SWIFT: Bringing SDN-Based Flow Management to Commodity Wi-Fi Access Points

Seppo Hätönen*, Petri Savolainen*, Ashwin Rao*, Hannu Flinck† and Sasu Tarkoma*
*University of Helsinki, †Nokia Bell Labs

***Abstract*—Wi-Fi networks are largely served using over-the-counter commodity Wi-Fi routers, access points (APs), and possibly with wireless controllers. Existing approaches to bring the benefits of Software-defined Networking (SDN) to such Wi-Fi networks suffer from availability, hardware requirements, and scalability issues. For instance, these approaches do not support enterprise APs whose firmware cannot be modified. Furthermore, we observe that using SDN controllers for managing all the flows traversing these APs and routers requires more than just installing an SDN switch on these devices. In this paper, we present our solution called SWIFT: Software-defined Wi-Fi Flow Management, for bringing SDN-based flow management to Wi-Fi networks using commodity Wi-Fi APs, Wi-Fi routers, and Wi-Fi controllers. We also detail its benefits, shortcomings, and possible use cases. Specifically, our solution significantly lowers the barrier-to-entry for deploying and conducting research on software-defined Wi-Fi networks.**

***Index Terms*—Network Management, SDN, Wi-Fi.**

I. Introduction

Wi-Fi is becoming the communication medium of choice in our homes and offices. This has compelled manufacturers of televisions, home-entertainment systems, and other devices that traditionally used wires for connectivity, to support Wi-Fi. Paralleling the growth of Wi-Fi is the growing demand to programmatically compose and manage communication networks using Software-Defined Networking (SDN) principles. However, in spite of its growing presence and importance, Wi-Fi has received significantly less attention in the SDN community compared to its wired siblings.

The growing importance of Wi-Fi and SDN, and the limited Wi-Fi support in existing SDN solutions, highlights the importance of addressing the roadblocks in bringing the benefits of SDN to networks which use commodity Wi-Fi access points (APs). The key roadblock is that commodity Wi-Fi APs are typically Wi-Fi hubs/bridges that simply forward packets. These include both open-source solutions such as *hostapd* [1] and *OpenWrt* [2], and also proprietary APs used in enterprise or home networks. Furthermore, these APs cannot take intelligent forwarding decisions because they cannot implement the match/action rules used by SDN switches such as Open vSwitch (*OVS*) [3]. As detailed in §II-B, *simply installing an OpenFlow switch such as OVS on an AP is not enough for managing the network traffic flows traversing the AP*.

The seminal work on bringing the SDN to Wi-Fi networks was OpenRoads [4] which used protocols such as the Simple Network Management Protocol (SNMP) for managing the Wi-Fi APs. The insights from OpenRoads were leveraged by several solutions such as ÆtherFlow [5], BeHop [6], and OpenSDWN [7]. However, these solutions cannot be used as-is in existing Wi-Fi networks because they suffer from scalability, hardware, and availability issues (see §II). We therefore focus on *bringing SDN-based flow management to Wi-Fi networks built using over-the-counter commodity APs and controllers.*

In this paper, we present SWIFT, an architecture for bringing SDN-based flow management to existing Wi-Fi networks. Our architecture leverages on commonly available technologies: Client Isolation and SDN switches such as *OVS*. Client Isolation prevents the Wi-Fi clients associated with an AP from communicating with each other. This feature is supported by a wide range of enterprise and consumer APs.[1] We bring SDN functionality to Wi-Fi networks by leveraging on Client Isolation to take control of all flows in the Wi-Fi network. The Intelligent AP technique achieves this by empowering devices running *OpenWrt* with *OVS*. In contrast, the Thin AP technique allows Wi-Fi APs that support Client Isolation to offload the flow management to external SDN switches.

Our key contributions are as follows.

- We enable existing SDN controllers to manage the network traffic flows in Wi-Fi networks built using commodity APs and routers. This significantly lowers the barrier-to-entry to deploy and experiment on software-defined Wi-Fi networks.
- Our Thin AP technique offloads the management of flows traversing APs to external SDN switches. This technique can be used in Wi-Fi networks and testbeds which use APs that cannot run an SDN switch such as *OVS* within the AP. For example, enterprise APs typically do not allow installation of custom firmware such as *OpenWrt*, and custom software such as *OVS*. Similarly, legacy commodity APs may not have the resources for running SDN switches.
- Our Intelligent AP technique integrates *OVS* with *OpenWrt*, and leverages the computational power of modern APs for implementing the forwarding decisions mandated by the SDN controller. This enables SDN controllers to manage the traffic flows at the edge of Wi-Fi networks.

Roadmap. In §II, we discuss related works and our motivation. We then detail our two techniques and their use cases in §III, and present the results of experimental evaluation of our techniques in §IV. We finally conclude in §V.

[1]Client Isolation has many aliases such as Wireless Isolation, AP or Station Isolation, Peer-to-Peer Blocking *etc.* In this paper we use Client Isolation.

ISBN 978-3-903176-08-9

TABLE I
COMPARISON TO THE STATE OF THE ART

Metric	CAPWAP	OpenRoads	ÆtherFlow	BeHop	OpenSDWN	SWIFT
Association control at controller	✓	-	-	✓	✓	✓
Association control at AP	-	✓	✓	-	-	✓
Configure AP using OpenFlow	-	-	✓	✓	-	-
Configure AP using other protocols	✓	✓	-	-	✓	✓
Manage and control commodity APs	-	✓	✓	✓	✓	✔
Manage and control enterprise APs	✓	-	-	-	-	✔
Packet forwarding at the controller	✓	-	-	-	-	-
Packet forwarding at the AP	✓	✓	✓	✓	✓	✓

Existing approaches come with a trade-off of scalability versus the ability to manage the flows between clients associated to the same AP. A ✔ implies that SWIFT allows AP management tools including enterprise AP management tools such as the Cisco Wireless LAN Controller to manage the APs.

II. Background and Motivation

In this section, we first present other proposed approaches for bringing SDN functionality to Wi-Fi networks. We then motivate our work by discussing the roadblocks preventing these approaches from being used in Wi-Fi networks using commodity APs and routers.[2]

A. Existing approaches to integrate Wi-Fi networks with SDNs

In Table I, we compare the existing approaches for bringing SDN functionality to Wi-Fi networks, namely, a) the Control and Provisioning of Wireless Access Points (CAPWAP) protocol [8], [9], b) OpenRoads [4], c) ÆtherFlow [5], d) BeHop [6], and e) OpenSDWN [7].

