Unified Programmability of Virtualized Network Functions and Software-Defined Wireless Networks

Julius Schulz-Zander[1], Carlos Mayer[2], Bogdan Ciobotaru[2], Stefan Schmid[3], Anja Feldmann[2],
[1]Fraunhofer HHI, Einsteinufer 37, 10587 Berlin, Germany
[2]TU Berlin, Marchstr. 23, 10587 Berlin, Germany
[3]Aalborg University, Selma Lagerlofs Vej 300, 9220 Aalborg, Denmark

The quickly growing demand for wireless networks and the numerous application-specific requirements stand in stark contrast to today's inflexible management and operation of wireless networks. While most research focuses on mobile networks, WiFi is often left out of the purview.

In this paper, we present and evaluate OPENSDWN, a novel WiFi architecture based on a joint SDN and NFV approach. OPENSDWN exploits virtualizaton across the wired and wireless network and introduces datapath programmability to enable service differentiation and fine-grained transmission control, facilitating the prioritization of critical applications. OPENSDWN implements per-client virtual access points and per-client virtual middleboxes, to render network functions more flexible and support mobility and seamless migration. Moreover, OPENSDWN also increases the security of upcoming WiFi HotSpot architectures by following a functional split approach. Finally, OPENSDWN can also be used to out-source the control over the home network to a participatory interface or to an Internet Service Provider.

***Index Terms*—Software-defined networks, network function virtualization, middleboxes, WiFi, IEEE 802.11 wireless networks and cellular networks, virtual networks, access networks, home networks, enterprise networks and campus networks**

I. INTRODUCTION

The popularity of WiFi networks is increasing at a fast pace, with more and more mobile end-devices becoming WiFi enabled. Today, many hotels and cafés—and sometimes also entire cities—offer free WiFi services. Several mobile operators also plan massive WiFi HotSpot as well as HotSpot 2.0 deployments for traffic offloading from cellular and future Internet-of-Things networks [1], [2].

The increasing demand for WiFi networks imposes new requirements, e.g., on security, optimized medium utilization, and mobility support. The (last) wireless hop is often critical for network performance, as it can contribute a non-negligible delay and may constitute a bandwidth bottleneck.

These requirements stand in stark contrast to the state-of-the-art: The management and operation of off-the-shelf WiFi networks is often very inflexible, and today's networks largely ignore the specific needs of users and/or applications. Moreover, WiFi networks are often deployed in an unplanned and uncoordinated manner: different parties in a house or neighborhood typically deploy and run their own dedicated infrastructure; neighboring access points as well as public access points cannot be leveraged—but rather interfere with each other, introducing unnecessary transmission delays, and reducing network capacity. Mobility support is often very limited, depriving users from essential services. Some deployments (e.g., public WiFi hotspots) in untrusted environments also come with security issues such as eavesdropping, requiring new and more flexible deployments.

Software-Defined Networking is an interesting new paradigm which allows overcoming network ossification by introducing programmability. In a nutshell, Software-Defined Networks (SDNs) consolidate and outsource the control over a set of network devices to a logically centralized software controller. The decoupling of the data plane and control plane allows the control plane to evolve independently of the data plane, enabling faster innovations. Moreover, OpenFlow, the standard SDN protocol today, introduces interesting generalizations. Openflow is based on a match-action paradigm, where switches can match not only the Layer-2 header fields of packets, but also Layer-3 and Layer-4 fields. These flexibilities can be used, e.g., to implement fine-grained traffic engineering [3], enforce complex network policies [4], [5], improve resource utilization in wide-area networks [6], [7], or enable network virtualization in datacenters [8].

SDN is also an enabler for a second paradigm shift in the Internet: *Network Functions Virtualization (NFV)*. Modern networks include many middleboxes to provide a wide range of *network functions (NFs)* to improve performance as well as security. For example, middleboxes are used for caching and load-balancing, as well as for intrusion detection. NFV aims to *virtualize* these network functions, and replace dedicated network function hardware with software applications running on generic compute resources. The resulting orchestration flexibilities can be exploited for a faster and cheaper service deployment. SDN can be exploited to steer flows through the appropriate network functions [9], [10], [11], [5]. Thus, SDN and NFV together, recently also called *SDNv2* in the context of carrier WAN networks, support fine grained service level agreements, as well as an accurate monitoring and manipulation of network traffic.

In this paper, we argue that there is a major potential of introducing programmability and virtualization in *wireless networks*, i.e., following a Software-Defined Wireless Networking (SDWN) approach. Wireless networks are very different from wired networks—the domain where SDN/NFV has been studied most intensively so far. In wireless networks, communication happens over a shared medium whose charac-

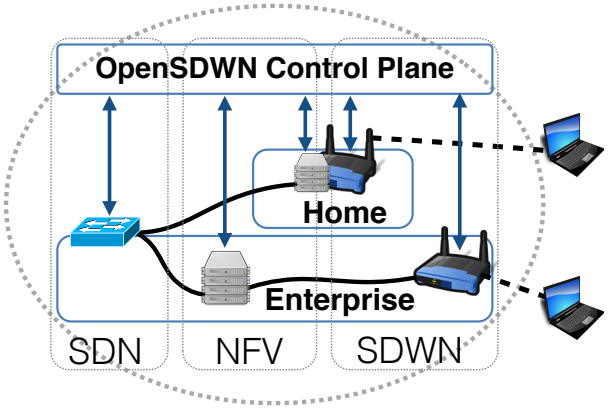


Figure 1: OpenSDWN introduces programmability in home and enterprise WiFi networks using an SDN and NFV approach.

teristics can change quickly over time and in an unpredictable manner, as users are often mobile and associations dynamic. WiFi networks offer several unique knobs to influence the probability of successful transmissions, such as transmission rate and power, as well as retry chains. This introduces opportunities for a fine-grained and application specific transmission control, e.g., for service differentiation.

Today's OpenFlow protocol is not well suited for WiFi: it is mainly restricted to programming flow table rules on Ethernet-based switches, and it is not possible to match on wireless frames, nor can measurements of the wireless medium be accommodated or per-frame receiver side statistics reported; it is also not possible to set per-frame or per-flow transmission settings for the WiFi datapath. In general, SDN+NFV have not received as much attention yet in the context of wireless

A. Our Contribution

This paper shows how to reap the benefits of SDN and NFV in home and enterprise WiFi networks. In particular, we present the design, implementation, and evaluation of OpenSDWN, a flexible WiFi architecture based on a unified, programmable control plane as illustrated in Figure 1. OpenSDWN allows to manage both the virtualized middleboxes as well as the wired and wireless datapath, e.g., to apply per-flow PHY and MAC layer transmission settings. In particular, OpenSDWN brings typical enterprise features to off-the-shelf WiFi APs and like most of today's state-of-the-art enterprise systems also works in congested environments, *i.e.*, it introduces typical enterprise features such as band steering, client-based load-balancing, infrastructure-driven handovers, and prioritization of traffic to cheap off-the-shelf WiFi APs which are typically found in the user's premises. However, OpenSDWN only brings the following benefits to WiFi APs which are under its control.

OpenSDWN comes with interesting use cases: (1) It enables service differentiation, and allows administrators or users to specify application and flow priorities on their wired and wireless network. These priorities are implemented using a fine-grained wireless transmission control. (2) Using its per-client virtual access points and virtual middleboxes, OpenSDWN supports seamless user mobility, as well as flexible function allocation (e.g., function collocation at night

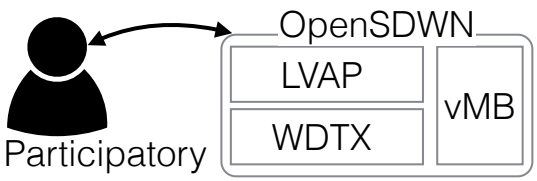


Figure 2: OpenSDWN builds upon the four building blocks: a fine-grained WiFi datapath programmability (WDTX) module, a abstraction for virtualized middleboxes (vMB), a light virtual access points (LVAPs) abstraction, and a participatory interface.

to save energy). (3) OpenSDWN enables deploying secure and efficient WiFi hotspot services via virtual access points hosted in the Cloud, where all user traffic is en/decrypted. By splitting the control and the data plane related traffic, overheads can be minimized. (4) Network functions such as firewalls and NATs can be deployed flexibly, e.g., outside user premises. (5) OpenSDWN introduces flexibilities in terms of network control: the system exposes a participatory interface à la [12], which allows users to indicate application specific priorities. The control can also be outsourced to an Internet Service Provider (ISP), e.g., for troubleshooting.

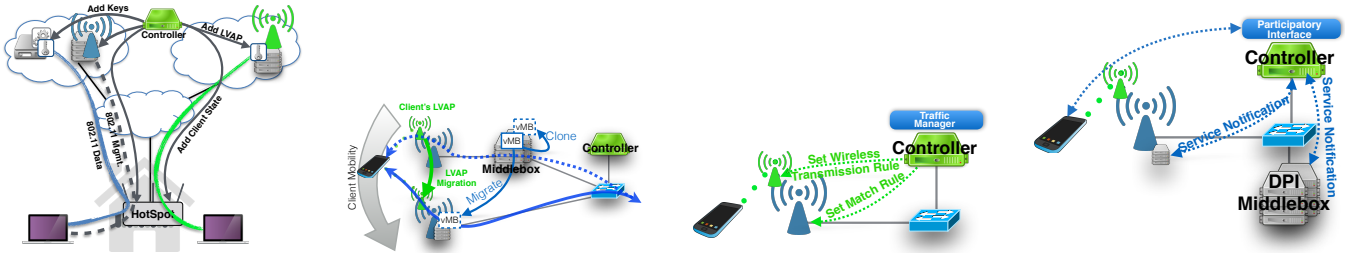
OpenSDWN builds upon four building blocks (see Figure 2): (1) The *Light Virtual Access Point (LVAP)* [13] abstraction that allows us to address the specific requirements of WiFi networks, whilst allowing for unified management of the wired and wireless portions of the network. (2) WiFi datapath programmability, e.g., for fine-grained wireless datapath transmission control (WDTX): settings include transmission power, transmission rate as well as tailored retry chains. (3) A unified SDN and NFV abstraction through virtualized middleboxes (*vMB*) and access points, e.g., to facilitate an easy handling and migration of per-client state. (4) A participatory interface which allows to share network control.

