Realizing Software-Defined Wireless Networks

- Achieving fine-grained Wi-Fi programmability with off-the-shelf hardware

vorgelegt von
Julius Schulz-Zander (Dipl.-Inf.)
egeb. in Kiel

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades
DOKTOR DER INGENIEURWISSENSCHAFTEN (DR.-ING.)

genehmigte Dissertation

Promotionsausschuss:
Vorsitzender: Prof. Dr.-Ing. Slawomir Stanczak
Gutachterin: Prof. Anja Feldmann, Ph. D., TU Berlin
Gutachterin: Konstantina (Dina) Papagiannaki, Ph. D., Telefonica I+D
Gutachter: Prof. Dr.-Ing. Jörg Ott, TU München
Gutachter: Prof. Dr. Stefan Schmid, Universität Aalborg


Berlin 2016
Ich versichere an Eides statt, dass ich diese Dissertation selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Datum Julius Schulz-Zander (Dipl.-Inf.)
Abstract

Fueled by the increasing popularity of wireless enabled mobile end-devices and the advent of the Internet-of-Things networks, the demand for wireless access technology is growing rapidly. Today, many hotels and cafés—and sometimes also entire cities—already offer free WiFi services. Moreover, several mobile operators operate massive WiFi HotSpot, as well as HotSpot 2.0, deployments for traffic offloading, i.e., reducing the stress on their cellular networks. Specifically, WiFi (also known as IEEE 802.11 or WLAN) stands as an attractive inexpensive wireless alternative to cellular access technologies due to its simplistic MAC layer design, reduced costs due to its prevalence in consumer off-the-shelf devices, and operation in the unlicensed spectrum.

Supporting the ever increasing number of WiFi capable devices across residential, public, and enterprise networks is non-trivial. In particular, the (last) wireless hop is often critical for network performance, as it can contribute a non-negligible delay and may constitute a bandwidth bottleneck. Thus, future WiFi architectures are challenged by optimized medium utilization, mobility support, and network management. The latter challenge is integrating wired, cellular, and WiFi networks seamlessly. While today point solutions exist for some of the WiFi-specific network challenges, commodity off-the-shelf hardware is outside the purview of such ossified, expensive, and vertically integrated solutions.

Software-Defined Networking (SDN) is a new paradigm which introduces programmability to overcome network ossification. The key idea of SDN is to decouple the control and data plane to consolidate and outsource the control over a set of network devices to a logically centralized software controller. This allows the control plane to evolve independently of the data plane, enabling faster innovations. 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 to improve performance as well as security. NFV aims to virtualize these network functions, and replace dedicated hardware appliances by software applications running on generic compute resources. The resulting orchestration flexibilities can be used for a faster and cheaper service deployment. SDN can be exploited to steer flows through the appropriate network functions.

Decoupling data plane and control plane operations, à la Software-Defined Networking, can greatly simplify network management and improve the overall performance and utilization of wired networks. However, SDN and NFV have not yet received as much attention in the context of wireless networks, due to fundamental differences between wireless and wired networks. First and foremost, wireless networks feature many peculiarities and knobs that often do not exist in wired networks. For example, wireless networks need interfaces for flexible resource allocation, client mobility, client-based load balancing, and fine-grained traffic engineering is paramount. Furthermore, today’s trend towards Bring-Your-Own-Device (BYOD), implies that the network has to accommodate a more diverse set of user device types of different generations. Moreover, today’s home networks, unlike enterprise networks, typically suffer from a non-existing dedicated control channel, rendering fine-grained centralized control challenging. But, applying the SDN and NFV concepts of softwarization and virtualization to WiFi networks has the potential to render WiFi ready for its future. However, little is known today on how to introduce and benefit from the concepts in WiFi networks.

In this thesis, we present a Software-Defined Wireless Network (SDWN) approach which combines SDN and NFV with wireless access technology. With our SDWN approach we
overcome the aforementioned challenges in WiFi networks. We make the following contributions: (i) we present an SDN framework, called Odin, that introduces our Light Virtual Access Point Abstraction (LVAP), which abstracts the complexities of the upper-802.11 MAC, and a control plane that allows the orchestration of WiFi and wired networks in unison, by leveraging OpenFlow for the Ethernet-based portion of the network. (ii) We present the design, implementation, and evaluation of OpenSDWN, a flexible, novel WiFi architecture based on a joint SDN and NFV approach. Specifically, OpenSDWN introduces WiFi datapath programmability to enable service differentiation and fine-grained transmission control, facilitating the prioritization of critical applications. OpenSDWN implements per-client virtual middleboxes, to render network functions more flexible and support mobility and seamless migration jointly with LVAPs. Moreover, control over the network is exposed through a participatory interface. (iii) We present the design and implementation of a novel WiFi-SDN control plane architecture called AeroFlux. It exploits locality in SDN control plane operations for scalability reasons. Specifically, AeroFlux is based on a 2-tiered control plane that handles frequent, localized events close to where they originate, \textit{i.e.}, close to the data plane, by so-called near-sighted controllers. Global events, which require a broader picture of the network state, are handled by the centralized part of the control plane. (iv) We present LegoFi, a modularized Software-Defined Wireless Network that follows the trend of NFV. \textit{i.e.}, realizing WiFi function blocks as virtualized and programmable wireless network functions (WVNFs), that are allocated (and composed) where and when they are most useful. Specifically, through WVNFs, we achieve a functional decomposition of the WiFi architecture, allowing to overcome inflexibilities found in today’s monolithic, vertically integrated and expensive WiFi architectures.

Thus, SDWN allows us to overcome today’s ossified WiFi architectures by orchestrating and modularizing the WiFi building blocks. Moreover, it provides the necessary abstractions to introduce common features of enterprise networks to residential and hotspot deployments, \textit{i.e.}, WiFi networks based on off-the-shelf commodity hardware. The practicality of our approaches has been successfully demonstrated at several international conferences and is currently deployed and running in two WiFi access networks, \textit{i.e.}, one university enterprise and one larger (30 household) residential network. Moreover, it has gained commercial interest by network vendors and operators. Therefore, we believe that our SDWN approach constitutes a relevant step forward to modern and future-proof WiFi networks.
Zusammenfassung


Um der stetig steigenden Anzahl an WLAN-kompatiblen Geräten in heutigen Netzen und der damit verbundenen steigenden Datenaufkommen gerecht zu werden, haben eine Vielzahl europäischer Provider wie die Deutsche Telekom AG (DTAG)\(^{10}\) und Swisscom AG\(^{16}\) bekannt gegeben, vermehrt WLAN Hot-Spots zur Entlastung ihrer Mobilfunknetze zu integrieren. DTAG hat im Frühjahr 2013 bekannt gegeben, den Ausbau bis 2016 auf knapp über 2,5 Millionen Hot-Spots deutschlandweit voranzutreiben. Eine Vielzahl dieser neuen WLAN Hot-Spots werden beim Kunden durch den Austausch bestehender Geräte realisiert oder durch Kooperationen mit externen HotSpot Anbietern erreicht.


\(^{1}\)https://shar.es/16F778
\(^{2}\)http://www.gartner.com/newsroom/id/2939217


der WLAN Architektur in LegoFi entspricht einem modularen Baukasten zur Realisierung von skalierbaren und zukunftssicheren WLAN Netzwerken.


Parts of this thesis are based on the following peer-reviewed papers that have already been published. All my collaborators are among my co-authors.

**Pre-Published Papers**

**International Conferences and Journals**


Thorsten Fischer, Thomas Hühn, Robin Kuck, Ruben Merz, Julius Schulz-Zander, and Cigdem Sengul. *Experiences with BOWL: Managing an Outdoor WiFi Network (or How to Keep Both Internet Users, Researchers Happy?).* In *25th Large Installation System Administration Conference (USENIX LISA ’11)*, 2011

**Workshops, Extended Abstracts and Demos**

Julius Schulz-Zander, Stefan Schmid, James Kempf, Roberto Riggio, and Anja Feldmann. *LegoFi the WiFi Building Blocks!* Accepted to *ACM MobiArch ’16*, 2016 (to appear)


Contents

1 Introduction .................................................. 17
   1.1 Problem Statement ........................................ 19
   1.2 Contributions ........................................... 20
   1.3 Outline ................................................ 21

2 Background ................................................... 23
   2.1 IEEE 802.11 .............................................. 23
      2.1.1 WiFi Architectures ................................... 23
      2.1.2 Programmability of the IEEE 802.11 data path ......... 26
      2.1.3 Single Channel WiFi Architecture ...................... 27
   2.2 Software-Defined Networking ............................. 28
      2.2.1 Terminology .......................................... 29
      2.2.2 The OpenFlow SDN Interface ........................... 30
   2.3 Network-Function Virtualization ........................ 31
   2.4 The Berlin Open Wireless Lab (BOWL) Testbed .......... 31
      2.4.1 Outdoor Testbed ....................................... 31
      2.4.2 Indoor and Smoketest Testbeds ......................... 32
      2.4.3 Operating System and Node Management ................ 33
      2.4.4 OpenFlow support for OpenWrt ......................... 34

3 Odin: Virtualizing Wireless Access Points ................. 35
   3.1 Motivation ............................................... 35
   3.2 Overview ............................................... 36
   3.3 Use Cases ............................................... 37
   3.4 The Odin System ......................................... 37
      3.4.1 System Components .................................... 37
      3.4.2 Light Virtual Access Points .......................... 38
      3.4.3 Reactive and Proactive Odin Applications ............. 42
   3.5 Odin on Commodity Hardware ............................ 42
      3.5.1 Controller ............................................ 42
      3.5.2 Agent ............................................... 43
      3.5.3 ACK Generation ....................................... 43
      3.5.4 LVAP Assignment ..................................... 43
   3.6 SDN Applications on top of Odin ......................... 44
      3.6.1 Mobility Manager ...................................... 44
      3.6.2 Client-based Load Balancing .......................... 45
      3.6.3 Wireless Troubleshooting .............................. 47
      3.6.4 Automatic Channel Selection .......................... 47
      3.6.5 Energy Efficient WiFi Networks ....................... 48
      3.6.6 Guest Policy Enforcement ............................. 49
   3.7 System Evaluation ........................................ 50
      3.7.1 Controller Load due to Publish-Subscribe Subsystem ... 50
      3.7.2 LVAP Handoff Micro-Benchmark ......................... 50
      3.7.3 Probe Request Serving Latency ......................... 51
      3.7.4 Compatibility with Clients ........................... 52
   3.8 Related Work ............................................ 52

13
3.9 Discussion ........................................ 53
3.10 Summary ........................................ 55

4 OpenSDWN: Programmability of the lower MAC 57
  4.1 Motivation ....................................... 57
  4.2 Contribution .................................... 58
  4.3 Use Cases and Overview .......................... 59
    4.3.1 Use Cases .................................. 59
    4.3.2 Overview .................................. 60
  4.4 The OpenSDWN System ........................... 61
    4.4.1 Wireless SDN ................................ 61
    4.4.2 Virtual Middleboxes .......................... 61
    4.4.3 Participatory Interface ....................... 62
  4.5 Evaluation ...................................... 62
    4.5.1 Deployments and Methodology .................. 63
    4.5.2 User-Defined Service Differentiation .......... 63
    4.5.3 User Mobility ................................ 66
    4.5.4 Smart Direct Multicast Service ............... 68
  4.6 Prototype Implementation ....................... 69
    4.6.1 Interfaces ................................ 69
    4.6.2 Control Plane ................................ 72
  4.7 Related Work ................................... 75
  4.8 Summary ....................................... 77

5 AeroFlux: A scalable Near-Sighted Control Plane 79
  5.1 Motivation ....................................... 79
  5.2 Overview ....................................... 80
  5.3 Benefits and Use Cases .......................... 80
  5.4 Architecture .................................... 81
    5.4.1 AeroFlux Applications ......................... 82
    5.4.2 Global Controller (GC) ....................... 83
    5.4.3 Near-Sighted Controller (NSC) ............... 83
    5.4.4 Radio Agent (RA) ............................ 83
    5.4.5 Light Virtual Access Point (LVAP) .......... 84
    5.4.6 WiFi Datapath Transmission (WDTX) Rule .... 84
  5.5 Prototype ....................................... 85
    5.5.1 Global Controller ................................ 85
    5.5.2 Near-Sighted Controller ....................... 85
    5.5.3 Radio Agent ................................ 85
    5.5.4 WDTX rules ................................ 85
  5.6 Evaluation ....................................... 86
  5.7 Discussion ...................................... 88
    5.7.1 Challenges and Extensibility .................. 88
    5.7.2 Deployment ................................ 88
  5.8 Related Work ................................... 88
  5.9 Summary ....................................... 89

6 LegoFi the WiFi Building Blocks! 91
  6.1 Motivation ....................................... 91
  6.2 Overview ....................................... 93
Nowadays almost all portable end-devices are WiFi enabled. The WiFi Alliance [18], the certification authority for WiFi devices, reports that 10 billion WiFi devices were shipped as of January 2015 and predicts an annual growth of 10 percent for the next five years\(^1\). Moreover, Gartner predicts that by 2018, more than 50% of the users will use wireless enabled mobile end-devices such as a tablet or smartphone for all their online activities\(^2\). Gartner also predicts, that roughly 40% of the non-mobile devices such as Desk-Phones and Projectors will be WiFi enabled. In the near future, with the advent of the Internet-of-Things networks [48,140], wireless access technology is likely to extend to even more objects. Thus, WiFi traffic is likely to grow over the next years. Fueled by the increasing popularity the demand for new wireless access technology becomes more and more urgent.

IEEE 802.11 (also known as WiFi or WLAN) is an attractive wireless access technology due to its operation in the unlicensed spectrum. It offers an inexpensive alternative to cellular access technologies due to its simplistic MAC layer design and its prevalence in consumer off-the-shelf devices. Thus, more and more mobile operators plan massive WiFi HotSpot, as well as HotSpot 2.0, deployments for traffic offloading from the cellular networks, \textit{i.e.}, to reduce the stress on their mobile networks. To highlight this need, we point to the fact that large operators such as Deutsche Telekom AG (DTAG) [10] and Swisscom [16] are offloading data from their cellular networks to WiFi networks. Indeed, DTAG aims to deploy 2.5 million WiFi hotspots by 2016. Moreover, nowadays many airports and cafés—and sometimes also entire cities—offer free WiFi services.

Supporting the ever increasing number of WiFi capable devices across residential, public, and enterprise networks is non-trivial. Specifically, the trend towards Bring-Your-Own-Device (BYOD) implies that operators have to accommodate an even more diverse set of user device types of different generations. Thus, network operators are challenged by optimizing network management and radio resource management (RRM). The former is to integrate wired and wireless networks seamlessly. The latter includes to support seamless mobility, efficient medium utilization, and dynamic bandwidth allocation. In particular, the wireless access is often critical for network performance, as it can contribute a non-negligible delay and jitter especially for high definition media [67,122] and may constitute a bandwidth bottleneck. However, today, 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. While today point solutions exist for some of the WiFi-specific network challenges, commodity off-the-shelf hardware is outside the purview of such ossified, expensive, and vertically integrated solutions.

\(^1\)https://shar.es/16F781
\(^2\)http://www.gartner.com/newsroom/id/2939217
Chapter 1 Introduction

The situation is worsened by the fact that non-enterprise 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 infrastructures; 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. Supporting stronger and more easily managed security between Hotspots is one example while simple and fast deployment of innovative new network services like seamless mobility, i.e., depriving users from essential services, is another.

Recent advances outside the wireless domain have simplified network management and increased network performance. The best example is Software-Defined Networking (SDN), which is an emerging new paradigm which allows to overcome 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.

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 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 [30, 54, 59, 106]. 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.

The SDN paradigm has recently received significant interest in wired networks through the OpenFlow protocol [91], today’s standard SDN interface for ethernet switches. Specifically, it has been discussed and successfully applied in many use cases, ranging from fine-grained traffic engineering [55], enforcing complex network policies [54, 66], improve resource utilization in wide-area networks [73, 78], or enable network virtualization in datacenters [53], etc.

In general, SDN and NFV have not yet received as much attention in the context of wireless networks. Indeed, wireless networks feature many peculiarities and knobs that often do not exist in wired networks. Specifically, due to its feature-driven and complex PHY- and MAC-layer design WiFi inherits more state management, e.g., authentication and association state handling, power-save management, and rate control stats keeping. For instance, in WiFi, every AP needs to keep track of a client’s state machine. Moreover, wireless networks use a shared medium whose characteristics change quickly over time and in an unpredictable manner, as users are often mobile and associations dynamic. Communication over a shared medium implies several link impairments, i.e., high packet loss due to collisions and hidden terminals or lost transmission opportunities due to exposed nodes. Specifically, the distributed coordinated function (DCF) of the IEEE 802.11 standard suffers from a significantly high collision probability [38] when the number of competing WiFi devices increases. Thus, the performance of WiFi networks in very dense environment such as stadiums or lecture halls can be substantially degraded due to the high collision probability in such environments.
1.1 Problem Statement

Moreover, WiFi links can be operated in a number of different regimes with different channel characteristics, e.g., dense, residential, and rural networks. Specifically, due to the non-stationary characteristic of the wireless channel, permanently adjusting settings such as transmission rate and power is crucial for the performance of WiFi networks and brings significant benefits in the service quality, e.g., through reducing the packet loss probability. Thus, permanently adjusting transmission properties is paramount.

To this end, today’s WiFi hardware offers several unique knobs to influence the probability of successful transmissions, such as changing transmission rate and power, as well as retry chains. Today’s rate and power control, however, is mainly done on the WiFi device itself. But it is rarely optimized to the application-layer demands and their diverse traffic requirements, e.g., their individual sensitivity to packet loss or jitter. This introduces opportunities for a fine-grained and application specific transmission control, e.g., for service differentiation.

In summary, WiFi introduces more complexity for the network management compared to wired networks, due to its complexity at the PHY and MAC layer. However, none of these knobs are addressed by today’s wired SDN.

1.1 Problem Statement

This thesis is driven by the challenges today’s and future WiFi deployments based on off-the-shelf commodity hardware face.

- **Integrating and interoperating heterogeneous networks:** First and foremost, today’s network operators face the challenge to support the ever increasing number of WiFi capable devices across residential, public, and enterprise networks. This raises the question how to manage all the different networks in unison all the way to the users’ premises. In particular, the challenge is to integrate wired, cellular, and wireless network management seamlessly. However, this is non-trivial, today’s WiFi networks based on off-the-shelf commodity hardware do not provide the means allowing network operators a unified network management.

- **Unplanned and uncoordinated deployment:** Today’s non-enterprise networks often suffer from poor performance as a consequence of an unplanned and uncoordinated deployment: Different parties in a house or neighborhood typically deploy and run their own dedicated infrastructure. Furthermore, 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. Thus, bringing typical enterprise features such as channel selection, mobility management, and client-based load balancing to home networks can dramatically increase user experience. However, today’s off-the-shelf commodity WiFi APs typically do not provide infrastructure controlled handovers, thus renders optimized network operation difficult.

- **Heterogeneous devices:** With the trend of Bring-Your-Own-Device (BOYD), today’s and future networks have to accommodate an even more diverse set of user device types of different generations. This leaves the challenge of providing Network Access Control (NAC) and abstracting from the inner workings of the IEEE 802.11
MAC, i.e., hiding the complexities of the 802.11 protocol stack from the network operator. Thus, a proper abstraction of the WiFi ideally integrates with OpenFlow, e.g., providing NAC control.

- **Limited resource management:** Today’s home networks lack the possibility to differentiate traffic in an application aware manner, i.e., applications requirements are not considered at the last hop. While such a differentiation may not be possible in the Internet due to network neutrality policies, it is not only legal but also often desirable to differentiate between applications on the last hop in private home networks. Specifically, due to network neutrality reasons, the Internet downlink usually offers no specific traffic classification. Indeed, WiFi networks offer several unique knobs to influence the probability of successful transmissions, such as transmission rate and power, as well as retry chains. To overcome this limitation, existing solutions can be used for traffic identification. This introduces opportunities for a fine-grained and application specific transmission control, e.g., for service differentiation. However, this requires programmability of the WiFi datapath and integration with the wired counterpart.

- **No possibility for participation:** Today, there is neither an interface letting users defining traffic priorities, nor can applications specify and report their requirements to the WiFi AP. As a result, today’s users are deprived from controlling application awareness in their home networks.

- **Monolithic and inflexible control:** Today’s ossified, expensive, and vertically integrated enterprise solutions often only tackle a limited set of WiFi-specific network challenges. For instance, the performance of WiFi networks in very dense environment such as stadiums or lecture halls can be substantially degraded due to the high collision probability in such environments. Accordingly, a de-duplication filter for WiFi frames at a more centralized location within the network can increase the transmission probability and thus improve the overall networks performance. However, while these point solutions focus on one specific use case, there is limited room for flexibility in terms of network deployment and architecture. Moreover, commodity off-the-shelf WiFi hardware is typically outside the purview of such solutions. Thus, allowing a flexible placement of WiFi network functions within a network can be exploited for a performance improved, faster, and cheaper service deployment.

This thesis is motivated by the observation that the flexibilities enabled by SDN and NFV can be an attractive means to overcome the limitations of today’s WiFi networks. However, SDN and NFV has been studied most intensively for wired networks so far. Thus, little is known today on how bring the benefits of SDN and NFV to WiFi networks. This is the key research challenge addressed in this thesis.

### 1.2 Contributions

In this thesis, we show that there can be a major benefit of introducing programmability and virtualization in wireless networks, i.e., following an Software-Defined Wireless Networking approach. In a nutshell, in this thesis we make the following contributions:

- **Virtualizing WiFi Access Points:** We provide an SDN framework, called Odin, that abstracts the upper-802.11 MAC and allows the orchestration of WiFi networks.
Thus, allowing today's network operators to manage WiFi access networks in uni-
sion with their wired counterparts. With Odin, we make the following contributions: 
(i) Light Virtual Access Points (LVAPs), a novel programming abstraction for address-
ing the IEEE 802.11 protocol stack complexity, (ii) a design and implementation for 
a software-defined WiFi network architecture based on LVAPs, and (iii) a prototype 
implementation on top of commodity access point hardware without modifications to 
the IEEE 802.11 client, making it practical for today's deployments.

• WiFi datapath programmability and NFV management: We present the de-
  sign, implementation, and evaluation of OpenSDWN, a flexible, novel WiFi architec-
ture based on a joint SDN and NFV approach. Specifically, with OpenSDWN, we reap 
the benefits of SDN and NFV for home and enterprise WiFi networks. OpenSDWN 
exploits datapath programmability to enable service differentiation and fine-grained 
transmission control, facilitating the prioritization of critical applications. Specifically, 
it provides an abstraction of the actual PHY layer transmission properties such as 
physical data rates and flow control. To this end, 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 
can also be used to out-source the control over the home network to a participatory 
interface or to an Internet Service Provider.

