SDNizing the Wireless LAN - A Practical Approach

Manzoor A. Khan Patrick Engelhard Tobias Dörsch

DAI Labor, TU Berlin, Berlin Germany,
{manzoor-ahmed.khan,patrick.engelhard,tobias.doersch}@dai-labor.de

Abstract

The emerging Internet of Things (IoT) paradigm and a plethora of diverse applications provision more flexible network management. Software Defined Networking (SDN) occupies the pivotal role in realizing such flexible network management. However, the gain of this potential panacea is still unmeasurable in a real sense especially when wireless medium is part of the equation, as the validation frameworks mostly skip capturing realistic system dynamics. In this paper, we study the performance gain of SDN control implemented in a physical testbed comprising of a virtualized core and a WLAN access network. With this contribution, we aim at realizing a more realistic environment where the impact of system dynamics on the stakeholders (users and operators) may be studied. We developed a mechanism to map the logical wireless channels over the physical wireless interface of the access point. SDN (OpenDaylight) control application for mobility management, the mapper tool, a visualization and control GUI, and Android applications are amongst the main contribution of this work.

Keywords: software defined networking, OpenFlow, Open vSwitch, experiment automation

1 Introduction

The recent past has advocated a rapid evolution in mobile communication technologies, which enables the transition from a monolithic architecture to a shared architecture and ensures ubiquitous connectivity. Such transformation, when coupled with envisioned bandwidth hungry applications, have triggered the need for high data rates and extremely low communication latencies. Now that dust around 4G has settled down to a great extent, the community is looking for newer technologies to achieve the envisioned requirements of future mobile networks, also referred to as 5G, aiming at improving the stakeholders objective functions. For this concept to be realized, the path is paved by the following technical and economic evolutions: widely available broadband Internet, reduced connectivity costs, more devices being manufactured with built-in sensors and WiFi capabilities, and immense market penetration of smart phones. We believe that 5G - in contrast to earlier generations - will not only improve the end user services, but also go beyond enabling communication between people by realizing machine-to-machine communication and the concept of Internet-of-Things (IoT). The challenges stemming from realizing the vision of *connectivity everywhere* for highly dynamic device layer entities and immensely heterogeneous applications define a major portion of 5G. The communication requirements are even more stringent when the users or sensors are mobile. The expanding networks, the inclusion of many entrants and their dynamic relationships, and virtualized network sections result in a very complex management task of the network. SDN and autonomic network management are seen as the enabling concepts that come to rescue and help in solving the pressing challenges. The SDN architecture propagates the separation of the control plane from the data plane, which enables the flexible hosting of network control functions in different settings and platforms e.g., in centralized or distributed fashions, in physical machines or virtual instances in a cloud network. When applying SDN solutions to the wireless access and mobile networks, it promises to provide efficient management and control over wireless operations by providing a unified management and control platform. Yet, the abstraction of centralized control on one hand and simple forwarding elements on the other, can not be achieved easily in wireless networks. The current SDN technology mostly caters to the needs of wired networks in data center or enterprise network settings and needs to be advanced to allow for monitoring of wireless parameters and for exerting control on wireless network infrastructure within the SDN control layer. A number of approaches to SDNized wireless networks exist in the research literature (e.g., (Kreutz et al., 2015), (Xia et al., 2015) and references therein). OpenFlow is still in its infancy and is evolving to meet the requirements of wireless communication. The Open Networking Foundation presented a report (McKeown et al., 2008) identifying challenges of future wireless networks and how SDN can be a solution to these issues. Research however struggles to include SDN in wireless networks: Some works rely on mobile node cooperation (Schulz-Zander et al., 2015), others create configuration overhead (Suresh et al., 2012), which leads to more load on all involved nodes during handover. Aetherflow (Yan et al., 2015) and its predecessor TinyNBI (Casey et al., 2014) are frameworks that allow the controller to manage the capabilities of a wireless access point (e.g., mapping of logical ports to physical ports, transmission power), to receive events corresponding to the wireless network, or to gather statistics of wireless ports.