Indeed, middleboxes are an integral part of OpenSDWN. First, to abstract and decouple user-specific state, OpenSDWN introduces the notion of per-client virtual middleboxes (*MB*s). Second, to identify and classify flows, and hence enable service-differentiation, OpenSDWN relies on a *Bro Intrusion Detection System (IDS)* [14]. Once flows have been detected, per-flow transmission rules are installed according to specific requirements such as policies specified by the users. Bro may also be used to tag packets, e.g., for a live streaming application where key frames should be transmitted in a prioritized way, as these frames are more critical for service quality.

We demonstrate the feasibility and usefulness of our system by reporting on different case studies and experiments conducted using two deployments: one at our university and one in a large home network.

B. Paper Organization

The remainder of this paper is organized as follows. Section II gives an overview of the goals and benefits of OpenSDWN. Section III presents the architecture of OpenSDWN, and Section IV reports on our deployments and experiments. Section V discusses the prototype implementation. After reviewing related literature in Section VI, we conclude our work in Section VII.



(a) **Secure scalable HotSpots:** LVAPs can be placed into the Cloud and allows separation of encrypted management as well as data traffic through separate programmable network equipment.

(b) **Mobility support:** Virtual middleboxes (e.g., encapsulating firewall connection state) can be migrated in the presence of mobility and cloned for redundancy.

(c) **Transmission control:** The controller sets specific wireless transmission and OpenFlow rules for per-flow wireless transmission control.

(d) **Participatory interface:** A participatory application provides an interface to the user. Service detection is achieved through DPI.

Figure 3: Three basic operations supported by OPENSDWN.

II. USE CASES AND OVERVIEW

OPENSDWN is based on programmable network devices in the spirit of SDN and NFV. Before we give an overview of the architecture, we discuss some use cases for the envisioned system. See also Figure 3 for some illustrations.

A. Use Cases

1) ***Service differentiation:*** OPENSDWN offers visibility into the network's state and supports a fine-grained transmission control, by allowing administrators and users to set per-flow and per-packet specific transmission settings (such as transmission rate and power, frame retransmissions, and flow control, *i.e.*, by leveraging *Request and Clear to Send (RTS/CTS)*). For instance, as we will demonstrate, OPENSDWN can protect latency-sensitive flows (e.g., live media streams) from competing with background traffic (e.g., Dropbox synchronization). OPENSDWN allows to prioritize traffic belonging to a particular service such as Spotify or Netflix at the wireless access. Today, however, decisions are usually made on a traffic class (e.g., based on the ToS or DSCP field) and thus, for all packets belonging to that class. This provides only coarse-grained traffic prioritization.
2) ***Mobility and migration:*** By virtualizing not only the per-client access points, but also the middleboxes, OPENSDWN supports both seamless user mobility and dynamic resource allocation. This enabled a more dynamic resource management which allows the adjustment and migration of resources and functionality with the user, e.g., for flexibly scaling up or down resources depending on the demand. By collocating network functions, e.g., at night, also energy may be saved.
3) ***Secure virtual WiFi AP:*** Public WiFi hotspots are often deployed in untrusted environment which raises security concerns with regards to *eavesdropping*. OPENSDWN enables deploying WiFi hotspot services via virtual access points hosted in the Cloud, where all user traffic is en/decrypted. This prevents eavesdropping attacks on a client's traffic, since all encrypted IEEE 802.11 traffic is forwarded to the Cloud. Moreover, to minimize overheads, OPENSDWN allows to split control and the data plane related traffic, *i.e.*, all data traffic is handled independently of the management traffic. For instance, the data traffic is decrypted in the datapath by specific network devices or middleboxes, *e.g.*, an ISP could deploy the decryption key in a BRAS or DSLAM. Alternatively, in the case of *Fibre To The Building*, the key could be deployed in an aggregation box in the basement of a building.
4) ***Flexible deployment:***: Network functions (firewalls, NATs, functionality for service differentiation) can be allocated and deployed flexibly. For instance, the different users of a house may use a shared box, outside their individual user premises, to run a middlebox or controller . The specific deployment requirements will depend on the scenario (fiber-to-the-home, endpoints of encryption tunnels, etc.).
5) ***Flexible control and participatory networking:*** OPENSDWN provides unified programmability and control over the network devices and middleboxes. It also offers customization flexibilities through a participatory interface à la [12]: the interface can be used by the users to specify priorities over different applications (e.g., Youtube over Dropbox), and the control may also be handed over to an Internet Service Provider (ISP) for troubleshooting, for defining requirements, and updating transmission rules. A local controller can also maintain connectivity between users in a neighborhood when Internet uplink failures occur.

B. Overview

OPENSDWN is based on an SDN+NFV (a.k.a. *SDNv2*) approach and consists of the following components:

1) **Unified Programmability and Abstractions:** The logically centralized control plane unifies SDN and NFV through programmatic abstractions. That is, OPENSDWN virtualizes both access points and virtualized middleboxes (see Figure 3(b)), which facilitates an easy handling and migration of per-client state, also beyond CPE boundaries. The former is realized through our *LVAP* abstraction, which is a virtualized per-client AP which simplifies the handling of client associations, authentication, handovers, and unified slicing of both the wired and wireless portions of the network. Specifically, the *LVAP* concept abstracts the complexities of the IEEE 802.11 protocol stack (*e.g.*, client associations, authentication, and handovers) and encapsulates the client's

Openflow state to achieve network slicing and client mobility. Moreover, OPENSDWN additionally introduces per-client virtual middleboxes, short vMB*s*, which can be transferred seamlessly across the network. Specifically, a *vMB* encapsulates the client's *MB* state as a virtual *MB* object. Thus, OPENSDWN achieves control logic isolation as SDN/NFV applications running on top the controller can only operate on their respective *LVAP*s and *vMB*s. OPENSDWN, as shown in Figure 3(a), also enables to outsource the en/decryption of the 802.11 traffic to separate network equipment and middleboxes.

2) **Programmable Datapath:** The programmable datapath gives the possibility to set per-flow specific transmission settings as shown in Figure 3(c). The settings include transmission power, transmission rate as well as tailored retry chains. It is even possible to differentiate between different packets of the *same flow* (5-tuple): *e.g.*, giving key frames of a live stream a higher priority. This is achieved by using an Intrusion Detection System (*IDS*, in our case: *Bro*) for packet classification and *tagging*: transmission settings are chosen depending on the tag.
3) **Participatory Interface:** OPENSDWN's participatory interface allows us to define flow priorities as well as priorities over customers. The chosen priorities are translated by the controller into meaningful network policies. Priorities can be adjusted anytime. Figure 3(d) depicts the participatory interface.

III. THE OPENSDWN SYSTEM

We first describe the wireless SDN component of OPENSDWN, then the virtual middlebox, and finally the participatory interface.

A. Wireless SDN

WiFi networks have several unique properties which do not exist in wired networks. For instance, WiFi networks offer several knobs to influence the probability of successful transmissions, such as transmission power or rate. This introduces opportunities for a fine-grained and application specific transmission control.

The wireless subcomponent of OPENSDWN inherits from [13]: (1) The Light Virtual Access Point (LVAP) abstraction: essentially the client's association state (the BSSID, SSIDs, client IP address, and OpenFlow rules). (2) Mobility support: by migrating a client's LVAP between physical APs, the infrastructure can control the client's attachment point to the network, without triggering a re-association at the client. (3) Slicing: the accommodation of multiple logical networks on top of the same physical infrastructure with different policies and control applications. A network slice is a virtual network with a specific set of SSIDs, where for example, the traffic may be VLAN tagged or directed to a specific destination port.

OPENSDWN introduces service differentiation through per-flow WiFi datapath transmission rules, organized into per-flow transmission rule tables. Rules are bound to one or more OpenFlow rules and assign meta or direct transmission properties to one or more OpenFlow entries. Specifically, fine-grained wireless transmission control is achieved by combining Openflow match-action rules with *wireless transmission rules* (WDTX) within the wireless access points. Regarding actions, assigning fixed and/or meta transmission settings is possible. Meta transmission settings include: *best probability rate*, *best throughput rate*, *second best throughput rate*, *common maximum rate* or *fixed rates* (e.g., a basic rate or a specific modulation and coding scheme rate). Based on the capabilities of the WiFi NIC, the transmission settings can be set for the device multirate retry chains. We demonstrate the benefit of *WDTX* rules over per-packet decisions in Section IV-B, *e.g.*, by reducing latency when transmitting low latency UDP traffic with the *best probability rate* which effectively reduces the number of Layer 2 retransmissions.

Furthermore, in order to account for the dynamic nature of the wireless network and in order to support client mobility, agents in OPENSDWN implement a publish/subscribe interface, allowing the controller to subscribe to network events (see Section V for more details).

B. Virtual Middleboxes

Middleboxes are an integral part of OPENSDWN. First, our service differentiation mechanism relies on a deep-packet inspection middlebox, to identify and classify flows. Moreover, OPENSDWN integrates *MB*s in the virtual network, and allows us to set and migrate state to support client mobility and to scale dynamically.

At the core of our system lies the concept of virtual *MB*s, short vMB*s*. *vMB*s are used to fully reap the virtualization benefits: the handling of vMBs is important to guarantee the decoupling of the per-client middlebox state and the inner workings of the middlebox from the physical instance.