• Hierarchical control plane: With AeroFlux, we present the design and implemen-
tation of a novel WiFi-SDN approach that exploits locality in SDN control plane 
operations for scalability reasons. Specifically, it tackles the risk of overloading the 
control plane, or of adding too much latency, when maintaining statistics or events 
globally, when there is limited benefit in maintaining these statistics globally.

• Functional decomposition of the WiFi architecture: We present LegoFi, a mod-
  ularized Software-Defined Wireless Network that follows the trend of NFV, i.e., realiz-
ing function blocks as virtual network functions (VNFs). Specifically, with LegoFi, we 
present a functional decomposition of the WiFi architecture. With LegoFi, individual 
(virtualized and programmable) WiFi function blocks are allocated where (and when) 
they are most useful. Thus functions that require low latency such as transmission 
rate control can be allocated on the Access Point, while functions that can benefit 
from having broader visibility across the local wireless network (e.g., association man-
agement, duplicate frame filtering and transmit power control) can be allocated onto 
a virtualized server platform or programmable network equipment further back in the 
network. LegoFi was designed to cope with the scalability and flexibility aspects of 
designing and operating future WiFi networks.

• Cross-Layer control: Finally, to the best of our knowledge, our approach offers the 
first solution to cross-layer programmability in WiFi networks.

1.3 Outline

The remainder of this document is structured in the following way:

Chapter 2 provides background information relevant for the remainder of this thesis: It 
gives basic introduction to the concepts of SDN, NFV and foundations of WiFi. In particular,
Chapter 1 Introduction

we explain the basics of the concepts and give the necessary details of the IEEE 802.11 Standard to follow the remainder of this thesis.

Chapter 3 presents Odin, a framework that provides programmability of the upper MAC 802.11 functionality through a novel per-client virtual AP abstraction.

Chapter 4 describes how to our SDWN framework is designed in accordance of the vision of the so called SDNv2 concept. Specifically, OpenSDWN provides programmability of the WiFi datapath and per-client virtual middleboxes, to render network functions more flexible and support mobility and seamless migration.

Chapter 5 is mainly concerned with the scalability aspects of an SDWN towards enterprise and ISP networks. Moreover, it presents an approach that exploits locality in SDN control plane operations for scalability reasons, that tackles the risk of overloading the global logically centralised control plane.

Chapter 6 presents a functional decomposition of the WiFi architecture, i.e., a modularized SDWN that follows the trend of NFV where function blocks are allocated where (and when) they are most useful.

Chapter 7 provides an outlook and possible directions for future work.
Background

We next present the necessary context of our work: (i) We give an overview of today’s WiFi architectures, the programmability aspects of WiFi, and the Single Channel Architecture where all WiFi APs operate on the same channel. (ii) We discuss recent trends in Networking including Software-Defined Networking and Network Function Virtualization. We review OpenFlow, the de facto standard interface for SDN, which defines a programming interface for forwarding devices such as switches for remote control and monitoring of their data plane. (iii) Finally, we describe the BOWL network with its testbeds that we use to evaluate our systems.

2.1 IEEE 802.11

In 2015 the IEEE 802.11 standard [2] (also known as WiFi or WLAN) celebrated its 25th anniversary. The IEEE 802.11 standards enable a wide range of wireless networking applications, e.g., wireless Internet access from offices, homes, cafés, and airports, but also car to car communication. Over the years, several amendments were proposed and released in order to cope with the increasing demands on performance, security, and radio resource management etc. Today WiFi offers speeds of several “Gigabits” (up to 6.77 Gbit/s) with its latest 802.11ac standard. In this section, we next present the necessary background on WiFi architectures, programmability of WiFi, and single channel architectures.

2.1.1 WiFi Architectures

Today’s enterprise WiFi architectures can roughly be divided into three main planes:

1. **Management Plane:** The management plane functions are in charge of centralized, non-real-time administrative tasks, such as provisioning APs, AP firmware update, fault surveillance, and security monitoring.

2. **Control Plane:** The control plane consists of functions necessary to be able to send and receive 802.11 frames: (i) set up and maintain the wireless link, (ii) control the transfer of data packets between the device and the AP, and (iii) share the wireless link with other devices (through the distribution system) [20].

3. **Data Plane (or data path):** The data plane consists of the devices which implement the WiFi data path. Specifically, the data path provides the necessary functions to actually transmit and receive IEEE 802.11 frames. The data plane carries IEEE 802.11
Chapter 2 Background

encapsulated frames though sometimes encapsulated in a tunnel, e.g., if a centralized controller is in place.

As of today, network operators can choose from a wide range of different WiFi architectures including:

**Centralized**: Centralized architectures offload the Management and/or Control Plane to a centralized controller.

RFC4118 [144], the IETF CAPWAP standard, specifies a reference architecture for centralized control of WiFi networks. It defines two components: a centralized Access Controller (AC) and Wireless Termination Points (WTPs). CAPWAP defines three architectural variants: Local (or Full) MAC, Split MAC and Remote MAC. RFC 4118 divides the functions into those that are specified by IEEE 802.11 and those that are network management functions (called “CAPWAP functions” in the RFC).

- **Local (or Full) MAC**: This variant places the entire 802.11 MAC on the access points and only places the network management and provisioning on the AC. Here, the WTPs have the same functional composition as an 802.11 AP as defined by the 802.11 standard.

- **Remote MAC**: This variant moves all the MAC functions (including the real-time critical) to the AC. This approach has seen little deployment.

- **SplitMAC**: The Split MAC variant allows the decoupling of the non-realtime functions from the WTP. For instance, the handling of authentication or association can be moved to the AC. All MAC functions with realtime constraints remain on the WTP. I.e., WTPs which only handle realtime functions are called *Thin APs*.

The primary motivation of centralized architectures is to solve issues such as network control and security. However, centralized architectures usually constitute a bottleneck and thus can suffer from performance issues. Most enterprise WiFi solutions [4, 6, 12, 15] are built along the lines of the Split MAC architecture where the AP management is performed by a centralized controller. The controller is either realized as a hardware appliance or “virtualized” software solution. These solutions offer a wide range of features such as seamless mobility, interference management, and intrusion detection and prevention system (IDS and IPS). However, these solutions are usually expensive, closed source, and vertically integrated.

**Cloud-based**: The controller resides on a cloud infrastructure while most of the Control Plane is handled on the APs.

Most Cloud-based solutions [7, 8, 13, 17] offer AP management and monitoring as well as accounting as a “Service” product where the controller resides on a cloud infrastructures such as Amazon EC2. Thus, most Cloud-based WiFi vendors rely on the Full MAC variant. These solutions mainly target network providers which do not want to run their own dedicated server/controller infrastructure. The motivation is to enable them to easily manage their WiFi APs. In addition, it provides tools for collecting data and performing analysis. All network management functions are
deployed in a public cloud-based network management system. Most of these solutions either provide a specific firmware for off-the-shelf devices or sell their own APs.

Aruba [4] and Meraki [11] offer such services for enterprise WiFi deployments. These vendors offer this “as a Service” product, where the vendor provides the management functions and the customer follows a plug and play approach with the APs. Specifically, these solutions come with their own custom hardware to offer typical enterprise features such as interference management, fine grained access control, and seamless mobility when deployed in a LAN. Off-the-shelf hardware is outside the purview of these solutions.

**Controller-Less:** This can be seen as a special case of cloud-based architectures, where the control plane is distributed between the APs.

AeroHive [3] promotes the controller-less architecture where the WiFi management plane is outsourced in the cloud and all intelligence is distributed “to the edge”. Specifically, their controller-less mesh-enabled AP products implement all features such as client-based load balancing, seamless mobility, and interference management in a distributed fashion. This architecture tries to overcome bottleneck issues that arise with “Gigabit” WiFi in centralized architectures where all WiFi traffic is handled by the controller.

**Distributed:** Similar to the centralized approach, but traffic forwarding and slicing is done at the AP and not at the centralized controller.

Several traditional enterprise WiFi solutions [4,6,12,15] which are built along the lines of the Split MAC architecture also offer a distributed forwarding architecture. The distributed forwarding architecture tries to overcome the issue of the potential bandwidth bottleneck at the controller in centralized architectures. These architectures perform the slicing directly at the AP and relies managing the wired and wireless portion of the network in unison. i.e., this architecture often relies on tunneling and slicing techniques at the MAC or IP layer (e.g., VLANs, MPLS, or GRE) to achieve a distributed forwarding architecture. However, in a simplified case just the authentication and association is performed by the controller.

**ISP Hotspots and Home APs:** Similar to the cloud-based approach, but the management plane is realized by a dedicated Auto Configuration Server managed by the network operator.

Remote management of different Internet access devices such as of ISP Hotspots, modems, routers, gateways, and Home APs (also known as customer-premises equipment (CPE)). ISP Hotspot networks on the other hand follow the Full MAC architecture, with management provided by a remote network management system through the CPE WAN Management Protocol (CWMP or TR-69) [21], a Broadband Forum HTTP-based protocol standard for remote AP management allowing pushes of static configurations and firmware updates. The standard defines an application layer protocol for communication between customer-premises equipment (CPE) and Auto Configuration Servers (ACS) solely for auto configuration and remote management.
These architectures do not support any enterprise level features such as interference management, seamless mobility, and client-based load balancing.

2.1.2 Programmability of the IEEE 802.11 data path

The 802.11 datapath can be divided into two parts: (i) the upper-level MAC is responsible for packet state handling which are not time critical. For example, the association and authentication state machine is handled in the upper-level MAC. (ii) The lower-level MAC which interfaces directly to the PHY transmission path and handles all wireless transmissions and receptions. In particular, the lower-level MAC performs all time critical aspects of the IEEE 802.11 MAC such as sending acknowledgements or performing flow control.

Programmability of the upper-level MAC: Since users are often mobile and associations dynamic, every AP needs to keep track of its clients through a per-client state machine. This state machine is controlled through management frames exchanged between the AP and client station. Management frames have less stringent requirements on the timings and thus, are handled in the upper-layer MAC. Typical IEEE 802.11 management frames are association, authentication and probe frames which are either used to form an association with an AP or to exchange capabilities. Management frame are sent at a basic transmission rate, and usually with a higher medium access probability. Because of the non-time critical requirements these frames can be handled by a management software running on the AP or forwarded to and handled by a centralized controller.

Programmability of the lower-level MAC: Moreover, WiFi links can be operated in a number of different regimes with different channel characteristics, e.g., dense, residential, and rural networks. Thus, due to the non-stationary characteristic of the wireless channel, permanently adjusting PHY layer settings such as transmission rate and power control is crucial for the performance of WiFi networks and brings significant benefits in the service quality, e.g., through reducing the packet loss probability. However, transmission rate control requires a tight control loop to adjust transmission settings in a timely manner due to the permanently changing characteristic of the wireless medium. For instance, the coherence time\(^1\) (also a function of the client mobility) can easily exceed the expected time of the successful transmission of multiple data frames [82], rendering optimized control difficult.

Minstrel, the de facto standard transmission rate control algorithm in the Linux kernel, relies on a mechanism called retry chain in modern WiFi cards. This mechanism allows to specify a sequence of different physical transmission setting for a single transmission of a frame. Modern WiFi cards allow chains with up to four triples defining the rate, power, and retry count for a single transmission of a frame or an aggregate. Specifically, frame aggregation is a feature that increases the efficiency of the medium utilization by aggregating two or more data frames in a single transmission. More specifically, frame aggregation significantly reduces the overhead of radio level headers (preamble), MAC frame fields, inter-frame spacing, and acknowledgements to to achieve a more efficient medium utilization. For example, the retry chain allows to specify that the first four transmission attempts of a frame should be with 54 Mbit/s, then three times with 24 Mbit/s at a higher transmission rate.

---

\(^1\)The time over which the radio channel is presumed to be steady.
transmission power. As a reference, the Minstrel rate algorithm adjusts the transmission rate on a wireless link based on transmission success probability statistics every 100 ms. Minstrel in its current form is a pure transmission rate control algorithm.

Communication over a shared medium implies several link impairments, i.e., high packet loss due to collisions and hidden terminals, lost transmission opportunities due to exposed nodes and the co-existence of devices with different capabilities. Thus, the IEEE 802.11 standard specifies a mechanism called Request to Send/Clear to Send (RTS/CTS). It allows to reserve the medium for a certain amount of time, i.e., this mechanism relies on the Network Allocation Vecor (NAV) which is akin to virtual carrier sensing.

Besides rate control, Transmit Power Control (TPC) is critical to improve spectral efficiency and reduce interference. Accordingly, TPC is a key feature of the radio resource management of modern enterprise WiFi solutions [4, 5]. Home networks environments based on off-the-shelf WiFi hardware, however, suffer from (co-)channel interference and the lack of TPC. Recent approaches [58, 75, 76] try to overcome this issue through a distributed approach. However, centralized approaches as found in enterprise solutions usually benefit from more advanced monitoring mechanisms based on spectral samples. Besides TPC, enterprise solutions also support Dynamic TPC (DTPC) [14] to reduce the transmit power on a per-link basis. Specifically, DTPC will dynamically adjust the per-link transmission power as a client moves closer or further away from an AP. Minstrel-Blues [76] is such an approach targeting off-the-shelf access points by improving the Minstrel rate control to perform a joint power and rate optimization on a per-link basis.

In this thesis, we present an approach (cf. Section 4) to remotely control the transmission settings on a per-packet, per-flow, per-station, and per-group basis. We also discuss TPC and DTPC in more detail in (cf. Chapter 3 and Chapter 4).

2.1.3 Single Channel WiFi Architecture

In the Single Channel WiFi Architecture (SCA) all physical APs operate on a single radio channel (frequency) which creates a continuous region of overlapping WiFi coverage. The benefit of this architecture is that it allows to realize seamless handover mechanisms which do not require any client side modifications. In other words, it eliminates the need for a handover mechanism involving the client. Moreover, clients can be assigned to the nearest AP with enough available capacity within the communication range.

We next go into detail why the IEEE 802.11ac standard is beneficial for realizing an SCA for home and hotspot networks. It introduces a couple of mechanisms to increase the performance of WiFi. First and foremost, it specifies an increased channel bandwidth (up to 160 MHz) for the operation in the 5 GHz band. Specifically, the 40 MHz channel bandwidth of 802.11n is extended to 80 and 160 MHz. This significantly increases the performance of WiFi. Other mechanisms which increase the performance are Multi-User MIMO (MU-MIMO) and higher modulation schemes (256 QAM). However, the approaches within this thesis benefit the most from the increased channel bandwidth and support for MU-MIMO.

A single transmission with 160 MHz channel bandwidth consumes almost all available channel bandwidth in the 5 GHz frequency spectrum. 160 MHz channel bandwidth, however, is only favourable in home environments since there are only 1 (or 2 with Dynamic Frequency Selection enabled) 160 MHz channels available. For enterprise networks 80 MHz channel bandwidth can be more beneficial since it allows to leverage an additional 80 MHz channel.
Chapter 2 Background

Specifically, the 5 GHz spectrum allows to access five 80 MHz channels in total. Furthermore, the higher number of channels allow that WiFi enterprise deployments can re-use more channels for radio frequency planning of non-adjacent cells.

Another optimization of 802.11ac comes with support for up to eight spatial streams and support for MU-MIMO. In contrast, IEEE 802.11n only defines up to four, although there are to date only few chips and APs using more than three spatial streams. Moreover, a large-scale measurement [40] by Meraki Networks in 2015 shows that only 25% of the 802.11n enabled clients support multiple spatial streams. In particular, this is most likely due to the lack of space for antenna diversity within mobile devices such as smartphones and tablets. To overcome this issue IEEE 802.11ac specifies support for MU-MIMO which allows an AP to transmit different spatial streams to several targeted clients simultaneously.

Accordingly, Meru Networks argues that 802.11ac makes the single channel WiFi architecture substantially more effective because of the limited number of available channels. In this thesis, we present a virtual access point abstraction which is a natural fit for the single channel WiFi architecture (cf. Chapter 3).

2.2 Software-Defined Networking

Software-Defined Networking (SDN) is an emerging networking concept to overcome today’s vertically integrated network architectures. The term SDN refers to a novel network architecture which breaks the vertical integration by separating the network’s control logic (the control plane) from the underlying routers and switches that forward the traffic (the data plane). Thus, with the separation of control and data plane, network equipment become simple networking devices just handling the packet forwarding.

In particular, SDN introduces programmability: the forwarding state (Forwarding Information Base or FIB) in the data plane (e.g., in the switches) is directly programmable by a remote control plane (e.g., running on commodity hardware within a data center) via a well-defined API. Moreover, SDN consolidates network control into a single control plane managing a multitude of data plane elements. Accordingly, decoupling the control plane from the data plane allows both to evolve independently, hence enabling faster innovations.

The control logic, is usually handled by a logically centralized control plane. It is typically realized through software control programs (so called SDN applications) that implement algorithms and mechanisms to control the forwarding. In particular, the control plane abstracts the underlying infrastructure for SDN applications and network services. Thus, the control plane is often referred to as the Network Operating System (NOS). SDN allows a wide range of network innovations [43, 44, 68, 84, 94, 142, 150] such as simplifying policy enforcement, network (re)configuration, and network debugging and verification. Thus, with a softwarized control plane, network innovation can in principle occur at the speed of software development. Note, the centralized control plane does not postulate a physically centralized control plane. This would raise questions of performance, scalability, and reliability and thus, would preclude such a solution. That said, a logically centralized control plane can be either realized as a cluster within a data center [35] or distributed across a WAN [86,105].

The Open Networking Foundation (ONF) is a non-profit mixed academic-industrial consortium which drives the advancement of SDN. Specifically, the ONF is leading the advancement

[^2]: http://blog.merunetworks.com/blog/2013/07/can-single-channel-really-work/
of SDN including the definition of APIs, configuration protocols, and architectures targeting carrier grade, data center, and enterprise networks. According to the ONF\textsuperscript{3}, the SDN architecture is designed in accordance with the following five principles:

1. **Directly programmable**: Network control is directly programmable because it is decoupled from forwarding functions.

2. **Agile**: Abstracting control from forwarding lets administrators dynamically adjust network-wide traffic flow to meet changing needs.

3. **Centrally managed**: Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch.

4. **Programmatically configured**: SDN lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves because the programs do not depend on proprietary software.

5. **Open standards-based and vendor-neutral**: When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols.

(Source: ONF Website\textsuperscript{4}, Jan. 6th, 2015.)

### 2.2.1 Terminology

Based on the previous principles we now describe (bottom-up) a simplified SDN architecture (as shown in Figure 2.1) to review the essential terminology used throughout this work:

1. **Data Plane**: The data plane consists of SDN enabled logical wired or wireless network devices that allow remote control over the forwarding and its inner workings through an open south-bound interface (see below). The datapath within the network devices is usually controlled by an agent which implements the south-bound interface.

2. **South-bound Interface**: The south-bound API defines the instruction set of the forwarding device. The south-bound interface also defines a protocol that is implemented by the forwarding devices and the control plane.

3. **Control Plane**: Forwarding devices such as WiFi access points or switches are programmed by the control plane, also known as Network Operating System (NOS). Specifically, the job of the control plane is twofold: (i) it needs to translate control instructions from the management plane down to the data plane (and vice versa) and (ii) it exposes an abstract view of the network to the SDN applications running in the management plane.

\textsuperscript{3}Source: ONF Website\textsuperscript{4}, Jan. 6th, 2015.

\textsuperscript{4}https://www.opennetworking.org/sdn-resources/sdn-definition
4. **North-bound Interfaces**: The control plane can also offer a quite broad variety of north-bound APIs towards the management pane such as RESTful APIs, multi-level programming interfaces, and file system based interfaces. However, today there is still no common northbound interface defined by the ONF yet.

5. **Management Plane**: SDN applications are software programs that control the network behaviour through the north-bound interface. SDN Applications only operate on an abstracted view of the network exposed by control plane plane.

### 2.2.2 The OpenFlow SDN Interface

The OpenFlow [91] protocol, the *de facto* standard SDN interface for wired networks, and was originally designed and specified in the scope of the Stanford Clean Slate Program\(^6\). Nowadays, the OpenFlow protocol is specified by the ONF.

OpenFlow allows to manipulate forwarding rules within the forwarding table of a switch and to obtain traffic statistics. With OpenFlow forwarding decisions are *flow-based* instead of destination-based. Specifically, OpenFlow is based on a match/action paradigm: the controller installs rules which match packets (belonging to a flow) and applies actions which are executed for each flow rule. In OpenFlow Version 1.3 a flow-entry in the *flow table* can be an arbitrary combination of layer-2 (MAC), layer-3 (IPv4 and IPv6), and layer-4 (TCP, UDP) packet header fields, some of which (*e.g.*, IP or MAC addresses) allow partial wildcarding of bits. For example, a flow entry can be to redirect all incoming packets at a specific port matching TCP destination port 80 to a specific output port of a switch. Moreover, recent versions of OpenFlow provide an advanced feature set ranging from support for MPLS and scheduling of flow entry updates to chaining of flow tables.

Nowadays, there exists a plethora of available *Open Source* OpenFlow-related software\(^7\) ranging from OpenFlow switch software (*e.g.*, Open vSwitch and Indigo), stand-alone OpenFlow stacks (*e.g.*, Loxigen or Click) to controller platforms (*e.g.*, Floodlight and ONOS).

---

\(^6\)[http://cleanslate.stanford.edu]

\(^7\)[http://yuba.stanford.edu/~casado/of-sw.html]
2.3 Network-Function Virtualization

A second paradigm shift in the Internet: Network Functions Virtualization (NFV). In today’s modern networks middleboxes play a key role [60, 81, 106, 121, 124, 141]. They provide a wide range of network functions to improve performance as well as security, e.g., middleboxes are used for caching and load-balancing as well as for intrusion detection. However, a vast majority of today’s middleboxes are realized as closed source proprietary hardware appliances each type implementing its own unique control interface. Thus, today’s network operators face the challenge of finding the space and power to accommodate these boxes into their networks: compounded by the increasing costs of energy, capital investment challenges and the rarity of skills necessary to design, integrate, and operate increasingly complex hardware-based appliances. Moreover, hardware lifecycles are becoming shorter as technology and services innovation accelerates, inhibiting the roll out of new revenue earning network services and constraining innovation in an increasingly network-centric connected world. Thus, hardware-based appliances rapidly reach end of life, requiring much of the procure-design-integrate-deploy cycle to be repeated with little or no revenue benefit.

NFV aims to virtualize these network functions into so called Virtual Network Functions (VNFs), and replace dedicated middlebox hardware-based appliances with software applications running on generic compute resources. The resulting orchestration flexibilities can be exploited for a faster and cheaper service deployment and thus, capital expenditures (CAPEX) and operating expenses (OPEX) can be reduced.

