be performed in the network. The architecture only describes basic functions that are required, but does not preclude additional functions, allowing a wide range of scenarios and compliant implementations. Consequently, the architecture envisions particular services or applications to interrogate and manipulate the resources in the network [17].

Fig. 1 illustrates the ONF-based architecture, which is composed of four planes: Data, Controller, Application, and Management, which is transversal to the first three [17]. The *Data* plane comprehends several network resources and is in charge of handling the traffic in the data path according to the instructions received from the Controller plane. Examples of operations in this plane are switching, routing, packet encapsulation, etc. Network resources expose their capabilities and receive instructions on how to handle the traffic via the D-CPI interface which connects the Data plane with the Controller plane. The OpenFlow protocol [21] is the most widely spread protocol for this interface.

The *Controller* plane is in charge of configuring the appropriate rules on the network resources as to enforce a specific network behavior. To accomplish this task, the controller plane includes several cooperating modules devoted to the creation and maintenance of an abstract resource model of the underlying network which is then exposed to the Application plane via the A-CPI interface. Although there is no standard protocol for the A-CPI interface, it is commonly implemented through Rest API.

The *Application* plane comprises several applications/services whose main goal is to define the network behavior and may have exclusive control of a set of exposed resources (e.g., network gateway). Applications may belong either to the network operator or to clients with the former usually having a broader scope and higher privileges than the latter. It is worth noticing that applications that primarily support the operation of the data plane (e.g., network topology discovery) are not considered part of the Application plane.

The *Management* plane spans its functionality across all planes and is in charge of monitoring, configuring, and maintaining the network. These functionalities are largely the same as in the Controller plane, therefore the two planes (Management and Controller) are often seen as a continuum [17]. However, a clear distinction between the two planes resides in the entities interacting with them: a human operator in the Management plane and applications in the Controller plane.[2]

[2]An extensive discussion on the differences between Management and Controller planes can be found at [22].