The *vMB* keeps solely the user-specific state information which can be transferred from one *MB* instance to another. On top of a *MB* (*e.g.*, a virtual machine or docker container running on a physical host) runs a MB *agent* which needs to accomplish three primary tasks: *(i)* interface with the resources of the *MB*, *(ii)* handle *vMB*s and *(iii)* expose the control of the *MB* to a remote entity (the controller). In a nutshell, the *vMB* basically abstracts the client state from the inner workings of a specific *MB* such as a Linux or BSD firewall. The middlebox agent also provides the necessary hooks for the controller (and thus applications) to instantiate, destroy, monitor and manage its functionality.

In OPENSDWN, a stateful *vMB* is characterized by a configuration file (a *MB*-specific list of tunable parameters), state of the active connections, statistics (counters), and a list of subscribed events in order to completely define its behavior. When a *vMB* is moved from one *MB* Agent to another, the new *MB* is able to handle the user's traffic in exactly the same way as the old one. *vMB*s were designed to give applications the possibility to manage user related *MB* state across physical *MB*s, without any awareness of the user's traffic.

In order to support e.g., scale-out upon certain network events, or to monitor the middlebox, OPENSDWN implements a publish/subscribe interface (see Section V).

C. Participatory Interface

OPENSDWN's participatory interface allows the WiFi users, the network provider or even the content provider, to express their preferences in terms of flow differentiation. Specifically, we allow external entities to rank—by assigning priorities—their transmissions. The rational behind this prioritization approach is simplicity: the participatory interface hides network complexity from end-users. Concretely, a user could express his or her preference to prioritize Netflix over Dropbox, by assigning a higher priority to the former. This preference will then be taken into account by the controller, which installs transmission rules which favor flows tagged as Netflix over flow tagged as Dropbox. This could be done, for example, by assigning different AC Queues or setting distinct rate chains.

As a static service mapping based on, e.g., content server IPs is cumbersome and unreliable, OPENSDWN uses a signature-based Intrusion Detection System (IDS) which also considers packet payload. Once the IDS detects a service of interest, it immediately informs the OPENSDWN controller, which applies the necessary policies accordingly.

In order to keep the system evolvable, and to account for the advent of new services, our participatory API also supports the installation of new signatures by external applications. This for example also enables content providers to install their own signatures, ensuring a better probability of correctness.

Technically, the participatory interface can be implemented based on a *URI* included in a HTTP GET request, or a domain name within a certificate.

IV. EVALUATION

The key benefit of OPENSDWN is its flexibility and the potential use cases it *enables*. How to optimally *exploit* the resulting flexibilities (e.g., in order to provide QoS guarantees) or how to fine-tune performance (e.g., of function migration), are orthogonal questions, and also depend on the context.

Nevertheless, in order to show the potential of OPENSDWN, we implemented and evaluated different applications using our proof-of-concept prototype. The first case study focuses on the system's service differentiation capabilities, and in particular, we consider the optimization of a video-on-demand application. In the second case study, we consider an optimized multicast service based on direct multicasting. The third focuses on the middlebox virtualization, and we discuss the migration of a personalized stateful firewall.

Deployments and Methodology

Our proof-of-concept implementation of OPENSDWN has been deployed in two real networks:

- Our research group's indoor WiFi network. This deployment consists of more than 25 IEEE 802.11n enabled APs, distributed across one floor of an office building.
- A centrally administrated home network which covers an entire building of ~21500 square feet. It provides internet connectivity for roughly 30 households with more than 70 active devices per day, using Ethernet and 10 WiFi APs (indoor and outdoor).

All APs run OpenWrt/LEDE with the ath9k Linux driver, user-level Click modular router [15], and Open vSwitch (OvS) version 2.3.90 supporting OpenFlow (*OF*) version 1.3 and *conntrack* table management. The off-the-shelf WiFi access points are either based on ARM, MIPS or x86. The variety of WiFi AP hardware ranges from IEEE 802.11g only to IEEE 802.11abgn boards equipped with one or more WiFi NICs based on Atheros chipsets.

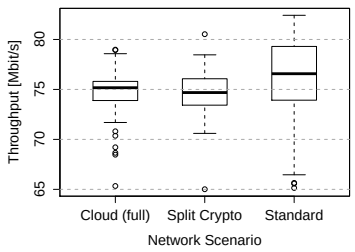
Our controller and *MB*s are evaluated on non-virtualized servers with 4 CPU cores supporting hyper-threading and at least 8 GB RAM. We leverage an identical server for the datapath encryption. All servers run a Debian-based OS with OvS 2.0.2 or 2.3.90. We monitor data through a dedicated port for the IDS at the core switch. We did not hit CPU or memory limitations in any of our experiments. Furthermore, for the performance evaluation of the controller and middleboxes, we use three dedicated servers: 1) an OpenFlow controller, 2) a middlebox, and 3) a traffic generator which either generates artificial traffic using *iperf3* or replays traffic through *tcpreplay*, *e.g.*, previously captured packet traces of a live video streaming or on demand service.

A. Secure virtual WiFi AP

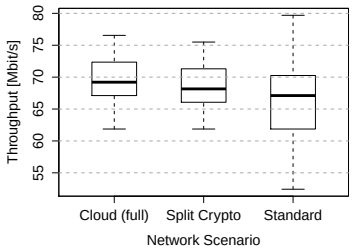
In the first case study (see Figure 3(a)), we present OPENSDWN's ability of hosting virtualized public WiFi hotspot APs in the control plane to prevent *eavesdropping* attacks. Specifically, public WiFi hotspots are often deployed in untrusted environment where the device is physically accessible. This raises security concerns with regards to eavesdropping. OPENSDWN enables deploying WiFi hotspot services via virtual access points hosted in the Cloud, where all user traffic is en/decrypted. This prevents eavesdropping attacks on a client's traffic, since all encrypted IEEE 802.11 traffic is forwarded to the Cloud. Furthermore, OPENSDWN allows splitting the control and the data plane related traffic, *i.e.*, all data traffic is handled independently of the management traffic. This allows reduce the load on the control plane, since all data traffic can be en/decrypted by programmable network equipment/middleboxes in the datapath. For instance, an ISP could deploy the decryption key in a BRAS or DSLAM. Alternatively, in the case of *Fibre To The Building*, the key could be also deployed in an aggregation box in the basement of a building.

Benchmarking of Cloud-assisted Virtual WiFi: First, we evaluate the base-line performance of running the virtual access points in the Cloud, managed by the control plane. Here, all 802.11 management and data frames are encapsulated and forwarded to the Cloud. In order to prevent fragmentation on the lower networking layers, we set the MTU of the link and signal the TCP MSS accordingly. We compare our results against the base-line *Standard* case, where all functions are performed directly on the AP.

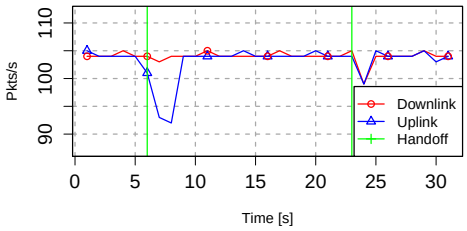
Figure 4(a) shows the throughput performance on the downlink. We observe, that in the full *Cloud* case, one can achieve almost as much throughput as in the *Standard* case. However, the maximum throughput is slightly lower when moving all 802.11 frames to the Cloud or middlebox in the datapath, *i.e.*,



(a) Downlink performance of an encrypted 802.11n link. Moving en/decryption to the Cloud has little performance implications.

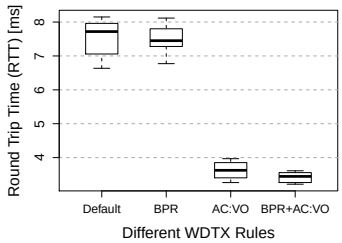


(b) Uplink performance of an 802.11n link with cryptography enabled. Moving en/decryption to the Cloud has little performance implications.

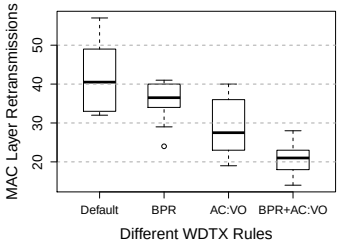


(c) Impact of virtual AP migration on a voice call. Compared to legacy WiFi, we observe no severe performance impact during handovers, *i.e.*, a handover equals a virtual AP migrations.

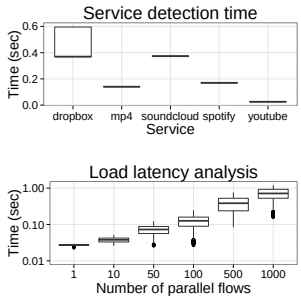
Figure 4: Evaluation of the secure virtual WiFi AP in the Cloud



(a) Latency when assigning different wireless transmission settings to a flow.



(b) MAC layer retransmissions when assigning different wireless transmission settings to a flow.



(c) Service detection time until rule installation and latency under different workloads.

Figure 5: Evaluation of OPENSDWN's fine grained transmission control through *WDTX* rules and *vMB* handling.

the bandwidth decreases due to the limitation of the uplink to 100 $Mbit/s$ and the additional tunneling overhead. This is also the case on the uplink as indicated in Figure 4(b). However, we observe no significant performance impact with our solution.

Benchmarking of Crypto in the Datapath: Next, we evaluate the separation of the 802.11 management and data frames. Here, all management traffic is sent to a remote authenticator and all data traffic is handled by a crypto box in the datapath. We observe that the throughput is nearly the same compared to the previous case. Figure 4(a) and Figure 4(b) show that the throughput is close to 70 $Mbit/s$ and again close to the Standard case. The advantage of this case is to handle functions such as traffic en/decryption, en/decapsulation, and deduplication directly in data plane instead of moving everything to operator's data center.