The research community has also been actively ad-

dressing the inherent scalability issue of SDNized networks. (Curtis et al., 2011) proposes to offload statistics gathering and management of micro-flows onto switches, thus reducing delays introduced by controller involvment. DIFANE (Rexford et al., 2011) includes switches in the control-plane. *Authority Switches* are introduced, which store a partition of all rules required to operate the network. Instead of switches sending every cache-miss to the controller, they forward the packet which triggered the miss to an *Authority Switch*, which in turn caches the required rules in the ingress switch. In (Rexford et al., 2011), flooding of packets causing a cache-miss is used to reduce packet-loss during handover of mobile nodes. If a node changes its point of attachement, the edge switch the node was connected to floods packets for the mobile node through the network in order to reach the new acces point. With this approach, the new point of attachment may receive packets sent during handover and can forward them to the mobile node, thus reducing packet loss. However, evaluation of the approach is only done by emulating the network and not in a physical testbed.

When it comes to SDN (wireless) networks, researchers often use network emulators like Mininet to evaluate new approaches to utilizing the advantages of SDN. Even though this approach has the advantage of not having to deal with challenges like setting up and configuring physical networks or creating a suitable network environment to simulate real-world conditions, it still lacks the capability to capture the realistic wireless dynamics. In general, the validation frameworks for most of the approaches are abstracted to higher levels, which we believe does not realistically capture the system dynamics. In this work, we discuss a full scale SDNized wireless LAN testbed that we developed using both virtual and physical network entities. The testbed is targeted to study the performance of different contributions (e.g., mobility management, location management, resource allocation, network selection, etc.) in a more realistic environment which involves both wireless and physical mediums, where the topology thickness may be dynamically adjusted for different settings, and where the network management procedures and protocols are fully implemented similarly to that of real deployment. Hence, we claim that our contributed and developed experimental setup will serve as a fitting validation framework for SDN control of wireless networks.

2 SDNized Wireless LAN Testbed

The testbed is designed and developed to function in different modes, of which the two prominent ones are: i) Controller communication mode - in this mode, the trigger events from the forwarding entities are sent to SDN controller, where the decision of trigger handling is carried out. ii) Inter-entities communication mode - inspired by (Jia, 2015), we propose to handle triggering events in a way different to the SDN paradigm of centralized control, i.e. the events are sent to other forwarding entities, which may take over partial control responsibilities delegated by the controller. In this paper, we focus on providing the details of the former due to space restriction. However, it should be highlighted that the two functional modes add additional capability to the testbed. Solutions which deviate from the classical SDN control principles (where based on the Open-Flow protocol, all the trigger events must be forwarded to controller) may be tested and their performance gains may be measured against those following the classical SDN rules. In what follows next, we provide the details of the developed testbed and its components for controller communication mode. Figure 1 pictorially presents the testbed environment, which comprises the following major components:

2.1 Central Manager

It is responsible for high-level policy definition and visualization. The policies are translated into northbound applications, which are then executed by the SDN controller positioned at this layer.

2.2 Aggregator and Transport Network

This component comprises a mesh of forwarding entities, which are implemented using virtual and physical switches. This component forms topology design and decides the topology thickness for different experiment settings (e.g. experiments to study the impact of topology thickness on the trigger propagation, or flow rules definition in the switches on a path).

2.3 Network Edge

This component comprises a set of edge switches that interface the Access Points (APs). It is worth highlighting here that an AP is connected to a port of an edge switch, whereas the wireless access port of the AP provides connectivity to multiple mobile devices. This provisions the change in per-port handling of OpenFlow i.e., a disconnected mobile device does not imply the disconnection of the edge switch's port to which the AP is connected. Hence, the controller needs to implement efficient host tracking to realize SDNized wireless networks. To achieve this, we implement the *WiFi-Monitor* tool, the details of which will follow later in this section.

2.4 Android Measurement Application

This component is a network traffic generating Android application running on Mobile- and Static Node. It can act as both a sender and a receiver. As sender, the application sends UDP packets to a receiving device with a configurable rate. These packets contain the sending timestamp and a packet id. The receiver stores these received packets. In order to automate experiments, the application is able to switch betwen access points by itself. After the experiment execution, the sender sends a *stop*-packet to inform the receiver of the end of the experiment. As logging from the Android device during an experiment would affect the networks performance, the application generates logs at