The CAPWAP protocol is one of the seminal techniques for managing Wi-Fi APs. While CAPWAP predates SDN, they share the same underlying principles: a logically centralized CAPWAP controller manages the APs and takes decisions based on the network state. The specifications of the CAPWAP protocol are fairly detailed, and its proprietary siblings from Cisco [10], Aruba Networks [11] and Ubiquiti Networks [12] manage vendor-specific Wi-Fi hardware. In spite of its limited support in open source solutions such as *OpenWrt*, CAPWAP serves as a cornerstone for *Software-defined Wi-Fi networks.*

OpenRoads, ÆtherFlow, BeHop, and OpenSDWN, primarily differ on the techniques used for managing client associations and client mobility. Client association can be handled either at the AP or at the controller. While handling client associations at the AP is easy to implement, a controller handling associations can be extended to manage hand-overs for implementing seamless connectivity. OpenRoads and ÆtherFlow handle client associations at the AP, while OpenSDWN and BeHop handle associations at the controller.

OpenSDWN and BeHop support client mobility by creating virtual APs (VAP) for the Wi-Fi clients. The Wi-Fi interface of APs typically support multiple networks (BSSID), which are extended as VAPs. OpenSDWN creates a unique VAP for each client, and during client mobility the controller migrates this VAP from one physical AP to another AP. BeHop uses a similar approach, where a single VAP can serve multiple clients. BeHop can also allow clients to locally select the best physical AP. However, OpenSDWN and BeHop do not scale as commodity APs may limit the number of VAPs that can simultaneously run on a physical AP; *e.g.* a Cisco 3700 AP supports 16 networks, *i.e.* 16 VAPs [13]. Furthermore, running multiple VAPs incurs significant performance overheads due to beacon frames [14]. This is a serious shortcoming given the increasing number of Wi-Fi devices at home and at work [15].

Each approach presented in Table I uses a different technique for managing APs. While OpenRoads uses SNMP, ÆtherFlow and BeHop extend OpenFlow to include commands to manage APs. ÆtherFlow uses a modified CPqD [16] OpenFlow switch to change AP configuration while BeHop uses a local agent. Similarly, OpenSDWN uses an agent at the AP that exposes configuration hooks to the controller.

B. Shortcomings of existing approaches

The existing approaches have limitations when dealing with flows between clients associated to the same AP. OpenRoads, ÆtherFlow, BeHop, and OpenSDWN are all built on top of *OpenWrt*, which in turn uses the Linux IEEE mac80211 driver [17]. This driver maintains a list of clients associated with the AP, which is used to directly forward packets between associated clients; packets between clients do not traverse the networking stack of the AP. A consequence of this optimization is that *an SDN switch such as OVS or CPqD running on an AP is unable to manage the flows between Wi-Fi clients associated with the same AP*. OpenSDWN and BeHop address this issue by creating VAPs. However, many APs support only a limited number of VAPs, limiting the number of clients served by a physical AP. Multiple VAPs, *i.e.* SSIDs, also incur heavy overheads in the Wi-Fi due to beacon frames, making them unsuitable for dense Wi-Fi networks or locations with multiple overlapping networks [14]; the current Wi-Fi design principles set the maximum number of SSIDs to only four.

The existing SDN approaches are also not suitable for enterprise APs such as those from Cisco. Typically enterprise APs neither offer support for SDN controllers nor do they support customization by third parties. While these APs can be configured to operate with a proprietary Wi-Fi controller or work autonomously, the limited customization support currently makes them impractical for many SDN research activities or integrating them to SDN-based networks.

Similarly, the existing approaches are also not suitable for legacy APs with limited storage space and limited computational capacity. The match-action rules mandated by SDN are computationally expensive compared to the direct packet forwarding. The legacy APs typically also have limited storage space (in the order of a few MB [18]) which might not be sufficient to add the binaries of SDN switches.

[2]Please note that, in this paper we use the terms Wi-Fi router and Wi-Fi AP interchangeably. A Wi-Fi AP typically acts as a bridge between Wi-Fi and wired networks. In contrast, Wi-Fi routers also include routing capabilities.

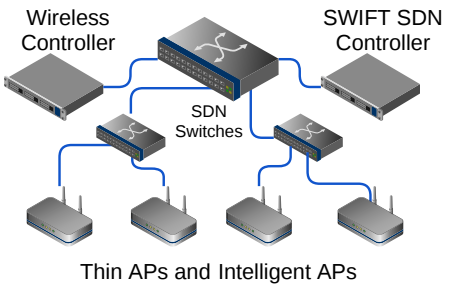


Fig. 1. **Example SWIFT topology**. *The SWIFT SDN controller manages the flows traversing the network, while the Wireless Controller manages the APs.*

C. Motivation

Existing approaches suffer from scalability, hardware, and availability issues for bringing the benefits of SDN-based traffic management to Wi-Fi networks built using commodity APs. This motivates us to find a solution which achieves this requiring only minimal changes to the APs and can be deployed on existing networks.

Programmatically managing a Wi-Fi network involves a) managing and provisioning the APs and b) managing the flows traversing these APs. The AP management and provisioning includes managing the radio interface parameters such as signal strength, channel, *etc.* This can be done either through existing wireless LAN controllers such as Cisco Wireless Lan Controller (WLC) [10], or through other network management systems. These are solid and mature solutions for managing commodity and enterprise APs. At the same time, SDN switches such as *OVS* [3] and SDN controllers[3] such as OpenDaylight [19] and Ryu [20] are solid and mature solutions for flow management. However, the existing approaches cannot combine SDN solutions with existing Wi-Fi networks built using commodity and enterprise APs. In particular, the SDN controllers cannot manage the flows between Wi-Fi clients associated with the same AP, while existing Wi-Fi APs do not support SDN.

In the following, we present our SWIFT architecture for bringing SDN-based flow management to existing Wi-Fi networks. Our solution enables existing SDN controllers to manage all the Wi-Fi traffic flows, including the flows between clients associated with the same AP, while allowing existing Wi-Fi controllers such as WLC to continue to manage the client associations, the radio interfaces, *etc.*, of these APs.

III. SWIFT: Software-defined Wi-Fi Flow Management

In this section, we present SWIFT, our software-defined Wi-Fi flow management architecture. The SWIFT architecture is illustrated in Figure 1, which depicts an enterprise network with a Wi-Fi controller. The Wi-Fi controller is used to manage the existing commodity APs, while the SWIFT controller manages the network traffic flows.

