Virtual Network Functions Orchestration in Enterprise WLANs

Roberto Riggio*, Tinku Rasheed*, Rajesh Narayanan†
*CREATE-NET, Italy; Email: rriggio@create-net.org
†Dell Inc., U.S.A.; Email: n_rajesh@dell.com

***Abstract*—Network Function Virtualization (NFV) has recently been proposed as a tool to optimize the deployment of network functions by shifting the processing from dedicated middleboxes to general purpose and inexpensive hardware platforms. In this paper, we propose a novel NFV–based management and orchestration framework for enterprise WLANs. Our framework is compatible with the ETSI NFV architecture and leverages on hybrid nodes combining the forwarding capabilities of a programmable switch with the storage/computational capabilities of a server. We propose an algorithm for Virtual Network Function placement which optimizes the functions deployment according to application level constraints. A proof-of-concept implementation of the proposed framework and a preliminary performance evaluation of selected VNFs are also presented.**

I. Introduction

Network Function Virtualization (NFV) promises to reduce the cost to both deploy and operate large infrastructures by replacing the dedicated and typically proprietary hardware that is currently used to run network services with general purpose and off–the–shelf components and by allowing network services such as Deep Packet Inspection, Firewalls, Load Balancing, etc., to be hosted on virtual machines (VMs).

At the same time Software–Defined Networking is reshaping the way networks are operated by providing operators and administrators with a programmatic interface to control the switching fabric from a logically centralized controller. SDN solutions have also recently appeared in the wireless networking domain. In our previous work [1] we tackled the challenges of providing network operators with high–level and expressive programming abstractions to control their enterprise WLANs. The proposed solution effectively allowed to deploy new services as *Network Apps* on top of a WLAN controller.

However, in [1] we focused on the primitives that are needed to program an enterprise WLAN and we did not address the broader challenge of providing an NFV orchestration framework. Such a scenario requires in fact the possibility to deploy and orchestrate the behavior of Virtual Network Functions (VNF) whose scope goes beyond the tasks of a WLAN controller. Similarly, *Network Apps* running on top of a WLAN controller may require further intelligence to be deployed within the network in order to achieve the desired results, e.g. a load balancing VNF may involve both handover and resource management logic to be implemented in the radio access network as well as traffic shaping and prioritization policies to be deployed in the backhaul.

The recent advances in general purpose computing platforms paved the way to a new generation of software routers. However, many of these solutions focus on improving the raw packet processing speed [2], [3], [4] but do not tackle the problem of deploying and orchestrating VNFs. In parallel there are significant efforts toward the management and orchestration of VNFs. In particular the European Telecommunications Standards Institute (ETSI) has recently tackled the NFV concept [5] while the OPNFV project [6] is working toward an open source carrier grade platform for NFV.

In this paper, we present a VNF management and orchestration framework for enterprise WLANs. The framework relies on hybrid nodes supporting both forwarding and computational/storage capabilities. Moreover, we evaluate the performances of a VNF placement algorithm which is in charge of deciding where VNFs must be placed within the network in order to satisfy application level constraints (typically latency). Finally, we report on a preliminary proof–of–concept implementation of the proposed framework deployed over a small scale (20–nodes) programmable enterprise WLAN testbed.

The reminder of this paper is structured as follows. Section II presents the physical network model, the VNF request model, and the VNF placing algorithm. The proposed algorithm is evaluated in Sec. III. The proof–of–concept and its validation are presented in Sec IV. Finally, Sec. V draws some preliminary conclusions pointing out the future work.

II. Network Model

In the VNF placement problem the input consists of Service Function Chains (SFC) consisting of a variable number of VNFs, whereas the substrate network provides the physical constraints in terms of bandwidth and capacity. Note that in this context the term capacity is not related only to pure computational resources such as number of CPU cores and memory. On the contrary it refers to packet forwarding and processing capabilities. From an architectural standpoint the VNF placement algorithm resides in the *Orchestrator* and is the engine of the resource allocation component in the Network Function Virtualization Infrastructure (NFVI). Before introducing the proposed solution we need to detail specific notations for the NFVI and the SFC requests.