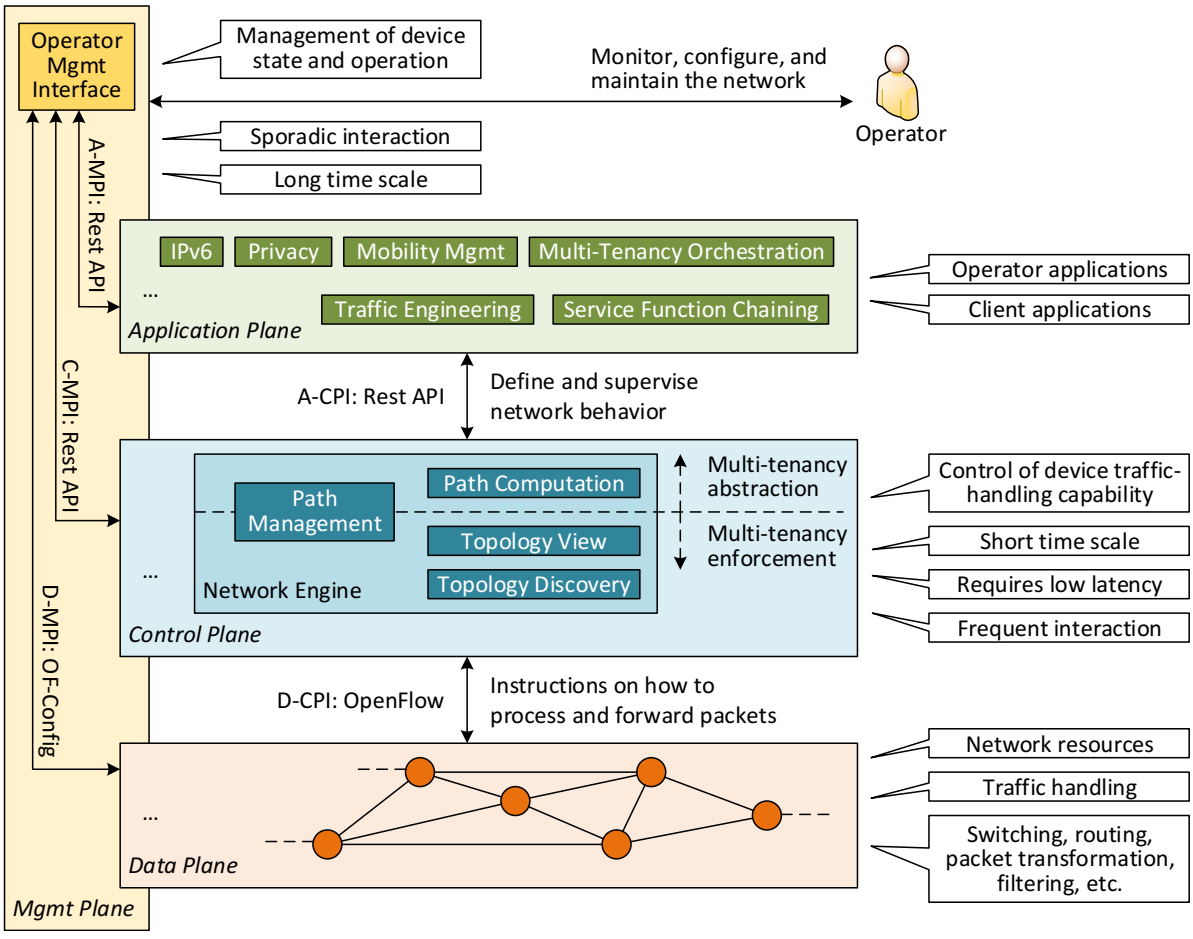

Fig. 1. Implemented SDN architecture.

Subsequently, the two planes differ in terms of (i) timescale and reactiveness, whereas the Management plane works at longer timescale than the Controller plane, and (ii) scope, with the Management having having greater scope and privilege. Indeed, it is in the scope of the Management plane to perform via the D-MPI interface the initial configuration of the network resources in the Data plane, such as the assignment of the SDN controller(s) they need to connect to and the configuration of queues and ports. This configuration is commonly done via the OF-Config protocol [23] in case of OpenFlow-capable switches. In the Controller plane, the Management uses the C-MPI interface to configure the policies defining the scope of the control given to the SDN applications, to monitor the performance of the system, and to configure the parameters required by the SDN controller modules. In the Application plane, Management configures through the A-MPI the parameters of the applications and the service level agreements. In addition to these interfaces, the Management plane provides a dashboard to network operators for configuring and tuning the network at each layer.

B. Controller plane: design and implementation

ONF provides an architecture framework and some design guidelines, which can then be taken as starting point when specifying a functional and operational system. In this section we undertake the challenge of identifying and designing those modules required for a comprehensive, implementable and operating architecture tailored to quick service provisioning. For that reason, we adopt a service-oriented architecture whereas services can be activated by triggers fired upon incoming events. Notably, the principles of service-orientation are independent of any vendor, product, technology, or implementation. In addition, we also adopt an event-driven communication paradigm as a complement of our service-oriented architecture. This paradigm enables the creation of loosely coupled software components and services while increasing responsiveness compared to asynchronous communication, being this aspect fundamental in an environment like a mobile network.

Based on the guidelines set by the general ONF framework, we designed the Controller Plane modules marked in blue in Fig. 1. These modules form the Network Engine, which supports multi-tenancy and implements the following functions: (i) discovery of the network topology and building of the resource model for the different tenants, and (ii) computation of the paths within the network and exposure of a management interface for the paths tailored to the different tenants. The Topology Discovery module is in charge of discovering the

TABLE I
CONTROLLER PLANE IMPLEMENTATION DETAILS

MODULE	DATA REQUIRED	EVENTS REQUIRED	EVENTS PROVIDED	API
Topology Discovery	Connected netw. nodes	Netw. node enter/leave/update	Netw. node enter/leave/update	Get netw. node
			Link appear/disappear/update	Get link
Topology View	Connected netw. nodes	Netw. node enter/leave/update	Netw. node enter/leave/update	Set network view
	Available links	Link appear/disappear/update	Link appear/disappear/update	Delete network view
				Get network view
				Get netw. node
				Get link
Path Computation	Network view	Netw. node enters/leaves/update	Paths computed	Set traffic class constraint
		Link appear/disappear/update		Set path
				Delete path
Path Management	Available paths	Paths computed	-	Set path for flow
				Delete path for flow

network topology, while the Topology View builds one or more abstract resource models of the network depending on the multi-tenancy configuration received from the Management plane. The Path Computation module uses these abstract resource models to compute the optimal paths for the different tenants, traffic classes, and requirements, such as latency and bandwidth. The Path Management ensures that each path set-up request received by applications can be accomplished (i.e., QoS requirements) and does not violate any constraint (i.e., maximum capacity of the links) or policy (e.g., SLA agreement of the tenant). Table I reports the implementation details of each module in terms of data and events required, and the API provided to external modules.

The *Topology Discovery* module implements the Link Layer Discovery Protocol (LLDP [24]) in order to discover the network topology. The Topology Discovery module raises an event whenever a network node enters or leaves the network, or any change occurs on the network node ports. Similarly, an event is raised whenever a link appears or disappears in the network or suffers any changes (i.e., available bandwidth).