[3]In the rest of the paper we use the term SDN controller to refer to SDN-based network and flow management systems such as OpenDaylight and Ryu.

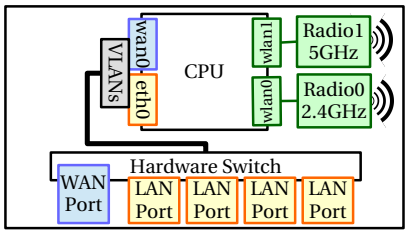


Fig. 2. **A Wi-Fi router that internally uses VLANs**. *Using VLANs to separate WAN and LAN ports reduces costs as only one physical interfaces is needed at the CPU.*

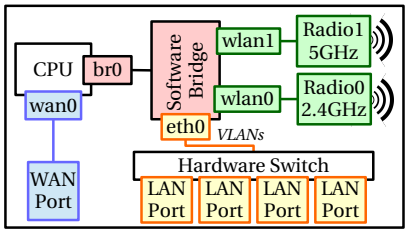


Fig. 3. **Interfaces on an *OpenWrt* router**. *Regardless of the internal wiring, OpenWrt uses a* Software bridge *to manage the Wi-Fi network and the LAN.*

A key building block for SWIFT is Client Isolation, a feature for preventing Wi-Fi clients from communicating with other clients associated to the same AP. When Client Isolation is enabled on an AP, the AP stops bridging the traffic between Wi-Fi clients associated with that AP. In III-A, we discuss Client Isolation and present ways in which it can be used to enable SDN switches to manage the Wi-Fi network traffic flows. We then detail the following two techniques for bringing SDN-based flow management to Wi-Fi networks.

a) Intelligent AP: In this technique (see §III-B), we run *OVS* inside the AP. This enables APs to exert fine-grained control over flows traversing its communication interfaces.

b) Thin AP: In this technique (see §III-C), an AP offloads the flow management to a remote SDN switch and in essence becomes a remote Wi-Fi interface on this switch; the AP and the SDN switch form the Thin AP.

In III-D, we discuss the steps the SWIFT SDN controller needs to take to manage flows traversing APs implementing our techniques, and why simply adding *OVS* to an AP is not enough to bring SDN to Wi-Fi networks.

A. Client Isolation

We now use the internals of *OpenWrt* running on commodity Wi-Fi routers to present an overview of Client Isolation.

In Figure 2, we present the connectivity between the interfaces of widely used commodity Wi-Fi routers. A typical Wi-Fi router has a WAN interface for the Internet connectivity, an Ethernet switch for the wired LAN ports, and at least one Wi-Fi interface. These interfaces are typically exposed to the CPU using Virtual LANs (VLANs). In Figure 2 we present a router (Netgear WNDR4300v1) that internally uses two VLANs, one VLAN for the WAN and another VLAN for the LAN. As shown in Figure 3, *OpenWrt* abstracts these wiring internals and exposes a software bridge that connects

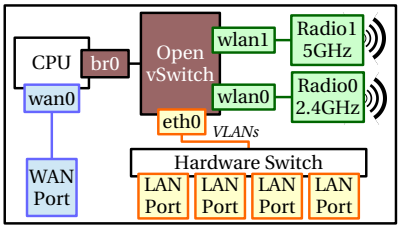


Fig. 4. **AP configured for the Intelligent AP**. *OVS replaces the default bridge provided by OpenWrt, and Client Isolation is enabled on the AP. This allows the OVS to manage the flows traversing the AP.*

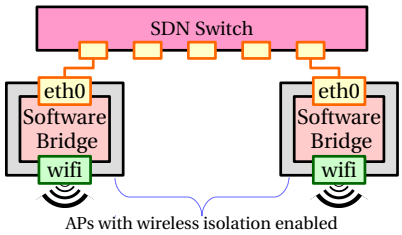


Fig. 5. **APs configured for the Thin AP**. *The APs have Client Isolation enabled, and an external SDN switch manages the network traffic flows traversing the AP.*

the Wi-Fi and wired LAN interfaces. However, because of the internal optimizations in the wireless LAN driver discussed in §II-B, the packets exchanged between the clients associated with the same Wi-Fi interface do not traverse this bridge.

This can be mitigated by enabling Client Isolation. However, we have observed the three following implementations in enterprise and consumer APs [2][10].

a) Permissive Isolation: The driver sends all traffic flows to the AP's network stack. An SDN switch running on the AP can therefore be used to manage the flows traversing the AP.

b) Restrictive Isolation: The driver only allows Address Resolution Protocol (ARP) messages to reach the network. An external SDN switch connected to the AP can use these ARP messages to impersonate the other hosts in the network. This can be achieved by using various techniques such as a) Proxy ARP for Private VLANs (PVLAN), also known as VLAN Aggregation [21][22], or b) the SDN controller sending an ARP reply with the MAC address of the switch in response to ARP requests from clients associated with the AP.

c) Total Isolation: The driver discards all traffic between wireless clients, which makes it impossible to extend Client Isolation to allow SDN switches to manage the traffic flows.

OpenWrt-based APs use *Permissive Isolation*, and enterprise APs may use any of the above implementations; the supporting documentation may provide hints on the implementation used by a given AP. For example, the Cisco 1131ag AP and Cisco WLC can be configured to either use *Restrictive Isolation* or *Total Isolation* [10]. In the following, we present two techniques to combine SDN switches with APs that implement either *Permissive Isolation* or *Restrictive Isolation*.

B. Intelligent AP

In this technique we run *OVS* on an *OpenWrt*-based AP. As shown in Figure 4, we replace the Linux Bridge created by *OpenWrt* (see Figure 3) with *OVS*. All interfaces that were plugged to the bridge are moved to the *OVS*, and Client Isolation is enabled on the Wi-Fi interfaces. Client Isolation forwards all packets arriving on these interfaces to the *OVS*. This allows the *OVS* to manage all flows between Wi-Fi clients and the flows traversing the AP to the wired network.

The Intelligent AP technique also supports multiple Wi-Fi networks (SSID) on the same AP. Each SSID appears as a logical Wi-Fi interface on *OpenWrt*, which is then plugged to the *OVS*. To manage the flows between Wi-Fi clients in different SSIDs, the SDN controller only needs to know the *OVS* port corresponding to a given SSID. The key benefit of this technique is the implementation of SDN match/action rules at the edge of the Wi-Fi network.

C. Thin AP

This technique is suitable for APs falling into one or more of the following categories.