A. Network Function Virtualization Infrastructure Model

Let $G_{nfvi} = (N_{nfvi}, E_{nfvi})$ be an undirected graph modeling the physical network, where N_{nfvi} is the set of $n =$

$|N_{nfvi}|$ physical nodes that compose the substrate network and E_{nfvi} is the set of edges or links. An edge $e^{nm}_{nfvi} \in E_{nfvi}$ if and only if a point–to–point connection exists between $n \in N_{nfvi}$ and $m \in N_{nfvi}$. With respect to the physical network, links are actual wiring media, e.g., an Ethernet cable interconnecting the two nodes[1]. Two weights, $w_p(n), w_r(n)$, are assigned to each node $n \in N_{nfvi} : w_{r,p}(n_{nfvi}) \in \mathbb{N}^+$ representing the packet and radio processing resources *available* on that node; for the sake of simplicity we assume that all resources can be expressed as an integer number. Nodes with both weights equal to 0 (zero) are assumed to be pure packet forwarding nodes. Another weight $b(e_{nfvi})$ assigned to each link $e_{nfvi} \in E_{nfvi} : b(e_{nfvi}) \in \mathbb{N}^+$ represents the capacity of the link connecting two nodes. In order to avoid exceeding the nominal capacity of the substrate links, traffic shaping is implemented at the nodes with packet and radio processing capabilities, i.e. nodes with $w_p(n) \geq 0$ or $w_r(n) \geq 0$. Finally, let P be the set of all substrate paths and $P(s,t)$ the set of all substrate paths between nodes s and t.

A sample substrate network is sketched in Fig. 1: as shown, the physical network is composed by 16 nodes interconnected together. In order to improve readability link weights have been omitted. The substrate network in this example consists of 6 nodes supporting both radio and packet processing capabilities (at the bottom of the picture), 4 switches, 2 of which supporting just basic forwarding capabilities, and 6 packet processing nodes (at the top of the picture). As we will see in Sec: IV, nodes with packet and/or radio processing capabilities consist of general purpose embedded platforms running multiple instances of the Click Modular Router [7]. The weights w_p associated to the substrate nodes represent the number of concurrent Click instances supported by that node, while the weights w_r is specific to WiFi–enabled nodes and represent the number of users that can be handled by that node. For example the weights $w_r = 50, w_p = 10$ means that the node can handle up to 50 active wireless terminals and up to 10 Click instances. Similarly the weight $w_p = 50$ associated to the pure packet processing nodes signifies that the node can run up to 10 parallel Click instances.

The resources required to embed a request G_{sfc} onto a substrate network G_{nfvi} are quantified using the substrate node (N^P, N^R) and edge (E^S) stress [8] defined as:

$$\begin{aligned} N^P(n_{nfvi}) &= \sum_{n_{sfc} \rightarrow n_{nfvi}} w_p(n_{sfc}) \\ N^R(n_{nfvi}) &= \sum_{n_{sfc} \rightarrow n_{nfvi}} w_r(n_{sfc}) \\ E^S(e_{nfvi}) &= \sum_{e_{sfc} \rightarrow e_{nfvi}} b(e_{sfc}) \qquad (1) \end{aligned}$$

We define the residual node capacity as the available radio $\rho^R(n_{nfvi})$ and packet processing $\rho^P(n_{nfvi})$ capacity of the substrate node $n_{nfvi} \in N_{nfvi}$:

$$\rho^P(n_{nfvi}) = w_p(n_{nfvi}) - N^P(n_{nfvi})$$

[1]In this work we consider undirected links for simplicity

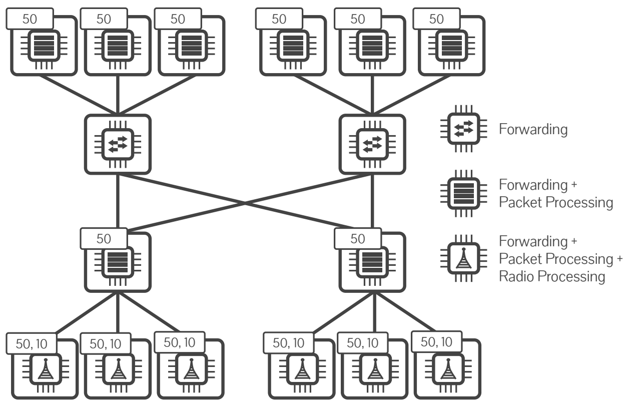

50
50
50
50
50
50
Forwarding
Forwarding + Packet Processing
50
50
Forwarding + Packet Processing + Radio Processing
50, 10
50, 10
50, 10
50, 10
50, 10
50, 10