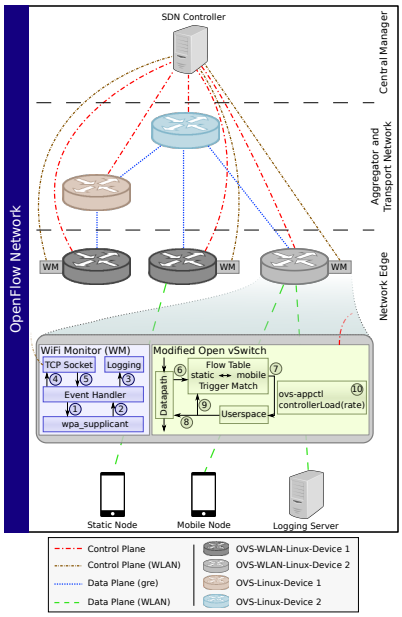

Figure 1. SDNized Wireless LAN - Experimental Setup

the end of experiments.

2.5 Experiment Automation

Running an experiment in a physical testbed is slow compared to an emulated environment. In order to automate both the setup and the execution of the experiment, we develop several tools. The previously discussed Android application is used to trigger the handover process while streaming data.

A machine attached to the network runs the experiment and configures the OpenFlow network using a script. After the setup phase, this machine sends a signal to both Android devices to start the experiment. It is also used to collect logs produced during the experiment from the log server, as well as to present a user interface to show the experiment status and results in real time. We also extended OVS to create load on the controller (configurable via ovs-appctl) by frequently sending dummy *PACKET_IN's* to the controller. As sending this many requests to the controller may have an impact on the switch performance, we use an OVS-Node which is not playing a role in the handover process.

WiFi-Monitor

We develop this tool to achive efficient tracking of mobile hosts required for wireless networks. The tool is implemented using C++ and runs on linux-based systems. As shown in Figure 1, the *WiFi-Monitor* registers itself as a receiver of WiFi-Events to the *wpa_supplicant* module (Malinen) (1). The *wpa_supplicant* is responsible for managing L2-authentication of mobile hosts. As soon as a host connects (disconnects) to (from) an AP, the accounting *WiFi-Monitor* instance receives a callback (2), logs the event (3), and sends a message to the SDN controller application (4). With this message, we forward information about the host (MAC-Address), the event (connect/disconnect), and which AP this event corresponds to (MAC-Address). Using this information, the controller is aware of the position of all hosts in the system at any time after initialization. The *WiFi-Monitor* may also be used for various measurements, of which a few will be detailed in section 3. For these measurements, messages from the SDN controller application are received and logged (3, 5). It should be furthermore noted, that we also modify the OVS implementation to address scalability issues of central control by delegating specific control responsibilities to forwarding entities. However, the implementation details are ommited due to space constraints.

We also test the efficiency of the *Wifi-Monitor* against the Host Tracker service of the OpenDaylight controller (odl, *IfIptoHost*) and found that the built-in OpenDaylight Host Tracker is not well suited to track mobile hosts. If a mobile host changes its point of attachment, the information received from the Host Tracker is mostly outdated and refreshed only after a few seconds. This issue is overcome by the contributed *WiFi-Monitor*. The OpenDaylight application contains a server to which all *Wifi-Monitor* instances connect. On each L2-Event the controller receives from the *WiFi-Monitor* instances, the host-to-AP mapping is updated in order to keep track of the hosts' current positions.

OpenDaylight application

Our ODL application comprises three major parts:

Monitoring: This module contains the server to which the *WiFi-Monitor* instances connect. It internally stores the current Points of Attachement (PoA) of all hosts in the system.

Mobility Learning: The learning module receives a callback from the *Monitoring-Module* on every L2-Event. These events are evaluated in order to predict a host's next PoA. Thereby, a probability vector per host is created $p_h(t,s)$, which describes the probability of a host h to connect to AP s at the discrete time-step t. Each L2-Event triggers an update of this probability vector for the host related to the event. The update step follows the learning framework proposed in (Khan and Tembine, 2012). We avoid further details of the implemented algorithm due to space limitation and partially different focus of this work. However, interested readers are encouraged to refer to it for detailed information. After updating the probability vector and the payoff vector for a host, a new prediction is generated. This prediction estimates the target PoA of the host.