SDN can be exploited to steer flows through the appropriate network functions [30, 54, 59, 106]. Thus, SDN and NFV together, recently also called SDNv2 [26] in the context of carrier WAN networks, support fine grained service level agreements as well as an accurate monitoring and manipulation of network traffic.

2.4 The Berlin Open Wireless Lab (BOWL) Testbed

Wireless testbeds are invaluable for researchers to test their solutions under real system and network conditions. However, typically these testbeds remain experimental and are not designed for providing Internet access to users. In our project, BOWL [23, 28], we stepped away from the typical and designed, deployed and currently maintain a live outdoor wireless network that serves both purposes. In [57], we report on the DevOps challenges and lessons learned with our outdoor network. In addition to the outdoor network, we operate two smaller testbeds with less stringent requirements on network availability for early state testing of new software components and network configurations.

2.4.1 Outdoor Testbed

Our outdoor network covers almost the entire TUB campus in central Berlin (see Fig. 2.2). The benefits are twofold: (1) university staff and students have outdoor wireless network access and (2) researchers have a fully reconfigurable research platform for wireless networking experimentation that includes real network traffic (compared to synthetic traffic). Since the network serves as a semi-production network, it brings out several administration and development challenges. The network and its components, including traffic generators,
routers and switches interconnect with a variety of other networks and infrastructures which are not controlled by our project, adding to the complexity.

The outdoor network comprises more than 60 nodes deployed on the rooftops of TUB buildings. We span three different hardware architectures (ARM, MIPS and x86). Each node is powered by passive Power over Ethernet (PoE), which simplifies cabling requirements. All nodes are equipped with a hardware watchdog, multiple IEEE 802.11a/b/g/n radio interfaces and a wired Ethernet interface. One radio interface is always dedicated to Internet access, the additional radio interfaces are free to be used in research experiments, and the wired interface is used for network management and Internet connectivity. All nodes are connected via at least 100 Mbit/s Ethernet to a router that we manage. A VLAN network ensures that we have flat layer 2 connectivity from our router to each node. Our router ensures connectivity to our internal network, the TUB network and the Internet. In its default configuration (which we commonly call the rescue configuration), the network is set up as a bridged layer 2 infrastructure network. Association to the access interface and encryption of the traffic is protected by WPA2 Enterprise (from the standard IEEE 802.11i [2]). Authentication is performed with IEEE 802.1x and Remote Authentication Dial-In User Service (RADIUS).

From the operational point of view, authentication of users to our network has proven surprisingly complex. A project-specific RADIUS installation is used as the pivot point to integrate a number of other distributed and disparate authentication solutions. Our users include (i) centrally managed university IT accounts, (ii) users from our own department, (iii) project-only user accounts, and (iv) eduroam users. TUB user authentication is a critical part of our contractual relationship with the university central IT department. The major challenge we faced and still face is the recovering from errors that might lie in external authentication services that we rely on to support these accounts.

2.4.2 Indoor and Smoketest Testbeds

In addition to our outdoor network, we operate two additional BOWL testbeds: (i) a smoketest network, which provides serial console access for early development and testing and (ii) an indoor network, for small-scale deployment and testing. These networks are...
used for staging before a full-scale deployment and measurements in the outdoor network. Therefore, the research usage pattern of the outdoor network is more bursty, with periods of heavy activity followed by lighter usage, whereas the smoketest and the indoor networks have been in heavy use since their deployment in early 2008. In 2012 the indoor testbed was rebuilt from scratch after the FG INET group moved to the MAR building. The current indoor network consists of 23 x86-based PC Engines Alix 2D3 WiFi access points each equipped with an IEEE 11abg Atheros AR5212/AR5213 and an IEEE 802.11 11abgn Atheros AR9220/AR9280 WiFi card. One or more so-called 2dBi “rubber duck” antennas are attached to each card. Note, the testbed can be extended by the same number of APs from our smoketest network. We also redesigned our smoketest network in 2015 to serve as both, a testbed for teaching purpose and early stage testing of new software components. Unlike our indoor testbed, the smoketest testbed also consists of ARM-based WiFi access points.

2.4.3 Operating System and Node Management

Each node runs OpenWrt [24] as the operating system. The OpenWrt build system typically produces a minimally configured image. To tailor this image to each node, the image is configured at boot time by an auto-configuration system that applies a so-called configuration to the image. A configuration includes all the configuration files that go under the /etc/config directory (the layout is specific to OpenWrt), and additional files, scripts and packages that may be needed by the experimenter. By default, every node runs a default rescue image and uses the aforementioned rescue configuration. Researchers install guest images in extra partitions and use guest configurations. Crash of experiments are expected to occur in practice but their effect needs to be minimized as much as possible. We achieve this objective thanks to the locally installed images. Indeed, in case of failure detection, a crashed node can be rescued by an immediate reboot into the rescue image. This mode of operation is implemented by interfacing the hardware watchdog on the nodes with a failure detection mechanism. Details on how an experiment failure is detected can be found in [92].

Because of the unique needs of experiment monitoring and reconfiguration at run-time, we wrote our own network management and experiment monitoring systems. We first briefly describe our first setup which we used until 2012 (see [57] for more details) and then describe our current system. The former is based on two main components: a node-controller, which runs on each node and a central node-manager. Each node-controller connects to one node-manager. However, several node-managers can be run in parallel i.e. one for each experiment if several experiment are to be run in parallel or for development. Thanks to the underlying VLAN infrastructure and virtualization of the central router, the traffic generated by each experiment can be isolated. More details on this topic can be found in [92].

Our new configuration and monitoring system does not require any modifications or additional software components on the AP’s side anymore and is compatible with every standard OpenWrt. Our current node configuration and monitoring system only relies on the following components: our OpenWiFi node configuration controller and an Icinga2 instance for monitoring purpose. Since today’s OpenWrt provides authenticated remote access via HTTP/HTTPS to its internal message bus (called ubus) we leverage this control interface for our new node configuration system. The interface relies on standard JSON to configure
and monitor an OpenWrt operating system and all ubus enabled services its running. However, our current system is still work in progress. As future work, we plan to integrate our current system with PacketFense, a Network Access Control (NAC) software solution.

2.4.4 OpenFlow support for OpenWrt

All our nodes are OpenFlow Version 1.3 enabled through Open vSwitch (OvS) [104]. OvS is a multilayer software switch which was originally developed by Nicira. OvS consists of two parts: (i) a forwarding module running in the Linux kernel and (ii) userland tools to configure and manage the flow state within the Linux kernel module. The software is licensed under the open source Apache 2 license.

\[http://openvswitch.org\]
Odin: Virtualizing Wireless Access Points

With the prevalence of wireless access technology at the last hop, today’s network operators face the challenge to seamlessly integrate their wireless and wired networks. Especially, this is a challenging task due to the non-uniformity of feature sets in existing solutions and the lack of programmability with commodity off-the-shelf hardware.

In this chapter, we present Odin, an SDN-based solution to address this challenge. With Odin, we make the following contributions:

(i) Light Virtual Access Points (LVAPs), a novel programming abstraction for addressing the IEEE 802.11 protocol stack complexity,
(ii) a design and implementation for a software-defined WiFi network architecture based on LVAPs, and
(iii) a prototype implementation on top of commodity access point hardware without modifications to the IEEE 802.11 client, making it practical for today’s deployments.

To highlight the effectiveness of the approach we demonstrate six WiFi network services on top of Odin including load-balancing, mobility management, jammer detection, automatic channel-selection, energy management, and guest policy enforcement.

3.1 Motivation

Modern enterprise WiFi networks typically consist of few dozens to thousands of Access Points (APs) serving a multitude of client devices including smart-phones, laptops, and tablets. For performance and scalability reasons, these networks require services which include mobility management, load-balancing, interference management, and channel reconﬁgurations. These services have to be realized as applications on top of the basic management functionality of the individual access points. However, different devices from different vendors typically offer different interfaces and do not offer native support for the needed applications. Additionally, today’s enterprises and provider networks are Bring-Your-Own-Device (BYOD) networks, implying that the network has to accommodate an even more diverse set of user device types of different generations.

To manage this growing complexity, network operators need novel abstractions as well as new tools to uniformly manage the wired and wireless parts of their network, e.g., to verify network conﬁgurations, perform troubleshooting, or systematic debugging. In wired networks, recent advances in Software-Defined Networking (SDN) have enabled such features through programmatic control of networks. In an SDN, the control plane and data plane are decoupled, allowing network intelligence and state to be logically centralized. Using this centrally-available global view of the network, SDN allows operators to perform principled
control and management of networks through the use of abstractions [123]. The best known
SDN interface is OpenFlow, which specifies a protocol for a logically centralized controller
to remotely manage forwarding tables within switches.

However, OpenFlow does not address the complexities of WiFi protocols and WiFi networks
which include interference mitigation, mobility management, and channel selection tech-
niques. This is unfortunate, because point-solutions exist for these WiFi-specific network
problems but are often provided only by enterprise vendors through vertically integrated
solutions. However, most cheap, off-the-shelf commodity hardware as deployed in today’s
access networks is outside the purview of such enterprise solutions.

Yet, proposals exist for extensible and programmable WiFi networks [97, 151]. However,
these depend on client-side modifications which we argue is impractical to deploy. This is
an obstacle not only for provider networks, but also for enterprise deployments given the
trend towards BYOD.

3.2 Overview

In this chapter, we present Odin, an SDN-based solution which presents a programming ab-
straction which can provide the features enterprise and provider networks need. It bridges
the gap between the range of features required by network operators and the lack of pro-
grammability in today’s WiFi networks. In the process of designing Odin, we address the
following research questions:

1. What programming abstractions are needed to address the complexities of the IEEE
802.11 protocol stack?
2. How can these abstractions be fit into an SDN architecture?
3. Can the SDN architecture already be realized on top of today’s commodity access
point hardware and without client modifications?

We find that the above questions can be answered affirmatively through the following con-
tributions:

• The proposed Light Virtual Access Point (LVAP) abstraction captures the complexi-
ties of the IEEE 802.11 protocol stack.
• We present a prototype implementation of the LVAP approach which we have made
publicly available.
• We evaluate the framework by presenting six typical WiFi network applications.

Odin is extensible in accordance with the features required in today’s WiFi networks, whilst
being deployable on top of low-cost commodity access point hardware. While we intro-
duced the basic concept of LVAPs in our HotSDN workshop paper [134] and showed the
system’s capabilities in multiple demos [112, 133], this chapter is based on our conference
full paper [119]. The concept of the LVAP abstraction has also been adopted by other
systems [108,109,110].

The remainder of the of the chapter is organized as follows. We next present the use-
cases that lead us to design Odin (§ 3.3). We then explain Odin’s architecture and the

\footnote{Odin source: \url{http://sdn.inet.tu-berlin.de}.}
LVAP abstraction in § 3.4. In § 3.5, we explain implementation details and challenges encountered in building Odin. We then present six WiFi network applications built on top of our framework in § 3.6. Next, we evaluate the core framework through a series of benchmarks in § 3.7. In § 3.8, we position our work with respect to the related work. In § 3.9, we present a discussion that describes the lessons learned and guidelines for further work. § 3.10 summarizes our work.

3.3 Use Cases

Odin has been designed for the following use cases:

**Traffic Offloading and Client Mobility:** Offloading user’s devices to WiFi allows operators to reduce stress on their cellular infrastructure. To this end, it is beneficial to provide users with consistent authentication credentials across their home networks, hotspots, and cellular connections, whilst managing client mobility. This will prevent the user from having to maintain multiple authentication credentials, whilst allowing operators to offload a user’s traffic onto a hotspot when available. This is similar to what is proposed by the Hotspot 2.0 initiative, which however requires clients to support IEEE 802.11u. Furthermore, mobility management is an important feature within enterprise WiFi deployments, typically offered by today’s vendors [12] and also explored by the research community [36, 63, 96, 97].

**Network Performance Management:** Channel selection, load balancing and wireless troubleshooting are crucial for the performance of WiFi networks, particularly within dense deployments like large enterprises or residential networks. Channel selection [72, 93, 128] involves continuously monitoring and then reacting to changes in the wireless environment. Load-balancing [34, 101] typically requires control of clients’ attachment points to the network or the ability to hand off clients between WiFi access points. Lastly, there is a need for the ability to measure, detect, and localize interferers. This is because interference caused by non-WiFi devices can severely impact the achievable throughput of WiFi devices within the same vicinity [107], since both kinds of devices share the same wireless spectrum.

3.4 The Odin System

In this section, we describe the components of Odin and the Light Virtual Access Point (LVAP) abstraction.

3.4.1 System Components

Figure 3.1 illustrates the components of the proposed design and their interactions. In line with the SDN concept, the design decouples the control from the data plane. This is done by having a logically centralized controller that leverages OpenFlow for the wired network, and a separate control plane protocol for the wireless part (elaborated upon in § 3.9). We chose to have separate protocols for programming the wired and wireless parts. This is because in its current state, OpenFlow does not extend well into the realm of the IEEE 802.11 MAC, as its scope is restricted to programming flow table rules on Ethernet-based switches. For instance, it cannot perform matching on wireless frames, cannot accommodate measurements of the wireless medium, report per-frame receiver side statistics, or be used
for setting per-frame or -flow transmission settings for the WiFi datapath.

We now describe the individual components in Odin:

**Odin Controller:** The controller enables network applications to programmatically orchestrate the underlying physical network. It exposes a set of interfaces to the applications (the northbound API) 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, and OpenFlow switches, which the Odin applications can then control.

**Odin Agents:** Agents run on the wireless APs and expose the necessary hooks for the controller (and thus applications) to orchestrate the WiFi network and report measurements. 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. This realizes a distributed WiFi split-MAC architecture. In addition, they perform matching on incoming frames to support a publish-subscribe system wherein network applications can subscribe to per-frame events.

**Odin Applications:** For wireless network applications to take effective control decisions, they 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 in Odin work either reactively or proactively by accessing measurements from multiple layers. This includes (i) measurements collected by the agents, (ii) OpenFlow statistics and (iii) measurements collected by external tools (e.g., `snmpd`). SDN applications can program the network through the northbound API offered by the controller.

### 3.4.2 Light Virtual Access Points

The Light Virtual Access Point (LVAP) is the abstraction in our system 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. The LVAP is a 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. It enables a port-per-source view of WiFi networks akin to that of wired networks. In doing so, it remains orthogonal, but complementary, to trends in physical layer virtualization and RF spectrum slicing [74]. LVAPs are hosted on the agent, and their assignment to agents is controlled by the controller.

**LVAPs as per-client APs**

In regular IEEE 802.11 networks, clients need to associate with a physical AP before sending data frames. The association process begins with the discovery phase, where a client either actively scans for APs by generating probe requests, or passively learns about APs through beacon frames generated by the latter. During an active scan, APs that respond with probe response messages become candidates for the client to associate with. The client then decides which AP to associate with via a locally made choice. At this point, the association is defined between the client’s MAC address and the BSSID of the AP. The BSSID of an AP is a MAC address of the AP’s wireless interface and is different from the SSID, which is a network name.

This design of the WiFi protocol is inconvenient; there is no mechanism for centralized control over the client’s association because the client makes the association decision entirely on its own. Furthermore, the infrastructure cannot instruct the client to re-associate without introducing additional signaling techniques such as [97].

The approach of LVAPs overcomes these difficulties without introducing additional signaling mechanisms between clients and the infrastructure, and thus conforms to our objective of not introducing client-side modifications. With LVAPs, every client receives a unique BSSID to connect to, essentially making them client-specific APs. Figure 3.2 indicates the decision flow in handling a client’s association using LVAPs.

When a client probe scans, a new LVAP is spawned within the Odin agent on the physical AP. This LVAP then responds to the client with a probe response as instructed by the controller, following which, the clients completes the association handshake with its LVAP.
As a result, a physical AP hosts a unique LVAP for each connected client. Every LVAP periodically unicasts beacon frames to its corresponding client. This ensures that a client never processes a beacon frame from another client’s LVAP. The overhead of per-client beacon generation can be reduced by increasing the beacon interval, by setting the NO_ACK bit on the beacon frame, and also leveraging higher data-rates because of the unicast transmission. Note, beacons are typically broadcasted but are identical to probe response frames which are unicasted. Unicasting beacons does not confuse client devices (cf. 3.7.4).

As long as the client receives ACKs for the data frames it generates and receives beacons from the AP it is associated to (in this case, an LVAP), the client stays associated. If the state corresponding to the client’s LVAP is migrated to and instantiated at another Odin agent fast enough, the client does not attempt to re-scan (since from the client’s point of view, its AP is still available). Thus, by migrating a client’s LVAP between physical APs, the infrastructure can now control the client’s attachment point to the network, without triggering a re-association at the client. The LVAP is thus an abstraction for the client’s association state, and simplifies the expression of any handoff-based service like mobility managers and client load-balancers in the form of network applications. Since it does not introduce any additional signaling mechanism between the infrastructure and the client, it is legacy client compatible. In addition, it brings a port-per-source view of WiFi networks akin to that of wired networks, which simplifies fine-grained policy enforcement. Note, if a client experiences significant signal strength reduction as a result of a LVAP being migrated to a distant AP, the client will perform a regular re-scan.

While the notion of per-client BSSIDs is employed commercially to handle mobility [12], the concept of an LVAP is new. The LVAP as a programming abstraction solves problems that extend beyond mobility management, as we will demonstrate in Section 3.6.

**State Encapsulated by LVAPs**

Figure 3.3 represents the state that is bound to each LVAP. For every associated client (identified by the client’s WiFi MAC address), there is a corresponding LVAP which comprises the following information: a unique virtual BSSID, one or more SSIDs, the IP address of the client, and a set of OpenFlow rules. With encryption, the session key will be part of the LVAP state. When an LVAP is migrated from one physical AP to another, all corresponding state (the BSSID, SSIDs, IP address of the client, and OpenFlow rules) is migrated as well. Since the LVAP’s BSSID is always consistent, the client does not perform a re-association. By binding a set of OpenFlow rules to the LVAP and allowing applications to program the wireless and wired side of the AP, we integrate our framework with OpenFlow.

**Slicing and Control Logic Isolation with LVAPs**

Accommodating multiple logical networks on top of the same physical infrastructure with different policies and control applications is called network slicing. 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. Figure 3.4 indicates how slicing can be layered on top of LVAPs. A slice is defined as a set of physical APs (or agents), clients (and thus LVAPs), network applications, and one or more unique SSIDs. When clients attempt to associate to a particular SSID, they are automatically assigned to the slice to which the SSID belongs. Thus, the client and its LVAP are now assigned to the same slice.
Applications operating on this slice can now manage the client (e.g., perform migrations, or add/remove/update OpenFlow rules on the client’s LVAP (cf. § 3.6.6)). The controller ensures that an application is only presented a view of the network corresponding to its slice. Since LVAPs are the primitive type upon which applications make control decisions, and applications do not have visibility of LVAPs from outside their slice, we thus achieve control logic isolation between slices.

**Supporting Authentication Through LVAPs**

Our architecture is compatible with the two most commonly deployed approaches for authentication.

**WPA2** is the de-facto standard for authentication in today’s WiFi networks (defined by IEEE 802.11i). In *WPA2 Enterprise*, a client authenticates against an authentication server with the AP acting as an authentication proxy to negotiate a session key. This session key is added to the client’s LVAP state (cf. 3.4.2) and then used to encrypt the connection. Accordingly, with *WPA2 Personal* the negotiated session key is added in the same way.

**Guest WiFi**: In this mode, a client’s first HTTP request is redirected through OpenFlow rules associated with the LVAP to a login page. The authentication server returns a security token for the client to the controller after a successful authentication.

**Multi-Channel Operation**

Odin benefits from operating physical APs’ wireless interfaces on the same channel for performing seamless client migration. This is beneficial in the context of the Single Channel Architecture (cf. Section 2.1.3). However, when performing LVAP migrations between physical APs of different channels, the operation is similar to regular WiFi handovers where clients need to perform a re-association. For multi-channel operation, Odin can leverage the Channel Switch Announcement of IEEE 802.11h (restricted to 5GHz band) to instruct clients to switch to a different channel while keeping association state intact. Additionally, Odin’s
port-per-source approach to managing clients with LVAPs is complementary to upcoming trends in RF spectrum slicing such as [74]. This will enable multiple LVAPs on the same AP to operate on different channels using a single antenna.

### 3.4.3 Reactive and Proactive Odin Applications

Network applications written on top of Odin can function both reactively and/or proactively. Proactive applications are timer-driven whereas reactive applications use triggers and callbacks to handle events. The latter mode of operation is important particularly within WiFi networks due to the dynamic nature of the channel, and the system needs to react based on inputs from different measurement sources. To this end, in our current implementation, an application can utilize multiple measurement sources.

**Measurements from the agent:** Reactive applications make use of a publish-subscribe system of the Odin agent in order to have a handler invoked at the application whenever a per-frame event of interest occurs at the agents. In our current implementation, applications register thresholds for link-based (PHY and MAC layer) rx-statistics like receiver signal strength indicator (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 that can be collected by the agents.

**OpenFlow statistics:** OpenFlow provides flow and port-based statistics of entries in switches' flow tables. Applications can query these statistics through the controller to make traffic-aware routing decisions.

**External measurement sources:** In addition to the usual per-link and per-flow statistics, applications can access data from multiple measurement sources outside the Odin framework, too, including the CPU and memory utilization and the channel active/busy times collected by tools such as collectd. We demonstrate this in § 3.6.5.

### 3.5 Odin on Commodity Hardware

In this section, we describe implementation details of the Odin prototype.

#### 3.5.1 Controller

The controller is implemented as an extension to Floodlight OpenFlow controller. This allows us to use OpenFlow for Odin specific functionality such as tracking client IP addresses to be attached to their respective LVAPs by tapping into DHCP messages (cf. § 3.5.4). The initial assignment of agents to slices, the initial set of SSIDs per slice, and the network applications to run on each slice are defined via a configuration file. The controller uses a TCP-based control channel to invoke the Odin protocol commands on the agents (cf. 3.8). The controller organizes state on a per-slice basis, allowing it to present applications only a view of their respective slice in terms of associated clients, their LVAPs, and physical APs. Applications are expressed as Java code and run on top of the controller as threads. The programming API includes hooks for applications to view and control mappings of clients to APs, add/remove SSIDs to slices, and to register/unregister subscriptions for the pub-sub mechanism. As a result of using Floodlight, the controller is not distributed and runs on a single machine.
3.5 Odin on Commodity Hardware

3.5.2 Agent

Odin agents run on physical APs, and are implemented in the Click Modular Router [85]. The agents implement the WiFi split-MAC together with the controller, host LVAPs, and collect statistics on a per-frame and host basis. They notify the controller whenever a frame is received that matches a per-frame event subscription registered by a particular application (cf. § 3.4.3). Alongside the agents, we run Open vSwitch on the APs to host OpenFlow rules carried by LVAPs as well as those expressed explicitly by network applications and the controller (for instance, to handle DHCP acknowledgments as described in § 3.4.2). Excluding the OpenFlow rules, the state associated with each LVAP hosted by an agent is approximately 48 bytes in size, and up to 32 bytes per-SSID in the slice (slices can announce multiple SSIDs).