The *Topology View* requires the knowledge of the connected network nodes and the available links in order to build an adjacency list graph representation of the network. Such representation is enriched with additional information about network nodes' capabilities (e.g., power profiles, load of the CPU and performance statistics, traffic isolation, etc.). Next, the Topology View module creates an ad-hoc network view for each tenant configured by either the Management plane or the Multi-tenancy application (see Sec. III-C). Such view may be partial and only include a subset of the physical resources or capabilities. With the purpose of maintaining an up-to-date view of the network, the Topology View module subscribes to the events provided by the Topology Discovery module. In turn, the Topology View broadcasts for each tenant (according with their current configuration) an abstracted and enriched version of the events offered by the Topology Discovery. For instance, if a link is not part of a given tenant's view, any event related to that link will not trigger a view update for that tenant. In addition, the module maintains an up-to-date vision of the network by periodically querying the status of each link and network node.

Whenever the Topology View announces a change (or update) in the network for a given tenant, the *Path Computation* module updates the network view for that tenant and (re)computes the paths within the network. The computation occurs for different traffic classes and constraints. MAC bridges [25], M2M communications [26], D2D signaling [27], and fronthaul traffic [28] are examples of traffic classes subject to different constraints. Once the module has computed the paths, it raises a path-update event for that tenant. The module is kept as simple as possible implementing a standard Dijkstra's algorithm without any further functionality. If any advanced feature is required, the module's behavior can be overridden by an external application through the exposed API (i.e., Traffic Engineering).

The *Path Management* module works on the paths provided by Path Computation (which are specific to each tenant and updated periodically) and exposes a Rest interface toward applications. This API is used to request the setup of paths within the network in the scope of a single tenant. Applications ask Path Management to set-up (or remove) a path for a given flow. A flow may have several requirements such as bandwidth, latency, packet loss and nodes to traverse. At this point, the module ensures that the path can be configured by checking whether the requirements can be fulfilled and acknowledges accordingly the requesting application. Besides, the module can be configured by external applications to apply constraints to specific requests. Finally, the module uses a combination of MPLS-TP [29], which is a widely used transport network protocol, and OpenFlow meters to enforce traffic privacy and isolation in the Data plane. We note that the choice of using MPLS-TP is mainly driven by the limitation of the OpenFlow switch implementation used for our experimental evaluation in Section IV. An alternative defined in OpenFlow [21] is Provider Backbone Bridge Traffic Engineering (PBB-TE) [30], which however is not supported in our reference OpenFlow switch implementation. More details on how to implement MPLS-TP with OpenFlow switches can be found in [31].

TABLE II
APPLICATION PLANE IMPLEMENTATION DETAILS

MODULE	DATA REQUIRED	EVENTS REQUIRED	EVENTS PROVIDED	API
IPv6	Connected switches	Switch enter/leave/update	Neighbor appear/disappear	Enable IPv6 Ndisc on switch Disable IPv6 Ndisc on switch
Mobility Management	Available gateways Nodes distance UE profiles	Switch enters/leaves/update Paths computed UE connection	UE anchors selected	Configure gateway selection
Privacy	UE profiles	UE profile update	-	-
Traffic Engineering	Network view	Switch enter/leave/update Link appear/disappear/update	-	Define path Delete path
Service Function Chaining	Network view	Switch enters/leaves/update Paths computed	-	-
Multi Tenancy	Network topology	Switch enters/leaves/update Link appear/disappear/update	-	Define network view Delete network view

See [32] for a comparison between MPLS-TP and PBB-TE.

C. Application Plane: Design and Implementation

The modules implemented in the Application Plane provide the functionality required by the use cases described before in Section II. These modules are designed as stand-alone pieces of software which subscribe to the events offered by the Controller plane and make use of the exposed APIs to trigger changes in the network. These modules are reported as green boxes in Fig. 1.

The *IPv6* module provides basic IPv6 connectivity to the UEs. This module is enabled per-tenant basis and is a client application, thus working with limited scope and privileges. The module implements the IPv6 Neighbor Discovery Protocol [33] which is responsible for features such as address autoconfiguration, duplicate address detection, and maintaining reachability information. Similarly, an IPv4 module may be implemented providing DHCP [34] and ARP [35] functions.

Mobility Management is responsible of choosing the IP gateways for the UEs in the network. Similarly to IPv6, this module is also a client application and works on per-tenant basis with limited scope and privileges. Whenever a UE connects to the tenant's network (or performs a handover), the module may select different gateways for the UE depending on her profile. For example, one gateway may be selected for real-time traffic while another for best-effort traffic. The selection is also influenced by the proximity of the gateways to the UEs. Doing so, the module offers a two-fold benefit: UEs are always served by optimal gateways and the network core is offloaded. Once the gateways have been selected, Mobility Management asks Path Management to configure the paths for the UE between the selected gateway(s) and the access point(s) the UE is attached to.

The *Privacy* module is a client application and implements the location privacy service. The aim of the service is to maintain the same gateway for those UEs that want to hide their location changes from external nodes. This can be easily achieved by overriding the default gateway selection policy of Mobility Management. Privacy assigns a fixed gateway to the privacy-enabled UEs and communicates the assignment to Mobility Management which will always select that gateway for those UEs.