Routing: The *Routing-Module* of our controller application calculates and applies routes to the network. This module uses the *Monitoring-Module* to determine the endpoints (APs) to which the two communicating hosts are attached. Routing can be run in two different modes, namely *reactive* and *proactive*. In *reactive* mode, the routes are only applied when a *PACKET_IN* is received. The *DL_SRC* and *DL_DST* of incoming packets are matched

against installed flow rules, which means source- and/or destination MAC-address of a packet need to match for the flow to be used. When the *Monitoring-Module* receives a L2-Disconnect-Event, all routes for the disconnected host are immediately removed. In *proactive* mode, the controller always installs two routes per connection. For example, consider a host h_2 that has a connection to host h_1. The controller installs two routes in the network and both are associated with h_2. The first route is the active route, which is actively used and contains all flows to connect h_1 with h_2. The second route is inactive. This route is generated using the *Mobility-Learning-Module*. The route contains all flows to connect h_1 (at the current PoA) to h_2 at the predicted PoA. This route has a lower priority than the active route and is therefore never applied by OVS. When h_2 disconnects from its current PoA, the *Routing-Module* will receive a callback (from the *Monitoring-Module*) and all active flows of h_2 are removed, which leads to the inactive routes becoming active. On the next L2-Connection-Event, the behavior depends on the correctness of the last prediction. If the prediction was correct, the connection between h_1 and h_2 is already established. If the prediction was wrong, the installed flows are removed and correct flows are installed. Finally, a new inactive route is installed for the new prediction for future movement of h_2.

3 Mobility Management - A Use-Case Scenario Implementation

Consider the scenario presented in Figure 2, where the mobile user is streaming video from a video server while the mobile device is connected to AP1. The transport network consists of a set of switches. The user now enters the coverage of AP2 and thereby triggers the handover process. We now detail how we realized this scenario and discuss how the handover procedures are carried out in the two modes of developed testbed.

Figure 3 depicts the basic structure of technical components for the use case scenario implementation. All the nodes in the network (with the exception of the hardware switch HWS) are commodity laptops. Where multiple NIC's are required USB-(Ethernet/WiFi) adapters are used. As can be seen in Figure 3, all devices are connected to a switch via Ethernet. The OpenFlow network we use for this scenario, as described in Figure 1, is realized by connecting the OVS-Nodes using GRE Tunnels. Each end-point of the tunnels is seen by the OpenDaylight Controller as a logical port. On the device *OVS-WLAN-Linux-Device 1*, there are two instances of OVS running. Each instance has control over one physical WiFi interface. One of these WiFi interfaces is the built-in NIC, the other WiFi interface is a plugged in USB-WiFi adapter. For simplicity of the scenario, the IP addresses of the mobile nodes are static. This implies that the APs are using the same subnet.

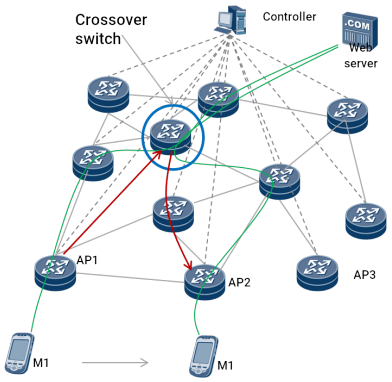

Figure 2. Use case scenario

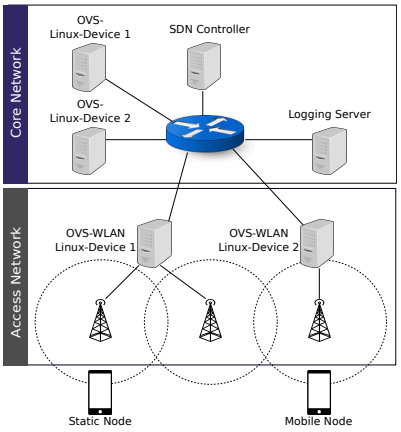

Figure 3. Physical Testbed Setup

3.1 Mobility Handling in Developed Testbed

Upon generation of the first handover trigger, i.e. host M1 disconnects from AP1, the edge switch running *WiFi-Monitor* generates a Disconnect-Event that is forwarded to the SDN-Control-Application. The Application reacts to this event by replying to the *WiFi-Monitor* and by deleting all active flows available for M1 while updating its internal database to store the hosts PoAs. Upon generation of the second handover trigger, i.e. host M1 connects to AP2, the edge switch generates a corresponding Connect-Event, which is forwarded to the SDN-Control-Application, which handles this event depending on the current mode (*reative/proactive*), as described in Section 2.