3.5.3 ACK Generation

As mentioned in Section 3.4.2, the agent needs to ensure the IEEE 802.11 requirement of generating ACKs for each data frame that the client sends to its LVAP. ACK frame generation is handled in hardware by the WiFi cards due to their strict timing constraint. On Atheros WiFi cards, this is implemented using a BSSID mask register which indicates the common bits of all the BSSIDs being hosted on that card. Whenever the card receives a valid frame, it verifies whether the destination address of the frame matches one of the BSSIDs it is hosting as per the bits set in the BSSID mask. If yes, an ACK frame is generated. However, a practical limitation exists with this mechanism. Consider the following two BSSIDs \(02:00:00:00:00:02\) and \(02:00:00:00:00:01\). In this case, the last two bits are uncommon between the two BSSIDs, causing the mask to be \(ff:ff:ff:ff:fc\). This leads to the hardware ignoring the last two bits of the destination address of an incoming frame to decide whether to generate an ACK frame. In this case, a frame destined to \(02:00:00:00:00:03\) will also cause the hardware to generate an ACK, even though it is not hosting a BSSID with that value: a false positive.

In Odin, since we use one BSSID per client, this needs to be handled carefully. One way to overcome this issue is to assign BSSIDs to client LVAPs such that the mask on the AP where the LVAP is being assigned retains as many set bits as possible and remains orthogonal to the masks of neighboring APs. This can be achieved in software by the controller. Spreading LVAPs over multiple NICs and APs will also alleviate the problem. Another approach is to suppress spurious ACKs by modifying the check that the hardware performs upon receiving a frame. Today’s low-end Broadcom WiFi cards support custom firmware such as OpenFWWF (our Atheros hardware does not support this). However, we conjecture that a programmable content-addressable memory for matching incoming frames in hardware enables possibilities beyond just selective ACK generation, with little increase in cost [1] and performance impact. This is particularly important as 802.11ac adoption is increasing, which supports throughputs on the order of 6.77 Gb/s. Recent work on software radios such as OpenRadio [32] will also aid in this direction.

3.5.4 LVAP Assignment

We now explain how Odin assigns LVAPs to clients.

Discovery: As per IEEE 802.11, clients perform active scans on all possible channels by broadcasting probe request messages. An agent that receives such a probe request forwards it to the controller. The controller then generates a BSSID unique to the client, and retrieves the list of SSIDs to announce (the union of SSIDs across all slices that the agent belongs to).
Chapter 3 Odin: Virtualizing Wireless Access Points

It then instructs the agent to generate a probe response for each of these SSIDs, through the client-specific BSSID. This is how clients discover SSIDs being hosted via Odin.

**Association:** When a client tries to associate to a specific SSID, it generates probe requests that specify the corresponding SSID. An agent that receives such a probe request forwards the message to the controller. If the controller has not already created an LVAP for the client, it spawns an LVAP for the client on the agent from which this probe request was first received. The client is mapped to the slice that the SSID belongs to (an SSID can only be part of one slice). Once the LVAP is spawned for the client at an agent, the association is performed between the client and the LVAP. If a client does not associate to its LVAP within a configurable amount of time, it is removed from the agent. The agent process maintains a lookup table with the mappings of the client’s MAC address to the LVAP’s state (cf. 3.4.2). It then makes use of this per client state to prepare the right 802.11 frames and ARP packets when communicating with clients.

**DHCP and ARP:** The IP address of the client is required for the agent to correctly handle ARP requests that concern the client. The IP address of each client is obtained dynamically by the controller which sets up OpenFlow rules in order to receive an OpenFlow PACKET_IN event whenever a DHCP-ACK packet is received at an AP. This is done when an agent first registers with the controller. After a client associates and begins to obtain an IP address over DHCP, the controller receives the DHCP-ACK via OpenFlow, obtains the IP address, updates the client’s LVAP, and then forwards the DHCP packet to the client via an OpenFlow PACKET_OUT.

### 3.6 SDN Applications on top of Odin

On top of our framework, we realized six different Odin applications which are correlated to the use cases described in §3.3. For the evaluations, we use ten APs from our indoor testbed distributed across the 16th floor (roughly 750 m²) of the TEL building at the TU Berlin campus. The WiFi APs are based on embedded hardware (PC Engines Alix 3D2) equipped with Atheros IEEE 802.11abgn cards. All APs are running OpenWrt with the ath9k Linux driver, user-level Click, and Open vSwitch supporting OpenFlow version 1.0. The Odin controller runs on a x86-based server equipped with 2 CPUs at 2.1 GHz and 4 GB of RAM. We did not hit CPU or memory limitations in any of our experiments.

#### 3.6.1 Mobility Manager

Supporting client mobility is a crucial feature in enterprise WiFi deployments. We have implemented a purely reactive mobility manager (89 source lines of code (SLOC)) on top of Odin, that leverages LVAP migrations. The application registers a subscription at the agents to be notified whenever an agent receives a frame at a receiver signal strength indicator (RSSI) above a specified value. Using context information passed through the corresponding callback (such as the exact value of the RSSI value and source that triggered the event), the application maintains a map of each client’s RSSI value from the point of view of different agents. It then assigns the clients to the agents where they can get the best RSSI value, whilst subjecting its decisions to a hysteresis to prevent spurious oscillations of a client between APs. With legacy switches in the core, a packet is sent out by the new AP to trigger the “backwards learning” mechanism (ARP flushing) to setup new flow entries. With OpenFlow in the core, this can be achieved by updating flow entries along the new path.

We evaluate the architectural consequence of our reactive mobility manager’s design, i.e., the number of notifications required before performing a client handoff under a given mobility
3.6 SDN Applications on top of Odin

Table 3.1 – Notifications generated between a handoff for two RSSI thresholds \((T_{rss})\) signal strength difference \((\Delta)\).

<table>
<thead>
<tr>
<th>Frame Reception Rate (frames/sec)</th>
<th>(T_{rss} = -96,\text{dBm})</th>
<th>(T_{rss} = -76,\text{dBm})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Delta = 5)</td>
<td>13.2</td>
<td>15.8</td>
</tr>
<tr>
<td>(\Delta = 20)</td>
<td>13.0</td>
<td>15.8</td>
</tr>
<tr>
<td>1000</td>
<td>731.66</td>
<td>910.4</td>
</tr>
<tr>
<td>5000</td>
<td>3373.4</td>
<td>4669.2</td>
</tr>
</tbody>
</table>

Experiment scenario. We show in § 3.7.2 that LVAP migrations have a negligible effect on the client’s throughput. We note that this is only one example of a mobility manager that can be built atop Odin. As demonstrated in 3.6.5, Odin applications can utilize different metrics from multiple sources to base mobility decisions on.

Experiment scenario: We use two APs and a single x86-based client. The client associates to the network and initiates a UDP flow. We vary three parameters for the evaluation: (1) the threshold \(T_{rss}\) the application sets for subscription notifications, (2) the threshold \(\Delta\), i.e., the minimum required difference of the client’s RSSI observed at its current AP and potential new AP for the mobility manager to perform a handoff, and (3) the client’s transmission rate. We artificially add a fixed offset to the client’s RSSI value being recorded by the APs. Using this, we initially set the client’s RSSI at the first AP to be 20\,\text{dB} more than at the other, and then reduce it by 0.1 unit every 100\,\text{ms} whilst increasing it at the other AP by the same amount. After 10s, the client’s RSSI is higher at the second AP. When the difference is above \(\Delta\), the client is LVAP-migrated to the new AP. Thus, only the relative RSSI values of the client at the two APs affects the results (not the absolute values), which enables testing the application using a stationary client. We conduct 5 runs for each combination of parameters and average the results.

Results: Table 3.1 shows the results of our experiments for different combinations of \(T_{rss}\), \(\Delta\), and the client’s transmission rate. A decreasing \(T_{rss}\) leads to an increased number of notifications generated. A smaller \(\Delta\) leads to the handoff being performed faster, and reduces the number of notifications in between handoffs. However, the dominant factor here is the transmission rate of the client itself. This shows that it is beneficial to introduce a rate-limiter for generating notifications by the agents. After all, for the same mobility scenario and during the handoff, there is a large number of notifications generated that do not further improve the mobility manager’s decisions. Note, the framework cannot track clients that do not transmit any frames at all. One workaround is to use Odin’s beacons as a mechanism to track idle clients at different physical APs. In regular WiFi, ACK frames do not contain the source address, but only the recipient address. Since beacons in Odin are unicast, they cause the client to generate an ACK frame addressed to their unique BSSID (which identifies the client). In order to reduce overhead, the system can set the NO_ACK bit on the beacons to avoid ACKs from active clients.

3.6.2 Client-based Load Balancing

The benefit of using a load-balancer in a WiFi setting is to increase the throughput for clients due to increased airtime fairness. To illustrate this, consider a scenario where there are multiple clients and one AP: each device gets almost the same share of channel access when operating at the same physical data rate. If only one of the clients generates upload
traffic whereas the other stations only download data via the AP, the total upload throughput almost equals the combined throughput of the downloaders (since all download traffic is transmitted by the AP and it has to share channel access with a single uploader). This leads to airtime unfairness among the clients. With more APs and proper load balancing, this unfairness can be alleviated. Furthermore, load-balancing can lead to better resource utilization due to spacial reuse and the capture effect when the collision probability is high. The 802.11k [22] amendment also attempts to address load-balancing, but requires modifications to the client.

Since LVAP-migrations are cheap, fast, and infrastructure-controlled (§ 3.7.2), client-migration based load-balancing is a good fit for an SDN Application. We implemented a load-balancer (76 SLOC) to demonstrate the feasibility of such an application on top of Odin. This application queries the framework once per minute to obtain the list of clients that can be seen by different APs and their corresponding RSSI values. It uses this information to build a map of clients to lists of agents that are candidates for hosting their respective LVAPs. The application then evenly re-distributes LVAPs (clients) across physical APs, constrained by the hearing map.

Experiment scenario: We use up to ten APs. 32 clients automatically associate to the network and request files from a server. We place the clients (8 in every cluster) and APs as shown in Figure 3.5. We use Harpoon [131] for flow-level traffic generation using a heavy-tailed flow size distribution, similar to traffic on the Internet. After the standard WiFi association, each client sends web requests to the Harpoon server. We conduct experiments with and without load-balancing enabled. Without load-balancing, the client is assigned to the first AP that receives the association request. With load-balancing, each LVAP is placed on a physical AP that has the highest RSSI and does not violate the client load on the AP. Because of the fixed PHY rate, no rate anomaly [135] can arise. We set the rate for management and data frames to the basic rate (6 Mbps). This ensures that all associated clients can exchange data with the APs.

Results: As expected, the overall TCP throughput increases when load-balancing is enabled (see Figure 3.6(a)). Furthermore, the total throughput is increased when increasing the number of APs. The gain in throughput is attributed to spacial reuse and the capture effect when collisions occur. We observe that TCP connections were established by at least 28 clients across all runs with a median of 30 clients requesting data from the server (see Figure 3.6(b)). Figure 3.6(c) shows the CDF of the per-client throughput of a single run. We observe an increase in fairness among clients with load balancing enabled: i.e., roughly 50% of the clients were able to transmit around 20 MB of data with load balancing enabled compared to 15% without load-balancing. The gain of per-client throughput can
3.6 SDN Applications on top of Odin

Figure 3.6 – Benefits of client-based Load Balancing

(a) Throughput comparison with and without load-balancing
(b) Number of clients contributing TCP traffic
(c) Total throughput per client with and without load-balancing.

Figure 3.7 – Troubleshooting detects jamming on frequency 2462MHz.

be attributed to the previously mentioned spacial reuse, capture effect, and medium access probability of the APs, where each client gets roughly an equal share of airtime at the AP.

3.6.3 Wireless Troubleshooting

Interference from non-WiFi devices such as microwave ovens, cordless phones, wireless security systems, and RF jammers can significantly impede the achievable throughput of nearby WiFi devices. To address this, interference identification systems (e.g., Cisco CleanAir) are starting to become a part of today’s enterprise deployments. These systems detect, localize, and quantify the interference impact caused by non-WiFi sources.

To this end, Odin leverages functionality of modern WiFi cards like Atheros AR9280 that provide coarse-grained energy samples per sub-carrier (frequency spacing of 312.5 KHz) of a WiFi channel. This provides the necessary interface for the development of systems like WiFiNet [107] on top of Odin for detection, localization and quantification of interference from a variety of non-WiFi interference sources.

Our troubleshooting application (102 SLOC) periodically (roughly every 5s) collects channel snapshots. Figure 3.7 shows the effect of a jammer (continuous stream of garbage frames) on channel 11 at 2462MHz over a period of 5 minutes. This data can be used by a jammer detection application, e.g., to localize a jammer via triangulation.

3.6.4 Automatic Channel Selection

Automatic Channel Selection (ACS) algorithms aim at automatically determining the best available channel for a WiFi interface. However, identifying combinations of channels for different APs while minimizing interference is challenging. Due the increasing amount of
different channel bandwidths within the 2.4 and 5 GHz band. On top of Odin, an ACS application can query different channel properties from the agent (or external sources) for data that characterizes the channel properties. This includes, but is not limited to, spectral samples from the sub-carriers or the active- and busy-time in order to estimate the amount of interference on the channel.

We implemented a simple ACS (97 SLOC) application on top of Odin that is based on a per-AP channel selection scheme. It scans across all available channels and computes the average and the max RSSI for each channel center frequency. Based on multiple subsequent spectral scans, the ACS application picks the channel with the smallest maximum and average RSSI. This example Odin application can be extended to also utilize additional channel properties provided by the Odin agent or external data sources in order to estimate the channel load, e.g., channel active- and busy-time. This information can then be used to implement functionality akin to [113].

Figure 3.8(a) shows a snapshot of channel load of all center frequencies within the 2.4 GHz band during the day in our office environment. These snapshots are aggregated by our ACS application over time in order to get to a view similar to the one in Figure 3.8(b). Based on this aggregated view, the application then performs channel selection according to the heuristic described above. As indicated within the snapshot and the aggregated view, it can be seen that channel 11 is less utilized than channel 1, which we confirmed to be correlated with the number of APs operating on each channel.

3.6.5 Energy Efficient WiFi Networks

The problem of energy consumption in telecommunications infrastructure is well known. In 2007, telecommunication infrastructures has been responsible for roughly 37% of the global ICT energy expenditure [138]. Of this, access networks consumed roughly 79% of the overall energy [41]. With the advent of massive hotspot deployments this is likely to increase even more. This raises questions on the energy efficiency of wireless access networks, as these networks do not always serve peak demands [27]. Hence, it is necessary to gracefully adjust the network to the current demand, improving both energy consumption and traffic pollution [89].

Jardosh et al. [79] and Goma et al. [61] propose saving energy by aggregating wireless users (and accordingly their traffic) onto a minimum number of WiFi APs allowing other APs to sleep. While Jardosh et al. show that the power consumption in enterprise networks
3.6 SDN Applications on top of Odin

Figure 3.9 – High-level design of the energy efficient WiFi networks architecture, realized through a joint mobility and energy management application and Energino. The APs are organized into clusters, with each cluster having a master AP and multiple slave APs.

can be reduced to 46%. The evaluation by Goma et al. indicates that up to 90% of the gateways in home networks can be switched off in urban settings. In particular, this also leads to performance improvements of up to 25% due to reduced crosstalk on the DSL uplink. However, introducing such mechanisms into residential networks without client modifications remains a challenge for today’s operators.

Thus, Odin is a natural fit to tackle this issue, since handovers are cheap and require no client side modifications. In [112], we have demonstrated an Odin application that combines Odin’s mobility management application with Energino [62], an integrated energy management system. Energino allows real-time energy consumption monitoring and remote power control, i.e., powering off the AP while only the Energino is powered.

The joint mobility and energy management application is sketched in Figure 3.9 and works as follows: The APs are organized into clusters, with each cluster having a master AP and multiple slave APs. The master APs always remain online and provide full coverage. Using a combination of observed network demands and an energy saving policy, the system activates or deactivates slave APs, and offloads clients between the master and the slaves accordingly. This is expressed as an energy manager written as an Odin application, which collects energy measurements via energy meters in order to make informed handover decisions.

Moreover, with a more fine grained power control, cell breathing [31] strategies can be applied.

3.6.6 Guest Policy Enforcement

Centralized policy enforcement is an important requirement in enterprise WiFi deployments. This is one avenue where LVAPs complement OpenFlow-based access control particularly well. A guest network application uses the framework’s API in order to instantiate a guest network on top of a slice of physical APs. It then attaches OpenFlow rules to all LVAPs of that slice which restricts the corresponding clients to be able to access only a certain set of subnets and ports. Since the OpenFlow entries follow the LVAP, other applications such as a mobility manager or load-balancer can operate on the same slice and perform LVAP migrations as well.
Chapter 3 Odin: Virtualizing Wireless Access Points

3.7 System Evaluation

In this section, we evaluate the CPU and memory utilization of the Odin controller as well as the latency involved in handling probe requests.

3.7.1 Controller Load due to Publish-Subscribe Subsystem

We evaluate the controller’s CPU and memory utilization when running the mobility manager (cf. Section 3.6.1) under synthetically generated load. The aim is to understand the load involved in running a realistic application that makes use of the publish-subscribe subsystem.

We use nine APs of our testbed. The mobility manager is notified whenever a frame is received by any of the APs above a given signal strength threshold. Based on these notifications, the mobility manager decides on whether or not to trigger a client handover. A load generator running on a dedicated server invokes RPCs on the agents in order to mock client associations from a fixed list of clients. It then creates 1000 mock frame receptions per client per second at the APs at varying signal strengths to simulate the reception of arbitrary 802.11 frames. Depending on the signal strength of each frame, the agents notify the controller. Across different runs of the experiments we vary the number of clients as well as the number of APs that can overhear a single frame transmission by a client (density factor). The density factor determines how many APs generate a notification for a single frame transmission by a client. Each run of our load generator for a particular parameter takes 250 seconds. We repeat the experiment 10 times for each combination of the parameters and observe the steady state CPU and memory utilization.

We find that an increase in the number of clients for a fixed density factor leads to an increase in the controller’s CPU utilization (see Figure 3.10). Furthermore, for a fixed number of clients, an increase in the density factor leads to an increased number of the mobility manager’s subscriptions being triggered, leading to more control messages to the controller. For 500 clients with density factors of 5 and 7, our APs were CPU bottlenecked before being able to saturate the controller. However, we note that 500 is already a very large number of clients to support with only 9 APs. The memory utilization at the controller is 180 ± 7MB across all runs.

3.7.2 LVAP Handoff Micro-Benchmark

Since LVAPs are a central primitive of Odin, we perform experiments to gauge their effectiveness. The goal is to understand what performance related assumptions Odin applications can make. To this end, we compare LVAP-handoffs against standard WiFi handoffs. We also demonstrate that frequent LVAP-based handoffs do not affect the throughput of a TCP connection.

We use a single client and two APs of our testbed. An HTTP server in the same network acts as a traffic end-point. Since DHCP and authentication related delays only appear in the first connection to the network, the client is provided a static IP and no authentication is performed. Note that an LVAP handoff is not susceptible to the authentication delay. We conduct this experiment on a 5 GHz channel during the night to limit interference.
3.7 System Evaluation

Comparison of Handoffs

For comparing the impact of handoffs, a client associates to an AP and begins an HTTP download of a large file. After 13 seconds, the client is made to handoff to another AP. When using Odin, the handoff uses an LVAP migration, whereas with regular WiFi, the client is explicitly told to perform a handoff using the `iw` command.

Figure 3.11 shows the TCP throughput over time with standard WiFi compared to Odin. For regular WiFi, the throughput drops to zero for several seconds before recovering. With Odin’s LVAP handoff, the TCP throughput is unaffected. As Figure 3.11 indicates, there is an overall reduction of throughput (close to 5 Mbit/sec) with Odin as opposed to regular WiFi. This is, because we currently use userspace Click to run the Odin agents, resulting in slower and jittery forwarding performance on our APs which makes TCP to throttle down. However, this is orthogonal to continuously maintaining L2 and L3 connectivity, which Odin successfully achieves through LVAP migrations.

LVAP-Handoff Frequency Benchmark

To understand how often an LVAP-handoff can be executed against a client without affecting its performance, a single `iperf`-based TCP flow is executed with the client as the source over a period of 30 seconds. Between the 5th and 25th seconds of the measurement, LVAP-handoffs are repeatedly triggered between the two APs at fixed rates. Figure 3.12 shows that LVAP-based handoffs are leading to no significant throughput degradation of the TCP flow. Specifically, even when repeatedly performing LVAP-handoffs every 100 ms the throughput degradation is negligible. This illustrates the inexpensive nature of this operation. Furthermore, in the event of LVAP oscillations due to poorly written control-logic, client performance will not be impacted significantly.

3.7.3 Probe Request Serving Latency

Since Odin invokes the controller for handling active-scans by clients, we evaluate whether our system can deliver probe responses to clients within the stipulated 30ms constraint.

For the experiment, a load-generator uses a hook on the agent that triggers the effect of a probe request reception. Nine APs of our testbed are used. We increase the rate of probe requests received at each agent. Each agent measures the time it takes in between receiving
Table 3.2 – Latency for serving probe requests (excluding transmission time on the channel) across 9 APs

<table>
<thead>
<tr>
<th>Scans per AP/s</th>
<th>Avg. Latency [ms]</th>
<th>Std-deviations [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.791</td>
<td>1.078</td>
</tr>
<tr>
<td>20</td>
<td>1.633</td>
<td>0.911</td>
</tr>
<tr>
<td>100</td>
<td>1.442</td>
<td>3.266</td>
</tr>
<tr>
<td>200</td>
<td>7.373</td>
<td>28.881</td>
</tr>
</tbody>
</table>

the probe request, informing the controller, having the controller respond with a BSSID, and then for the agent to construct a probe response message.

Table 3.2 shows, that the delays introduced due to our split-MAC design are well within the 30ms bound described above. We note that the latency is dominated by the network round-trip delay. Running the load-generator at 1,800 scans per-second (200 scans per-second-per-AP) lead to excessive queuing in the 100 Mbit/s Ethernet switch that our APs were connected to, which lead to the larger delays.

3.7.4 Compatibility with Clients

We have tested our framework with common WiFi client devices, such as Windows, Linux, Mac OS X, iOS and Android devices. Compatibility with a multitude of client devices was demonstrated at \[112,133\].

3.8 Related Work

We next position our work with respect to existing approaches that introduce programmability and/or perform centralized management of wireless networks.