The *Traffic Engineering* module is an operator application and decides how to route different traffic classes within the network. This module works with higher privileges and scopes than client applications and operates either on physical or tenants network. First, the module defines the path for a given traffic class or constraint between two points in the network. Next, the Traffic Engineering module contacts the Path Computation one, and overrides the decision made by the latter. In addition, the network operator can define manually the paths using the module's API. Finally, while many traffic engineering optimization problems have been proposed in literature [36], this module implements the linear programming formulation for MPLS networks as proposed in [37] to reduce the congestion level of the network.

The *Service Function Chaining* module is an operator application and facilitates the deployment of new services in the network. This module receives the configuration over the Management plane and contacts Path Management for configuring the constraints regarding the requested service. The Service Function Chaining module has access to the up-to-date view of available resources and connectivity from the Topology View module. This view is used to compute a logical overlay connection among the (physical and virtual) network functions composing a given service (chain). This logical overlay is then passed to the Path Management module to compute and implement the required links. The Path Management module does not only consider the logical path imposed by the service needs (e.g., order and location of the network functions that need to be interconnected), but also the requirements on the connectivity itself, e.g., in terms of bandwith, latency, isolation, geographical/topology constraints, affinity considerations, etc. For example, if a firewall is implemented as virtualized function on a gateway, the Service Function Chaining module interacts with the Path Management module, overriding its default behavior in such a way that all the UEs traffic associated to that gateway will pass through the firewall location.

TABLE III
EVALUATION OF IMPLEMENTATION EFFORT

MODULE	LINES OF CODE	TIME SPENT
Controller Modules	3218	95 hours
New Services	1589	20 hours 40 minutes
Mobility Management	356	4 hours
Privacy	153	40 minutes
Traffic Engineering	192	1 hour
Service Function Chaining	216	1 hour
Multi Tenancy	672	14 hours

TABLE IV
PROGRAMMER COMPETENCY MATRIX

SKILL / LEVEL	0	1	2	3
Data structures			X	
Algorithms			X	
System programming				X
Build automation			X	
Automated testing			X	
Problem decomposition				X
System decomposition				X
Code readability			X	
Defensive code			X	
Error handling			X	
API design				X
Framework design			X	
Requirements definition				X
Languages experience			X	
Platforms experience			X	
Domain knowledge			X	
Tools knowledge			X	
Upcoming technologies				X

The *Multi-tenancy* functionalities of the network are provided by different complementary components. On the one hand, the architecture and internal modules of the controller are multi-tenancy oriented since inception. One example of this design, can be found on the Topology View module which, by default, maintains a "real" or physical view of the network, while keeping multiple abstract "views" used by the different tenants. In addition, the allocation of resources to the data plane is done through a combination of an encapsulation supporting resource isolation (e.g., MPLS-TP) and traffic shaping (provided by the OpenFlow meter primitive and software queues configured at the resource level). On the other hand, this functionality is used in combination with the *Multi-tenancy* module. This module is responsible of managing the creation of different virtualized network views and to slice the resources in the network. The multi-tenancy module retrieves the physical network view from the Topology View module and it builds different views of the network according to the configuration received over the Management plane and to the switches capabilities (i.e., traffic isolation, resource reservation, etc.). Each view is a subset of the network resources and the module ensures that the sum of the network views' resources does not exceed the ones of the physical network by implementing a simplified version of the admission control mechanism for new tenant requests as proposed in [38].[3] Once the module assures that the view is consistent, it contacts Topology View and creates the network view for a given tenant. From this moment onwards, all the other modules (i.e., Path Computation and Path Management) will maintain multiple network views, each for tenant. Whenever a tenant contacts one of those modules, the module firstly identifies the corresponding view associated with the tenant, and secondly runs the required procedures as described in the previous section. Similar approaches to this implementation of multi-tenancy in two components, one integrated in the architecture (providing supporting functions) and a second module implemented as an application (providing the bookkeeping of resources) can be found in the literature [39].

[3]For the sake of evaluation simplicity, we only consider the network capacity constraint.

IV. PERFORMANCE AND FUNCTIONAL VALIDATION

The main goal of this paper is to provide evidence in a real-life environment of the benefits of applying SDN concepts to mobile networks in terms of easiness, flexibility and agility when deploying new services. To do so, we deployed an SDN test-bed based on off-the-shelf hardware running GNU/Linux.

A. Evaluation of service creation effort

In the following, we leverage our implementation to quantify the actual effort required for this. Even though it is generally claimed that reducing service creation time is one of the key advantages of SDN, to the best of our knowledge ours is the first attempt to quantify this benefit.