Fig. 1: Substrate network model used in our NFV architecture. The figure shows the three basic virtual resources: forwarding, packet processing, and radio access.

$$\rho^R(n_{nfvi}) = w_r(n_{nfvi}) - N^R(n_{nfvi})$$

Likewise, we define the residual capacity of a substrate link $\rho^E(e_{nfvi})$ as the total amount of resources available on the substrate link $e_{nfvi} \in E_{nfvi}$:

$$\rho^E(e_{nfvi}) = b(e_{nfvi}) - E^S_{nfvi}(e_{nfvi})$$

Finally, we can define the available bandwidth of a substrate path $p \in P^S$ as the residual capacity of the bottleneck link

$$\rho^E(p) = \min_{e_{nfvi} \in P} \rho^E(e_{nfvi})$$

B. Service Function Chain Requests

Users are allowed to request SFC as a *undirected* and *acyclic* graph $G_{sfc} = (N_{sfc}, E_{sfc})$. Where N_{sfc} denotes the set of nodes and $E_{sfc} \subseteq N_{sfc} \times N_{sfc}$ denotes the set of links. Notice that as opposed to the previous NFVI model, nodes in SFC requests represent virtual network functions through which packets must undergo before leaving the network. Nodes and links in the SFC request shares the same weights as for the NFVI substrate network.

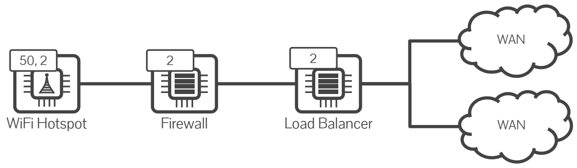
The *Orchestrator* is in charge of deciding whether a particular SFC can be accepted or if it must be refused. If a request is accepted then the *Orchestrator* is in charge of mapping the request onto the substrate network, i.e., network resources must be allocated and configured on both the substrate nodes and the substrate links and the VNFs must be instantiated on the selected nodes. The embedding of a SFC request G^V onto the substrate network is subject to the following constraints:

- *Node assignment*. Each node in the SFC request is mapped to a different substrate node with sufficient capacity. The mapping function $M_N : N_{sfc} \rightarrow N_{nfvi}$ from virtual nodes to substrate nodes is such that $\forall\ n_{sfc}, m_{sfc} \in N_{sfc}$,

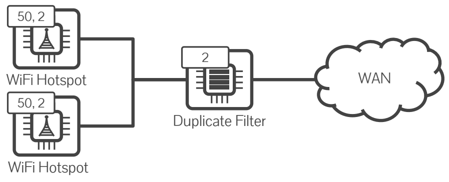
$$\begin{aligned} &M_N(n_{sfc}) \in N_{nfvi} \\ &M_N(m_{sfc}) = M_N(n_{sfc}), \quad \text{iff } m_{sfc} = n_{sfc} \ (2) \end{aligned}$$

subject to:

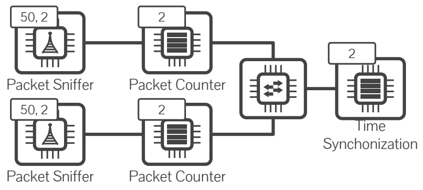
$$w_p(n_{sfc}) \leq \rho^P(M_N(n_{sfc})), \quad w_r(n_{sfc}) \leq \rho^R(M_N(n_{sfc}))$$



(a) Enterprise WLAN.



(b) WiFi duplicates filtering.



(c) WiFi network monitoring.

Fig. 2: Sample Service Function Chains Requests.

- *Link assignment*. Each virtual link is mapped to a *single* substrate path between the substrate nodes on top of which the two endpoints of the virtual link have been mapped. Only substrate paths with sufficient capacity on their bottleneck links are considered. Link assignment is defined by a mapping function $M_E : E_{sfc} \to P_{vfni}$ from virtual links to substrate paths such that $\forall\ e_{sfc} = (m_{sfc}, n_{sfc}) \in E_{sfc}$,

$$M_E(m_{sfc}, n_{sfc}) \subset P(M_N(m_{sfc}), M_N(n_{sfc}))$$

subject to:

$$\sum_{P \in M_E(e_{sfc})} \rho^E(p) \geq b(e_{sfc})$$

A few sample SFC requests are sketched in Fig. 2. Notice that wireless terminals are not represented in that they are outside the control of the orchestration framework. The SFC request in 2a consists of three VNFs including a WiFi hotspot, a firewall, and a load balancer. The WiFi hotspot request will also include parameters such as the name of the network and the authentication parameters (e.g. type of encryption, RADIUS server to be used, etc.) however these kind of information are purely functional and are thus omitted in this section. The SFC requests in Fig. 2b and 2c represents respectively a performance enhancing VNF and a wireless channel monitoring VNF.