Why not OpenFlow?: There have been efforts to bring OpenFlow to wireless APs (e.g., using OpenFlow together with SNMP \[146\]). However, we argue that OpenFlow in its current state is ill-suited to orchestrate WiFi networks for many reasons. It cannot perform matching on wireless frames, cannot accommodate measurements of the wireless medium, report per-frame receiver side statistics, or be used for setting per-frame or -flow transmission settings for the WiFi datapath. Yet, extending OpenFlow to accommodate these requirements does not yield any specific benefits. By implementing a custom protocol for handling Odin agents, we thus achieve a cleaner separation of concerns.

Vendor solutions: A plethora of commercial enterprise WiFi solutions exist. These solutions typically manage APs centrally via a controller which is hosted either in the local network \[12\], or remotely in the cloud \[11\]. Unfortunately, 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.

Virtual APs: Virtualization of APs have been studied in different contexts. \[12\] uses a one-BSSID-per-client approach to provide seamless mobility. SplitAP \[37\] pools together multiple APs in order to regulate air-time fairness. On the other hand, we demonstrate multiple use-cases for the LVAP abstraction as well as its utility as an API for building an SDN for WiFi networks.

Programmable wireless networks and centralized scheduling: Dyson \[97\] addresses the problem of extensibility in wireless LANs, by defining a set of APIs for clients and APs to be managed by a controller. The controller can query these nodes for channel information,
form a global view of the network, and then control the network’s behavior to enforce a
set of policies. Flashback [47] 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. DIRAC [151] 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, these systems require special software or
hardware on the client, which raises questions of practicality, and goes against the design
requirements for our framework. There are systems that do not modify the client in order
to deliver services. In DenseAP [96], channel assignment and association related decisions
are made centrally by taking advantage of a global view of the network. However, it does
not offer slicing of the WiFi, and provides a limited form of client association management
because explicitly forces clients to disconnect, and then perform a re-scan in order to change
the client’s attachment point to the network. Thus, they do not perform client handoffs
seamlessly.

CENTAUR [127] improves the data path in enterprise WiFi networks by using centralization
to mitigate hidden terminals and to exploit exposed terminals. It is a natural fit for an
application on top of Odin. FlowVisor [125] slices the network resources at the flow level and
delegates control of different slices to controllers for wired networks. It achieves this by acting
as a transparent proxy between OpenFlow switches and multiple OpenFlow controllers. This
results in isolation of slices by ensuring that a controller operating on one slice cannot control
traffic of another slice. With our framework, we have brought these concepts of isolation
into WiFi networks. [145] supports multiple concurrently running experiments using slicing
by SSIDs. However, as we show in this chapter, slicing by BSSIDs as is done in Odin offers
more powerful client isolation and management abilities.

3.9 Discussion

In designing Odin, we were careful to keep in mind upcoming trends in physical layer virtu-
alization techniques, datapath programmability, hardware-based packet matching and oper-
ator requirements.

Virtualization of the PHY layer: Although we have addressed isolation at the IEEE
802.11 MAC layer, our system does not handle virtualization of the PHY layer, which is a
logical next step. The IEEE 802.11 standard defines a Point Coordination Function (PCF),
for centrally scheduled channel access. However, the PCF is rarely implemented in to-
day’s WiFi hardware/drivers. Picasso [74] enables virtualization across the MAC/PHY. It
proposes a technique to perform spectrum slicing and allows a single radio to receive and
transmit on different frequencies simultaneously. MAClets [39] allows multiple MAC/PHY
protocols to share a single RF frontend. These advances can be used by Odin to oper-
ate multiple LVAPs with different characteristics on different channels on top of the same
AP. Alternative approaches, such as [136] and [143], are incompatible with today’s WiFi
MAC/PHY and thus do not fit our design requirements.

Programmability of the WiFi data path: Odin’s current implementation does not yet
provide programmability of per-flow WiFi PHY settings. This is well within the scope of
our design because the per-flow and -client transmission settings can be added as LVAP
state. Enabling per-flow transmission settings will allow applications to centrally implement
rate and power control. With OpenRadio [32], our system could also benefit from a clean-slate programmable network dataplane. This would allow Odin to work around hardware limitations such as that with the BSSID registers used for ACK generation (cf. § 3.5.3). We see OpenRadio, combined with Odin, as a steps towards WiFi networks that are fully programmable down to the PHY.

**Performance isolation between slices:** Odin in its current form achieves control logic isolation between slices. As of now, it is difficult to enforce FlowVisor-like bandwidth and CPU isolation (on an AP) between slices. First, per-flow bandwidth isolation can be performed on the agents using a token-bucket approach, but this only provides weak isolation on the physical layer, due to the dynamic characteristic of the wireless medium. Although modern WiFi cards are equipped with multiple queues to provide QoS, the assigned priorities and scheduling are hard to adjust. Hence, the FlowVisor approach of per-port queues does not suffice, and WiFi-specific QoS mechanisms need to be incorporated. Second, for agent CPU isolation, throttling control messages between the controller and agent does not suffice. This is because the performance of the pub-sub mechanism has a direct bearing on the effectiveness of a reactive application. If we throttle notifications being sent from an agent to the controller, it may negatively affect the decision-making at the application. We are currently exploring what the right design points are.

**Decision interference within the same slice:** Floodlight allows multiple applications or modules to operate on top of the same network, but provides no guarantees on performance interference. Odin applications that operate on top of the same slice are subject to this question as well. For instance, if there is a mobility manager and a load-balancer running on top of the same slice, and these applications attempt to perform LVAP migrations, they may interfere with each others’ decisions. This is part of an unsolved problem with OpenFlow-based architectures, which is “how do we reason about multiple applications that operate upon the same slice?”. Is it possible to have a network OS that mediates access to shared resources by different applications in the same sense as a host operating system does? One workaround employed by many controllers today is to have only one application per slice [64,86]. In Odin, the workaround is to delegate responsibilities of different primitives to different applications. For instance, within a single slice, LVAP migrations are solely handled by an application that does both mobility management and load-balancing, whereas a rate-control application would manage tx-rates per-client based on the time-varying view of the network. At the same time, an access control application can manage the OpenFlow rules attached with each LVAP.

**Division of responsibilities between the controller, agent, and applications:** In Odin, we have a well-defined division of responsibilities between the agents, controller, and the applications. Agents handle all management frames locally except for probe requests from clients that it is not hosting an LVAP for. It also handles rx-based triggers for the pub-sub mechanism. The controller handles probe requests from clients by informing the agents about the BSSID to use for a particular client. It also assigns LVAPs to associating clients. Applications operate purely on the LVAP view exposed by the controller, or using statistics-based triggers at the agents. In this split-up, applications describe higher-level functionalities on top of the WiFi network (such as mobility management). They do not control LVAP assignment to clients or management frames, which is done by the controller. However, the LVAP represents a client that is already associated to the network and has an attachment point through an AP. There are situations where applications want control over clients while they are associating. For instance, a policy manager that needs to reactively determine OpenFlow rules for an associating client’s LVAP based on its authentication.
credentials. This has to be done before the LVAP is assigned to the agent. Given such a situation, it may be useful to involve applications in the LVAP assignment process as well.

**Realizing a virtual BSS through LVAP grouping:** In order to deal with the limited number of independent BSSIDs handled by today’s WiFi cards. Odin’s LVAP abstraction could be extended to allow grouping of LVAPs under a single unique BSSID. Specifically, Odin could assign the same BSSID to stationary clients to reduce the number of unique BSSIDs in the system. This would be akin to today’s traditional WiFi networks where each AP hands out one or more unique BSSIDs per AP. The same could be realized in case of multiple SSIDs.

### 3.10 Summary

In this chapter, we introduce Odin, a Software-Defined Wireless Networking framework targeting WiFi networks. We present our novel Light Virtual Access Point (LVAP) abstraction, that addresses well the complexities of the IEEE 802.11 protocol as we show via six common network services. Odin runs on top of today’s commodity access point hardware without requiring client-side modifications, whilst being well-suited by design to take advantage of upcoming trends in physical layer virtualization and hardware extensions. Thus, with Odin, we present an solution to uniformly manage both wired and WiFi networks given the requirements of today’s network operators.

However, Odin addresses only one aspect of the envisioned flexible and programmable WiFi architecture. In particular, Odin does not provide fine-grained control over the WiFi data path or control over middleboxes. Thus, we next focus on a joint SDN and NFV approach where we reap the benefits of both concepts for WiFi, e.g., to migrate firewall state between WiFi access points to support seamless mobility.
OpenSDWN: Programmability of the lower MAC

The quickly growing demand for wireless capacity and the numerous application-specific requirements stand in stark contrast to today’s inflexible management and operation of off-the-shelf WiFi networks. The (last) wireless hop is often critical for network performance, as it can contribute a non-negligible delay and may constitute a bandwidth bottleneck. In particular, today’s data path programmability at the last WiFi hop is rarely optimized to the application-layer demands and their diverse traffic requirement, e.g., their individual sensitivity to packet loss or jitter. However, WiFi networks offer several unique knobs to influence the transmission characteristic, such as transmission rate and power, as well as retry chains. Thus, traffic and application-aware optimizations are feasible if an SDN for wireless can provide mechanisms to control the WiFi-specific transmission settings on a per-slice, per-client, and per-flow level.

In this chapter, we present OpenSDWN, a flexible, novel, and open Software-Defined Wireless Networking (SDWN) architecture. Specifically, with OpenSDWN, we reap the benefits of SDN and NFV for home and enterprise WiFi networks. OpenSDWN exploits 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, e.g., migrating firewall state between hotspots when performing a client handover. OpenSDWN out-sources the control over the network through a participatory interface, e.g., to the user, application or an Internet Service Provider.

4.1 Motivation

Today’s home and hotspot networks lack the possibility to differentiate traffic in an application aware manner, i.e., applications requirements are not considered at the last hop. While such a differentiation may not be possible in the Internet due to network neutrality policies, it is not only legal but also often desirable to differentiate between applications on the last wireless hop. Specifically, due to network neutrality reasons, the internet downlink usually offers no specific traffic classification. Indeed, WiFi networks offer several unique knobs to influence the probability of successful transmissions, such as transmission rate and power, as well as retry chains. To overcome this limitation, existing solutions can be used for traffic identification. This introduces opportunities for a fine-grained and application specific
transmission control, e.g., for service differentiation. However, this requires programmability of the WiFi datapath and integration with the wired counterpart.

4.2 Contribution

OpenSDWN is based on a unified, programmable control plane as illustrated in Figure 4.1. 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.

OpenSDWN comes with interesting use cases: (1) It enables service differentiation, and allows administrators or users to specify application and flow priorities on the wired and wireless portion of the 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 colocation at night to save energy). (3) Network functions such as firewalls and NATs can be deployed flexibly, e.g., outside user premises. (4) OpenSDWN also introduces flexibilities in terms of network control: the system exposes a participatory interface à la [56], through which users can indicate priorities for their applications. The control can also be outsourced to an Internet Service Provider (ISP), e.g., for troubleshooting.

OpenSDWN leverages the LVAP abstraction (see Chapter 3) and adds the following functionality (see Figure 4.2):

- 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.
- A unified SDN and NFV abstraction through virtualized middleboxes and access points, e.g., to facilitate an easy handling and migration of per-client state.
- 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 (vMBs). Second, to identify and classify flows, and hence enable service-differentiation, OpenSDWN relies on a Bro Intrusion Detection System (IDS) [103]. 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.

58
4.3 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 4.3 for some illustrations.

4.3.1 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, power, retransmission and RTC/CTS strategy). 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).

2. **Mobility and migration**: By virtualizing not only the per-client access points, but also the middleboxes (MBs), OpenSDWN supports both seamless user mobility and dynamic resource allocation. The more dynamic resource management introduced by OpenSDWN enables 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. **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.).

4. **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 [56]: 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 or for defining requirements and updating transmission rules.
Chapter 4 OpenSDWN: Programmability of the lower MAC

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

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

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

Figure 4.3 – Three basic operations supported by OpenSDWN.

A local controller can also maintain connectivity between users in a neighborhood during network failures on the uplink.

### 4.3.2 Overview

OpenSDWN is based on a joint SDN and NFV (also known as 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 4.3(a)), which facilitates an easy handling and migration of per-client state, also beyond CPE boundaries. The OpenSDWN abstractions can be seen as an extension of Odin (see Chapter 3) to NFV: Odin’s LVAP concept abstracts the complexities of the IEEE 802.11 protocol stack (client associations, authentication, and handovers), and enables the unified slicing of both the wired and wireless portions of the network. The former is achieved by encapsulating the client’s OpenFlow state. OpenSDWN additionally introduces per-client virtual middleboxes, short *vMBs*, 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 LVAPs and vMBs.

2. **Programmable Datapath:** The programmable datapath gives the possibility to set per-flow specific transmission settings as shown in Figure 4.3(b). 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): for instance, key frames of a live stream may be given 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 4.3(c) depicts the participatory interface.
4.4 The OpenSDWN System

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

4.4.1 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 builds upon Odin (cf. Chapter 3), from which OpenSDWN also inherits:

- **The Light Virtual Access Point (LVAP) abstraction**: essentially the client’s association state (the BSSID, SSIDs, client IP address, and OpenFlow rules).
- **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.
- **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 datapath 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.

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 4.6 for more details).

4.4.2 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 MBs 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 MBs, short vMBs. vMBs 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 user-specific state information and can be transferred from one MB instance to another. On top of a physical MB runs a MB agent which needs to accomplish three primary tasks: (i) interface with the physical resources of the MB, (ii) handle vMBs and (iii) expose the control of the MB to a remote entity (the controller). 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), the state of the active connections, the 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 the exactly same way the old one. vMBs were designed to give applications the possibility to manage user related MB state across physical MBs, 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 4.6).

4.4.3 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.

4.5 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.

4.5.1 Deployments and Methodology

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

- Our research group’s indoor WiFi network (see Section 2.4). 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 release Chaos Calmer with the ath9k Linux driver, user-level Click modular router [49], 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 middleboxes are evaluated on non-virtualized servers with 4 CPU cores supporting hyper-threading and at least 8 GB RAM. 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.

4.5.2 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.
Chapter 4 OpenSDWN: Programmability of the lower MAC

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 4.4(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 (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 4.4(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 4.4(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
4.5 Evaluation

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 4.5 depicts the measured service detection 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 \( \mu s \) 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 4.1 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.

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 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 mae80211, 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 DASH-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 MAC layer retransmissions (see Figure 4.7(a)).

### 4.5.3 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 4.6(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 $\mu$s for a single entry in a vMB object. Next, we evaluate the read time (getState) which decreases constantly. The average value stabilizes at around 270 $\mu$s (see Figure 4.6(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 40 $\mu$s (see Figure 4.6(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.

**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 Odin detects a client with a higher RSSI at a new AP, a handover event is generated and the client’s firewall state migrated to the AP before the handover. The firewall state migration service then decides whether the state associated with the mobile user needs to be migrated and executes the operation. The application keeps a mapping between APs and firewalls. If the client is moving over to an AP that corresponds to a different stateful firewall than the current, a migration of the client’s connection tracking state is performed.

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

Algorithm 1: Mobility Service

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 are virtually simultaneous because RPC method
calls are asynchronous, and called at different agents. Table 4.2 shows the measured 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 potential power that the simplicity of the vMB abstraction exposes to a network programmer. Note, the agent to kernel communication for a single rule is below one millisecond. The RPC interface and entry processing from the Linux Kernel netlink interface contribute the most to the processing time.

### 4.5.4 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 an 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 4.7(c) shows that the throughput and frame count increase after 28 seconds, when the application switches from multicast to unicast. Figure 4.7(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.
4.6 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 Floodlight OF controller. The MB agent is realized in python and implements a newly defined MB protocol.

4.6.1 Interfaces

Interfaces to the physical WiFi and middlebox resources are provided by agents. We describe the radio and middlebox interfaces in turn. Moreover, Table 4.3 depicts the south-bound interface between the agents and the controller.

Table 4.3 – Subset of South-bound APIs provided by the framework

<table>
<thead>
<tr>
<th>Radio API: (Controller to agent)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add/remove/set-lvap</td>
<td>Add/remove/update an LVAP on an agent</td>
</tr>
<tr>
<td>read-lvap-table</td>
<td>Obtain the list of LVAPs on an agent</td>
</tr>
<tr>
<td>read-net-stats</td>
<td>Query per-station rx-stats at the agent</td>
</tr>
<tr>
<td>(read/set)-subscriptions</td>
<td>Query/set per-flow transmission rules on an agent</td>
</tr>
<tr>
<td>(read/set)-channel</td>
<td>Query/set the list of subscriptions at the agent</td>
</tr>
<tr>
<td>(read/set)-beacon-interval</td>
<td>Query/set the channel the agent listens and transmits on</td>
</tr>
<tr>
<td>(get/set)-config</td>
<td>Get/Set configuration of parameters a virtual middlebox</td>
</tr>
<tr>
<td>(get/set)-stats</td>
<td>Get/Set the state of a virtual middlebox</td>
</tr>
<tr>
<td>getAvailableEvents</td>
<td>Get a list of available events supported by a middlebox.</td>
</tr>
<tr>
<td>subscribe/unsubscribe</td>
<td>Un-/subscribe from receiving notifications</td>
</tr>
</tbody>
</table>

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)
Chapter 4 OpenSDWN: Programmability of the lower MAC

Figure 4.8 – 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.

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 the minstrel rate control algorithm of the mac80211. 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 LVAPs or maximum transmission time to WDTX rules. WDTX rules are bound to OF rules through 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 4.8 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 4.9 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 space keeps track of all traffic passing through 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. 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 notifications it is no longer interested in, according to the ID. An `Event` message is sent every time a change occurs in the internal state of the MB.

### 4.6.2 Control Plane

The OpenSDWN controller exposes a set of interfaces to the applications 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, MBs and OF switches, which the applications can then control. The northbound API shown in the following Table 4.4:

<table>
<thead>
<tr>
<th>Table 4.4 – Subset of APIs provided by the framework</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>North-bound API for Radio</strong></td>
<td>Description</td>
</tr>
<tr>
<td><code>getClients()</code></td>
<td>Get slice-specific view of associated clients</td>
</tr>
<tr>
<td><code>getAgents()</code></td>
<td>Get a view of agents in the application’s slice</td>
</tr>
<tr>
<td><code>getDataFromAgent()</code></td>
<td>Query agent for per-client rx-statistics</td>
</tr>
<tr>
<td><code>register/unregister/Subscription()</code></td>
<td>Subscribe to a per-frame event of interest at agents</td>
</tr>
<tr>
<td><code>add/removeAgent/Event()</code></td>
<td>Add or remove an SSID to the application’s slice</td>
</tr>
<tr>
<td><strong>Northbound API specific for DPIs</strong></td>
<td>Description</td>
</tr>
<tr>
<td><code>start/stop DPI()</code></td>
<td>Start or stop the DPI daemon running on the agent</td>
</tr>
<tr>
<td><code>get/setInterfaceToMonitor()</code></td>
<td>Get/Set the network interface the DPI should monitor</td>
</tr>
<tr>
<td><code>uninstall/installService()</code></td>
<td>Un/Subscribe for services</td>
</tr>
<tr>
<td><code>getServiceInstances()</code></td>
<td>Get the list of services installed on a DPI instance</td>
</tr>
<tr>
<td><code>checkRunning()</code></td>
<td>Check whether the service is currently running</td>
</tr>
<tr>
<td><code>getEventTypes()</code></td>
<td>Get event types that DPI supports</td>
</tr>
<tr>
<td><code>getFieldsToSearch()</code></td>
<td>Get a list of header fields that DPI is able to inspect</td>
</tr>
<tr>
<td><strong>Virtual Middlebox Northbound API</strong></td>
<td>Description</td>
</tr>
<tr>
<td><code>migrate_vMB()</code></td>
<td>Move a vMB from one physical MB to another</td>
</tr>
<tr>
<td><code>add_vMB()</code></td>
<td>Add a vMB to a physical MB</td>
</tr>
<tr>
<td><code>remove_vMB()</code></td>
<td>Remove a vMB from a physical MB</td>
</tr>
<tr>
<td><code>clone_vMB()</code></td>
<td>Clone a vMB from a physical MB</td>
</tr>
</tbody>
</table>

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 receiver 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 access data from 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
4.6 Prototype Implementation

Figure 4.10 – The Participatory Interface allows users to assign priorities to a particular service on a per-device basis.

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 to apply 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, following 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 parsed to a flow and its tag, which is determined by Bro and indicates the service being carried in the flow’s payload. It then builds the Openflow matches that will be used later to set the rules, and checks the storage for the pre-installed policies for the client IP and the specific service (tag). The retrieved policies, which are defined in terms of Openflow actions, are installed in the switch currently responsible for the traffic of the given client. The exact switch can be determined, in case Odin is integrated, by querying which physical access point is hosting the LVAP associated to this client. After the proper Openflow actions are installed, the Traffic Manager installs a rule on the switch attached to the host running Bro, to drop all traffic regarding this flow, in order to make sure Bro does not spend any resource analyzing already tagged flows, hence avoiding unnecessary load on Bro.

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 triggers and callbacks to handle events. The latter mode of operation is particularly interesting in the context of WiFi networks due 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 use a publish-subscribe system of 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 receiver signal strength indicator (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 which can be collected by the agents.

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

- **Request-response model:** A controller interested in the contents of a remote middlebox, can send a request message to the MB’s agent and the corresponding action is performed: part of the existing internal state is read and sent back, modified or deleted, or new data is added. This feature is useful for the remote control over the behavior of the machine and represents the proactive behavior of the 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 register. Because 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 of entries through the switch flow tables. Applications can query these statistics through the controller to make traffic-aware routing decisions.

### 4.7 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 [125] 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 [12], or remotely in the cloud [11]), 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 [119, 134] and Aeorflux [117, 118], by introducing datapath programmability, network function virtualization and participatory interface. Over the last years, several interesting architectures have been proposed towards a more programmable WiFi, for example Dyson [97], an architecture for extensible wireless LANs which also defines a set of APIs for clients and APs to be managed by a controller. Flashback [47] 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 [147] is a programmable wireless testbed for dense WiFi networks as they occur in in residential and enterprise settings. Atomix [33] 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 [45] 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 [151] 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, and violates the design requirements for OpenSDWN. There are also systems that do not modify the client in order to deliver services. In DenseAP [96], 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 [127] seeks to improve the datapath in enterprise WiFi networks by using centralization to mitigate hidden terminals and to exploit exposed terminals.