The effort required to implement a new service depends on the complexity of the task that the service aims to accomplish, and the ease with which the platform that implements it can be used. While the complexity of the task is determined by the use case, the implementation effort is highly influenced by the tools offered to the developers. The goal of this section is therefore to evaluate the developer-friendliness of our architecture. In particular, we quantify the implementation effort associated to the exemplary use cases reported in Section II.

The network controller runs Ryu[4] as component-based SDN framework. Ryu APIs natively support the event-driven communication paradigm allowing a simpler prototyping of our architecture. Moreover, the Topology Discovery module is already provided by Ryu. All the other modules of the architecture have been implemented using Python. In the next paragraphs, we evaluate the Controller and Application Plane implementations efforts.

Table III reports the *Implementation effort*, both in terms of lines of code and the development time spent. Those figures

[4]http://osrg.github.io/ryu/

are of course specific to the implementation choices we made (e.g., the use of the Ryu controller and the Python language) and to the developer, whose skills are reported in Table IV in the form of a programmer competency matrix[5] which is commonly used for assessing a specialist's competences [40]. We believe that they provide a valuable insight on the complexity of our architecture and an good estimation of the required effort, although there might be differences across developers. We divide the architecture components in two groups: the core "Controller" modules, which provide the basic functionality of the architecture, and the "New services" modules, which build on the former to implement the advanced functionality of the network. As it can be seen in the table, the core modules require a relatively large implementation cost (almost 100 hours); however, this cost is incurred only once. In contrast, the application involve a much lower cost (approx. one fifth of the core modules for all the considered functionality). In general terms, this shows that our ONF-based architecture enables network operators to provision new services with a very low service creation time. We next summarize the implementation challenges of these modules.

The implementation of the *New Services* modules required relatively little time. The Multi Tenancy module was the most challenging application to implement (more than two thirds of the overall effort): while the implementation of its interface was straightforward and its engine to ensure consistency is relatively simple, we had to devote significant time in its testing and validation to proper handling concurrent requests. The implementation of the other services required much less time: Mobility Management[6] and Privacy modules required less than 5 hours and basically consist of a smart algorithm for gateway selection, while Traffic Engineering module, which is based on a solver for the linear programming formulation,[7] required one hour (i.e., one twentieth of the total effort implementing new services). The implementation of the Service Function Chaining is another key feature of our implementation, as it provides the network operator with an interface for defining and modifying paths between network elements, requiring a similar implementation effort.

To support the development of the above modules, we leveraged continuous integration (CI) tools for streamlining the code testing and debugging. For automatic testing we used a local installation based on Docker[8] of Travis CI,[9] which provides a customizable service for building and testing software projects. A series of unit tests are hence executed to (i) verify the correctness of the modules implementation against the expected behavior of the events and API, as defined in Table I and II, and to (ii) attest the robusteness of the implementation against unexepcted inputs or events. Moreover, we used Codecov[10] for assessing the coverage of the unit tests on the implemented code as to ensure that any line being developed is properly tested. While those tools provided the necessary testing functionalities in the context of this evaluation, we acknowledge that a more complete suite, like the one proposed in [41], is advisable to better automatize the development process and thoroughly detect undesirable bugs, as highlighted in [42].

For the sake of clarity, all the controller modules evaluated in this article have been implemented on a single controller. However, regardless the implementation of each controller module, the interface toward the Application plane remains the same and does not require any change on the applications thanks to the adoption of a service-oriented architecture. Indeed, one of the key benefits of such architecture is that interactions occur between loosely coupled software components that operate independently. Moreover, this architecture allows for service reuse, making it unnecessary to rewrite all the components when upgrades/modifications are needed only affect a subset of the modules. As a result, the evaluation of the effort required to implement the applications is still accurate although being developed on a proof-of-concept.

Notwithstanding, we note that our prototype faces the same challenges of centralized systems, in terms of reliability, resiliency and scalability. Because of these issues, we envision that a deployment of the controller in a realistic scenario might be distributed across a number of separate servers. Therefore, we envision that each controller module can be implemented in a distributed fashion relying on well-known high performance distributed computing (HPDC) techniques. For example, the Topology Discovery and Topology View modules can be implemented using a divide and conquer approach where the whole network domain is split in sub-domains and the global network view is created by combining all the partial views. Path Computation might follow the same approach whereas the computed paths are exposed via a distributed hash table (DHT). In this way, the module can scale to large number of paths and to handle continuous updates and requests. Furthermore, additional applications can be implemented in order to support and interoperate with legacy modules (e.g., ANDSF, HSS, AAA, IMS) following the same approach of our implementation.

B. Experimental validation

The experimental validation of our implemented architecture is based on a test-bed composed of 14 switches, a network controller, and a UE. All the switches and the controller are interconnected through Ethernet. Moreover, 3 switches expose IP gateway capabilities and 6 switches offer IEEE 802.11b/g connectivity to the UE.