- A custom firmware such as *OpenWrt* cannot be installed on the AP. For example, the flash memory size of the AP is not large enough to install *OpenWrt*.
- A custom software such as *OVS* cannot be installed on the AP. For example, the AP runs proprietary firmware which does not allow customization of the AP.
- The hardware restrictions make it impractical to implement the Intelligent AP approach.

Most enterprise APs fall into one or more of the above categories as they may not allow installation of third-party firmwares or software. Similarly, many legacy APs which support *OpenWrt* but have either limited storage or computational capacity to run *OVS* can use our Thin AP technique.

In this technique, the AP acts as a remote Wi-Fi interface for an SDN switch, and each Wi-Fi network of the AP becomes a port on that switch. This port and the AP connected to it form the Thin AP, which is now responsible for the flows traversing the AP. Furthermore, multiple APs can be connected to a single SDN switch, *i.e.* only a single SDN switch is required to turn a small Wi-Fi network into an SDN-managed one.

As discussed in §III-A, if the Client Isolation on an AP is of the type *Restrictive Isolation* then the SDN controller will be required to provide ARP responses to ARP queries made by clients associated with the AP. However, this impersonation may cause issues with device discovery, for example. We discuss these issues in Section §III-F. In contrast, Intelligent APs do not require the controller to handle ARPs because these APs do not redirect traffic to an external SDN switch.

The key benefit of the Thin AP technique is that it only requires Client Isolation on the AP. This enables the transformation of existing Wi-Fi networks to support SDN.

D. Flow Management using an SDN Controller

We now discuss the steps an SDN controller such as OpenDaylight or Ryu must take to manage flows traversing APs configured as either Intelligent AP or Thin AP.

1) Managing flows traversing Intelligent APs: The SDN controller must perform the following additional tasks to manage the flows traversing Intelligent APs. First, the controller must know which SDN switches are APs configured as Intelligent APs. Second, the controller also has to keep track of the hosts in the network that are associated with these APs, which is essential for forwarding packets between wireless clients associated with the same AP. For such flows, the packets received from the wireless interface of an Intelligent AP must be sent back to the wireless interface. The client's MAC address can be used for this. Additionally, for an Intelligent AP to support encrypted Wi-Fi traffic, packets with *EtherType 0x888e (EAP over LAN)* must be sent to the AP's network stack to be processed by the AP's WPA2 module.

2) Managing flows traversing Thin APs: A Thin AP has a few more requirements from the SDN controller than an Intelligent AP. In addition to Intelligent AP requirements, the SDN controller has to track the IP addresses of all the clients associated with each Thin AP. The SDN controller is expected to examine the ARP queries made by Wi-Fi clients for discovering the other wireless clients associated with the same Thin AP. If the clients are allowed to communicate with each other, the controller responds with an ARP reply with the MAC address of the SDN switch connected to the AP, for which the IP tracking is required. As a consequence of this, a Wi-Fi client α communicating with Wi-Fi client β will send a packet with the source MAC address of client α, source IP address of client α, destination MAC address of the SDN switch, and destination IP address of client β. For such packets the SDN controller commands the SDN switch to replace the source MAC address with the MAC address of the SDN switch, and replace the destination MAC address with the MAC address of client β, and finally send the packet to the wireless interface of the Thin AP. This allows the SDN switch to forward packets between clients associated with the Thin AP. Unlike the Intelligent AP, the Thin AP does not require any specific rules to support Wi-Fi encryption because these frames are managed locally by the AP or by a Wi-Fi controller.

3) Supporting Client Mobility: Most Wi-Fi controllers offer support for client mobility [10]. For the SDN controller to support mobility, the controller must update the flow table rules in SDN switches to ensure that packets are forwarded to the AP which the client is associated to. Both Wi-Fi and the SDN controllers can coordinate this using their respective north-bound APIs or react to network changes.

The above actions are straightforward to implement in any SDN controller, and we have implemented them in our SWIFT SDN controller based on the Ryu controller framework.

E. Use Cases

We now discuss some of the use cases of our work.

1) Bringing SDN-based flow management to enterprise Wi-Fi networks: Current enterprise APs are typically managed using a proprietary Wi-Fi management solutions and do not allow installation of custom firmwares. If these APs support either *Permissive Isolation* or *Restrictive Isolation*, and if SDN switches are added to the network, the Wi-Fi traffic flows can be controlled by SWIFT. With the steps discussed in §III-D, the existing Wi-Fi controllers can now function alongside SWIFT and manage both Wi-Fi APs and traffic flows.

2) Using existing Wi-Fi testbeds for SDN research: Many Wi-Fi testbeds contain a large number of legacy devices. The hardware in these devices might be obsolete by modern standards; for example, APs may have only 4MB of flash storage [18]. However, these devices can run *OpenWrt*, and our Intelligent AP or Thin AP techniques can be used on these devices as *OpenWrt* supports *Permissive Isolation*. This allows conducting SDN research in existing Wi-Fi testbeds.

3) Programmable Network-wide access control: An SDN controller with a fine-grained view of the devices in the network allows the implementation of network-wide access control. The SDN controller can be used to dynamically grant or deny devices access to network services and resources. This level of control has many use cases especially for Wi-Fi networks where many clients can enter and leave at will. First, guest devices can be granted only a limited Internet access while known devices have full access. Second, SDN controllers can coordinate with AP management controllers for creation of location specific access rules. By determining where each client is located, the SDN controller can manage which devices each client has access to. For example, access to streaming devices such as Chromecast or AppleTV can be restricted only to devices near the clients. Third, Wi-Fi networks are increasingly being used to connect Internet-of-Things (IoT) devices such as baby-monitors and security cameras, which are known to have vulnerabilities. With our techniques, these devices can be protected by creating an SDN-based security overlay which controls the set of devices with which such vulnerable devices can communicate.

4) Limit the number of SSIDs: Combined with the network-wide access control, the SWIFT can limit the number of SSIDs used to only a few while retaining the benefits of multiple SSIDs. Different security or group memberships can be allocated to Wi-Fi clients, allowing the dispensation of specific networks such as "guest" or "accounting".

F. Discussion

In this section, we presented our SWIFT architecture and the techniques it is built on. The techniques combine off-the-shelf components, mainly Client Isolation and SDN switches such as *OVS*, to provide SDN-based management of *all traffic* traversing a Wi-Fi network.