Picasso [74] enables virtualization across the MAC/PHY and uses spectrum slicing. It allows a single radio to receive and transmit on different frequencies simultaneously. MAClets [39]
allows multiple MAC/PHY protocols to share a single RF frontend. These advances can be used by OpenSDWN (and already Odin) to operate multiple LVAPs with different characteristics on top of the same AP. Alternative approaches, such as [136] and [143], are incompatible with today’s WiFi MAC/PHY and thus do not fit our design requirements. FICA [136] introduces a new PHY layer, that splits the channel into separate subchannels which stations can simultaneously use according to their traffic demands. Jello [143], 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 [32], 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 [46] (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 [80] simplifies the operation of cellular networks and supports high-level service policies to direct traffic through sequences of MBs. 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 [65] 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 [124]. Besides virtualization, middleboxes also play an important role in OpenSDWN for the fine-grained transmission control, which is based in deep-packet inspection [25, 52, 130]. Sekar et al. were one of the first to emphasize the importance of middleboxes, and in their middlebox manifesto [121], 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. [59] 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 instances. Concretely, the southbound interface of OpenNF deals with the network function 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. Merlin [132] is a language to provision network functions and entire network function chains. An interesting NFV platform is ClickOS [90], a virtualized software middlebox platform, based on light virtual machines. OpenSDWN is based on the Click modular router [49, 85].

Home networks have received particular attention over the last years [51, 149]. Users are offered more flexibilities on how their network can be optimized [95, 148], sometimes even over participatory interfaces [51], helping home users to improve performance [120]. Programmable middleboxes can also be exploited to provide a faster ISP service delivery [83].
4.8 Summary

We present OpenSDWN, an Open Software-Defined Wireless Networking framework. With OpenSDWN, we reap the benefits of SDN and NFV for home and enterprise WiFi networks. OpenSDWN implements per-client virtual access points and per-client virtual middleboxes, to render network functions more flexible and support mobility and seamless migration, e.g., migrating firewall state between hotspots when performing a client handover. Moreover, OpenSDWN exploits WiFi data path programmability to enable service differentiation and fine-grained transmission control, facilitating the prioritization of critical applications. Since the user is often left out of scope of this optimizations, we out-sources the control over the network to the user, application, or an Internet Service Provider through a participatory interface.

While constituting an important next step towards our envisioned flexible WiFi architecture, OpenSDWN still relies on a simple presumably centralized control plane. However, several WiFi related function are very latency sensitive and thus, require a tight and bandwidth efficient control loop. For instance, WiFi requires to frequently collect medium and link statistics to adjust the WiFi-specific transmission settings accordingly. Collecting all this information globally creates the risk of overloading the control plane, or of adding too much latency. To address this issue, we next study the effect of a scalable 2-tier control plane architecture where latency sensitive WiFi functions are offloaded to near-sighted controllers in order to reduce the risk of overloading the centralized control plane.
We have already argued in the previous chapters, that WiFi networks can benefit from the concepts of SDN and NFV. However, many WiFi related functions such as transmission rate and power control, interference management [107,128], and mobility management require a tight and bandwidth efficient control loop. In particular, permanently adjusting WiFi transmission settings in a timely manner is paramount due to the non-stationary characteristic of the shared wireless medium. For instance, users are often mobile and associations dynamic. Thus, reducing the risk of overloading the control plane, or of adding too much latency is important for the performance of a WiFi network. In other words, the load on the control plane should be kept at a minimum.

In this chapter, we argue that there is a need for an SDN and NFV interface for WiFi networks that supports fine-grained control of WiFi-specific transmission settings and that scales up to very large hotspot WiFi deployments. Accordingly, we present the design and implementation of AeroFlux, a novel SDWN approach that leverages locality in SDN control plane operations for scalability and responsiveness.

5.1 Motivation

Due to the non-stationary characteristic of the wireless channel, permanently adjusting settings such as transmission rate and power is crucial for the performance of WiFi networks and brings significant benefits in the service quality, e.g., through reducing the packet loss probability. Today’s rate and power control is mainly done on the WiFi device itself. But it is rarely optimized to the application-layer demands and their diverse traffic requirements, e.g., their individual sensitivity to packet loss or jitter. Therefore, if SDN for wireless can provide mechanisms to control the WiFi-specific transmission settings on a per-slice, per-client, and per-flow level, traffic and application-aware optimizations are feasible.

This, however, requires that controllers frequently collect link characteristics and, accordingly, adjust transmission settings in a timely manner. As a reference, the standard rate control mechanism in the Linux kernel adjusts the transmission rate on a wireless link based on transmission success probability statistics every 100 ms. Leaving rate control (and power control accordingly) to a centralized controller comes with a risk of overloading the control plane, or of adding too much latency, while there is limited benefit in maintaining these
Chapter 5 AeroFlux: A scalable Near-Sighted Control Plane

statistics globally. For instance, the coherence time\(^1\) (also a function of the client mobility) can easily exceed the expected time of the successful transmission of multiple data frames [82], rendering optimized control difficult.

5.2 Overview

We propose and present a 2-tiered approach for the design of a wireless SDN control plane. Our design, called AeroFlux, handles frequent, localized events close to where they originate, i.e., close to the data plane, by relying on Near-Sighted Controllers (NSC) [70, 114]. Global events, which require a broad picture of the network’s state, are handled by the Global Controller (GC). More specifically, GC takes care of network functions that require global visibility, such as mobility management and load balancing, whereas NSCs control per-client or per-flow transmission settings such as rate and power based on transmission status feedback information exported by the Access Points (AP), which include the rates for best throughput and best transmission probability. Put differently, we enable the global controller to offload latency-critical or high-load tasks from the tier-1 control plane to the NSCs. This reduces the load on the GC and lowers the latency of critical control plane operations.

As a result, with AeroFlux, we realize a scalable wireless SDN architecture which can support large enterprise and carrier WiFi deployments with low-latency programmatic control of fine-grained WiFi-specific transmission settings. The AeroFlux design introduces a set of new trade-offs and optimization opportunities which allow for advancements in the use of the shared wireless medium, and, as a result, in the user’s quality of experience. For instance, our prototype’s per-flow control allows application-aware service differentiation by prioritizing multimedia streams (Section 5.3). Another key feature of AeroFlux is that it does not require modifications to today’s hardware and works on top of commodity WiFi equipment.

The remainder of this chapter is organized as follows. In Section 5.3, we discuss different use cases highlighting the benefits and requirements of SDN in WiFi networks in general. Next, we explain our architecture (Section 5.4). We then present our prototype in Section 5.5, followed by an evaluation (Section 5.6) of our approach, based on network statistics from our university’s WiFi deployment. Next, we discuss the challenges, extensibility and deployment of our approach in Section 5.7. In Section 5.8, we position our work with respect to the related work. Section 5.9 summarizes our work.

5.3 Benefits and Use Cases

An SDN for WiFi in general and our proposed AeroFlux architecture in particular support new opportunities to algorithmically optimize the flow allocation and network management in a wireless setting, and to differentiate between slices (virtual networks), applications, and flows.

For instance, an SDWN control application can introduce additional (virtual) networks, with specific properties which depend on the network environment and the traffic demands. For example, an additional guest network with low priority transmission settings (e.g., low tx power) can be set up quickly during a conference. A control application may also configure

\(^{1}\)The time over which the radio channel is presumed to be steady.
5.4 Architecture

the network to transmit keyframes of a video stream at high transmission probability settings (e.g., low rate, high power), while the other, less-critical packets can be transmitted using more optimistic rate/power settings, optimized for throughput, as the loss of a non-keyframe does not negatively impact the video quality as much. This can be realized with a supporting middlebox, which adds corresponding tags to the packets of a video stream [54]. Those tags are then used by the WiFi AP to set the transmission rate and power for the packet accordingly. Another example is selective intrusion detection: To detect malicious behavior in a WiFi deployment, an IDS application can selectively monitor IEEE 802.11 frames, such as suspicious CTS frames that block the usage of the medium, from all or only a subset of the APs. The latter is used to detect a CTS jamming attack and to switch off the virtual carrier sensing. Furthermore, an SDN-based WiFi network can benefit from traditional IDS systems such as the Bro Network Security Monitor by routing or selectively duplicating flows for intrusion analysis [152].

AeroFlux is based on a distributed control plane that exploits opportunities for localized control to reduce communication overhead (between the controlled data plane elements and the remote controller), and to enable scalable and low-latency fine-grained control over WiFi transmission settings.

Optimized transmission control: The medium access and transmission control is arguably one of the most important tasks performed by a WiFi network. A fine-grained and fast transmission control is also a natural use case for localized SDN WiFi control. First, an optimized transmission controller needs to collect a lot of statistical data; communicating this data across the wide-area network (e.g., to a controller in the cloud) is inefficient in terms of communication overhead. Second, a fine-grained and efficient control relies on a fast and tight control-loop, which only a near-sighted controller (e.g., in the same network segment) can achieve.

Channel Selection: The set of available channels is usually constrained by a country’s regulations. Specific channels are for indoor or outdoor use only, with transmission power level limitations. Furthermore, some countries require the usage of a dynamic frequency selection in case of a radar detected nearby. This said, channel selection is critical for the network operator in terms of performance and legal operation. Hence, a channel selection application needs to frequently collect information about the channel state at the radio access, e.g., the state of the channel sub-carriers, medium busy states, radar detection and physical errors. To this end, an application should be able to leverage the channel measurement information and define certain requirements such as a set of channels, transmission power limitations, the number of APs that should be managed under one umbrella and traps for notifications and/or warnings. Channel selection does not necessarily depend on a global network view but benefits from a broader view of the wireless environment. However, the GC needs to be notified if clients need to be migrated/handed-off to another AP. In addition, the GC needs to be notified in case of a radar detection, since this could affect the entire network operation.

5.4 Architecture

The design of our AeroFlux architecture is mainly motivated by enabling application- and traffic-aware fine-grained transmission control while ensuring observance of control application constraints (e.g., to comply with 802.11 protocol requirements such as responding to
probe requests within tens of milliseconds. Scalability is paramount to support very large deployments.

To this end, AeroFlux uses a 2-tiered control plane: the global control plane GC and the near-sighted control plane NSC. Fig. 5.1 depicts the high level interactions of the architecture’s building blocks. The GC is logically centralized, e.g., a set of redundant controllers deployed in a data center, whereas the NSCs are located close to where they are needed, e.g., near to the APs. The Radio Agent (RA), a software daemon running on each AP, maintains the AP’s Light Virtual Access Points (LVAPs) [119]. LVAPs keep per-client association and authentication state, and they store per-client OpenFlow and WiFi Datapath Transmission (WDTX) rules. The interfaces are described in more detail in Section 4.6. In the following we describe AeroFlux’s building blocks.

### 5.4.1 AeroFlux Applications

Applications can control the wired and wireless portion of the network through a north-bound API exported by the GC. AeroFlux’s north-bound API provides network wide (1) programmatic control of all forwarding operations, (2) per client state management, (3) statistics reporting, and (4) event notification. Applications running on top of AeroFlux can be reactive and/or proactive. Reactive applications use triggers and callbacks to handle events, which is particularly crucial within WiFi networks due to the non-stationary characteristic of the wireless environment. For example, continuous adjustments of the wireless transmission settings can bring significant improvements to the achieved network performance. Proactive applications typically are driven by timers. A proactive application can control the forwarding or routing within the network, e.g., setup of routes to gateways or to enforce network wide client isolation through forwarding policies.
Since some WiFi control applications need to deal with latency-critical aspects of the IEEE 802.11 protocol, e.g., probe response frames must be answered within tens of milliseconds. Applications can specify such control plane requirements in the form of constraints. Specifically, control application constraints allow applications to specify requirements such that the GC knows which apps are needed to be handled by NSCs. Furthermore, the GC will use these constraints to decide, where in the control plane (on the GC itself, or on an NSC) to run the application.

5.4.2 Global Controller (GC)

The global controller handles events which are not time-critical [70] or events belonging to inherently global tasks [114]. Examples include authentication, wide-area mobility management, global policy verification (including loop-free forwarding sets), client load-balancing, and applications for intrusion detection or network monitoring. In addition, the global controller is best suited to manage MBs (firewalls, etc.), including their instantiation and the steering of flows for MB traversal. The GC also instantiates and controls the NSCs, e.g., the GC offloads selected control plane functions to the NSCs whilst monitoring their load.

5.4.3 Near-Sighted Controller (NSC)

The NSCs are located close to the APs. They handle time-critical and/or load-intensive control plane tasks, which do not require a global network view. An NSC can update WiFi datapath entries, collect link characteristics, and adjust WiFi transmission settings at a high frequency and in a timely manner. Essentially all functions exported by the Radio Agent on the APs (see below) can be used by control plane applications running on the NSCs.

The NSCs also implement an interface for MBs life-cycle management (delegated by the GC). To keep up with the dynamic demand for network functions, the NSCs can spawn, migrate, and remove MBs, in case the MB functionality is provided only through software, e.g., by a virtual machine running on a Linux-based hypervisor. We can safely assume that sooner rather than later, most network functions in enterprise and operator networks are provided by software running on standard server hardware, as envisioned by Network Function Virtualization (NFV).

This is essentially crucial in the presence of client mobility or per-flow application- and traffic-aware optimization, where MB state needs to be migrated or where MBs need to perform a deep packet inspection (DPI) for flow classification. Furthermore, per-flow application- and traffic-aware optimizations require a low latency control channel between the NSCs and WiFi APs for timely adjustments of the transmission settings to the channel conditions, due to the non stationary characteristic of the wireless channel.

5.4.4 Radio Agent (RA)

Each AP hosts a Radio Agent, which, through implementing knobs for controlling WiFi-specific transmission settings and for gathering wireless related measurement data such as link and channel statistics in a configurable interval (see Fig. ?? to the control plane, e.g., adjusting per-flow transmission power level or export channel utilization. This is necessary as the transmission rate depends on the environment and the coherence time. In case of
client mobility, frequent transmission rate adjustments are more likely than for static WiFi clients, where the coherence time is low. An SDN for WiFi needs to provide this interface in order to allow applications to quickly adapt to changes of the channel characteristics.

Also, the RA hosts Light Virtual Access Points (LVAPs, discussed next), which are virtual, per-client APs, that simplify the handling of client associations, authentication, handovers and slicing. For example, LVAPs maintain per-client WiFi transmission settings (WDTX entries, discussed below). All functionality provided by the LVAPs and WDTX entries is programmatically made available to the NSCs by the RAs.

In addition, RAs provide an interface for network applications to request to receive selected IEEE 802.11 frames. Specifically, RAs can perform matching on incoming frames to support a publish-subscribe like system wherein network applications can subscribe to per-frame events, e.g., a selective wireless IDS/IPS can subscribe to CTS frame that reserve the medium for a long duration.

5.4.5 Light Virtual Access Point (LVAP)

AeroFlux relies on Light Virtual Access Points as described in Section 3.4.2. In a nutshell, an LVAP is by default a per-client virtual AP that keeps all network state belonging to that client. In AeroFlux, LVAPs enable programmatic control of client associations, authentication, handovers. An LVAP comprises the following information: a unique BSSID\(^2\), one or more SSIDs\(^3\), the client’s IP address, a set of OpenFlow rules and IEEE 802.11 specific transmission settings (WDTX rules, described below). Furthermore, an LVAP can be configured to serve multiple WiFi clients by a single BSSID and a common set of TX settings. Also, LVAPs can span across multiple physical APs (in a single BSSID domain) to serve a larger number of WiFi clients, for example to support shared high-volume broadcast or multicast traffic.

LVAPs are instantiated by the GC and assigned to RAs. The GC migrates LVAPs from one RA to another (i.e., from one physical AP to another), to support seamless client hand-overs for client mobility or client load-balancing.

5.4.6 WiFi Datapath Transmission (WDTX) Rule

The WiFi datapath transmission (WDTX) rules extend OpenFlow rules by defining per-flow transmission settings, i.e., a WDTX entry can match one or more OF rules. WDTXs are kept on a WiFi AP and are linked to OpenFlow flow table entries. Through the AF protocol, NSCs (and the GC) manage the WDTX entries on an AP. Currently a WDTX entry allows to control the following 802.11-specific settings: transmission rate, number of retries, transmission power, and the usage of ACKs and RTS/CTS. WDTX rules can be extended to enable flow-based control over the transmission chain or the antenna used for the transmissions.

---

\(^2\)The BSSID specifies the MAC address of the (virtual) WiFi AP.

\(^3\)The SSID defines the name of the network.
5.5 Prototype

Our early AeroFlux prototype is based on our existing frameworks (cf. Chapter 3 and Chapter 4). In this section, we briefly discuss how we implemented the different components of AeroFlux.

5.5.1 Global Controller

The GC is based on the Odin controller, which in turn is based on the open source Floodlight controller. By relying on Odin, AeroFlux inherits an implementation of the LVAP concept. Currently, since we rely on Floodlight, we do not support a distributed GC control plane.

The GC currently implements three southbound interfaces, the OpenFlow protocol for the wired, a separate datapath control interface for the wireless portion of the network, and an interface for middlebox management as described in Section 4.6.1. Odin introduced a separate southbound interface for the WiFi specific transmission settings, since accommodating that extra functionality into the OpenFlow protocol does neither yield any specific advantages nor does it simplify the prototyping.

5.5.2 Near-Sighted Controller

The NSC and its interactions with the RA are similar to the interactions between the OpenSDWN controller and the RA as described in Section 4.6.1. The NSC manage pools of LVAPs that belong to a network slice. The communication interface between GC and NSC is currently under development. A basic stand-alone variant of the NSC exists as proof-of-concept based on the RYU controller, written in Python. The goal of the prototype is to investigate the feasibility and performance of RA’s components. The overhead of the WDTX control for non-meta transmission rates is discussed in 5.6.

5.5.3 Radio Agent

In the AeroFlux architecture the radio agent is realized as a separate daemon which can proxy Odin messages to our Click-based Odin agent. The Radio Agent module for handling of WDTX rules is realized as a pure C module for simpler integration with Odin’s Radio Agent and existing access point management software such as the hostapd.

5.5.4 WDTX rules

The WDTX rules are implemented within the mac80211 framework of the Linux Kernel to benefit from the WiFi driver abstraction and the Linux rate control algorithm (minstrel and minstrelHT) for best effort traffic. WDTX rules can overwrite the per-flow physical transmission rate ($TR_n$), power ($TP_n$) and retry count ($RC_n$). Assigning fixed and/or meta transmission settings is possible. Meta transmission setting include best probability rate, best throughput rate, second best throughput rate or basic rate. Based on the capabilities of the WiFi NIC, the transmission settings can be set for the device 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 minimum probability or maximum transmission time to WDTX rules. WDTX rules are bound to OF rules (see Section 4.6.1) by adding a tag to all packets that belong to an OF flow at the ingress port. A OF rule can tag packets which are passed through the Linux kernel down to the WiFi driver.

5.6 Evaluation

In this section, we evaluate the scalability aspects of AeroFlux in the context of an actually deployed network (the campus network of TU Berlin). We consider a power and rate control application: This application requires frequent communication (statistics requests) with the RA and does not rely on a global view of the network state; it is hence an ideal candidate for a more near-sighted control.

In the following, when estimating the control plane load, we will only take into account statistics requests. This yields a lower bound on the actual control plane overhead and hence benefit of the near-sighted control. The frequency at which WDTX entries are updated depends on the implemented rate and power control algorithm. Analogous to the Linux default rate control implementation, our application collects up-to-date per-client statistics once every 100 ms. A client’s statistics status update is 2.3kB in size on average. This takes dual stream 802.11n clients into account, with their large number of supported transmission rates.

Our evaluation is based on network statistics from the university WiFi deployment, serving up to 9,000 clients every day. This network covers a large campus as shown in Fig. 5.2. The circles show with their diameter how many APs are deployed in one building, and a circle’s color shows how many clients (darker = more) are present in that building. We observe six popular areas with more than 400 clients, two of which with more than a thousand clients. Fig. 5.3(a) depicts the fraction of clients served by the locations. We observe, that 60 percent of the clients are served by less than 10 hotspot locations. Figure 5.3(b) shows the distribution of clients and APs among four commercial controller (C1 to C4). The APs are load balanced between the controller. The CDF indicates a non-optimal AP to controller
5.6 Evaluation

![Graphs showing distribution of clients across locations and percentage of clients handled by each controller.](image)

(a) Distribution of clients across locations. (b) Percentage of clients handled each Controller (C1 to C4).

(c) Reduction of GC load with NSCs.

Figure 5.3 – Evaluation of AeroFlux under consideration of an existing University campus WiFi deployment.

Load balancing. Specifically, the load difference is up to 15 percent between the controller C2 and C4. Moreover, we observe that roughly 60 percent of the clients are served by less than a third of the WiFi APs.

Next, we study the reduction in statistics request load on the GC when incrementally deploying NSCs in Fig. 5.3(c). We deploy NSCs according to a straightforward, greedy approach: The location of a next NSC is chosen to be the most popular NSC-less location, i.e., the location with the greatest number of associated clients that is not yet served by an NSC. Alternative algorithms performing more rigorous optimizations (see Sec. 5.7.2 for a short discussion) will only improve the benefits of the near-sighted control further.

Fig. 5.3(c) shows that with zero NSCs deployed, the GC is responsible for performing rate and power control for all clients, thus it needs to collect all statistics. This results in almost 1.6 GBit/s traffic for the statistics alone. Deploying NSCs helps: With an NSC at the eight most popular locations of the university campus, the load on the GC is halved to about 800 MBit/s, while the average load on the NSCs is only 111 MBit/s. With our greedy deployment strategy and with only very few (one or two) NSCs deployed in the network, the average NSC load is below 200 MBit/s (easy to handle) while the benefit in terms of reduced GC load is already significant. We discuss more optimized deployment strategies in Sec. 5.7.2.
Chapter 5 AeroFlux: A scalable Near-Sighted Control Plane

5.7 Discussion

5.7.1 Challenges and Extensibility

NSC failure: One way to overcome a NSC failure is to increase the redundancy of the NSC infrastructure by assigning backup NSCs to the RAs. Another is, to spawn a new NSC and to re-install all state from the corresponding LVAPs. We favor the former approach, since the latter would result in a downtime until the new NSC is fully operational. Both approaches work fine with AeroFlux, since the GC keeps a backup of all active LVAP.

Extensibility: LVAPs are extensible and can support other MAC or physical access schemes such as the 802.11 Ad-Hoc mode or even TDMA-based approaches, e.g., LVAPs can be extended to host MAClets [39] code. In addition, with OpenRadio [32], our system could benefit from a clean-slate programmable network dataplane.

Frame aggregation and reordering: Setting transmission settings on a per-flow level creates challenges with 802.11 frame aggregation, a key feature of 802.11n for achieving higher link utilization by aggregating queued frames into a single IEEE 802.11 frame. The critical aspect with WDTX is that an aggregate uses the same transmission settings for all frames in an entire frame. Hence, our prototype can overwrite the default aggregation strategy to either transmit all frames individually or to assign a aggregation class to the flows, i.e., to allow aggregation for all frames that belong to the same transmission class. We are currently exploring what the right design points are for more efficient aggregation strategies.