As our SDN architecture does not devise any intervention on the UE, its hardware and software requirements are simply an IEEE 802.11b/g interface, and a standard IP stack. Employing IEEE 802.3 and 802.11 as link layer technologies does not affect the implementation nor the evaluation of our architecture since the modules work on an abstract network view as described in Section III. Clearly, the adoption of a different link

[5]http://sijinjoseph.com/programmer-competency-matrix/
[6]This module is available for download at: http://odmm.net/openflow/
[7]SciPy.org, Linear Programming Solver: https://docs.scipy.org/
[8]https://www.docker.com/
[9]https://travis-ci.org/
[10]https://codecov.io/

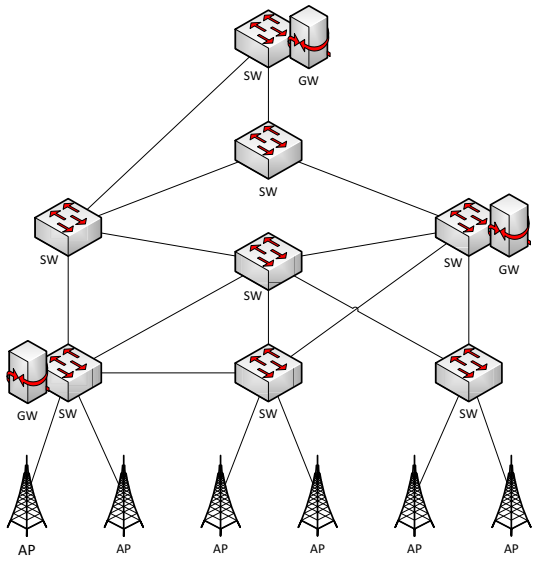

Fig. 2. SDN test-bed network topology.

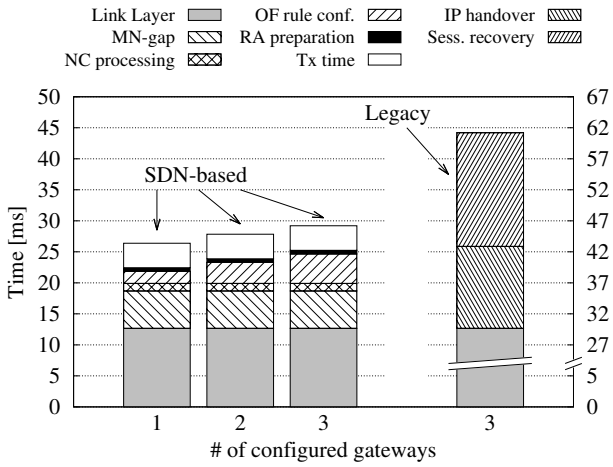

Fig. 3. Handover breakdown of SDN-based and Legacy [43] solutions in case of 1 UE and different number of configured gateways

layer technology may require some changes in the Network Engine which is in charge of building the network model.

All the switches run Open vSwitch 2.3.3 LTS[11] which provides an OF 1.3 interface. We then use Linux traffic-control[12] to provide QoS capabilities to the switches.[13] The connection between Open vSwitch and the SDN controller is performed out-of-band through standard OFP mechanisms. To evaluate the performance of our implementation, we configured the network topology shown in Fig. 2 on our test-bed. We choose Mobility Management as representative case for the performance assessment since is a key function of a mobile network, and very sensitive to incurred latency. We focus on the handover delay (i.e., the time an UE does not have connectivity as a result of a change of access point), by analyzing the components that affect more the overall latency. For this analysis, a node external to the test-bed generates *ping* traffic destined to the UE every $2ms$. We performed a total of 1000 handovers – and measured the average and standard deviation of the handover delay – by moving the UE from one access point to another, covering the cases in which the UE is simultaneously using 1, 2 and 3 gateways – so we can assess the impact of the number of gateways assigned.

The handover delay analysis can be studied looking at 3 different aspects: (i) the Link Layer handover: time elapsed since the old radio link is torn down until the new one is established; (ii) the IP Layer configuration: time required by the UE to obtain network layer connectivity (including the Link Layer handover); and (iii) the IP flow recovery: time interval during which an IP flow is interrupted due to the handover (including both the Link Layer and the IP Layer configuration). Our results show that for the case of 3 gateways, the Link Layer handover was $12.7ms$ ($\sigma = 4.4ms$), the IP Layer configuration took $25.9ms$ ($\sigma = 4.9ms$), and the IP flow recovery required $29.2ms$ ($\sigma = 6.7ms$). Fig. 3 shows all the measured results which are compared against [43], a legacy – non-SDN – implementation of an analogous mobility protocol evaluated on a test-bed of similar characteristics. It can be noticed that our solution takes few tens of milliseconds to provide the UE with the IPv6 configuration, and the Link Layer switch time is the major component. These figures are aligned with the results presented in [43]. Moreover, we observed a $6ms$ gap (MN-gap in Fig. 3) between the instant the UE completes the Link Layer handover, and the time it starts the IP Layer configuration. The processing time required by the network controller to run all the algorithms is about $1ms$ (NC-processing), while the time necessary to configure the OpenFlow rules on the network nodes is about $2ms$ for each gateway (OF rule conf.). The network controller spends $1ms$ for building and sending the IPv6 signaling used to convey the UE's IPv6 prefixes (RA preparation). The remaining time in the overall IP Layer configuration latency is due to the transmission delay (Tx time).