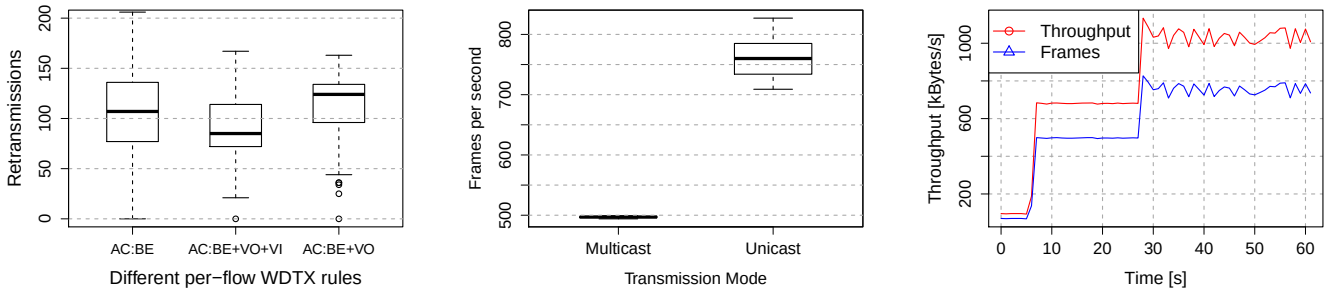
For instance, today's general purpose computing hardware deployed in the datapath can easily handle the traffic of hundreds of access points. Specifically, Intel's DPDK greatly boosts packet processing performance and throughput. Moreover, recent CPUs can easily handle several hundreds of gigabit of AES (block chain) traffic per second. Figure 4(c) shows the impact of a LVAP migration on a WiFi link. Compared to legacy WiFi, we observe no severe performance impact during handovers, *i.e.*, a handover equals a virtual AP migrations.

B. User-Defined Service Differentiation

The first case study concerns OPENSDWN's service differentiation capabilities. Before presenting our video-on-demand optimizer in more detail, we will discuss some more general aspects of our system.

Today, most public internet downlink traffic is sent as *best effort*, also due to *network neutrality* requirements. But also in small offices, home offices or home networks, traffic is often treated equally, although this is legally not required. We believe that there is a high potential benefit of differentiating services in home networks, e.g., by prioritizing voice traffic over regular web traffic. Especially given today's trend to deploy more and more wireless devices in the user's premises, traffic can significantly interfere, e.g., an unimportant system update for a device can easily interfere with requested *on demand* services such as Spotify or Netflix, resulting in poor performance.

Benchmarking the Transmission Rule Extension: There are several ways to prioritize traffic through specific *WDTX* rules, bound to a particular flow entry. We investigate, as a benchmark, the effect of assigning a meta transmission rate and a medium access priority, on the latency and MAC layer retransmissions of a single flow. To this end, we first study the effect on MAC layer retransmissions (*cf.* Figure 5(b)) when assigning a per-flow transmission rule to a latency sensitive UDP flow. In our experiment, we use two OPENSDWN APs and two clients. Each client is connected to one of the APs in our indoor testbed. We start generating best effort TCP traffic on the link between one AP and client, and start a latency sensitive flow on the link between the other client and the AP. In the beginning, the latency sensitive flow and the background traffic are treated equally, which results in a round trip time



(a) Flows sent as BE do not experience significantly more retransmissions in the presence of higher prioritized traffic.

(b) HD IPTV stream exceeds capacity of multicast bandwidth. Switching to unicast mitigates this problem.

(c) OpenSDWN's multicast Application switches to unicast for a single subscriber.

Figure 6: Evaluation of the service differentiation and smart multicast OpenSDWN applications

Table I: Service Detection Delay

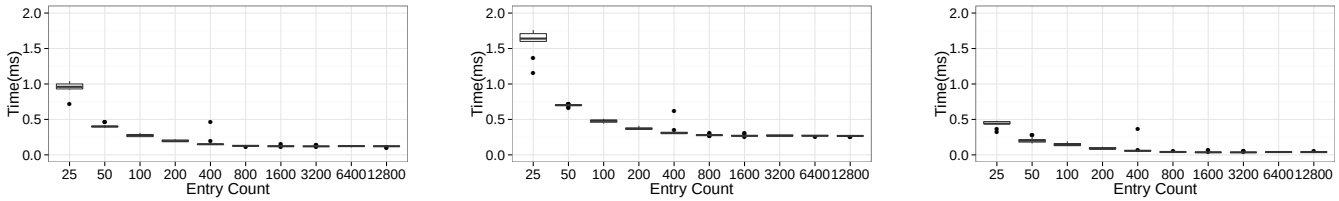
Frequency (ms)		2us	5us	10us	50us	100us	500us	1ms	5ms	10ms	50ms	100ms	1s
Same host	Bro - MB Agent	0.060651	0.058026	0.05992	0.058391	0.064939	0.049412	0.030326	0.009835	0.003022	0.000381	0.000371	0.000421
	Bro - Rule Installation	0.06167	0.0589	0.060837	0.059249	0.065867	0.050346	0.031277	0.011736	0.005432	0.002918	0.002871	0.003129
Different hosts	Bro - MB Agent	0.057146	0.063181	0.062309	0.064495	0.055805	0.031668	0.021651	0.001527	0.000388	0.000389	0.000394	0.000425
	Bro - Rule Installation	0.058662	0.064439	0.069839	0.06587	0.058102	0.033013	0.023731	0.003773	0.003733	0.003183	0.00362	0.003861

(RTT) of roughly 8 ms. Next we assign the *best probability rate* (BPR) to the flow; this leaves the RTT unchanged. When changing the medium access to the highest priority (AC:VO), i.e., the voice access category, the RTT drops by half to less than 4 ms as depicted in Figure Figure 5(a). This is as expected since a higher medium access probability constitutes a change in the RTT. Note, in today's home network traffic is typically sent as *best effort* and rarely differentiated as in OpenSDWN. However, only looking at the RTT of an UDP flow is not sufficient as it does not take the MAC-layer (L2) packet loss into account; this however has a significant effect on the jitter and performance of transport protocols (L4) such as TCP. Thus, we next study the effect of the meta transmission rates on the packet loss. We assign a *WDTX* entry to the *OF* flow rule that matches the flow, and assign the *best probability rate* and highest medium access priority (AC:VO) which increases the transmission probability on the L2. Figure 5(b) shows that this significantly reduces the MAC layer retransmissions compared to default flow properties. We conclude that combining the medium access strategy by a meta transmission rate within OpenSDWN significantly reduces the number of MAC layer retransmissions, and the the RTT by roughly 50%. That said, OpenSDWN can achieve a per-flow resource utilization which is better suited for the diversity of traffic requirements in today's home networks.

Benchmarking the DPI Interface: In order to understand latency and induced load of service discovery, we replay traces of typical streaming services collected at a university campus network in our testbed. Concretely, we replay the traces 100 times per service at first and then vary the number of simultaneous *youtube* flows: 1, 10, 50, 100, 500, 1000 to identify eventual bottlenecks on the service detection engine. The traffic is injected on one server and tapped on a second server running a Bro *MB* instance handled by our agent. The controller is hosted on a third server with a dedicated out-of-band control channel running a service discovery SDN/NFV application. Figure 5(c) depicts the measured service detection time and the load latency analysis, i.e., the latency added during high workload pattern.

In order to estimate possible performance bottlenecks of the service detection chain, we measure the delay added by the different components involved in the detection, in bursty scenarios. We are interested in how our system reacts to different rates of events. We mock the detection of a service by triggering an event from Bro at different intervals. Specifically, we schedule events sequentially, from a Bro script, adding a determined delay between 2 consecutive events. We send 300 events in total over multiple runs for each delay, starting from 2 μs up to 1 second. We run this procedure in two different scenarios: First, we keep both the controller and the MB Agent on the same host to eliminate the network delay. In the second scenario, we run the MB Agent and controller on a different host. Table I presents the results. They include, for each scenario, two distinct measurements: First, we measure the mean of the delay between the instant Bro sends the event and the MB Agent processes it (basically, the delay caused by the queue of events). Second, we measure the time between the instant Bro fires the event and the moment that our controller installs the required rules for this flow. This delay includes therefore the MB Agent processing time, the delay caused by the MB Protocol and the controller handling time of this event. As expected, our system presents a lower response time for smaller event rates (bigger delays). The worst performance, for the highest rate, indicates a total delay of around 60 ms. However, the *detection delay* could be mitigated by proactively installing coarse-grained flow rules based on DNS response messages. However, this might lead to side issues with today's CDNs which provide a wide range of different services through the same front servers.

Case Study: Medium Access Optimizer. Given these benchmarks, we now consider a simple case study: the optimization of a video-on-demand transmission. Our setup consists of a single AP and three clients. One client performs a system update, one requests a Video-on-Demand (VoD) stream and



(a) Per entry write latency. (b) Per entry read latency. (c) Per entry delete latency.

Figure 7: Latency for a stateful firewall vMB object read, write and delete operation. Latency in milliseconds (time) is normalized to a per-entry time. vMB object size is increased from 25 entries to 12,800 entries.

the third client does a UDP-based VoIP call. In the beginning, all flows are treated equally as best effort traffic. Next we put the voice flow into the highest priority queue. As expected, the prioritized traffic now achieves a slightly higher throughput than the best effort traffic. However, in mac80211, the voice queue does not perform aggregation and hence, can easily suffer from too many competing stations. Specifically, even if the medium access probability is high, the performance without 802.11 frame aggregation is significantly lower. That said, if a flow suffers from background traffic, e.g., caused by a neighboring WiFi network, switching to the highest queue with aggregation can significantly increase the throughput. Due to the bursty nature of *Dynamic Adaptive Streaming over HTTP (DASH)* [16] based VoD traffic, the BE traffic is just slightly decreased while VoD services benefit from a more aggressive medium access, which in turn leads to a faster switching of the video quality. BE flows do not experience significantly more retransmissions in the presence of higher prioritized traffic. In other words, using prioritization reduces the achievable throughput of BE flows without a big impact on the the MAC layer retransmissions (see Figure 6(a)).