C. Virtual Network Function Placement

The objective of our algorithm is to embed multiple SFC requests consisting of VNFs and links on top of a substrate network. We assume that, the substrate consists of tree–like networks, e.g. *fat–tree*, commonly found in datacenters and in enterprise/campus networks. SFC requests coming from users are randomly picked among the ones depicted in Fig. 2.

Algorithm 1 Virtual Network Function Placement Algorithm

```
1: procedure EmbedRequest($G^V, G^S$)
2:     $n_{sfc} \leftarrow$ SelectHighestDegreeVNF($G_{sfc}$)
3:     $n_{nfvi} \leftarrow$ SelectHighestDegreeNode($G_{nfvi}$)
4:     $M(n_{sfc}) \leftarrow n_{nfvi}$                          ▷ embed VNF
5:     $\phi(n_{nfvi}) \leftarrow 1$                    ▷ mark NFVI node as used
6:     if not EmbedVNF($n_{sfc}$) then
7:         Reject($G_{sfc}$)
8:     end if
9: end procedure
```

Algorithm 2 VNF Embedding Algorithm (Recursive)

```
1: procedure EmbedVNF($n_{sfc}$)
2:     $\psi(n_{sfc}) \leftarrow 1$                         ▷ mark VNF as visited
3:     for all $\{m_{sfc} \in neighbors(n_{sfc}) \mid \psi(m_{sfc}) == 0\}$ do
4:         $\Theta_{nfvi} = G_{nfvi} \setminus \{n_{nfvi} \in N_{nfvi} \mid \phi(n_{nfvi}) == 0\}$
5:         if $\Theta_{nfvi} == \emptyset$ then        ▷ No substrate node available
6:             return False
7:         end if
8:         $m_{nfvi} = \underset{m_{nfvi} \in \Theta_{nfvi}}{argmin} [W_{nfvi}(e_{sfc}, m_{nfvi})]$
9:         if $m_{nfvi} == \emptyset$ then
10:            return False
11:        end if
12:        $M(m_{sfc}) \leftarrow m_{nfvi}$                      ▷ embed node
13:        $\phi(m_{nfvi}) \leftarrow 1$                     ▷ mark node as used
14:        Allocate path between $M(m_{sfc})$ and $M(n_{sfc})$
15:    end for
16:    for all $\{m_{sfc} \in neighbors(n_{sfc}) \mid \psi(m_{sfc}) == 0\}$ do
17:        if not EmbedVNF($m_{sfc}$)) then
18:            return False
19:        end if
20:    end for
21:    return True
22: end procedure
```

The *Orchestrator* implements a recursive greedy algorithm based on a Breadth–first traversal of both the substrate network and the SFC request. At each step the algorithm tries to map a VNF to the substrate node that minimizes a virtual edge stress metric. The visit on both substrate network graph and on the SFC request graph begins from the two highest degree nodes.

The algorithm (see Alg. 1) begins by picking two nodes $n_{sfc} \in N_{sfc}$ and $n_{nfvi} \in N_{nfvi}$. Then, it maps n_{sfc} on n_{nfvi} and starts a breadth–first traversal (see Alg. 2) on the VNF request starting at virtual node n_{sfc}. At each step the visited node $m_{sfc} \in N_{sfc}$ is mapped to the substrate node with the minimum virtual edge stress. We define the virtual edge stress $W : E_{sfc} \times N_{nfvi} \to \mathbb{R}$ between a virtual edge $e_{sfc} = (n_{sfc}, m_{sfc}) \in E_{sfc}$ and a substrate node $m_{nfvi} \in$