The Intelligent AP technique empowers routers and APs which are supported by open-source firmwares such as *OpenWrt*, and allow installation of an SDN switch such as *OVS*. The Thin AP technique can be used with existing Wi-Fi APs which do not allow custom firmwares, or are otherwise incapable of processing the computationally intensive SDN match-action rules. Our two techniques can be used *to bring SDN-based traffic management to existing Wi-Fi networks*.

Our techniques are straightforward to implement in any SDN controller. The Intelligent and Thin AP approaches have

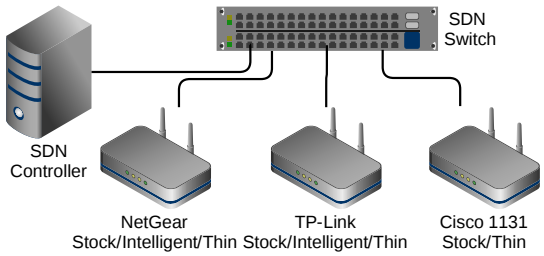


Fig. 6. **Testbed topology**. *The Netgear and TP-Link APs can be configured as Stock, Intelligent, or Thin APs, while the Cisco AP can be configured as Stock or Thin AP. We built our custom SDN controller using the Ryu framework.*

some specific needs that must be addressed. For example, when an AP is configured as a Thin AP, and a client sends out a broadcast packet, the SDN switch to which the AP is connected, sends broadcast packets back to the AP. This may trigger a loop detection algorithm on the AP, causing the AP to momentarily drop all packets sent back to it. This can either be disabled in the AP, or it can be circumvented with application-specific controller modules. For example, DHCP requests can be routed directly to a DHCP server.

Our two techniques can also be combined with the other solutions discussed in Table I. This would be beneficial as we currently lack the remote AP configuration capabilities, offered by for example Cisco WLC. At the same time, our techniques address the scalability issue of multiple networks on a radio interface; this is a serious shortcoming of existing solutions.

IV. Evaluation

In this section we present results of experiments conducted to a) quantify the overheads incurred by APs implementing our two techniques, and b) address scalability. Please note that the results presented here are for qualitative purposes only.

A. Experiment Setup for Quantifying Overheads of SWIFT

1) Devices Used and Network Topology: We used three commodity APs for our experiments: a) Netgear WNDR-4300v1, b) TP-Link WR1043NDv2, and c) Cisco 1131ag. The Netgear and TP-Link can run *OpenWrt* while the Cisco does not support custom firmware; each of these APs have different hardware capabilities. The Netgear and TP-Link can be configured either as a Stock, Intelligent, or Thin AP. We run these two APs in their Stock *OpenWrt* configuration for the baseline measurements. For the Intelligent AP tests, we install *OVS* on these APs, enable Client Isolation, and include a patch for *hostapd* to detect and exchange data with *OVS* to support WPA2 encryption. For the Thin AP tests, the default Linux bridge is used with Client Isolation enabled and the *OVS* is located in a remote host. The Cisco is used in its Stock configuration for baseline measurements, and Client Isolation is enabled only when the AP is configured as a Thin AP. The testbed topology is shown in Figure 6. We use a FIT-PC3 Pro as the main SDN switch for the testbed, and also as the external SDN switch for our Thin APs. In addition, we used two identical laptops as test clients.

2) SWIFT Controller: Our SWIFT controller extends Ryu SDN to support our Thin and Intelligent AP implementations. For each traffic flow, SWIFT installs corresponding flow rules after an SDN switch receives the flow; in our implementation, the forwarding decisions are based on the MAC addresses of the clients. These give a lower bound on overheads, while additional overheads can be incurred depending on the policies that determine the rules. Our main focus was on the overheads incurred by the redirection of packets to an SDN switch.

3) Test Scenarios: We use the following three scenarios. We believe that these three scenarios emulate small Wi-Fi networks and also testbeds used for Wi-Fi experiments.

Scenario A. In this scenario, our two laptops are associated with the same AP. This scenario emulates a small Wi-Fi network such as a home network with a single AP.

Scenario B. In this scenario, the two laptops are associated with different APs. This scenario emulates a Wi-Fi network in a small-office home-office scenario where the clients can be associated with different APs.

Scenario C. In this scenario, one laptop is associated with an AP, while the second laptop is in a high-speed wired network. This scenario emulates a network such as a university network where Wi-Fi clients communicate with servers present in a high-speed wired network. The client in the wired network is connected to the switch using a 1 Gbps Ethernet cable.

4) Traffic Generation: We use the *Flent network benchmarking tool* [23] for quantifying the overheads incurred by our changes to the APs. Our motivation for using Flent was that it includes a *Realtime Response Under Load (RRUL)* test for testing Bufferbloat [24]. During an RRUL test multiple TCP flows between our clients traverse our network at the same time. Furthermore, these competing flows can be configured to have different priorities. For different priorities, the Flent uses the Differentiated Services Code Point (DSCP) field in IP packets; all the packets of a given TCP flow have the same value in the DSCP field. This allows us to emulate the network traffic where clients use different applications such as VoIP and web browsing at the same time. This test is particularly important as *OpenWrt* internally uses the Linux networking stack where the default queue of the networking interfaces is `pfifo_fast`. In the `pfifo_fast` configuration, a queue has three bands labeled 0, 1, and 2, and the DSCP field in the IP header determines the band of a packet; packets in band 0 are served with a higher priority than those in band 1 which in turn have a higher priority than those in band 2 [25]. Each band is served First In First Out, and packets in a band are served only when there are no packets in a lower numbered band; for example, packets in band 2 are served only when there are no packets in band 0 and band 1. We use Flent's RRUL tests in the following three modes.

a) RRUL (default): In this mode, the Flent on each client creates multiple TCP and UDP flows with different priorities.

b) RRUL Best Effort: The Flent running on each client generates multiple TCP and UDP flows. However, in this test all traffic flows have the same priority, i.e., all TCP and UDP packets have the same value in the DSCP field in the IP header.

c) RRUL Ping: The Flent uses only ICMP messages to measure the round-trip time (RTT) without network load. This provides a baseline measurements on the impact of the Intelligent AP and Thin AP approach on the network latency.

We conduct the following experiments to quantify the overheads incurred by of our two techniques. For each of the three scenarios—Scenario A, Scenario B, and Scenario C—we run the Netgear and TP-link AP in the Stock, Intelligent AP, and Thin AP configuration, and the Cisco AP in the Stock and Thin AP configuration. Furthermore, we run 30 iterations of Flent in the default RRUL mode and the RRUL Best Effort mode; during each iteration Flent generated the TCP flows for 180 seconds. We also run one iteration of Flent for 300 seconds in the RRUL Ping mode.