C. Smart Direct Multicast Service

OPENSDWN can also be used in conjunction with group communication abstractions such as multicast. Especially with the advent of IPv6, the fraction of multicast traffic is likely to grow in the future: IPv6 realizes broadcast over multicast, and mDNS to broadcast features to neighboring stations.

In IEEE 802.11, multicast packets are typically sent at basic rate. However, wireless networks may benefit from a Direct Multicast Service (DMS): DMS has the potential of reducing the transmission time over regular multicast, by sending 802.11 packets as unicast. Unfortunately, DMS requires a client to signal its DMS capabilities to the AP, which is the reason why DMS is rarely used in 802.11 networks today.

With OPENSDWN, a controller can detect the number of subscriptions for a particular multicast service, and control the transmission accordingly. Specifically, a controller can install an OpenFlow rule to switch from multicast to unicast for the transmission. Moreover, OPENSDWN allows to assign a *WDTX* transmission rule to a particular stream of multicast data, to send the data at the *maximum common transmission rate* for a group of wireless devices.

We evaluate OPENSDWN's smart multicast application with a single access point and a IPTV set-top-box from a major European ISP. First, we transmit a IPTV continuous stream of multicast data to the box. With a single station, our application installs a rule to send the multicast stream as unicast on the wireless medium. With multiple stations, the application switches back to multicast at the maximum common rate for the transmission. Figure 6(c) shows that the throughput and frame count increase after 28 seconds, when the application switches from multicast to unicast. Figure 6(b) indicates that an HD IPTV stream easily exceeds the basic rate of IEEE 802.11g networks. Note, switching to unicast or to a higher datarate mitigates this issue.

D. User Mobility

As a second case study, we consider OPENSDWN's support for user mobility, where also middlebox functionality is migrated. Supporting client mobility is a crucial feature in WiFi deployments with multiple physical APs. The application migrates a stateful firewall vMB object between MBs, i.e., installs the client's flow state at the new AP before or during the handoff.

Benchmarking the Stateful Firewall vMB Interface: We study the performance of the vMB stateful firewall module of OPENSDWN for different workloads in more detail. Specifically, we measure the read, write and delete performance of a stateful FW vMB extension that utilizes the *netlink* interface of the Linux Kernel *conntrack* module for connection tracking, which is typically part of a stateful firewall. We repeat each experiment 12 times for each workload. The vMB object workloads vary from 25 to up to 12,800 entries. As shown in Figure 7(a), we first measure the performance of the per-entry execution time of the write (setState). The write duration for a single entry decreases constantly with the workload, and stabilizes at around 130 μs for a single entry in a vMB object. Next, we evaluate the read time (getState) which decreases constantly. The average value stabilizes at around 270μs (see Figure 7(b)). Finally, we evaluate the delete operation in order to fully understand the required time for the migrate operation, which requires a read, write and a delete of the old vMB object. The average value of a *delState* stabilizes at around 40μs (see Figure 7(c)). That said, a migrate operation takes at least the time of a combined read and write, times the number of entries. Thus, the time can be estimated by the measured results. Specifically, the delete of the old vMB state can be called after the object was correctly fetched and while it is installed into the new MB.

Algorithm 1: Mobility Service

```
begin
    if handoverEvent = True then
        oldMBid ← AP2MBmap.get(oldAPid) ;
        newMBid ← AP2MBmap.get(newAPid) ;
        vMB ← createvMB(clientIP, oldMBid) ;
        if vMB.migrate(newMBid) = True then
            signalOpenSDWN(migrationComplete) ;
```

Entry count	Mean execution time (ms)			
	Write	Read	Delete	Migrate
1	11.6	38.4	6.4	45.0
10	12.3	48.6	6.8	60.9
100	20.3	121.6	10.7	141.9
1000	115.9	778.0	43.0	893.9
10000	1119.3	5201.2	385.3	6320.5

Table II: Average execution time of the **`setState`**, **`getState`** and **`delState`** operations for different workloads.

Case Study: Firewall State Migration The firewall state migration service is a reactive application triggered through external events to move state between MB instances. The algorithm in form of pseudo-code is shown in Algorithm 1. For example, when OPENSDWN detects a client with a higher RSSI at a new AP, a handover event is triggered and the client's firewall state (connection tracking state) migrated to the new AP before the handover. In case the firewall is realized outside of the AP, *e.g.*, in several middleboxes throughout the network, the client's firewall state would be migrated to the middlebox associated with the new AP. Accordingly, the firewall state migration service keeps a mapping between APs and firewalls. Furthermore, it decides whether the state associated with the mobile user needs to be migrated and executes the operation.

During the state migration operation, the controller uses the three operations that were evaluated previously. The **`getState`** call on the serving middlebox is followed by a **`setState`** operation with the target MB identifier as argument. Finally, the state is removed through a **`delState`** call. The last two operations happen simultaneously since RPC calls are asynchronous, and called at different agents. Table II shows the average migration time for different vMB object sizes. The total time of a `migrate()` call on a vMB object with 100 entries averages at around 140 ms. This underlines the power that the simplicity of the vMB abstraction exposes to a network programmer. The RPC interface and entry processing from the Linux Kernel netlink interface contribute to most of the processing time. Note, the agent to kernel communication for a single rule is below one millisecond.

V. PROTOTYPE IMPLEMENTATION

This section presents more details about our prototype implementation. We first describe the different radio and middlebox interfaces implemented by OPENSDWN, then present the control plane, and finally discuss the support for reactive and proactive applications. The Radio Agent is implemented in C/C++ while the controller is based on the Java-based *ONOS OpenFlow* controller. The *MB* agent is realized in python and implements a newly defined *MB* protocol.

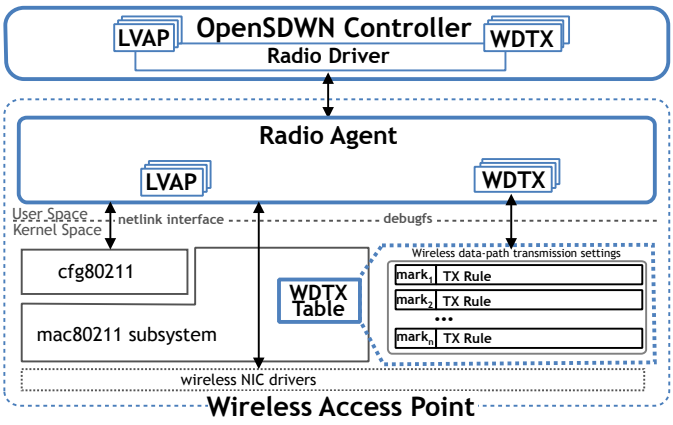


Figure 8: Wireless Radio Interface: Packets matching an OpenFlow rule are annotated with a *mark* and then matched by a *WDTX* rule to control wireless transmission settings on a per-flow level.

Table III: Subset of South-bound APIs provided by the framework

Radio API: (Controller to agent)	Description
`{add/remove/set}-lvap`	**Add/remove/update an LVAP on an agent**
`read-lvap-table`	**Obtain the list of LVAPs on an agent**
`read-rx-stats`	**Query per-station rx-stats at the agent**
`{read/set}-key`	**Query/set per-client crypto keys + state**
`{read/set}-wdtx`	**Query/set per-flow transmission rules on an agent**
`{read/set}-subscriptions`	**Query/set the list of subscriptions at the agent**
`{read/set}-channel`	**Query/set the channel the agent listens and transmits on**
`{read/set}-beacon-interval`	**Query/set the beacon interval on the agent**
Middlebox API: (Controller to Agent)	**Description**
`{get/set}-config`	**Get/Set configuration of parameters a virtual middlebox**
`{get/set}-state`	**Get/Set the state of a virtual middlebox**
`{get/set}-stats`	**Get/Set statistics (e.g. packet counter) of a vMBs**
`getAvailableEvents`	**Get a list of available events supported by a middlebox.**
`subscribe/unsubscribe`	**Un-/subscribe from receiving notifications)**

A. Interfaces

Interfaces to the physical WiFi and middlebox resources are provided by agents. We describe the radio and middlebox interfaces in turn. Moreover, Table III depicts the south-bound interface between the agents and the controller.

Radio Interface: OPENSDWN's wireless APs run a radio agent which exposes the necessary hooks for the controller (and thus applications) to orchestrate the WiFi network and report measurements. All time-critical aspects of the WiFi MAC protocol (such as IEEE 802.11 acknowledgments) continue to be performed by the WiFi NIC's hardware. On the other hand, non time-critical functionality including management of client associations, is implemented in software on the controller and the agents. Specifically, this realizes a distributed WiFi split-MAC architecture. In addition, matching on incoming frames is performed to support a publish-subscribe system wherein network applications can subscribe to per-frame events.

In order to realize the fine grained wireless transmission rule interface, we have extended the mac80211 subsystem of the Linux Kernel. Thus, OPENSDWN benefits from its driver abstraction and *Minstrel*, the *de facto* standard transmission rate control algorithm of the Linux Kernel mac80211 subsystem. Note, this algorithm fills a per-station rate table which defines the transmission settings on a per-link basis for a $100ms$ time interval. *WDTX* rules control per-flow physical layer settings. Assigning fixed and/or meta transmission settings is possible, e.g., assigning fixed MCS transmission rates or *best throughput rate*. Based on the capabilities of the WiFi NIC, the transmission settings can be set for the device's multirate retry

chains. With Atheros cards such as the AR9280, there are four segments for the transmission rate, power and retry count. We are currently investigating the possibility to assign functions such as a maximum common transmission rate for a given set of *LVAP*s or maximum transmission time to *WDTX* rules. *WDTX* rules are bound to *OF* rules trough a newly defined action that attaches a tag to all packets that match an *OF* flow entry at the ingress port. The defined tags are passed through the Linux kernel down to the WiFi driver. Figure 8 depicts OpenSDWN's *WDTX* interface.