N_{nfvi} as follows:

$$W(e_{sfc}, m_{nfvi}) = (1-\alpha-\beta) \min_{e_{nfvi}} \left[\rho^E(e_{nfvi}) - b(e_{sfc})\right] + \alpha\left[\rho^P(m_{nfvi}) - w_p(m_{sfc})\right] + \beta\left[\rho^R(m_{nfvi}) - w_r(m_{sfc})\right] \quad (3)$$

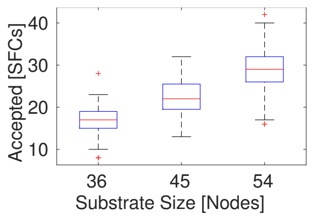
where $e_{nfvi} \in P(M_N(n_{sfc}), m_{nvfi})$. This distance is the convex combination of the residual capacity of the substrate node m_{nfvi} after mapping the virtual node m_{sfc} and the residual bandwidth on the bottleneck substrate link e_{nfvi} after mapping the virtual link e_{sfc}. The parameters $\alpha, \beta : 0 \le (\alpha+\beta) \le 1$ can be used to give priority to the residual capacity or to the residual bandwidth. For example, if $\alpha = 0.5$ and $\beta = 0.5$ the algorithm will embed the virtual node m_{sfc} to the substrate node with the highest residual capacity, instead if $\alpha = \beta = 0$ the algorithm will try to embed the virtual node m_{sfc} to the substrate node whose path from $M(m_{nfvi})$ has the smallest residual bottleneck link capacity.

After mapping the node $M_N(m_{sfc}) \leftarrow m_{nfvi}$ the algorithm maps the virtual edge e_{sfc} to the shortest path between $M_N(n_{sfc}) = n_{nfvi}$ and $M_N(m_{sfc}) = m_{nfvi}$. The procedure stops when all the virtual nodes have been visited or if the substrate network cannot accommodate the request. The latter case can happen if there are not enough nodes in the substrate topology ($\Theta_{nfvi} == \emptyset$) or if either the capacity or the link bandwidth has been exhausted.

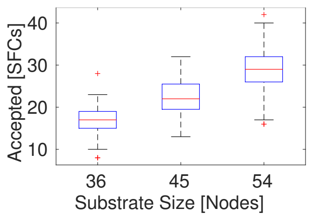
We observe that the algorithm has complexity $O((N_{nfvi})^2 \log N_{nfvi})$ in the number of substrate nodes, since it visits all N_{sfc} nodes of the input SFC, which are N_{nfvi} in the worst case, and for each node it then builds a spanning tree rooted at visited substrate node, at a cost $O(N_{nfvi} \log N_{nfvi})$; in the case of a bounded size N_{sfc} for the input SFC, the complexity indeed becomes $O(N_{nfvi} \log N_{nfvi})$.

III. Evaluation

In this section we shall first describe the simulation environment and then the performance metrics used for evaluating the VNF placement algorithm. The goal of this evaluation is to study the impact of SFC complexity on the NFVI substrate. Simulations are carried out in a simulator implemented in Matlab®. In our simulations we assume that SFC requests are embedded sequentially until the substrate network resources are exhausted. The reference substrate network topology is k–ary fat–tree with $k = 4, 6, 8$, where leaf nodes are WiFi Access Points (APs) rather than hosts. This results in a total of, respectively, 16, 54, and 128 WiFi APs. Two types of SFCs are considered in this evaluation: linear (see 2a) and star–shaped (see 2b). The number of nodes in each SFC is for each request is uniformly distributed between $[2, 5]$. In our experimental setup we used the values $\alpha = 0.45$ and $\beta = 0.1$ in order to weight equally between packet processing capacity and bandwidth giving minor weight to radio processing capabilities.

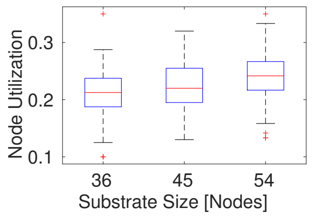


(a) Linear topologies.

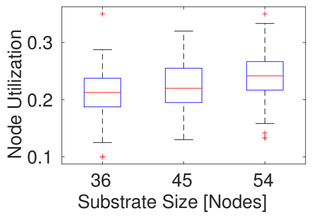


(b) Star topologies.

Fig. 3: Number of accepted SFCs for different substrate sizes.

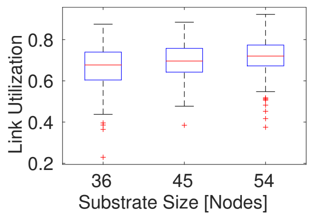


(a) Linear topologies.