In this experiment, we capture different delay components, which impact the overall handover cost. To measure the Propagation-Delay (the time the trigger event takes to reach the controller over the transport network) of a layer 2 event, the *WiFi-Monitor* starts a timer directly before sending an Event-Message (connect/disconnect) to the SDN-control-Application, which immediately answers to this event upon reception. The previously started timer is stopped as soon as the *WiFi-Monitor* receives the answer from the controller.

The process to measure L2-Delays is similar. All devices that run an instance of the *WiFi-Monitor* (i.e., ev-

ery AP) are synchronized using the NTP protocol (D.L. Mills, 1985). Every L2-Event is logged to our logging server. The difference in the timestamps of an L2-Event-Pair (i.e., Disconnect-Event and corresponding Connect-Event) is considered to be the L2-Delay.

The Pre-Compuation-Delay is measured within the SDN-Controller and describes how long the reception of the *PACKET_IN* and the handling of it are apart. It describes how long a *PACKET_IN* was queued before being processed.

The Compuation-Delay is also captured within the SDN-Controller and indicates how much time the Contoller needed for calculating the new routes and flows that are created due to a *PACKET_IN*.

The Flow-Setup-Delay is the counterpart to the Propagation-Delay. It describes the time between sending a *FLOW_MOD* from the SDN-Controller and applying it in the OVS. When the Controller sends a *FLOW_MOD*, a timer is started. Each OVS immediately answers to a *FLOW_MOD*. When the Controller receives this answer, the corresponding timer is stopped.

The Flow-Application-Delay is captured within the OVS and describes the time between the modification/insertion of a new flow and the first application of it.

3.1.1 Logging

We utilize the widely used logging module *rsyslog* (logging) for remote logging, which provides the option to forward the system logs to a remote server. We use this mechanism to forward log messages (i.e. previously described delay measurements) from all involved devices to our logging server, which is attached to our local network via Ethernet. Additionally, the logging server connects to one of the APs to be available from within the OpenFlow network. In the SDN-Control-Application, we realize special handling of the logging server. If the *Monitoring-Module* recognizes the MAC-Address of the logging server, an action is triggered to install routes from all APs to the logging server solely using the *DL_DST* field as a match. This way, the logging server is at all times reachable within the OpenFlow network. To realize the forwarding of the Android logs, we utilize the Android Application *Logcat to UDP* (Madzik). We configure *Logcat to UDP* to forward the logs of our Android Application to the remote logging server.

3.1.2 Running the Experiment

Having the testbed setup as described above, we carried out the experiments, in which two Android devices are used to measure the network's behavior during handover. Static Node (as given in Figure 3) is connected to WiFi interface 1 of *OVS-WLAN-Linux-Device 1*. Mobile Node is switching between the two other available WiFi networks. When the experiment starts, Static Node acts as a sender while Mobile Node runs in receiving mode. After Static Node starts sending, Mobile Node switches its PoA several times.

As we know that as the Android device associates to or disassociates from an Access Point, the new topology of the network requires changes to flow tables in switches. These changes are applied differently depending on controller mode. For instance, in reactive mode, the controller installs flows after the first *PACKET_IN* from the OVS-Node at the Access Point from which Mobile Node disconnected. The message to the controller contains the header of the received packet and, depending on the configuration, the packet content. The controller receives the *PACKET_IN* and calculates the path through the network and installs the flows in switches along that path.

However, if the controller is running in proactive mode, the controller predicts to which Access Point a mobile host will connect to in the future. This allows the controller to preconfigure the network with a second route from Static Node to the future location of Mobile Node. This route is installed with a lower priority than the actual route so that only the actual route is used. When Mobile Node disconnects from its current Access Point, the WiFi-Monitor detects the disassociation and sends the information about the topology change to the controller. The controller reacts by deleting the current route, resulting in the 2nd preinstalled route to be the active one. It should be highlighted that proactive controller mode reduces handover delay especially if the disassociation event is propagated to the controller as quick as possible. The vanilla OpenFlow implementation however does not immediately notice that a device left. Together with the *WiFi-Monitor* detecting (dis-)association events much faster, proactive controller mode can significantly reduce handover delay.