By comparing the above results with the standard latency target of $50ms$ for path restoration in transport networks [44], [45], we can notice that our implementation meets that target with a worst case of $35.9ms$ ($\mu = 29.2ms$, $\sigma = 6.7ms$) with 3 gateways. It is worth noticing that such result includes the time required to perform the Link Layer handover ($12.7ms$) and the MN-gap ($6ms$), which are independent of the implemented application. Nevertheless, the $50ms$ target is for the overall system and by focusing only in network controller operations, it can be noticed how the whole handover procedure is managed in about $5ms$ and depends linearly on the number of gateways and corresponding paths. On the contrary, the processing time at the network controller is constant for each UE. This result indicates that our prototyped SDN architecture and applications are able to react promptly to network events, especially because it has been obtained on commodity hardware without any kind of optimization.

[11] http://openvswitch.org/
[12] http://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html
[13] http://docs.openvswitch.org/en/latest/howto/qos/

V. Conclusions

Future networks will carry more traffic, and this traffic will exhibit disparate characteristics, imposing very different requirements and constraints on the network design. Current network architectures are very rigid and inflexible in terms of the way they manage users' traffic, and are not capable either of quickly deploying new services on demand to cope with the dynamic needs from the customers. Therefore, future network architectures should be characterized by an *enhanced flexibility*. We believe SDN is the key tool to provide this required flexibility. In this article we have adopted the general SDN framework proposed by the ONF and fully designed a compatible architecture suitable for future network operators. By implementing our proposed architecture and testing it on a medium sized test-bed, we have demonstrated how easy and quick would be for an operator to create and put into operation new (SDN-based) services.

References

[1] P. K. Agyapong and et al., "Design considerations for a 5g network architecture," *IEEE Communications Magazine*, vol. 52, no. 11, pp. 65–75, Nov 2014.

[2] Local IP Access and Selected IP Traffic Offload (LIPA-SIPTO), http://www.3gpp.org/DynaReport/23829.htm.

[3] G. Chen and et al., "Analysis of Failure Cases in IPv6 Roaming Scenarios," RFC 7445, Mar. 2015.

[4] V. Yazici and et al., "A new control plane for 5g network architecture with a case study on unified handoff, mobility, and routing management," *IEEE Communications Magazine*, vol. 52, no. 11, pp. 76–85, Nov 2014.

[5] T. T. Nguyen and et al., "Sdn-based distributed mobility management for 5g networks," in *2016 IEEE Wireless Communications and Networking Conference*, April 2016, pp. 1–7.

[6] I. Ahmad and et al., "New concepts for traffic, resource and mobility management in software-defined mobile networks," in *2016 12th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, Jan 2016, pp. 1–8.

[7] M. R. Sama and et al., "Software-defined control of the virtualized mobile packet core," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 107–115, Feb 2015.

[8] 3GPP, "Architecture enhancements for Control and User Plane separation of EPC nodes; Rel-14," 3rd Generation Partnership Project (3GPP), TS 23.214, Sep. 2017.

[9] M. Wernke and et al., "A classification of location privacy attacks and approaches," *Personal and Ubiquitous Computing*, vol. 18, no. 1, pp. 163–175, Jan 2014.

[10] V. A. Kachore and et al., "Location obfuscation for location data privacy," in *2015 IEEE World Congress on Services*, June 2015, pp. 213–220.

[11] C. J. Bernardos and et al., "Wi-fi internet connectivity and privacy: Hiding your tracks on the wireless internet," in *2015 IEEE Conference on Standards for Communications and Networking (CSCN)*, Oct 2015, pp. 193–198.

[12] T. He and et al., "Location privacy in mobile edge clouds: A chaff-based approach," *IEEE Journal on Selected Areas in Communications*, vol. PP, no. 99, pp. 1–1, 2017.

[13] A. Yegin and et al., "On Demand Mobility Management," draft-ietf-dmm-ondemand-mobility-12, Jul. 2017.

[14] E. Hernandez-Valencia and et al., "How will nfv/sdn transform service provider opex?" *IEEE Network*, vol. 29, no. 3, pp. 60–67, May 2015.