B. Experiment Results on SWIFT Overheads

In Figure 7, we present the results of our experiments conducted to quantify the overheads incurred by our two techniques. In the Stock configuration, the packets between two clients associated with an AP by-pass the kernel network stack of the AP because they are forwarded directly by the radio interface driver. Our Intelligent AP technique causes these packets to be redirected to the *OVS* running on the AP, while our Thin AP technique causes these packets to be redirected to an external SDN switch. We quantify the impact of these redirections using the mean TCP goodput and RTT observed in the different scenarios discussed above.

In Figure 7(a) we present the RTTs observed during the *RRUL Ping* test. The mean TCP goodput and the mean RTT observed during the *RRUL (default)* test are presented in Figure 7(b) and Figure 7(c) respectively; the corresponding observations of *RRUL Best Effort* test are presented in Figure 7(d) and Figure 7(e) respectively. In each figure, `S`, `I`, and `T` denote the Stock, Intelligent AP, and Thin AP configurations respectively; cisco, ng, and tp denote the Cisco, Netgear, and TP-Link AP respectively. The numbers above the X-axis represent the percentage change in the value over the corresponding Stock configuration. For instance, in Scenario B when the Netgear and TP-Link APs are configured as Intelligent APs for the RRUL BE test (ng-tp,I-I), we observe a 1.2% decrease in the mean TCP goodput in Figure 7(d) and a 226% increase in mean RTT in Figure 7(e) compared to the mean goodput and mean RTT observed when the APs were used in their Stock configuration (ng-tp,S-S).

Scenario A. In this scenario, two clients associated with the same AP communicate with each other. The Thin AP configuration is expected to incur a larger increase in RTT compared to the Intelligent AP configuration because the packets have to traverse an external switch. This increase is clearly visible in Figure 7(a), Figure 7(c), and Figure 7(e); the increase in RTTs for the Intelligent AP configuration are smaller than the increase in the RTTs for the Thin AP configuration. Note that the amount of increase in RTT is dependent on the network latency between the AP and the external SDN switch; for instance, the latency between our external switch and our APs was under 1 ms. Furthermore, for the Cisco and Netgear APs, the impact of this redirection on the RTT is small compared to the increased queuing delays faced when the network is loaded; the increase in RTT for Cisco APs in the Thin AP configuration reduces from 55% under no load (see Figure 7(a)) to 0.2% under load from Best-Effort traffic (see Figure 7(e)). In RRUL Best Effort Test, the mean round-trip for the Netgear AP configured as a Thin AP is 5.7% lower while the TCP goodput is 1.4% higher than the corresponding RTT and goodput in the Stock configuration; we make similar observations in the RRUL Best Effort test when the Netgear AP is configured as an Intelligent AP. While the increases are mostly marginal, we believe that these are due to more optimal efficient processing of queues by *OVS*. In contrast, we observe an opposite behavior when using the TP-Link in the Intelligent AP mode. These changes in performance point to the hardware of the APs; the TP-Link has the weakest hardware in terms of CPU capacity, and this is evident from the high RTTs observed in the Stock configuration in Figure 7(a).

Scenario B and Scenario C. In Scenario B, the two laptops are associated with different APs. While in Scenario C, one laptop is connected to an AP and the other is in the wired network. With the help of Figure 6 one can see that for Scenario B the path traversed by the packets in the Stock-Stock configuration and Thin-Thin configuration are identical; similarly, the Stock-Wired configuration and Thin-Wired configuration are identical for Scenario C. The overheads incurred between these configurations are therefore marginal.

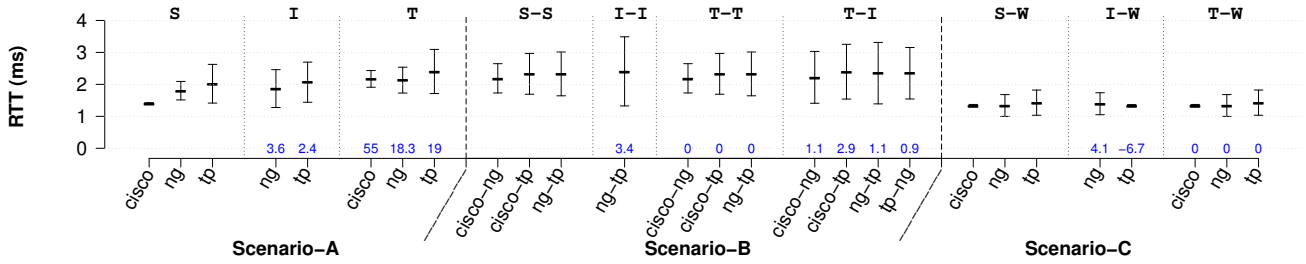
We observe a small difference between the TCP goodput in the Int-Int configuration and the goodput in the Stock-Stock configuration for Scenario B in Figure 7(b) and Figure 7(d). However, in Figure 7(c) and Figure 7(e), we can see that ICMP messages sent by Flent during the RRUL test and RRUL BE test incur a significantly higher RTT. Furthermore, for the Best Effort test, while the TCP goodput decreases by only 1.2% the RTT of the ICMP messages increases by 226%. Clearly, the Thin-Thin configuration performs better than the Int-Int configuration; *this highlights the benefits of offloading the flow management from commodity APs to an external switch*.

For Scenario C, the difference in the TCP goodput and RTT between the Intelligent-Wired configuration and the Stock-Wired configuration is marginal. In Scenario C, we observe a higher RTT compared to Scenario B and Scenario A because of the 1 Gbps wired link; the TCP flows from the faster wired network ensure that the queues at the AP are saturated.

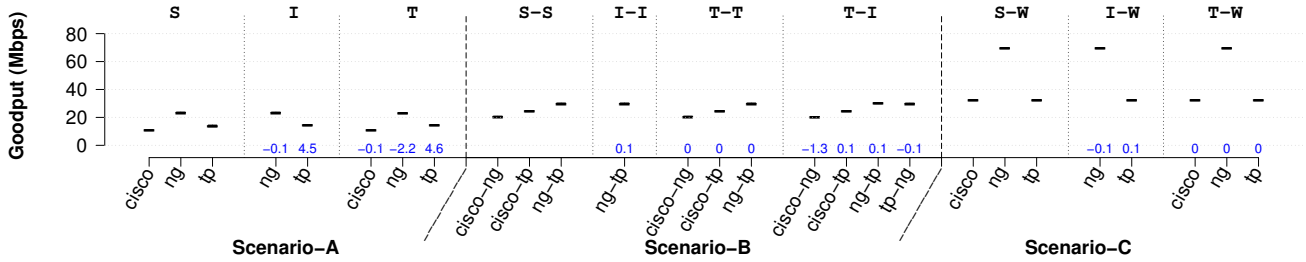
Across all scenarios, we observe that values for the TCP goodput and RTT in the RRUL test are significantly different from those in the RRUL Best Effort test. In the RRUL test, the ICMP messages used for the RTT measurements have the least priority and therefore have higher queuing delays. In the RRUL Best Effort test, all packets have the same priority; we therefore observe smaller RTTs compared to the RRUL test.