Moreover, for effective control decisions, wireless network applications need access to statistics not only at a per-frame granularity, but also measurements of the medium itself (for instance, to infer interference from non-WiFi devices operating in the same spectrum). Thus, applications can access measurements (e.g., RSSI, *OF* statistics or spectral measurements) from multiple layers, and work either reactively (e.g., trigger-driven) or proactively (e.g., timer-driven).

Middlebox Interface: A middlebox agent (MB Agent) runs either on a server or WiFi AP and accomplishes three primary tasks: interface the physical resources of the middlebox, handle virtual middleboxes (vMB) and expose the control of the middlebox to the control plane. In OpenSDWN, each agent handles exactly one MB functionality through the middlebox interface. Figure 9 depicts the agent's structure with its interfaces and abstractions.

We have implemented two interfaces for different types of middleboxes in OpenSDWN: 1) a stateful firewall and 2) an interface for deep packet inspection. The former targets firewall handling within the Linux Kernel. Moreover, we have implemented two versions of the stateful firewall vMB: 1) the first one uses wrappers of the `iptables` and `conntrack` user-space tools and 2) the other one uses the `python-iptables` and `pynetfilter_conntrack` libraries to communicate with the Linux kernel netfilter modules. For the latter, we had to extend the libraries to support insertion of new entries to the connection tracking table, and to monitor changes inside the connection tracking table for event generation. Specifically, the latter brings a significant performance improvement: e.g., a state insertion call of 10000 entries is almost 70 times faster over the former interface. However, the former brings advantages for simpler extensibility for non-time critical parts of the firewall handling. The connection tracking table inside the kernel keeps track of all traffic passing the firewall in both directions, and represents the internal *traffic-dependent* state. For each connection or flow, the number of bytes and packets sent in each direction is recorded. This serves as the *statistics* state of the middlebox.

Moreover, the SDN control plane needs to react to events such as threats like DoS attacks or load changes within the network. To this end, the MB agent implements a publish/subscribe system together with the controller. Our Bro IDS and stateful firewall abstraction implement an interface to receive events at the controller, e.g., if someone scans the network. In the case of the stateful firewall, events must be generated whenever something changes in the connection tracking table. The agent leverages the `pynetfilter_conntrack` API to filter events that match a subscription from the controller.

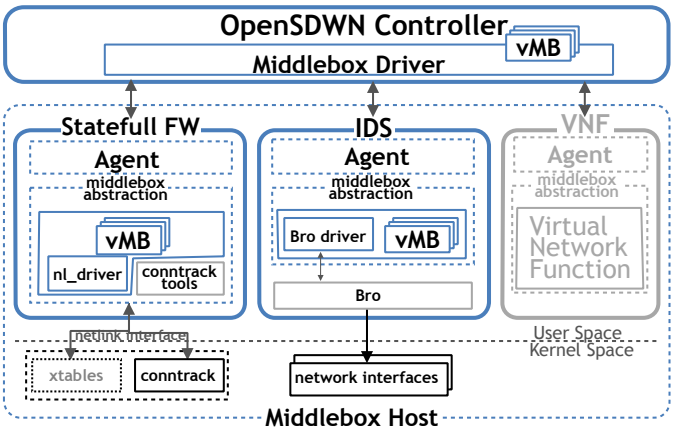


Figure 9: Middlebox Interface and vMB agent structure: vMB protocol interpreter and state machine; MB interface with specific handlers for different types of middleboxes.

Table IV: Subset of APIs provided by the framework

North-bound API for Radio	Description
`getClients()`	**Get slice-specific-view of associated clients**
`getAgents()`	**Get a view of agents in the application's slice**
`handoffClientToAp()`	**Perform an LVAP migration of a client to an AP**
`getRxStatsFromAgent()`	**Query agent for per-station rx-statistics**
`{register/unregister}Subscription()`	**Subscribe to a per-frame event of interest at agents**
`{add/remove}Network()`	**Add or remove an SSID to the application's slice**
Northbound API specific for DPIs	**Description**
`{start/stop}DPI()`	**Start or stop the DPI daemon running on the agent**
`{get/set}InterfaceToMonitor()`	**Get/Set the network interface the DPI should monitor**
`{unsub/sub}scribeForService()`	**Un/Subscribe for services**
`{uninstall/install}Service()`	**Un/Install the capability of detecting a service**
`availableInterfaces()`	**Get the network interfaces available at MB**
`getServicesInstalled()`	**Get the list of services installed on a DPI instance**
`isRunning()`	**Check whether the service is currently running**
`getEventTypes()`	**Get event types that DPI supports**
`getFieldsToSearch()`	**Get a list of header fields that DPI is able to inspect**
Virtual Middlebox Northbound API	**Description**
`migrate_vMB()`	**Move a vMB from one physical MB to another**
`add_vMB()`	**Add a vMB to a physical MB**
`remove_vMB()`	**Remove a vMB from a physical MB**
`clone_vMB()`	**Clones a vMB from a physical MB to another**

Specifically, the agent offers a list of parameters that can be used to create an event mask. The controller can request this information through the `Event_List_Req` message. For each event mask, the agent creates a filter and an event ID. In this way, the controller can deactivate outdated notifications, according to the ID. An `Event` message is sent every time a change occurs in the internal state of the MB.

B. Control Plane

The OpenSDWN controller exposes a set of interfaces to the applications (the northbound API shown in Table IV) and then translates these calls into a set of commands on the network devices (the southbound API). The controller also maintains a view of the network including clients, APs, *MB*s, and *OF* switches, which the applications can then control.

Reactive applications can leverage a publish-subscribe system of the radio and *MB* agent which invokes a handler at the application, whenever an event of interest occurs at the agents. Our current implementation supports applications to register thresholds for events to reduce the amount of events, e.g., receive link-based (PHY and MAC layer) rx-statistics like the *received signal strength indicator (RSSI)*, the bit-rate, and the timestamp of the last received packet, only when necessary. That said, an application can ask to be notified whenever a frame is received at a radio agent at an RSSI

greater than $-70dBm$. Moreover, applications can leverage multiple measurement sources outside the framework, too.

Participatory Interface:

The participatory interface is implemented as a RESTful API exposed by an SDN application, that we call *Service Ranking*. The *Service Ranking* is implemented as a simple Web service, written in NodeJS. It receives priorities as input and feeds the controller with requests through the Northbound API. Figure 10 depicts the components associated with the participatory interface.

A dedicated *Traffic Manager* module within the OpenSDWN Controller, is responsible for compiling requests into meaningful transmission rules, namely assigning a QoS class or *WDTX* rule on a matched flow. Concretely, the Traffic Manager is responsible applying network policies to flows, taking into consideration the service being carried by the flow. We define flows as a group of packets that share a 5 header tuple, composed by source and destination IPs, source and destination ports and the transport protocol, *i.e.*, according to Bro's connection concept.

The algorithm is shown in Algorithm 2. During its initialization (not shown by the algorithm), the Traffic Manager subscribes to all DPI events (by sending subscriptions to MB Agents associated to middleboxes whose type is DPI), passing as parameter the callback to handle the upcoming events. If an event occurs, the message is translated into flow rules and a corresponding *tag*, which is determined by Bro and indicates the service being carried in the flow's payload. It then installs the Openflow match rules after checking the existence of pre-installed policies for the client's IP and the specific service (tag) at the corresponding switch.

The retrieved policies, which are defined in terms of Openflow actions, are installed in the *OF* switch which handles the client traffic. In case of a WiFi AP, the *OF* switch is the AP currently hosting the client's LVAP. After installation of the proper *OF* actions, the Traffic Manager installs flow rules on the *OF* switch hosting the Bro IDS, to prevent unnecessary load on the IDS. In other words, this prevents Bro from spending resources on already identified traffic flows.

Furthermore, an application developer can interact with the *Service Ranking* interface by defining: 1) a Service Name, 2) a Device ID or IP address and 3) a Priority. The former identifies a content provider (e.g., Youtube, Spotify) or a generic application layer service which the OpenSDWN framework system detects through *deep packet inspection*, e.g., by looking at the SSL certificate, URI or IP address space.

New services can be added by the network operator or public database through the *Service Ranking* interface. With the user participatory interface, the controller exposes a list of detectable services through the northbound API along with a list of IPs of devices connected to a particular network slice. Moreover, the *Service Ranking* interface can also be configured to only expose the list of services to a specific (e.g., connecting) client.

Reactive and Proactive Applications

Network applications written on top of OpenSDWN can function both reactively and/or proactively. Proactive applications are timer-driven whereas reactive applications use

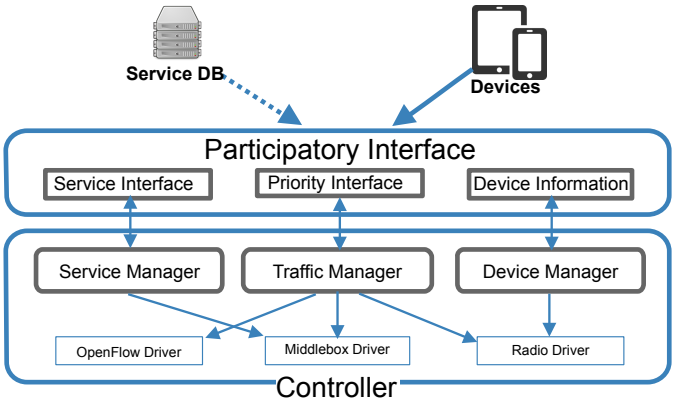


Figure 10: The Participatory Interface allows users to assign priorities to a particular service on a per-device basis.