3.1.3 Parsing and aggregating results

Each run of the experiment generates log files for each node running software: OVS-Nodes, controller, and Android devices. In order to obtain results, logged events have to be matched to their counterparts and events caused by a single move between Access Points aggregated to evaluate the networks behavior during a single handover. All events but those occuring on the Android devices are immediatly logged to the log server. As we can expect events to have a certain order, only information logged by the Android devices have to be mapped to their respective groups of network events.

The *WiFi-Monitor* is used to improve detection of connection events. We compare the speed and accuracy of Host Tracker to the developed *WiFi-Monitor*. Table 1 shows the statistics of Host Tracker. The *Bad Answers* column describes how often Host Tracker returned a wrong PoA for a queried host. The *Time Lost* column shows how much time was lost until a correct answer was received. This time also depends on the request frequency, which is depending on the amount of *PACKET_IN*s generated. One can see that the behavior is quite unstable with high peaks in the amount of bad answers as well as the lost time. When using the *WiFi-Monitor*, we achieve an average of 9.42*ms* delay over 400 handovers due to the propagation

Table 1. Error Measurement of Host Tracker in 136 Handovers

	Bad Answers	Time Lost(ms)
Average	1.65	121.7
Sum	225	16,556
Max	16	2415

delay. This means that the maximum delay using the Host Tracker (2415*ms*) is higher than the overall delay in 136 runs using the *WiFi-Monitor* ($9.42ms \cdot 136 = 1,281.12ms$). Since the control application reacts to the L2-Events, we do not obtain any bad answers.

4 Conclusions

In this paper, we provided the details of authors' designed and developed testbed for SDNized wireless LAN. We discussed different components of the testbed and elaborated on the technologies used therein. To give better insight of the demonstrator and assist the researchers of SDN topics, we have also provided a discussion on our proposed SDN application and its implementation. The use case scenario summarizes the use of the developed testbed for mobility management.

Acknowledgment

This work is partially funded by iMoveFAN, a collaborative project with Huawei Research Germany.

References

C Jasson Casey, Andrew Sutton, and Alex Sprintson. tinyNBI: Distilling an API from essential openflow abstractions. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 37–42. ACM, 2014.

Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Sujata Banerjee, Praveen Yalagandula, and Puneet Sharma. DevoFlow: scaling flow management for high-performance networks. *ACM SIGCOMM Computer Communication Review*, 41(4):254–265, August 2011.

D.L. Mills. Network Time Protocol (NTP). RFC 958, Sept. 1985.

Xiaozhou Jia. SBMP: An SDN-based Mobility Management Protocol to Support Seamless Handover. Master's thesis, The University of Tokyo, 2015.

M. A. Khan and H. Tembine. Random matrix games in wireless networks. In *Global High Tech Congress on Electronics (GHTCE), 2012 IEEE*, pages 81–86, Nov 2012. doi:10.1109/GHTCE.2012.6490129.

Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

logging. RSYSLOG The Rocket-Fast System For Log Processing. Website. `http://www.rsyslog.com/`; revised July 12, 2016.

J. Madzik. Chemik/logcatudp. Website. `https://github.com/Chemik/logcatudp`; revised July 12, 2016.

J. Malinen. Linux wpa/wpa2/ieee 802.1x supplicant. Website. `https://w1.fi/wpa_supplicant/`; revised July 12, 2016.

Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2): 69–74, March 2008. ISSN 0146-4833.

odl. Controller projects' modules/bundles and interfaces. Website. `https://wiki.opendaylight.org/view/Controller_Project's_Modules/Bundles_and_Interfaces`; revised July 12, 2016.

Jennifer Rexford, Michael J Freedman, Minlan Yu, and Jia Wang. Scalable flow-based networking with DIFANE. *ACM SIGCOMM Computer Communication Review*, 41(4):351–362, October 2011.

Julius Schulz-Zander, Carlos Mayer, Bogdan Ciobotaru, Stefan Schmid, and Anja Feldmann. Opensdwn: programmatic control over home and enterprise wifi. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 16. ACM, 2015.

Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feld- WLANs with Odin. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 115–120. ACM, 2012.

W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie. A Survey on Software-Defined Networking. *Communications Surveys Tutorials, IEEE*, 2015. ISSN 1553-877X. doi:10.1109/comst.2014.2330903.

M. Yan, J. Casey, P. Shome, A. Sprintson, and A. Sutton. Aetherflow: Principled wireless support in SDN. In *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, pages 432–437, Nov 2015. doi:10.1109/ICNP.2015.9.