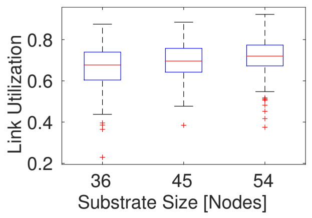


(b) Star topologies.

Fig. 4: Node utilization for different substrate sizes.



(a) Linear topologies.



(b) Star topologies.

Fig. 5: Link utilization for different substrate sizes.

The packet processing and the bandwidth resources on the substrate nodes are uniformly distributed between 50 and 100. The packet processing and bandwidth requirements for SFC requests are uniformly distributed between, respectively, 1 and 20, and 1 and 50. The evaluation metrics used in this study are standard ones adopted in several other related work (see, e.g., [9], [10], [11]). Namely the number of accepted SFC requests and the average node and link utilization of the substrate network computed as the averages of, respectively, the node stress and the link stress (1).

Figure 3 shows the absolute number of accepted SFC requests for the different substrate networks. As expected the number of accepted SFC requests increases with the substrate size. Moreover, as it can be seen in Fig. 4 and Fig. 5 the average node and link utilization at the end of the embedding, i.e. when the last SFC has been placed, also increases with the size of the NFVI substrate. This is promising in that it signifies that the algorithm is capable of making better use of the network resources available in larger infrastructures.

IV. Implementation

We prototype the NFV architecture discussed in this work in order to show its suitability to address the requirements of practical enterprise WLAN environments. In this section we shall first describe the system architecture and its design

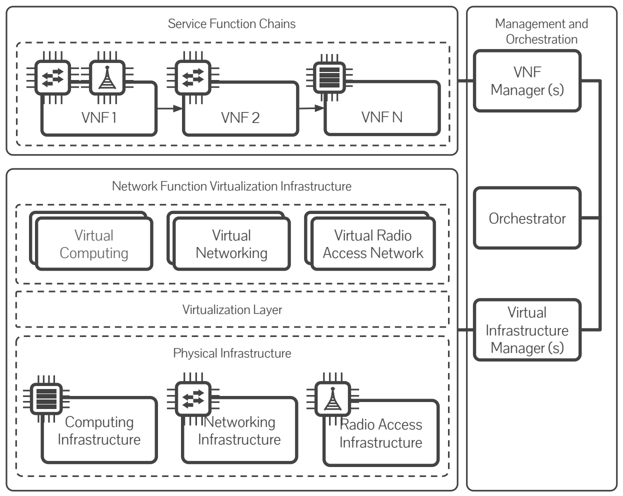

Service Function Chains
VNF 1
VNF 2
VNF N
Management and Orchestration
VNF Manager (s)
Orchestrator
Virtual Infrastructure Manager (s)
Network Function Virtualization Infrastructure
Virtual Computing
Virtual Networking
Virtual Radio Access Network
Virtualization Layer
Physical Infrastructure
Computing Infrastructure
Networking Infrastructure
Radio Access Infrastructure

Fig. 6: Reference network function virtualization architecture [5].

choices, then we will report on the implementation of a sample NFV application and on its evaluation.

A. Orchestration Framework Architecture

The proposed framework is a preliminary implementation of the ETSI reference NFV Architecture [5]. As it can be seen in Fig. 6, the architecture is conceptually divided into three layers. The bottom layer represents the NFV Infrastructure (NFVI). The second layer includes the virtualized resources exposed by a virtualization layer. Finally, in the third layer we have the actual VNFs which are the software implementation of a particular network function which is capable of being executed over the NFVI. The Management and Orchestration plane covers the orchestration and the management of physical and/or virtual resources that support the NFV infrastructure as well as the lifecycle management of the single VNF, i.e. creation, configuration and destruction.

Our architecture currently accounts for three kinds of NFVI resources, namely: basic forwarding nodes, packet processing nodes which support advanced packet manipulation, and radio processing nodes which, in addition to the features supported by the generic packet processing node, also embed specialized hardware in the form of one or more 802.11 Wireless NICs. All packet processing nodes leverage general purpose computing platform and operating systems.