5.7.2 Deployment

The question of how many and where to deploy NSCs is an interesting algorithmic problem. For our simulations and as a simple lower bound for the benefit of the near-sighted control, we have considered a greedy heuristic. While a detailed theoretical study of the problem is beyond the scope of this paper, we observe that the problem is related to the Facility Location Problem (FLP): a solution to the FLP problem jointly optimizes the number of opened facilities (i.e., the NSCs) as well as the distance of the customers (i.e., WiFi users) to their closest facility. While the problem is computationally intractable in general, there exist efficient approximation algorithms for the metrical variant. Currently, the best known algorithm is by Li [87] (approximation ratio 1.488).

5.8 Related Work

While SDN has been discussed intensively in the context of wired networks (such as datacenter networks, campus networks, or WANs), much less is known today about how to introduce SDN in wireless networks.

There have recently been several studies on innovation opportunities in cellular networks. C-RAN [46] (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 [80] simplifies the operation of cellular networks and supports high-level service policies to direct traffic through sequences...
of middleboxes. 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 offers a number of benefits for mobile networks, including wireless
access segments, mobile backhaul networks, and core networks. SoftRAN [65] 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.
OpenRadio [32] proposes a novel design for a programmable wireless data plane that provides
modular and declarative programming interfaces across the entire wireless stack.

We are not the first to observe the advantages of a more distributed SDN control, and
scalable SDN solutions have been discussed for many years already in the context of wired
networks. For example, DIFANE [150] and DevoFlow [50] extend data plane mechanisms
to reduce the load towards the controller. This is orthogonal to our approach, as instead of
extending the switches, we move control plane functions closer to switches. There also exist
many approaches to distribute the control plane while maintaining logically centralized,
eventually consistent network state. [42, 139] However, they assume that all applications
require the network-wide state, and cannot be used for local control applications.

There are also studies on near-sighted controllers. In their seminal work, [70], Yeganeh
and Ganjali propose a 2-layer control architecture where the bottom layer is a group of
controllers with no interconnection, and no knowledge of the network-wide state, and where
the top layer is a logically centralized controller that maintains the network-wide state. Our
work is inspired by Kandoo [70], but tailored towards the specific use case of WiFi networks.
In [71], Heller et al. study the problem of how many controllers should be placed where, in
order to achieve a sufficiently fine-grained control. Their algorithm is evaluated on different
WAN topologies. Finally, Schmid and Suomela [114] classify different applications according
to their locality (“nearsightedness”), and show fundamental tradeoffs between locality and
optimality.

We are not aware of any work on near-sighted control approaches in wireless networks. This
is surprising, given the naturally fine-grained control needed for optimized transmissions.

5.9 Summary

With AeroFlux, we propose and present a 2-tiered approach for the design of a scalable
wireless SDN control plane targeting large scale WiFi deployments. AeroFlux handles fre-
quent, localized events close to where they originate, i.e., close to the data plane, by relying
on Near-Sighted Controllers. Global events, which require a broad picture of the network’s
state, are handled by the Global Controller (GC). More specifically, GC takes care of network
functions that require global visibility, such as mobility management and load balancing,
whereas NSCs control per-client or per-flow transmission settings such as rate and power
based on transmission status feedback information exported by the Access Points (AP). Put
differently, we enable the global controller to offload latency-critical or high-load tasks from
the tier-1 control plane to the NSCs. This reduces the load on the GC and lowers the latency of
critical control plane operations. As a result, with AeroFlux, we realize a scalable wireless
SDN architecture which can support large enterprise and carrier WiFi deployments with
low-latency programmatic control of fine-grained WiFi-specific transmission settings. Another key feature of AeroFlux is that it does not require modifications to today’s hardware and works on top of commodity WiFi equipment.

However, while the aforementioned architectures are important components of a future-proof WiFi architecture, there are additional opportunities to render the deployment of WiFi networks even more flexible. In particular, WiFi networks today provide a wide range of functionality related to performance and security. Today, these functionalities are often integrated and located at a single location. Given the different requirements and characteristics of these functions, this is suboptimal. A functional decomposition of the WiFi building blocks can aid to improve the overall networks performance. Thus, we study next an approach of placing WiFi network functions in a more flexible manner, i.e., enabling a faster and cheaper service deployment.
LegoFi the WiFi Building Blocks!

In the previous chapters, we have already shown that WiFi networks can benefit from the concepts of SDN and NFV to realize an important step towards a future-proof WiFi architecture. However, we in this chapter identify additional opportunities for rendering the deployment of WiFi networks even more flexible. Today, WiFi consists of a large number of control and data plane network functions that are still integrated and located at a single location, e.g., tied to the control plane or WiFi AP by following the Split-MAC architecture. Given the different requirements and characteristics of these functions, this is suboptimal. A functional decomposition of the WiFi building blocks can aid to render WiFi more flexible and to increase the overall network performance.

In this chapter, we present LegoFi: a functional decomposition of the WiFi architecture. With LegoFi individual (virtualized and programmable) WiFi function blocks are allocated where (and when) they are most useful. Thus functions that require low latency such as transmission rate control can be allocated on the Access Point, while functions that can benefit from having broader visibility across the local wireless network (e.g., association management, duplicate frame filtering and transmit power control) can be allocated onto a virtualized server platform or programmable network equipment further back in the network. Functions that have loose latency constraints but need to have visibility across deployments (e.g., authentication management) can be moved into an operator data center. This allocation can also change, e.g., during a failover or a scale-out. Specifically, we present a rough functional decomposition and explore how LegoFi can benefit four different deployment scenarios, namely home, operator hotspots, fibre to the building (FTTB), and enterprise. Coupled with an orchestration system LegoFi can also serve as the basis for rapid and flexible operator service innovation.

6.1 Motivation

The increasing demand and requirements for WiFi conflict with today’s WiFi architecture, and suggest an architecture oriented towards more flexible management and deployment. Today’s WiFi is known for its notorious inefficiencies especially in dense environments where collisions at all APs are frequent [38]. In particular, the current WiFi architecture ties wireless network functions either to a single Wireless LAN Controller or to a single Access Point (AP). In the first case, a controller typically manages a set of APs through a standardized [99,100] or proprietary [102] control protocol. However, today’s standardized solutions are inflexible and limited [19] (and have hardly been implemented (cf. [19])), and proprietary solutions are expensive.
Indeed, only a few vendors [3, 137] implement part of the standard with the CAPWAP tunneling protocol [99] being the most frequently implemented part. Additionally the CAPWAP protocol has not been updated to include features introduced into the 802.11 standard, such as MIMO, since the CAPWAP protocol was standardized. While we agree that the standardization of the radio link protocol is necessary to ensure interoperability between devices and access points, we believe that in order to support faster innovation, more flexible service deployment, and cheaper evolution of the WiFi architecture, the network functions should be modular and open.

We argue that an architecture based on this modular approach is also more likely to address the increasingly diverse nature of WiFi network deployment scenarios of Figure 6.1. Indeed, different WiFi deployments often come with different requirements for placing wireless network functions. However, today’s centralized, monolithic control however do not properly reflect the various requirements of the individual network functions. The architecture restricts how wireless network functions can be placed, so a suboptimal deployment is likely. Accordingly, we make the case for decomposing, virtualizing, distributing and chaining [69, 77, 88, 129, 132] of the WiFi network functions in a flexible manner: the WiFi functions are located where they are most useful or efficient.

Today’s choices for deployment consist of the AP and the Cloud. Indeed, most deployments today either follow the Split MAC architecture (e.g., in the enterprise context) with WiFi network functions distributed between a centralized Cloud- or LAN-based AC and Thin APs, or the Full MAC variant (e.g., in the home context) with no AC and autonomous APs and the management functions sometimes centralized in a Cloud- or server-based management system.

The AP can provide low latency, which is attractive for fine-grained transmission control or for reacting quickly to packet losses. Deployment in a data center in the wired network, generically in the Cloud, imposes a significantly higher latency and results in higher communication costs. On the other hand, the Cloud can be more secure than a physically accessible AP, and can give the network operator a single point where they can more easily manage many different WiFi deployments.

But a third possibility has opened recently with operator fibre deployments. The advent of Fibre-To-The-X (FTTX), namely to the Building (FTTB) or distribution-point (FTTdp), offers a location close to the AP but far enough back in the network so that multiple APs can be served by the same network function: a pico data center consisting of a virtualized server in the basement of a building. This option is attractive, as it increases the operator’s ability to manage the network, like a Cloud deployment, while keeping latencies low, like an AP deployment.

For example, moving the association function onto a virtualized platform topologically near the AP allows device associations to be managed across multiple APs. Such a functionality placement can support seamless mobility, but can also be an attractive means to improve throughput: when an association function can handle a collection of APs, clients can be balanced across APs and channels in an informed and strategic manner, e.g., based on RSSI values or supported feature set such as 802.11n greenfield mode. In contrast, locating the association function on a single AP as is traditionally done restricts the ability to flexibly handle traffic, while moving it into the Cloud could potentially increase association latency in an unacceptable manner. In a virtualized environment, device allocations to APs may even be dynamic. This can potentially increase fault-tolerance and scalability, by allowing
for automatic failover if an AP fails or by graceful autoscale up and down of the network function depending on traffic load.

Similarly, a modular WiFi architecture is also better prepared to leverage the trend towards programmable switches [9,98] and Fibre-To-The-Cabinet, since these offer additional deployment points for WiFi network functions. LegoFi is similar in spirit to recent trends in cellular networks [46,65,80] and inspired by Network Functions Virtualization [59,90,121,132].

6.2 Overview

This Chapter makes the case for the LegoFi architecture, in which the WiFi control and data planes are decomposed into their network functions which can be virtualized and flexibly deployed and chained according to the needs of the deployment scenario. We also introduce the concept of Light Virtual Network Function, or LVNF, a programming abstraction allowing network programmers to implement complex services by composing several elementary packet processing blocks, i.e., the LVNFs, into a more complex packet processing sequence. We illustrate the advantages of LegoFi over the existing WiFi architecture for supporting four use cases (see Figure 6.1): home deployments, operator hotspots, fibre to the building/curb, and enterprise deployments. LegoFi is particularly interesting in open, programmable and virtualized networks, where allocations can become dynamic, and automatic failover or scale-out are supported.

6.3 State of the Art WiFi

Today’s WiFi architecture can roughly be divided into three main planes:

1. Data Plane (the data path),
2. Control Plane, and

The management plane functions are in charge of centralized, non-real-time administrative tasks, such as provisioning APs, AP firmware update, fault surveillance and security monitoring. The control plane consists of functions involved in sending and receiving 802.11 frames which set up and maintain the wireless link and which control the transfer of data packets between the device and the AP and share the wireless link with other devices [20]. The data plane consists of the devices which implement the wired or wireless data path. For instance, the WiFi data path provides the necessary functions to actually transmit and receive IEEE 802.11 frames. In case of WiFi, the data plane carries IEEE 802.11 encapsulated frames though sometimes encapsulated in a tunnel, e.g., if a centralized controller is in place, or transferred to an 802.3 frame and tagged with an 802.1Q VLAN tag, if the AP is deployed standalone.
6.3.1 Centralized CAPWAP Control Architecture

The IETF CAPWAP standard [144] architecture for centralized control of WiFi networks specifies two components: a centralized Access Controller (AC) and Wireless Termination Points (WTPs). CAPWAP defines three architectural variants: Local (or Full) MAC, Split MAC and Remote MAC. The Local (or Full) MAC variant places the entire 802.11 MAC on the access points, and only places the network management and provisioning on the AC. In the Local MAC variant, the WTPs have same functional composition as an 802.11 AP as defined by the 802.11 standard. The Split MAC variant places the MAC functions with realtime constraints on the WTP but allows the decoupling of the non-realtime functions from the WTP, which are called Thin APs. For instance, authentication can be moved to the AC. The Remote MAC variant moves all the MAC functions, including the realtime functions, to the AC, but this approach has seen little deployment. RFC 4118 divides the functions into those that are specified by IEEE 802.11 and those that are network management functions (called "CAPWAP functions" in the RFC).

6.3.2 Proprietary WiFi protocols

CAPWAP is the only standardized protocol to control and provision WiFi networks today. However, CAPWAP is hardly implemented. Instead, vendors have continued implementing their proprietary solutions. The main reasons given for not implementing CAPWAP are its inflexibility and limitations. According to Aruba [19], any vendor moving to CAPWAP for its current offering would find significantly less functionality available. CAPWAP’s least common denominator approach to AP control, does and cannot well support the different architectures, features and required pace of new development. CAPWAP does not take into account any of the recent extensions (802.11ac, 802.11n, and 802.11k).

6.3.3 Open and Programmable WiFi

With a more open architecture, network functions for supporting new radio features can be added without the need for modifying a protocol standard, resulting in faster and less expensive solutions. While there has been a lot of interest in overcoming network ossification by opening wired computer networks and even cellular networks and in making them more virtualized and programmable, there are few initiatives in the context of WiFi networks. Some examples are the LVAP abstraction of Odin [119] and programmable datapath architectures promoted by MAClets [39], OpenRadio [32] and OpenSWDN [115]. LegoFi goes beyond these efforts in that it advocates modularizing and virtualizing all the WiFi control and data functions so they can be deployed where they will best serve the needs of the overall deployment requirements.

6.4 The Case for LegoFi

WiFi networks rely upon multiple network functions, henceforth called building blocks. Example functions are handling client associations, authentication, frame deduplication, traffic encryption/decryption, or transmission control. We argue for a more flexible and modular WiFi architecture, where individual function blocks can be placed at the location where they
6.4 The Case for LegoFi

Figure 6.1 – Upcoming network architectures offer a new unique deployment opportunity for WiFi functions, e.g., a pico datacenter located the basement in case of fiber to the building (FTTB). LegoFi targets a flexible deployment of WiFi functions such as encryption/decryption or duplication filtering of 802.11 frames. LegoFi is interesting in open, programmable and virtualized networks, where allocations can be dynamic, and automatic failover or scale-out are supported. The figure shows four use-cases where the advantages of LegoFi are illustrated, namely: home or residential networks possibly with fibre to the building/curb; operator hotspots which provide WiFi services in public spaces such as along a shopping street, in a mall or at a coffee shop; and enterprise deployments in which the WiFi access network is connected to a routed and managed enterprise network.

are most useful in terms of performance, visibility, or security. LegoFi aims to translate the aforementioned features into Lego-like building blocks, in order to realize flexible WiFi deployments where building blocks can be located (and possibly service-chained) anywhere where virtualized compute/network resources are available, depending on timing constraints and other deployment considerations. Figure 6.1 contains a high level diagram showing how LegoFi applies to the four deployment scenarios. The small Lego-bricks in the figure indicate virtualized WiFi network functions. For instance, on the network equipment such as a programmable switch, like a programmable DSLAM, Open-Flow switch or the like.

It is well known that the different functional building blocks come with different requirements: some are realtime critical (e.g., ACK or RTS/CTS generation) while others are latency sensitive (e.g., fine-grained transmission control or responding to probe messages), and should hence be allocated close to the user, e.g., on the AP. Other functions, however, can benefit from an increased visibility, and being able to optimize network functions across multiple APs (e.g., applications supporting mobility). Finally, security critical applications resp. applications requiring global visibility (e.g., endpoints of RADIUS tunnels) should be located in a safe building (of the operator cloud).

Next, we discuss some use cases in more detail.

6.4.1 Deduplication

We believe that dense WiFi-based deployments, with many APs operating on the same channel, offer a unique optimization opportunity for LegoFi. Consider for example a football stadium equipped with a large number of Access Points that provide Internet connectivity to ten thousands of spectators (cf. Figure 6.2(a)). Similarly dense WiFi-based deployments, with many APs operating on the same channel, can also be found, for example, in cities or airports.

It is well-known that in such dense scenarios, the collision probability is high and the performance suboptimal accordingly [38]. Thus, due to the unreliable nature of the wireless
medium, today’s WiFi usually implements Layer-2 reliability through a two way handshake for each unicast data transmission. More specifically, each successfully received unicast data frame is normally acknowledged by the transmitter. However, if the transmission of an WiFi ACK frame fails, the sender will retry, which can cause duplicate frames at the destination. Today’s WiFi access points filter out duplicate frames in order to prevent duplicate frames from being received by the destination, potentially causing misinterpretations in TCP.

With LegoFi, we may exploit capture effects occurring in dense settings: as the collision probability is likely to vary across access points (e.g., due to different signal strengths), several Access Points may successfully “overhear” packets, and could forward them to a nearby, centralized network function performing deduplication.

In other words, with a LegoFied deduplication, we can dramatically increase the transmission probability. Hence, by consolidating this filtering of duplicate frames to a more centralized point in the network, and composing a couple of APs in a hearing domain, we can increase the receive probability and ensure a reduced RTT on the uplink. In case of group related traffic such as broadcast or multicast, which lacks a Layer-2 reliability mechanism, this can also significantly increase the receive probability.

LegoFi’s deduplication building block could be placed on the AP itself, behind a group of APs, e.g., on the connecting switch 6.2(a) or a centralized controller. Note, the trend towards programmable switches [9,98] offer a unique opportunity deploying software components on switch the hardware itself.

### 6.4.2 Load Balancing

An interesting application arises in the context of a more centralized client association management. By balancing clients across APs, throughput or airtime fairness among clients could be significantly increased.
Today’s WiFi networks use IEEE 802.11 management frames to handle associations, authentication and discovery. In IEEE 802.11-based networks, clients need to associate with a AP before sending data frames. The association process begins with the discovery phase, where a client either actively scans for APs by generating probe requests, or passively learns about APs through beacon frames generated by the latter. During an active scan, APs that respond with probe response messages become candidates for the client to associate with. The client then decides which AP to associate with via a locally made choice. Moreover, probe response frames are used by WiFi APs to signal capability information to clients, and vice versa.

Today, probe frames are typically handled by the AP itself. In enterprise networks, probe frames are usually handled directly on the AP while a copy can be forwarded to the centralized controller in order to build a global hearing map. However, we note that handling of probe frames underlies time constrains. Specifically, during an active scan a client will send a probe request frame on a channel and then waits for probe responses from APs on that channel for a short period of time, before switching to the next channel.

While responding to probe frames directly on the AP may provide the best latency, we argue that it can be worthwhile to detach this functionality from the AP and move it further back into the network, making it possible to influence the association decisions of clients, through near-sighted load-balancing. LegoFi provides a probe responder building block which can be coupled with the authentication building block to manage client associations in accordance with a hearing map across APs. The building block can be placed behind a group of APs, e.g., on the connecting switch (see Figure 6.3(a)). Thus, to provide a timely feedback loop, we coordinate the responses across access points, and hide APs from clients when performing an active scan. The association decision can be influenced through blacklisting of clients at particular APs, before the client performs the association.

A simple controller less client-based load balancing can be realized through a probabilistic probe responder. Clients can also be balanced among access points by controlling the visibility of APs to the clients together with blacklisting on the APs. This function changes the probability of responding to probe messages based on the client to AP ratio at neighboring APs. To this end, the function receives statistics periodically via the centralized controller. Note, this works as an alternative to the aforementioned function in case of high latency uplinks, as indicated in Figure 6.2(b), where authentication and association frames
can’t be handled by a centralized controller. Thus, this function can be realized almost controller-less.

6.4.3 En/Decryption Mobility Domain

AAA functions are not as time-critical but highly security-critical. These functions today restrict client mobility: most traffic is en/decrypted directly at the AP or forwarded to the centralized controller. We see a potential for consolidating this functionality across a number of access points as shown in Figure 6.3(b). Thus, mobility domains are enlarged, and we believe that decryption of WiFi data frames in a trusted environment can also aid to improve the security of the system. Especially, in dense urban deployments, we can leverage the WiFi AP functionality of a neighboring AP with a better reception rate. For instance, a pico datacenter in the realm of the ISP, e.g., in the basement of a building can easily overtake this functionality.

6.4.4 ePDG

The Evolved Packet Data Gateway (ePDG) is a crucial component of the Evolved Packet Core (EPC), specified for the upcoming mobile networks. The ePDG is responsible for interworking between the EPC and untrusted non-3GPP networks that require secure access, such as a WiFi, LTE metro, and femtocell access networks. The ePDG terminates and manages subscriber-initiated IPsec/IKEv2 tunnels which are used to secure traffic over an untrusted network. Moreover, WiFi calling is closely aligned with VoLTE, known as Voice over LTE. Operators worldwide are currently launching VoLTE which replaces the current calling mechanism. In order to handoff calls between LTE and WiFi, traffic from the latter needs to be routed to the IMS through the Evolved Packet Core (EPC), which results in a scalable deployment opportunity for network operators.

Figure 6.4(a) demonstrates the integration of the ePDG as close to the customer in a FTTB setup for more flexibility. Since the ePDG stores the tunnel key material, even the internet connection over an open WiFi hotspot can be secured. Moreover, through the flexible placement of the ePDG along with the composition of multiple surrounding APs can be used for WiFi calling. Thus, WiFi calling can be used as a complement to VoLTE, allowing operators to reduce stress on their cellular networks.

6.4.5 Wireless Intrusion Prevention

A Wireless Intrusion Prevention System (WIPS) is a typical building block of enterprise WiFi deployments that monitors the radio spectrum for the presence of unauthorized APs, e.g., known as rogue APs, spectrum jammers, or intrusion detection. WIPS usually take countermeasures automatically. WIPS are typically implemented by the WLC and are not designed as a network function that can be deployed in a flexible manner. This, however, raises scalability concerns for upcoming cloud based WiFi deployments based on cheap off-the-shelf APs, as found on today’s user’s premises.

In order to provide enterprise like WIPS over the Internet for residential and hotspot WiFi deployments, the WIPS needs to be decoupled from the WLC and modularized, allowing for more flexible deployment strategies, e.g., located close to a group of hotspot APs. Moreover,
this also enables a more flexible scaling in enterprise networks, since this feature can be placed in the dataplane, similarly to today’s intrusion detection and prevention systems.

By consolidating this functionality for a number of access points, standard security features can be applied to today’s residential and hotspot networks and scaled flexibility in enterprise deployments (see Figure 6.4(b)). Moreover, consider the case in Figure 6.2(b), offloading wireless network functions to an off-site can dramatically reduce the amount of control traffic at the centralized control plane.

6.5 The LegoFi Architecture

We first present a rough breakdown of the WiFi architecture, then describe our design abstractions, and finally discuss the chaining of network functions.