Algorithm 2: Service Differentiation through DPI

```
begin
    if ServiceDetectEvent = True then
        (5 tuple, service) ← parsevMB(vMBMessage);
        match ← toOFMatch(5 tuple);
        policy ← getNetworkPolicy(ipClient,service);
        action ← buildOFAction(ipClient,markPkt,setTOS);
        lvap ← getLVAP(ipClient);
        if lvap exists then
            physicalAP ← getPhysicalAP(lvap);
            switchAP ← getOFSwitch(physicalAP);
            if physicalAP and switchAP then
                # Install rules at last hop;
                addOpenFlowRule(switchAP, match, action);
                addWDTXRule(lvap, policy);
                # Install Selective Tap at DPI;
                addOpenFlowRule(switchDPI, match, action:drop);
```

triggers and callbacks to handle events. The latter mode of operation is particularly interesting in the context of WiFi networks where channel quality can change quickly. To this end, in our current implementation, an application can utilize multiple measurement sources.

Radio agent interface: Reactive applications can make use of a publish-subscribe system which is exposed by the radio agent. The former can register a handler to receive notifications on a per-frame granularity. In our current implementation, applications register thresholds for link-based (PHY and MAC layer) rx-statistics like signal strength (*e.g.*, RSSI), bit-rate, and timestamp of the last received packet. For instance, an application can ask to be notified whenever a frame is received at an agent at an RSSI greater than -70dBm. In addition, applications can make use of measurements such as spectral scans or the channel busy time of the wireless environment.

Middlebox agent interface: Communication over the southbound interface is realized through the exchange of messages according to the vMB protocol. It functions based on two mechanisms:

- **Request-response model:** A controller can control a middlebox's state through a *Create Read Update Delete (CRUD)* API exposed by the MB's agent. This invokes the following action: the internal state can be read, modified or deleted, or new state installed. This is necessary for *proactive* remote control over the behavior of a middlebox by a controller.
- **Publish-subscribe model:** The role of the publisher is

taken by the agent, where the controller acts as the subscriber. The agent offers a set of events and event parameters to which the controller can subscribe. Since a controller is usually not interested in all event messages that can be sent by the publishing agent, a filter is used for selecting the content or the type of event messages.

OpenFlow statistics: OpenFlow provides per-flow and port-based statistics for entries in the flow tables of a switch. Applications can query these statistics through the controller to make traffic-aware routing decisions.

VI. Related Work

While software-defined networking and network virtualization principles have been studied intensively for wired environments, not much is known today about how to reap the corresponding benefits in the wireless and home network context. In general, it is difficult to port systems such as FlowVisor [17] to WiFi networks, and provide, e.g., bandwidth and CPU isolation on the access point.

While there exist a plethora of commercial enterprise WiFi solutions, which typically manage APs centrally via a controller (hosted either in the local network [18], or remotely in the cloud [19]), these solutions do not extend into the purview of cheap low-cost commodity AP hardware that is used by provider networks, nor do they support common, open and programmable interfaces.

OpenSDWN exploits the LVAP abstraction and builds upon Odin [20], [13] and Aeroflux [21], by introducing datapath programmability, network function virtualization, and a participatory interface. Over the last years, several interesting architectures have been proposed towards a more programmable WiFi, for example Dyson [22], an architecture for extensible wireless LANs which also defines a set of APIs for clients and APs to be managed by a controller. Flashback [23] proposes a control channel technique for WiFi networks, by allowing stations to send short control messages concurrently with data transmissions, without affecting throughput. This ensures a low overhead control plane for WiFi networks that is decoupled from the data plane. BeHop [24] is a programmable wireless testbed for dense WiFi networks as they occur in in residential and enterprise settings. Atomix [25] is a modular software framework for building applications on wireless infrastructure which achieves hardware-like performance by building an 802.11a receiver that operates at high bandwidth and low latency. FlexRadio [26] aims to unify RF chain techniques (MIMO, full-duplex and interference alignment), into a single wireless node, and enables a flexible RF resource allocation. DIRAC [27] proposes a split-architecture wherein link-layer information is relayed by agents running on the APs to a central controller to improve network management decisions. However, the requirement for special software or hardware on the client side, violates the design requirements of OpenSDWN. There are also systems that do not modify the client in order to deliver services. In DenseAP [28], channel assignment and association related decisions are made centrally by taking advantage of a global view of the network. However, slicing is not supported and also client association management is limited. Also Centaur [29] seeks to improve the datapath in enterprise WiFi networks by using centralization to mitigate hidden terminals and to exploit exposed terminals.

Picasso [30] enables virtualization across the MAC/PHY and uses spectrum slicing. It allows a single radio to receive and transmit on different frequencies simultaneously. MAClets [31] allows multiple MAC/PHY protocols to share a single RF frontend. These advances can be used by OpenSDWN (and already Odin) to operate multiple *LVAP*s with different characteristics on top of the same AP. Alternative approaches, such as [32] and [33], are incompatible with today's WiFi MAC/PHY and thus do not fit our design requirements. FICA [32] introduces a new PHY layer, that splits the channel into separate subchannels which stations can simultaneously use according to their traffic demands. Jello [33], a MAC overlay where devices sense and occupy unused spectrum without central coordination or dedicated radio for control. Enabling per-flow transmission settings will allow applications to centrally implement rate and power control. With OpenRadio [34], our system could also benefit from a clean-slate programmable network dataplane.

There is also a number of interesting works in the context of programmable *cellular* networks. C-RAN [35] (i.e., *Cloud-RAN*), is a new cellular network architecture for the future mobile network infrastructure. It combines centralized processing, cooperative radio and cloud, to render the radio access network more flexible. SoftCell [36] simplifies the operation of cellular networks and supports high-level service policies to direct traffic through sequences of *MB*s. Fine-grained packet classifications are pushed to the access switches, and to ensure control-plane scalability, a local agent at the base station caches the service policy. Openflow-based SDN also offers a number of benefits for mobile networks, including wireless access segments, mobile backhaul networks, and core networks. SoftRAN [37] uses SDN principles to redesign the radio access network, and seeks to provide the "big-base station abstraction": it coordinates radio resource management through its logically centralized control plane, managing interference, load, QoS, etc. through plug and play algorithms.

OpenSDWN promotes a unified programmable control over network and middleboxes. Middleboxes are ubiquitous in today's computer networks [38]. Besides virtualization, middleboxes [39], [40], [41] also play an important role in OpenSDWN for the fine-grained transmission control, which is based in deep-packet inspection [42], [43], [44]. Sekar et al. were one of the first to emphasize the importance of middleboxes, and in their middlebox manifesto [45], the authors argued for software-centric middlebox implementations running on general-purpose hardware platforms that are managed via open and extensible management APIs. Also Gember-Jacobson et al. [10] argue for a joint control of NFV and SDN components, and present the OpenNF architecture to coordinate the different control plane tasks, and to enable an efficient reallocation of flows across *network function (NF)* instances. Concretely, the southbound interface of OpenNF deals with the *NF* state diversity and seeks to minimize modifications. The northbound interface allows control applications to flexibly move, copy, or share subsets of state between *NF* instances. [46] presents a system for dynamic up- and down-

scaling of middleboxes following a split/merge approach. Merlin [47] is a language to provision *NFs* and entire *NF* chains. An interesting NFV platform is *ClickOS* [48], a virtualized software middlebox platform, based on light virtual machines. OPENSDWN also leverages the Click [15] framework.

Home networks have received particular attention over the last years [49], [50]. Users are offered more flexibilities on how their network can be optimized [51], [52], sometimes even over participatory interfaces [49], helping home users to improve performance [53]. Programmable middleboxes can also be exploited to provide a faster ISP service delivery [54].

Bibliographic Note. Some material of this paper were presented at ACM SOSR 2015 [55] and at the ACM CoNEXT 2016 CAN Workshop [56].

VII. CONCLUSION

We have presented OPENSDWN, a programmable and virtualized WiFi network which may be used as a prototype to experiment and demonstrate a more flexible and fine-grained network management environment for WiFi networks. We understand our system as an *enabler* of more flexible WiFi networks. How to optimally *exploit* the resulting flexibilities (e.g., in order to provide QoS guarantees) or how to fine-tune performance (e.g., of function migration), are orthogonal questions, and are left for future research.

Acknowledgment We are thankful for many discussions with Henry Owen, James Kempf, and Thomas Hühn. We also like to thank Sven Zehl and Tobias Steinicke for their efforts on the wireless datapath. Research supported by the Federal Ministry of Education and Research (BMBF) Software Campus "SDWN" Project Grant (Reference number 01IS12056) as well as by Aalborg University's PreLytics project.

REFERENCES

[1] Cisco, "Cisco Service Provider Wi-Fi: A Platform for Business Innovation and Revenue Generation," in *Cisco*, 2015.

[2] P. Valerio, "Using carrier wifi to offload iot networks," in *InformationWeek: Network Computing*, 2014.

[3] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN," *Queue*, vol. 11, no. 12, pp. 20:20–20:40, Dec. 2013.

[4] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: A Software Defined Internet Exchange," in *Proc. SIGCOMM '14*.

[5] S.K. Fayazbakhsh et al., "Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags," in *Proc. USENIX NSDI*, 2014,.

[6] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving High Utilization with Software-driven WAN," *SIGCOMM Comput. Commun. Rev. 2013*.

[7] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," *SIGCOMM Comput. Commun. Rev.*, 2013.

[8] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *Internet Computing, IEEE*, vol. 17, no. 2, pp. 20–27, March 2013.

[9] B. Anwer, T. Benson, N. Feamster, D. Levin, and J. Rexford, "A Slick Control Plane for Network Middleboxes," in *Proc. HotSDN '13*.

[10] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: Enabling Innovation in Network Function Control," in *Proc. ACM SIGCOMM*, 2014.

[11] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying Middlebox Policy Enforcement Using SDN," in *ACM SIGCOMM '13*.

[12] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory networking: An api for application control of SDNs," *SIGCOMM Comput. Commun. Rev.*, 2013.