B. Prototype Design

Packets processing. In our prototype we use Click [7] as a single solution for advanced packet processing. Click allows to build complex VNFs using simple and reusable components, called *elements*. Click includes over 300 elements supporting functions such as packet classification, access control, deep packet inspection. Elements can be composed in order to realize complex functionalities. Finally, Click is easily extensible with custom processing elements making possible to support features that are not provided by the standard elements.

Data plane. We implemented our prototype on top of general purpose and off–the–shelf components. In particular the switching fabric is composed of Pronto 3290 OpenFlow switches supporting pure forwarding capabilities and Dell Split Data Path (SDP) switches [12]. The SDP architecture combines the features of merchant silicon switching with the features of advanced and deep flow programmable network processor. SDP nodes essentially consist in standard OpenFlow switches combined with an host processor capable of running a Linux distribution (Debian–based in our case). We exploited this capability to dynamically deploy instances of the Click Modular Router that are in charge of performing the desired network functions. Finally, also the WiFi data–path has been implemented as a VNF running on general purpose embedded computing modules (PCEngines ALIX).

Management plane. As Virtual Infrastructure Managers (VIM) we use a combination of frameworks. Floodlight is used in order to configure resources in the switching fabric. The SD–RAN controller proposed by the authors in [1] is used in order to control the wireless VNF instances. Finally, a proof–of–concept *Orchestrator* has been implemented on top of the Tornado Framework [13]. This *Orchestrator* provides a web interface trough which the users can specify their SFC by composing the available VNFs. Additional VNFs can also be uploaded by the users in the form of Click scripts. The request is then mapped on the NFVI substrate by the *Orchestrator* which then deploys the VNFs on the selected nodes and configures the VIMs. Notice how, although the current *Orchestrator* relies on the north–bound interfaces exposed by the various VIMs, its architecture can be easily extended in order to accommodate for other managers such as OpenStack and OpenDaylight.

C. VNF: Wireless Link Monitoring

Resource allocation in wireless networks is a complex task due to the stochastic nature of the of the wireless channel. For example, performing handover decisions using as an input only the RSSI strength between clients and APs can lead to sub–optimal performances. This is due to the fact that RSSI measurements are a poor indicators for link–layer performance metrics such as frame loss. However, if the downlink frame loss can be easily tracked by the APs to which a client is currently associated, tracking the uplink frame loss rate is much harder. This is due to the fact the amount of frames actually sent by a client is an unknown variable at the network side. Implementing a wireless link monitoring VNF capable of estimating in real–time the uplink delivery probability would allow network operators to gain a real–time knowledge about the link conditions which in time can be leveraged to drive resource allocation and handover decisions.

The proposed wireless link monitoring VNF, sketched in Fig. 2c, leverages on the broadcast nature of a wireless link and in particular on the fact that each transmission can be recorded by multiple in–range APs. *Packet Capture* VNFs are deployed at different WiFi APs where they can track the meta–data associated to any link–layer event. In particular for each transmission the following information are gathered:

- *Transmitter Address*. The MAC address of the transmitter.

- *TSFT*. The 802.11 MAC's 64-bit Time Synchronization Function Timer. Each frame received by the radio interface is timestamped with a $1\mu sec$ resolution clock.
- *Sequence*, The 802.11 MAC's 16-bit sequence number. This counter is incremented by the transmitter after a successful transmission.
- The frame *RSSI* (in dB), *Rate* (in Mb/s), *Length* (in bytes), *Duration* (in μsec), *Type* and *subtype*,

The collected meta–frames are then forwarded to a *Packet Counter* VNF where they are aggregated by source address and transmission rate. Aggregated meta–frame are then delivered to a *Time Synchronization* VNF. This VNF aims at building a globally synchronized view of all link–layer events in the network. The VNF implements a synchronization algorithm that does not require the WiFi APs to share a common clock (which would be impractical for μsec–level precision). The algorithm looks for simultaneous receptions of the same frame (identified using the sequence number contained in the 802.11 MAC header) at different points in order to synchronize the timestamp of the various frames to a common reference. Once the global clock synchronization has been achieved, the time synchronization VNF can track in real–time the frame delivery rates between one or more wireless terminals and the APs.