C. Experiment for Testing Scalability of SWIFT

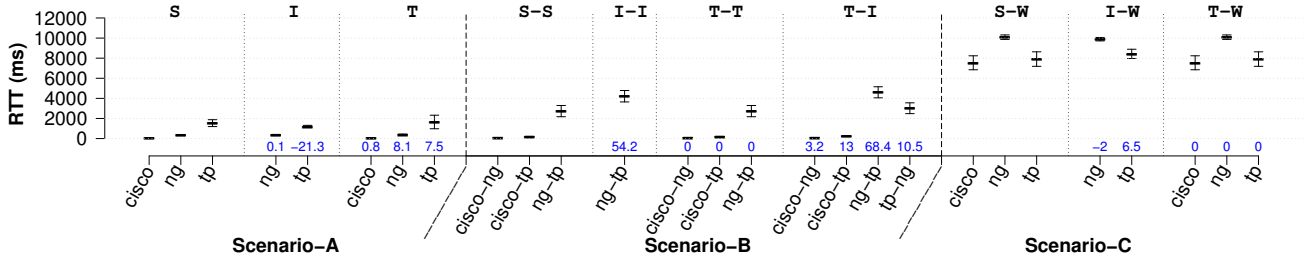
We also performed an experiment to showcase the scalability of our architecture. As discussed in §II, the previous



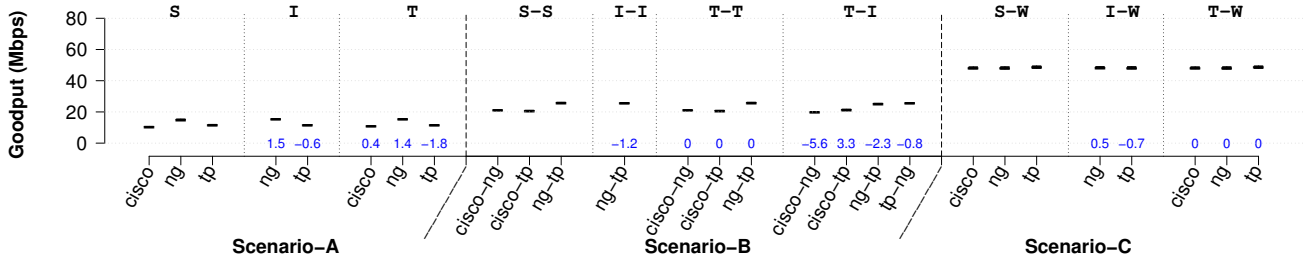
(a) **Mean round-trip times during RRUL Ping test.** *The error bars represent the standard deviation of the observed round-trip times. The numbers above the X-axis present the percentage change in the measured value (mean round-trip time) over the value observed in the corresponding stock configuration.*



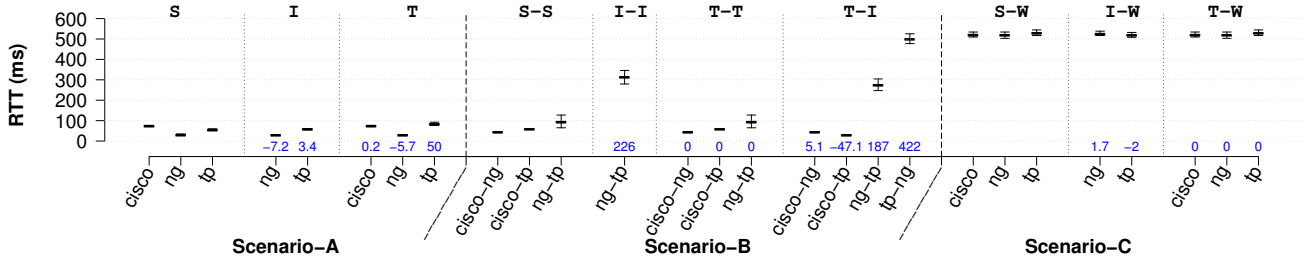
(b) **Mean goodput during the default RRUL test.** *The error bars represent 99% Confidence Intervals for the mean TCP goodput.*



(c) **Mean round-trip time during the default RRUL test.** *The error bars represent 99% Confidence Intervals for the mean round-trip time.*



(d) **Mean goodput during the RRUL BE test.** *The error bars represent 99% Confidence Intervals for the mean TCP goodput.*



(e) **Mean round-trip time during RRUL BE test.** *The error bars represent 99% Confidence Intervals for the mean round-trip time. Note the lower round-trip times compared to the default RRUL test. The ICMP messages have the same priority as the TCP flows in this test.*

Fig. 7. **Experiments Results**. *In each figure,* `S`*,* `I`*, and* `T` *denote the Stock, Intelligent AP, and Thin AP configurations respectively; cisco, ng, and tp denote the Cisco, Netgear, and TP-Link AP respectively. For instance,* `T-I` *with cisco-ng represents a scenario where the first client was associated with a Cisco AP configured as a Thin AP and the second client was associated with a Netgear AP configured as an Intelligent AP. Similarly,* `W` *represents a scenario when one of the clients was in the wired network. For instance,* `I-W` *with ng represents a scenario where one client was associated with the Netgear AP configured as an Intelligent AP and the second client was in the wired network. The numbers above the X-axis represent the percentage change in the value over the corresponding Stock configuration. For instance, in Scenario B when the Netgear and TP-Link APs are configured as Intelligent APs for the RRUL BE test (ng-tp I-I), we observe a 1.2% decrease in the mean TCP goodput in figure (d) and a 226% increase in mean round-trip time in figure (e) compared to the mean goodput and mean round-trip time observed when the APs were used in the Stock configuration (ng-tp S-S). We observe that our two approaches incur negligible overheads in terms of goodput, however their impact on the round-trip time varied with the tests.*

approaches are limited to a single client per VAP if all traffic flows need to be managed; however, the maximum number of VAPs is limited by the hardware and drivers of the APs. We show that our system can scale beyond the maximum number of available SSIDs on the Cisco AP hardware, *i.e.* beyond 16.