[13] J. Schulz-Zander, L. Suresh, N. Sarrar, A. Feldmann, T. Hühn, and R. Merz, "Programmatic Orchestration of WiFi Networks," in *Proc. USENIX ATC '14*.

[14] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Comput. Netw. Dec. 1999*.

[15] "Click modular router project," http://read.cs.ucla.edu/click.

[16] *ISO/IEC 23009-1:2014 - Dynamic adaptive streaming over HTTP (DASH)*, 2014.

[17] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the production network be the testbed?" in *OSDI '10*.

[18] "Meru Networks," http://www.merunetworks.com. [Online]. Available: http://www.merunetworks.com

[19] "Meraki," http://www.meraki.com/. [Online]. Available: http://www.meraki.com/

[20] J. Schulz-Zander, L. Suresh, N. Sarrar, A. Feldmann, T. Hühn, and R. Merz, "Programmatic Orchestration of WiFi Networks," in *Proc. USENIX ATC '14*.

[21] J. Schulz-Zander, N. Sarrar, and S. Schmid, "AeroFlux: A Near-Sighted Controller Architecture for Software-Defined Wireless Networks," in *Proc. Open Networking Summit (ONS)*, 2014.

[22] R. Murty, J. Padhye, A. Wolman, and M. Welsh, "Dyson: an architecture for extensible wireless LANs," in *Proc. USENIX ATC '10*.

[23] A. Cidon, K. Nagaraj, S. Katti, and P. Viswanath, "Flashback: decoupled lightweight wireless control," in *ACM SIGCOMM '12*.

[24] Y. Yiakoumis, M. Bansal, A. Covington, J. van Reijendam, S. Katti, and N. McKeown, "BeHop: A Testbed for Dense WiFi Networks," in *Proc. WiNTECH '14*.

[25] M. Bansal, A. Schulman, and S. Katti, "Atomix: A Framework for Deploying Signal Processing Applications on Wireless Infrastructure," in *Proc. NSDI*, 2015.

[26] B. Chen, V. Yenamandra, and K. Srinivasan, "FlexRadio: Fully Flexible Radios and Networks," in *Proc. NSDI 15*, 2015.

[27] P. Zerfos, G. Zhong, J. Cheng, H. Luo, S. Lu, and J. J. Li, "DIRAC: a software-based wireless router system," in *MobiCom*, 2003.

[28] R. Murty, J. Padhye, R. Chandra, A. Wolman, and B. Zill, "Designing high performance enterprise Wi-Fi networks," in *Proc. NSDI '08*.

[29] V. Shrivastava, N. Ahmed, S. Rayanchu, S. Banerjee, S. Keshav, K. Papagiannaki, and A. Mishra, "CENTAUR: realizing the full potential of centralized wlans through a hybrid data path," in *Proc. MobiCom '09*. [Online]. Available: http://doi.acm.org/10.1145/1614320.1614353

[30] S. S. Hong, J. Mehlman, and S. Katti, "Picasso: Flexible RF and Spectrum Slicing," in *ACM SIGCOMM 2012*.

[31] G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, and I. Tinnirello, "MAClets: active MAC protocols over hard-coded devices," in *Proc. CoNEXT '12*.

[32] K. Tan, J. Fang, Y. Zhang, S. Chen, L. Shi, J. Zhang, and Y. Zhang, "Fine-grained channel access in wireless LAN," in *ACM SIGCOMM 2010*.

[33] L. Yang, W. Hou, L. Cao, B. Y. Zhao, and H. Zheng, "Supporting demanding wireless applications with frequency-agile radios," in *USENIX NSDI '10*.

[34] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "OpenRadio: a programmable wireless dataplane," in *HotSDN '12*.

[35] China Mobile Research Institute, "C-RAN: The road toward green RAN," in *White Paper*, 2011.

[36] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "SoftCell: Scalable and Flexible Cellular Core Network Architecture," in *CoNEXT '13*.

[37] A. Gudipati, D. Perry, L. E. Li, and S. Katti, "SoftRAN: Software Defined Radio Access Network," in *Proc. HotSDN '13'*.

[38] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. ACM SIGCOMM 2012*.

[39] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," Technical Report, Tech. Rep., 2013.

[40] S. Rajagopalan, D. Williams, and H. Jamjoom, "Pico replication: A high availability framework for middleboxes," in *Proc. 4th annual Symposium on Cloud Computing (SOCC)*, 2013.

[41] J. Khalid, A. Gember-Jacobson, R. Michael, A. Abhashkumar, and A. Akella, "Paving the way for nfv: Simplifying middlebox modifications using statealyzr," in *Proc. 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016, pp. 239–253.

[42] "Re-examining the performance bottleneck in a {NIDS} with detailed profiling," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 768 – 780, 2013.
[43] H. Dreger, C. Kreibich, V. Paxson, and R. Sommer, "Enhancing the accuracy of network-based intrusion detection with host-based context," in *Proc. DIMVA*, 2005.
[44] R. Sommer, M. Vallentin, L. De Carli, and V. Paxson, "HILTI: An Abstract Execution Environment for Deep, Stateful Network Traffic Analysis," in *Proc. IMC*, 2014.
[45] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi, "The middlebox manifesto: Enabling innovation in middlebox deployment," in *Proc. ACM HotNets*, 2011.
[46] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/Merge: System Support for Elastic Execution in Virtual Middleboxes," in *Proc. NSDI*, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2482626.2482649
[47] R. Soulé, S. Basu, P. J. Marandi, F. Pedone, R. Kleinberg, E. G. Sirer, and N. Foster, "Merlin: A language for provisioning network resources," in *Proc. CoNEXT*, 2014.
[48] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the Art of Network Function Virtualization," in *Proc. NSDI*, 2014.
[49] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and P. Bahl, "An operating system for the home," in *Proc. NSDI*, 2012.
[50] Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown, "Slicing home networks," in *Proc. HomeNets '11*.
[51] R. Mortier, T. Rodden, T. Lodge, D. McAuley, C. Rotsos, A. W. Moore, A. Koliousis, and J. Sventek, "Control and understanding: Owning your home network," in *Proc. COMSNETS*, 2012.
[52] Y. Yiakoumis, S. Katti, T.-Y. Huang, N. McKeown, K.-K. Yap, and R. Johari, "Putting home users in charge of their network," in *Proc. UbiComp '12*.
[53] M. S. Seddiki, M. Shahbaz, S. Donovan, S. Grover, M. Park, N. Feamster, and Y.-Q. Song, "FlowQoS: QoS for the Rest of Us," in *Proc. ACM HotSDN*, 2014.
[54] K. R. Khan, Z. Ahmed, S. Ahmed, A. Syed, and S. A. Khayam, "Rapid and scalable isp service delivery through a programmable middlebox," *SIGCOMM Comput. Commun. Rev. 2014*.
[55] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, and A. Feldmann, "Opensdwn: Programmatic control over home and enterprise wifi," in *Proc. ACM Sigcomm Symposium on SDN Research (SOSR)*, 2015.
[56] J. Schulz-Zander, R. Lisicki, S. Schmid, and A. Feldmann, "Secuspot: Toward cloud-assisted secure multi-tenant wifi hotspot infrastructures," in *Proc. ACM CoNEXT Workshop on Cloud-Assisted Networking (CAN)*, 2016.

Julius Schulz-Zander is a Postdoctoral Researcher at the Fraunhofer Heinrich-Hertz-Institute (HHI), Germany. He also continues working as a consulting Postdoc in Prof. Anja Feldmann's group at Technische Universität Berlin (TU Berlin), Germany. Previously, he was a research assistant at TU Berlin and Deutsche Telekom Innovations Laboratories (T-Labs) in Berlin, Germany. From 2007-2011, he was a student worker working at the T-Labs and a research scholar in Prof. Nick McKeown's group at Stanford University in 2009. His current research is centered around network architectures and security, virtualization, wireless networking, machine learning, and embedded systems.

Carlos Mayer is a software engineer in a Berlin-based startup company working on distributed services-oriented architectures. He received a B.Sc. in Electrical Engineering from the Federal Technical University of Paraná, Brazil, and a M.Sc. in Computer Engineering from the Technische Universität Berlin in 2015. During his masters studies, he focused on computer networking, specifically on wireless networking, Network Function Virtualization, and SDN.

Bogdan Ciobotaru is a software developer in a Berlin-based startup working on Backend Developer and Data Scientist. He received his M.Sc. in Computer Engineer from Technische Universität Berlin and his B.Sc. from University Politehnica of Bucharest in 2015 and 2010, respectively. His main research interest are telecommunication systems, wireless networking, network architectures and security.

Stefan Schmid is an Associate Professor at Aalborg University, Denmark (since 2015). He received his MSc (2004) and PhD degrees (2008) from ETH Zurich, Switzerland. Subsequently, Stefan Schmid worked as postdoc at TU Munich and the University of Paderborn (2009). From 2009-2015, he was a senior research scientist at the Deutsche Telekom Innovations Laboratories (T-Labs) in Berlin, Germany. His research interests revolve around the fundamental and algorithmic problems of networked and distributed systems.

Anja Feldmann Since 2006 Prof. Anja Feldmann, Ph.,D. is a full professor at the Technische Universität Berlin, Germany. From 2000 to 2006 she headed the network architectures group first at Saarland University and then at TU München. Before that (1995 to 1999) she was a member of the Networking and Distributed Systems Center at AT&T Labs – Research in Florham Park, New Jersey. Her current research interests include Internet measurement, traffic engineering and characterization, network performance debugging, intrusion detection, and network architecture. She has been Co-Chair of ACM SIGCOMM and IMC and Co-PC-Chair of ACM SIGCOMM, ACM IMC, and ACM HotNets. She is a member of the German Academy of Sciences Leopoldina, the BBAW, and the supervisory board of SAP SE. She received a M.S. degree in Computer Science from the University of Paderborn in 1990 and M.S. and Ph.D. degrees in CS from CMU in 1991 and 1995, respectively.