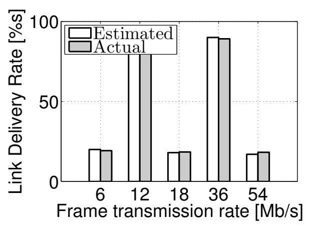
We evaluated the accuracy of this VNF with a network setup composed of a single client and three WiFi APs. Traffic is injected from the wireless client as a single UDP stream. Measurements are taken for different transmission rates. Impairments on the link between client and the WiFi APs are simulated by dropping received frames with probability p. For all measurements a total of 6000 frames were generated. Confidence interval were very small for all the measurements and have been omitted to improve readability. The packet generation rate and payload are kept fixed at, respectively, 100 packets/s and 1472 bytes.

Figure 7 reports the actual and estimated frame delivery rates at the three APs in the network for different transmission rates. During this campaign the dropping probability of two links has been set to $p = 0.8$ (poor channel conditions) while the dropping probability of the third link has been set to 0.05 (good channel conditions). This scenario is representative of the case where at least one of the in–range receivers is experiencing a good channel condition. As it can be seen, also in this case the VNF can closely track the actual link delivery rates for all the transmission rates.

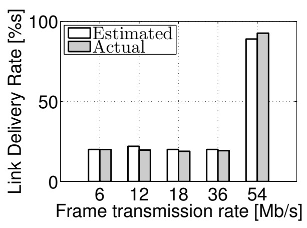
V. Conclusions

In this paper we presented a NFV–based management and orchestration framework for enterprise WLANs. Its architecture is compatible with the generic ETSI NFV model. We also moved the first steps towards the definition of a SFC request model for enterprise WLANs and we designed and evaluated a VNF placement algorithm. Finally, we reported on a preliminary implementation and evaluation of the proposed framework over small scale indoor testbed.

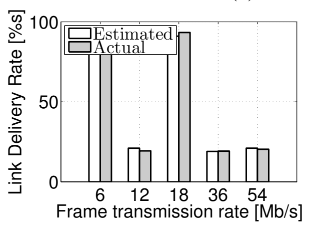
As future work we want to generalize the VNF request model in order to account for packet processing functions with different execution complexity. Moreover, we also plan to devise a more detailed resource request model for wireless resources which can allow the user to describe the performance he/she expects from the network leaving to the *Orchestrator* the task of allocating/deallocating VNFs dynamically in order to meet the input requirements.



(a) Packet Sniffer 1.



(b) Packet Sniffer 2.



(c) Packet Sniffer 3.

Fig. 7: Actual and estimated frame loss at the three Packet Sniffers for different transmission rates where at least one of the in–range receivers is experiencing a good channel condition.

References

[1] R. Riggio, T. Rasheed, J. Schulz-Zander, S. Kuklinski, and M. K. Marina, "Programming Software–Defined Wireless Networks," in *Proc. of IEEE CNSM*, 2014.

[2] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "Routebricks: Exploiting parallelism to scale software routers," in *Proc. of ACM SOSP*, 2009.

[3] S. Han, K. Jang, K. Park, and S. Moon, "Packetshader: A gpu-accelerated software router," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 195–206, Aug. 2010.

[4] K. K. Ram, A. L. Cox, M. Chadha, and S. Rixner, "Hyper-switch: A scalable software virtual switching architecture," in *Proc. of USENIX ATC*, 2013.

[5] E. T. S. I. (ETSI), *ETSI GS NFV 002 Network Functions Virtualisation (NFV); Architectural Framework*, December 2014.

[6] "OPNFV: Open Platform for Network Function Virtualization." [Online]. Available: https://www.opnfv.org/

[7] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[8] R. Riggio and D. Pellegrini, "Progressive virtual topology embedding in openflow networks," in *Proc. of IEEE IM*, 2013.

[9] M. Chowdhury, M. R. Rahman, and R. Boutaba, "ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping," *Networking, IEEE/ACM Transactions on*, vol. 20, no. 1, pp. 206 –219, February 2012.

[10] Y. Minlan, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, Mar. 2008.

[11] Y. Zhu. and M. Ammar, "Algorithms for Assigning Substrate Network Resources to Virtual Network Components," in *Proc. of IEEE INFOCOM*, Barcelona, Spain, April 23-29 2006.

[12] R. Narayanan, S. Kotha, G. Lin, A. Khan, S. Rizvi, W. Javed, H. Khan, and S. A. Khayam, "Macroflows and microflows: Enabling rapid network innovation through a split sdn data plane." in *Proc. of EWSDN*.

[13] "Tornado Web Server." [Online]. Available: http://www.tornadoweb.org/