6.5.1 Architectural Breakdown

Table 6.1 shows a decomposition of the WiFi architecture into network function building blocks that are candidates for flexible deployment. Due to space considerations, not all possible network functions are included, and some of those included represent aggregations of functions that, in a more detailed analysis, could be broken down further. Different functional building blocks come with different requirements. These are indicated by the columns Latency and Bandwidth. The Placement column indicates where the candidate function can be placed, based on its latency and bandwidth constraints. AP indicates that the function must run on the AP with HW further indicating that it will most likely be implemented in hardware. TFL, for Tight Feedback Loop, indicates that the function can run either on the AP or potentially further back in the network but topologically close to the AP, like on a pico datacenter (virtualized rack or server in the basement). LFL, for Loose Feedback Loop, indicates that the function can run at an operator data center or other location topologically distant from the AP. Finally, Datapath (DP) indicates that the function can run on a programmable switch, like a programmable DSLAM, OpenFlow switch or the like.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Capability</th>
<th>Visibility</th>
<th>Placement</th>
<th>Plane</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radio Resource Management</td>
<td>Channel Selection</td>
<td>medium</td>
<td>group</td>
<td>LFL</td>
</tr>
<tr>
<td></td>
<td>Co-channel and RF Interference</td>
<td>medium</td>
<td>group</td>
<td>LFL</td>
</tr>
<tr>
<td></td>
<td>Transmit Power Control</td>
<td>low</td>
<td>low</td>
<td>TFL</td>
</tr>
<tr>
<td></td>
<td>Transmit Rate Control</td>
<td>low</td>
<td>low</td>
<td>TFL</td>
</tr>
<tr>
<td>Client Association</td>
<td>Beaconing</td>
<td>realtime</td>
<td>direct</td>
<td>AP+HW</td>
</tr>
<tr>
<td></td>
<td>Probe Handling</td>
<td>realtime-low</td>
<td>low</td>
<td>TFL+C</td>
</tr>
<tr>
<td></td>
<td>Association Handling</td>
<td>medium</td>
<td>low</td>
<td>all</td>
</tr>
<tr>
<td></td>
<td>Probe Handling</td>
<td>medium</td>
<td>low</td>
<td>all</td>
</tr>
<tr>
<td></td>
<td>Association Handling</td>
<td>medium</td>
<td>low</td>
<td>all</td>
</tr>
<tr>
<td></td>
<td>Association Handling</td>
<td>medium</td>
<td>low</td>
<td>all</td>
</tr>
<tr>
<td></td>
<td>Transmission Frame Control</td>
<td>low</td>
<td>low</td>
<td>TFL</td>
</tr>
<tr>
<td></td>
<td>Channel Selection</td>
<td>medium</td>
<td>low</td>
<td>TFL</td>
</tr>
<tr>
<td></td>
<td>Channel Selection</td>
<td>medium</td>
<td>low</td>
<td>TFL</td>
</tr>
<tr>
<td></td>
<td>Channel Selection</td>
<td>medium</td>
<td>low</td>
<td>TFL</td>
</tr>
<tr>
<td></td>
<td>Prevention</td>
<td>medium</td>
<td>Visibility</td>
<td>L4</td>
</tr>
</tbody>
</table>
Control frame generation functions such as ACK or RTS/CTS generation need to be deployed on the AP, preferably in hardware, because they must obey realtime constraints specified by the 802.11 standard. Some other functions, such as beacon generation, also need to be deployed on the AP because they involve specific timing constraints. Other functions are latency sensitive, for example probe and association control, but not sensitive enough that they have realtime constraints. These functions can be deployed further back in the network, giving them more visibility over multiple access points, for example for better load balancing in the case of association control. Radio resource management functions such as channel selection and security functions such as 802.1x don’t need such tight coordination with the radio layer but do require global visibility across multiple access points and potentially multiple deployments, including APs in a neighboring building in the case of radio resource management, so they can be allocated into operator data centers. Finally, data plane functions such as duplicate filtering can be allocated onto programmable data plane elements close to the AP, if such elements are available.

Deployment of function blocks is controlled by an orchestration layer that is run out of the operator data center with agents on the APs and pico datacenter. Each function block needs to define the requirements for the orchestration layer as shown in Table 6.1. The orchestration layer then deploys the function block in accordance with the constraints.

### 6.5.2 LegoFi Design Abstractions

The main LegoFi abstraction is the **Wireless Virtual Network Function (LVNF)**. A LVNF encapsulates a piece of control or data plane functionality necessary for deploying a WiFi network and exposes a high level, declarative interface suitable for that functionality, in addition to an interface for composing with other LVNFs. LVNFs can be composed to perform complex operations on data plane traffic, or to implement various control plane operations. In LegoFi, everything, including radio access, is treated as a virtual network function (VNF) and is implemented as a specialized LVNF. Notice however that, some LVNFs may require dedicated hardware. For example radio LVNFs require wireless NICs and/or Software Defined Radios (SDRs).

The LVNF abstraction covers three fundamental elements of VNF composition, namely: layer management (i.e., how to form a virtual network), state management (i.e., how to migrate VNFs from one node to another), and network monitoring (i.e., how to expose the network status):

- **State Management.** Programmers not need to be exposed to the details of handling VNF state, nor should they have to deal with the details of polling the network elements for statistics. The LVNF management framework tackles this requirement.

- **Layer Management.** Programmers should simply specifying the logical sequence of VNFs that traffic should traverse, leaving the details of the routing to an underlying runtime system. The Virtual Port abstraction (described below) addresses this requirement.

- **Network Monitoring.** Network managers should have a means to easily query the status of the network using high-level primitives. Such information includes network statistics and topology changes, e.g. nodes/links going offline or becoming congested.
A Virtual Port links an output port on one LVNF to the input port on another. The link is formed by specifying the portion of the flowspace that must be routed across the ports. For example, Figure 6.5 depicts a LVNF chain in which a WiFi AP is deployed as a collection of LVNFs. The composed WiFi AP can be viewed itself as a LVNF. This LVNF is responsible for handle the pure packet TX/RX operations including rate control, ACK, or RTS/CTS generation, and power control with some of these function offloaded to the Wireless NIC firmware. The radio LVNF has four output Virtual Ports. The first three emit, respectively, Probe Requests, Authentication Requests, and Association Requests and are on the control plane, while the fourth output emits data frames and is in the data plane. Management frames are then forwarded to three different LVNFs handling the generation of the corresponding response messages, while data frames are forwarded to a duplication filtering LVNF. HTTP traffic is forwarded to a DPI LVNF. Finally, all traffic is redirected to a gateway LVNF (Gateway) which connects the WiFi network to the Internet. Figure 6.6 depicts the internal structure of the probe responder LVNF which is built in the same way.

6.6 Prototype Implementation

We implemented our proof-of-concept LVNF management and orchestration framework by combining different virtual infrastructure managers. A virtual infrastructure manager is a software module that manages the allocation, placement, and control of virtual infrastructure. An example of such a module is the Ryu OpenFlow controller which controls the forwarding on switches. The goal is to integrate all VIMs under a common north bound API. While our prototype currently targets 802.11 networks, we believe the LegoFi architectural concept and possibly also the LVNF API may also be applicable to 5G wireless networks as well.

The LegoFi architecture currently accounts for three kinds of resources: (1) basic forwarding nodes (i.e., OpenFlow switches), (2) packet processing nodes, and (3) radio access nodes. The latter, in addition to the features supported by the packet processing node, also embed
specialized hardware in the form of one or more 802.11 Wireless NICs. Our prototype builds upon Click [49] as a single solution for advanced packet processing. Click allows complex VNFs to be built using simple and reusable components, called elements. Click includes over 300 elements supporting functions such as packet classification, access control, and deep packet inspection. Finally, Click is easily extensible with custom processing elements making it possible to support features that are not provided by the standard elements.

6.7 Summary

Motivated by the increasing and different requirements on WiFi networks, we promote to decompose, open and virtualize the WiFi network function blocks, making it possible to deploy them when and where they are most useful. The resulting flexibilities can be exploited to introduce innovative new network services, such as a smart client association management or packet efficient packet deduplication. Indeed, while we agree to the need of standardizing the radio link protocol, we are convinced that the network functions should be open.

We believe that our work opens several interesting directions for future research. In particular, while we understand LegoFi as an enabler, it will be interesting to study how to optimally reap the benefits of LegoFi in the different use-cases. Moreover, we are currently exploring additional use cases for LegoFi, e.g., in terms of automatic failover (the WiFi network continuous offering ad-hoc network services to the neighborhood even if connectivity to the Internet is lost) or security (the WiFi controller exploits increased visibility to detect malicious access points). LegoFi is open source and we have demonstrated an early prototype at [111].
Conclusion and Outlook

Fueled by the increasing popularity of wireless enabled mobile end-devices and the advent of the Internet-of-Things networks, the demand for wireless access technology is growing rapidly. However, supporting the ever increasing number of WiFi capable devices across residential, public, and enterprise networks is non-trivial. In particular, the (last) wireless hop is often critical for network performance, as it can contribute a non-negligible delay and may constitute a bandwidth bottleneck. Moreover, non-enterprise 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. Also mobility support is often very limited, depriving users from essential services. Thus, future WiFi architectures are challenged by optimized medium utilization, mobility support, and network management. The latter challenge is integrating wired and wireless networks seamlessly. While today point solutions exist for some of the WiFi-specific network challenges, commodity off-the-shelf hardware is outside the purview of such ossified, expensive, and vertically integrated solutions.

Decoupling data plane and control plane operations, à la Software-Defined Networking, can greatly simplify network management and improve the overall performance and utilization of wired networks. However, SDN and NFV have not yet received as much attention in the context of wireless networks, due to fundamental differences between wireless and wired networks. First and foremost, wireless networks feature many peculiarities and knobs that often do not exist in wired networks. For example, wireless networks need interfaces for flexible resource allocation, client mobility, client-based load balancing, and fine-grained traffic engineering is paramount. Furthermore, today’s trend towards Bring-Your-Own-Device (BYOD), implies that the network has to accommodate a more diverse set of user device types of different generations. Moreover, today’s home networks, unlike enterprise networks, typically suffer from a non-existing dedicated control channel, rendering fine-grained centralized control challenging.

In this thesis, we show that there can be a major benefit of introducing programmability and virtualization in wireless networks, i.e., following an Software-Defined Wireless Networking (SDWN) approach. With our SDWN approach we combine the benefits of SDN and NFV with wireless access technology. We present novel abstractions that hide the complexities of the IEEE 802.11 protocol stack and allow network operators to manage their wired and wireless portion of the network in unison. We decouple the control and data plane to consolidate and outsource the control over a set of network devices including WiFi APs, switches, and routers to a logically centralized software controller. This allows the control plane to evolve independently of the data plane, enabling faster innovations. Moreover, we make the
case for a functional decomposition of WiFi into its building blocks by following an NFV approach to virtualize network functions as software components running on generic compute resources and on programmable network devices. The resulting orchestration flexibilities can be used for a faster and cheaper service deployment. SDN can be exploited to steer flows through the appropriate network functions.

With our SDWN approach we overcome the aforementioned challenges in today’s WiFi networks. We now summarize the key contributions and take-aways of each chapter individually, then give directions for future work in the area.

7.1 Summary

The key contributions of each chapter are summarized below.

**Odin:** An SDN framework that provides programmability and virtualization of the upper IEEE 802.11 MAC functionality with off-the-shelf commodity hardware.

Chapter 3 proposes a Software-Defined Wireless Networking framework targeting WiFi networks. With Odin, we present our novel Light Virtual Access Point (LVAP) abstraction, that addresses well the complexities of the IEEE 802.11 protocol stack, and a control plane that allows the orchestration of WiFi and wired networks in unison, by leveraging OpenFlow for the wired portion of the network. We show the benefit via six common network services realized as SDN Applications. Odin runs on top of today’s commodity access point hardware without requiring client-side modifications, whilst being well-suited by design to take advantage of upcoming trends in physical layer virtualization and hardware extensions. Thus, with Odin, we present an solution to uniformly manage both wired and WiFi networks given the requirements of today’s network operators. However, Odin addresses only one aspect of the envisioned flexible and programmable WiFi architecture. In particular, Odin does not provide fine-grained control over the WiFi data path or control over middleboxes.

**OpenSDWN:** A joint SDN and NFV framework that provides programmability of the WiFi datapath and per-client virtual middleboxes, to render network functions more flexible and support mobility and seamless migration.

Chapter 4 presents a flexible, novel WiFi architecture for home and enterprise networks based on a joint SDN and NFV approach. OpenSDWN implements per-client virtual access points and per-client virtual middleboxes, to render network functions more flexible and support mobility and seamless migration, e.g., migrating firewall state between hotspots when performing a client handover. Moreover, OpenSDWN introduces IEEE 802.11 lower MAC (datapath) programmability to enable service differentiation and fine-grained transmission control, facilitating the prioritization of critical applications. Since the user is often left out of scope of this optimizations, we out-sources the control over the network to the user, application, or an Internet Service Provider through a participatory interface.
**AeroFlux**: A 2-tiered control plane that addresses the scalability aspects of an SDWN towards enterprise and ISP networks by exploiting locality in SDN control plane operations for scalability reasons, to tackle the risk of overloading the global logically centralised control plane.

Chapter 5 presents a 2-tiered approach for the design of a scalable wireless SDN control plane targeting large scale WiFi deployments. AeroFlux handles frequent, localized events close to where they originate, i.e., close to the data plane, by relying on Near-Sighted Controllers. Global events, which require a broad picture of the network’s state, are handled by the Global Controller (GC). More specifically, GC takes care of network functions that require global visibility, such as mobility management and load balancing, whereas NSCs control per-client or per-flow transmission settings such as rate and power based on transmission status feedback information exported by the Access Points (AP). Put differently, we enable the global controller to offload latency-critical or high-load tasks from the tier-1 control plane to the NSCs. This reduces the load on the GC and lowers the latency of critical control plane operations. As a result, with AeroFlux, we realize a scalable wireless SDN architecture which can support large enterprise and carrier WiFi deployments with low-latency programmatic control of fine-grained WiFi-specific transmission settings. Another key feature of AeroFlux is that it does not require modifications to today’s hardware and works on top of commodity WiFi equipment.

**LegoFi**: A functional decomposition of the WiFi architecture where WiFi function blocks are allocated where (and when) they are most useful.

Chapter 6 presents a modularized SDWN approach that is designed along the lines of SDN and NFV for WiFi networks. However, while the aforementioned architectures are important components of a future-proof WiFi architecture, there are additional opportunities to render the deployment of WiFi networks even more flexible. In particular, WiFi networks today provide a wide range of functionality related to performance and security. Today, these functionalities are often integrated and located at a single location, i.e., either implemented on the AP or the control plane. Given the different requirements and characteristics of these functions, this is suboptimal. A functional decomposition of the WiFi building blocks can aid to improve the overall networks performance. With LegoFi, WiFi function blocks are realized as virtualized and programmable Wireless Virtual Network Functions (WVNFs), and are allocated (and composed) where and when they are most useful. Specifically, through WVNFs, we achieve a functional decomposition of the WiFi architecture, allowing to overcome inflexibilities found in today’s monolithic, vertically integrated and expensive WiFi architectures.

To wrap up, by orchestrating and modularizing WiFi along the lines of SDN and NFV we can overcome today’s ossified WiFi architectures. Moreover, we present the necessary abstractions to introduce common features of enterprise networks to residential and hotspot deployments, i.e., for WiFi networks based on off-the-shelf commodity hardware. The practicality of our approaches has been successfully demonstrated at several international conferences and are currently deployed and running in two WiFi access networks, i.e., one.
university enterprise and one larger (30 household) residential network. Moreover, our contributions have gained commercial interest by network vendors and operators. Therefore, we believe that the contributions in this thesis constitute a relevant step forward to modern and future-proof WiFi networks.

### 7.2 Future Directions

This thesis encourages a multitude of directions as future work, both in terms of research as well as in terms of open source code development.

**Long Term Research Directions:** With LegoFi, we propose a radical and consequent approach to further flexibilize the WiFi architecture. While we have made a first step towards this vision, the topic deserves an in-depth study. In particular, as a first step towards such a study, an implementation and evaluation of an orchestrator may provide important insights into the design space. The questions in the scope of LegoFi are: What are the right interfaces between the virtual wireless function blocks? Which abstraction can be used for the functions blocks? How and where should the function blocks be placed? What are the design requirements to achieve a scalable logically centralized control plane? How and where should the control logic be placed, e.g., on an intermediate node close to the datapath or in the cloud? Is it beneficial to realize a modularized control plane where control logic can be deployed in a distributed fashion?

In the direction of 5G there is a plethora of open research directions: Can we leverage the modularized concept of LegoFi for 5th generation mobile networks (also known as 5G)? For instance, can we apply the LVAP concept to 5G networks? In other words, how can we leverage the LVAP abstraction for other wireless access technology? Thus, can we use LVAPs as a generic abstraction in SDWNs. Is the LVAP abstraction suitable for vertical handover, i.e., can we leverage the LVAP abstraction for fixed-mobile convergence to perform state migration between cellular and WiFi? Can participatory networking help in cases such as QoS provisioning or mobile-edge cloud, i.e., can the network provide resources such as computing and storage blocks?

LVAPs provide fast handover which enables seamless mobility. This seamless mobility is desirable for a couple of different network applications such as seamless WiFi access on highways, underground service, trains etc. Thus, further research in this direction can be to investigate the design requirements of an architecture that supports frequent handover at a large scale. In particular, how control traffic can be minimized in such environments?

The systems in this thesis do not fully support multi-tenancy. For instance, they do not support enough isolation and airtime fairness at the PHY layer. However, WiFi networks at scale should also allow to operate multiple tenants on top of the infrastructure. FlowVisor [126] or OpenVirteX [29] enable operators to create and manage virtual SDNs of multiple tenants on top of their own infrastructure. Specifically, OpenVirteX and FlowVisor act as OpenFlow proxies between an operator’s network and the tenants’ network controller to perform slicing etc. With these approaches tenants can specify their own topology and addressing schemes with performance isolation between slices. This challenge was not addressed yet in the scope of Software Defined Wireless Networking. Specifically, how to host a multitude of different tenants on top of a shared wireless infrastructure. How can we achieve
virtual addressing in the scope of 5G networks? How can we achieve performance isolation between slices of the wireless access down to the PHY?

**Short Term Directions:** The SDN framework Odin in Chapter 3 currently builds on the Click Modular Router [49] to realize the Agent functionality and Light Virtual Access Point (LVAP) abstraction. The control protocol is an open ASCII-based protocol which is used to communicate with the Click Control Socket. However, currently the Odin protocol relies on base64 encoding to exchange binary data such as spectral scan samples or entire IEEE 802.11 packets. The OpenFlow protocol supports nested types within the control messages and allows to transfer binary data. Accordingly, switching to the OpenFlow protocol would reduce the amount of control traffic. Thus, can we realize the Odin protocol as an extension of the OpenFlow protocol, i.e., by leveraging the vendor extension?

Another disadvantage of using Click is that it does not support IEEE 802.11n yet, which is likely to be the case also in the near future. Thus, a next step can be to investigate how the Odin approach can be realized within the mac80211 subsystem of the Linux kernel. This also enables us to (re-)use more advanced power saving schemes for the stations. Moreover, aggregating stationary clients into larger virtual AP cells with a single BSSID could also allow us to overcome the limitation of the current BSSID scheme (cf. Section 3.9). However, experimental features usually do not get easily accepted into the mainline Linux kernel. Thus, this would require to get the necessary interfaces into the mac80211 subsystem to keep the constant maintenance overhead at a minimum level when maintaining patches outside of the Linux kernel.

With OpenSDWN, in Chapter 4, we present a system which allows to build more flexible WiFi networks. Specifically, OpenSDWN enables fine-grained programmability of the WiFi datapath and control over middlebox functionality. We propose to investigate the orthogonal questions of 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) next. To speed up the process of service detection we propose to find a better bootstrapping approach. For instance, by following a two step approach where we first rely on DNS to match flows and then rely on our existing mechanism with bro signatures to update the existing rule with a more precise matching rule.

To follow-up on AeroFlux (Chapter 5), we suggest to continue from the current prototype in the direction of an generic approach for “control logic offloading”. In particular, we propose looking into mechanisms to describe the requirements of control logic functions and mechanisms for dynamic loading of control functions. Another direction is to investigate a proper bootstrapping and load-balancing mechanism for the AP to controller assignment. For instance, in a citywide HotSpot deployment it can be beneficial to assign all APs on a shopping street to the same controller to reduce the amount of inter-controller control messages, e.g., caused by frequent handovers between APs.
# List of Figures

2.1 SDN Reference Architecture. ........................................... 30
2.2 Coverage of the BOWL network on the TU-Berlin campus. ........... 32
3.1 High-level design of the Odin architecture. .......................... 38
3.2 Processing path for WiFi frames on Odin APs ......................... 39
3.3 Per-Client LVAP state ................................................. 39
3.4 Odin applications operate upon a per-slices view ..................... 41
3.5 Floor plan and placement of APs and clients (8 per cluster) .......... 46
3.6 Benefits of client-based Load Balancing ................................ 47
3.7 Troubleshooting detects jamming on frequency 2462MHz ............... 47
3.8 Automatic Channel Selection leveraging spectral scans ............... 48
3.9 High-level design of the energy efficient WiFi networks architecture.. 49
3.10 Normalized CPU utilization (2-cores) on the controller per density factor. 51
3.11 Impact of a LVAP migration and WiFi handoff on TCP throughput ..... 51
3.12 TCP throughput during Odin LVAP-handoffs .......................... 51
4.1 High-level architecture of OpenSDWN ..................................... 58
4.2 OpenSDWN’s relation to Odin .......................................... 58
4.3 Basic operations supported by OpenSDWN ............................... 60
4.4 Evaluation of OpenSDWN’s fine grained transmission control .......... 64
4.5 Evaluation of OpenSDWN’s service detection time and latency ......... 65
4.6 Latency for a stateful firewall vMB object read, write and delete operation 67
4.7 Evaluation of the smart direct multicast service ........................ 69
4.8 OpenSDWN’s Radio Agent and WDTX interface ......................... 70
4.9 vMB agent structure .................................................... 71
4.10 The Participatory Interface .............................................. 73
5.1 AeroFlux: NSCs are deployed close to the APs .......................... 82
5.2 Existing University WiFi deployment ..................................... 86
5.3 Evaluation of AeroFlux under consideration of an existing WiFi deployment 87
6.1 LegoFi targets a flexible deployment of WiFi functions ................ 95
6.2 VNFs for deduplication and load balancing .............................. 96
6.3 VNFs for probe responding and crypto mobility domain ............... 97
6.4 Deployment Scenarios in the scope of LegoFi ........................... 99
6.5 A simple WVNFS chain illustrating the concept of Virtual Port ........ 102
6.6 Internal structure of the probe responder WVNFS ....................... 102
List of Tables

3.1 Mobility Manager: Notifications triggered for two RSSI thresholds. .......... 45
3.2 Probe request serving latency. ........................................... 52

4.1 Service Detection Delay ................................................. 66
4.2 Stateful Firewall: Performance of the read, write and delete operation .... 68
4.3 Subset of South-bound APIs provided by the framework ...................... 69
4.4 Subset of APIs provided by the framework .................................. 72

6.1 Subset of Building Blocks ................................................. 100


[34] Bejerano, Y., and Jae Han, S. Cell breathing techniques for load balancing in wireless lans. *Mobile Computing, IEEE Transactions on* 8, 6 (June 2009), 735–749.


Bibliography