In the experiment, we used the same testbed as above with a single SSID. However, we used 20 different devices, including laptops, mobile phones, and a Chromecast. We associated the devices to the Cisco AP operating as Thin AP, and through the SWIFT controller we pushed isolation rules to the AP to isolate several of the devices from other devices.

As expected, the SWIFT controller was able to control all the traffic flows traversing the Thin AP. The isolated devices still retained the Internet access, but could not communicate with other devices, *i.e.* full control of the flows was achieved.

D. Evaluation Summary

The goal of our evaluation was to quantify the impact of our two techniques, and address the scalability of our architecture. The overhead results presented in this section show that our two techniques can be used on commodity APs. Furthermore, the differences in performance between Stock, Intelligent, and Thin configurations are negligible, and the performance largely depends on an AP's hardware capabilities. Specifically, the performance of the Intelligent AP technique depends heavily on an AP's hardware capabilities. In contrast, the Thin AP technique has fewer demands from the AP's hardware than the Intelligent AP, making it more suitable for APs with less powerful hardware. Our results also highlight the benefits of offloading the flow management from commodity APs to an external switch. *This implies that switches procured for SDN research can be used in Wi-Fi testbeds having commodity APs.*

The scalability experiment shows that our architecture can support networks with a large number of clients with only a single SSID, *i.e.* we can limit the overheads caused by a large number of SSIDs, and push beyond the hardware limitations of the APs which limit the other approaches.

Note that our goal was not to evaluate the performance of *OVS*; this has been done by Pfaff *et al.* [3]. We are also not quantifying the overheads of SDN policies because the observed latencies and goodputs can be policy specific and some policies can flood the switch with rules [26].

V. Conclusions

In this paper, we presented SWIFT, an architecture for bringing SDN-based traffic flow management to Wi-Fi networks built using commodity enterprise and consumer devices. SWIFT combines widely available technologies—Client Isolation and SDN switches such as *OVS*—and can therefore be used in almost all existing networks. The techniques used by SWIFT require only a small number of tasks, which are straightforward to implement in any SDN controller. We strongly believe that our techniques provide a deployable way to bringing SDN-based traffic management to Wi-Fi networks, and significantly lower the barrier-to-entry for conducting SDN research on Wi-Fi networks. For instance, our techniques can be used to bring new features such as AP-specific access control to enterprise networks, or conduct research on evaluating the benefits of SDN-based flow management in IoT networks which use Wi-Fi.

The instructions for deploying SWIFT are available at [27].

Acknowledgment

This work is supported by the Nokia Center for Advanced Research (NCAR), the Tekes PraNA project, and Tekes Take-5 project.

References

[1] "hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator," http://w1.fi/hostapd/.
[2] "OpenWrt," https://openwrt.org/.
[3] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *In Proc. of USENIX NSDI*, 2015.
[4] K.-K. Yap, M. Kobayashi, R. Sherwood, T.-Y. Huang, M. Chan, N. Handigol, and N. McKeown, "OpenRoads: Empowering Research in Mobile Networks," *ACM SIGCOMM CCR.*, pp. 125–126, 2010.
[5] M. Yan, C. J. Casey, P. Shome, A. Sprintson, and A. Sutton, "Ætherflow: Principled wireless support in SDN," *CoRR*, vol. abs/1509.04745, 2015.
[6] Y. Yiakoumis, M. Bansal, A. Covington, J. van Reijendam, S. Katti, and N. McKeown, "BeHop: A Testbed for Dense WiFi Networks," in *Proc. of ACM WiNTECH '14*, 2014, pp. 1–8.
[7] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, and A. Feldmann, "OpenSDWN: Programmatic Control over Home and Enterprise WiFi," in *Proceedings of SOSR '15*. New York, NY, USA: ACM, 2015.
[8] P. Calhoun, M. Montemurro, and D. Stanley, "Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification," *Internet RFCs*, vol. RFC 5415.
[9] ——, "Control and Provisioning of Wireless Access Points (CAPWAP) Protocol Binding for IEEE 802.11," *Internet RFCs*, vol. RFC 5416.
[10] "Cisco Wireless LAN Controller," www.cisco.com/c/en/us/products/wireless/wireless-lan-controller/index.html.
[11] "Aruba Networks Mobility Controller," www.arubanetworks.com/products/networking/controllers.
[12] "Ubiquiti Networks UniFi," www.ubnt.com/enterprise.
[13] "Cisco Aironet 3700 Deployment Guide," http://www.cisco.com/c/en/us/td/docs/wireless/technology/apdeploy/8-0/Cisco_Aironet_3700AP.html.
[14] "SSID Overheads," http://www.revolutionwifi.net/revolutionwifi/2013/10/ssid-overhead-how-many-wi-fi-ssids-are.html.
[15] "Total wi-fi device shipments to surpass ten billion this month," http://www.wi-fi.org/news-events/newsroom/total-wi-fi-device-shipments-to-surpass-ten-billion-this-month, January 2015.
[16] "CPqD OpenFlow 1.3 Software Switch," http://cpqd.github.io/ofsoftswitch13/.
[17] "Linux Wireless," wireless.wiki.kernel.org.
[18] "Linksys WRT54G Series," https://en.wikipedia.org/wiki/Linksys_WRT54G_series.
[19] "The OpenDaylight Platform," https://www.opendaylight.org/.
[20] "Ryu SDN Framework," https://osrg.github.io/ryu/.
[21] D. McPherson and B. Dykes, "VLAN Aggregation for Efficient IP Address Allocation," *Internet RFCs*, vol. RFC 3069.
[22] S. HomChaudhuri and M. Foschiano, "Cisco Systems' Private VLANs: Scalable Security in a Multi-Client Environment," *Internet RFCs*, vol. RFC 5517.
[23] T. Høiland-Jørgensen, "Flent: The flexible network tester."
[24] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *ACM Communications*, pp. 57–65, January 2012.
[25] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, and P. Larroy, "Linux advanced routing & traffic control," in *Ottawa Linux Symposium*, 2002, p. 213.
[26] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *IEEE Local Computer Network Conference*, Oct 2010, pp. 408–415.
[27] "SWIFT," https://version.helsinki.fi/swift/.