[15] ONF, "Relationship of SDN and NFV," TR-518, Open Networking Foundation, Oct. 2015.

[16] ETSI NFV, "Report on SDN Usage in NFV Architectural Framework," NFV-EVE 005, Dec. 2015.

[17] ONF, "SDN architecture, Issue 1.1," TR-521, Open Networking Foundation, 2016.

[18] R. Vilalta and et al., "Network virtualization controller for abstraction and control of openflow-enabled multi-tenant multi-technology transport networks," in *Optical Fiber Communication Conference*. Optical Society of America, 2015.

[19] X. Li and et al., "5g-crosshaul network slicing: Enabling multi-tenancy in mobile transport networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 128–137, 2017.

[20] A. Mayoral and et al., "Multi-tenant 5g network slicing architecture with dynamic deployment of virtualized tenant management and orchestration (mano) instances," in *ECOC 2016; 42nd European Conference on Optical Communication*, Sept 2016, pp. 1–3.

[21] ONF, "OpenFlow Switch Specification, Version 1.3.2," TS-025, Open Networking Foundation, 2013.

[22] E. Haleplidis and et al., "Software-Defined Networking (SDN): Layers and Architecture Terminology," RFC 7426, Jan. 2015.

[23] ONF, "OpenFlow Management and Configuration Protocol," TS-016, Open Networking Foundation, 2014.

[24] IEEE, *IEEE 802.1ab-rev - Station and Media Access Control Connectivity Discovery*, IEEE Std. 802.1ab, 2009.

[25] ——, *IEEE 802.1d - MAC bridges*, IEEE Std. 802.1d, 2004.

[26] ETSI, *Machine-to-Machine communications (M2M); M2M service requirements*, ETSI Std. TS 102 689, 2013.

[27] 3GPP, "Study on enhancements for infrastructure-based data communication between devices; Rel-13," 3rd Generation Partnership Project (3GPP), TS 22.807, Sep. 2014.

[28] IEEE, *IEEE 1914.3 - Radio over Ethernet (PAR for a New IEEE Standard)*, IEEE Std. 1914.3, 2016.

[29] M. Bocci and et al., "A Framework for MPLS in Transport Networks," RFC 5921, Jul. 2010.

[30] IEEE, *Provider Backbone Bridge Traffic Engineering*, IEEE Std. 802.1Qay, 2009.

[31] J. Medved and et al., "MPLS-TP Pseudowire Configuration using OpenFlow 1.3," draft-medved-pwe3-of-config-01, Jul. 2012.

[32] R. Vaishampayan and et al., "Application driven comparison of t-mpls/mpls-tp and pbb-te - driver choices for carrier ethernet," in *IEEE INFOCOM Workshops 2009*, April 2009, pp. 1–6.

[33] T. Narten and et al., "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861, Sep. 2007.

[34] R. Droms and et al., "Dynamic Host Configuration Protocol," RFC 2131, Mar. 1997.

[35] D. C. Plummer, "An Ethernet Address Resolution Protocol," RFC 892, Nov. 1982.

[36] A. Mendiola and et al., "A survey on the contributions of software-defined networking to traffic engineering," *IEEE Communications Surveys Tutorials*, vol. 19, no. 2, pp. 918–953, Secondquarter 2017.

[37] A. Mereu and et al., "Primary and backup paths optimal design for traffic engineering in hybrid igp/mpls networks," in *2009 7th International Workshop on Design of Reliable Communication Networks*, Oct 2009, pp. 273–280.

[38] V. Sciancalepore and et al., "Mobile traffic forecasting for maximizing 5g network slicing resource utilization," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[39] X. Li and et al., "5g-crosshaul network slicing: Enabling multi-tenancy in mobile transport networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 128–137, 2017.

[40] V. Lytvyn and et al., "A method for constructing recruitment rules based on the analysis of a specialist's competences," *Eastern-European Journal of Enterprise Technologies*, vol. 6, no. 2, pp. 4–14, December 2016.

[41] P. A. A. Gutirrez and et al., "NetIDE: All-in-one framework for next generation, composed SDN applications," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, June 2016, pp. 355–356.

[42] L. J. Jagadeesan and et al., "Programming the network: Application software faults in software-defined networks," in *2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Oct 2016, pp. 125–131.

[43] F. Giust and et al., "Analytic evaluation and experimental validation of a network-based ipv6 distributed mobility management solution," *IEEE Transactions on Mobile Computing*, vol. 13, no. 11, pp. 2484–2497, Nov 2014.

[44] ITU-T, "Ethernet linear protection switching," ITU Telecommunication Standardization Sector, Recommendation G.8031/Y.1342, Jan. 2015.

[45] B. Niven-Jenkins and et al., "Requirements of an MPLS Transport Profile," RFC 5654, Sep. 2009.