TOWARDS A PRINCIPLED WIRELESS SUPPORT IN SDN

A Thesis

by

PRITHVIRAJ SHOME

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Alex Sprintson
Committee Members,	Paul Gratz
	Radu Stoleru
Head of Department,	Miroslav M. Begovic

May 2016

Major Subject: Computer Engineering

ABSTRACT

Software Defined Networking (SDN) has recently emerged as a transformational tool to design and operate communication networks and services. While the SDN approach has significant benefits for both wireline and wireless radio networks, the support for wireless networks in SDN technologies is still in its infancy as compared to wired networks. One of the key features of SDN is that networks can be managed in a programmatic manner. The challenge for building such a model for wireless radio networks is that there is a plethora of radio protocols that need to be supported, each having its own nuances. To address this, we need to build fundamental abstractions that provide enough visibility so that a programmer can implement protocols, while at the same time being rigid enough not to expose excessive details that will complicate the application development process. The purpose of this work is to introduce a principled approach towards building a cross-layer architecture for wireless networks so that they can receive the same level of programmability as wireline interfaces. Specifically we aim to integrate wireless protocols into the general SDN framework and to provide a logical and consistent view of physical layer radio resources. This is achieved by proposing a new set of abstractions and their interfaces based upon exisitng SDN terminology and the basic building blocks of Software Defined Radio (SDR) in wireless devices. We validate our approach by implementing our design as an extension of an existing OpenFlow data plane and deploying it in an IEEE 802.11 accesspoint as well as in a typical SDR system.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr. Alex Sprintson for the guidance, inspiration and the mentoring he provided all throughout my graduate career. He always provided great source of encouragement and was always available to discuss various problems that eventually led to this thesis. I would also like to thank the entire Flowgrammable Team who supported and provided valuable inputs throughout the work. I greatly benefited from the discussions and hope that we continue to work together in other projects. I am also thankful to Muxi Yan for the interactions and brain storming sessions during the ÆtherFlow project. A special mention also goes out to Jalil Modares and Dr. Nicholas Mastronarde from Univerity at Buffalo for the collaborative effort which went into CrossFlow project. Finally, I want to thank Dr. Paul Gratz and Dr. Radu Stoleru for having served as my committe members.

Last but not least, I want to thanks my family and friends for their immense support and encouragement, and most importantly, I would like to thank my wife Punam for her understanding and support, without whom none of this would have been possible.

NOMENCLATURE

SDN	Software Defined Networking
SDR	Software Defined Radio
TLS	Transport Layer Security Protocol
TCP	Transmission Control Protocol

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Software Defined Networking (SDN) drastically changes the meaning and process of designing, building, testing, and operating networks. The core principle of the SDN paradigm is a separation of the network control and data planes. It enables network administrators to have a centralized view of the network and provides a standardized interface for remote configuration of network devices. In particular, the SDN approach provides an abstraction of the underlying data plane and an interface to manipulate that abstraction. This approach provides the capability to manage and operate a large network through a logically centralized controller and to define custom network behaviors.

The current support for wireless networking in SDN technologies has lagged behind its development and deployment for wired networks. The academic and industrial communities have focused primarily on wireline networks, while wireless networks have received significantly less attention. Currently published SDN standards, the most popular of which is OpenFlow [16], do not provide support for wireless protocols, which poses a major obstacle to developing SDN-enabled heterogeneous networks with wireless components. Attempts to support wireless networking within that framework have been ad hoc, and true network visibility is missing with respect to wireless protocols. The wireless protocols are also constantly changing and new protocols are being developed which have made the task for management of cross-layer characteristics in wireless radio networks incredibly difficult.

With such ever-changing wireless standards and protocols, there has been a conscious shift towards a programmatic approach for designing and implementing wireless radios. This has led to a tremendous interest in Software Defined Radios (SDR). SDR

is a powerful concept in which filters, amplifiers, modulators and other complex signal processing blocks are realized in software, instead of on specialized hardware. As the task of signal processing is handed over to software, it is possible to use inexpensive general purpose hardware, connected to an RF front end, to create powerful and highly flexible radios.

While the SDR paradigm has revolutionized the design of wireless radios, it does not provide an efficient method to control a network of radios. While SDRs can be reconfigured to provide a wide variety of radio functionalities, there does not exist a consistent interface to expose the SDR's functional modules to the application developer. As modules can be added, removed or changed any time, the interface framework must be able to adapt to these changes. As such, the requisite framework should allow control of various constituent modules while hiding their complexity from the network operator. This level of abstraction is necessary because, as the network grows and becomes more heterogeneous, it is impossible for the operator to keep track of each and every wireless radio module. Here, by the notion of heterogeneous networks, we take into consideration a network containing both wired and wireless devices. Hence the architecture should enable network control, meet requirements of users and at the same time abstract away the details of the implementation. The goal of this work is to fill the gap by extending the basic concepts of SDN to support wireless networks in a principled way. We also aim to integrate two key technologies, SDR and SDN, to provide a consistent interface to manage underlying abstractions of physical layer radio resources in SDRs. Note that any reasonable design must not be specific to a single protocol or implementation of SDN, but applicable to every viable implementation. Furthermore, any solution must not be tailored to a single application, but enable potentially *any* application. Some examples are:

- *Physical layer adaptation* including (i) *frequency hopping* to resist narrowband interference and prevent unauthorized interception; (ii) *transmission power control* to maintain a target link quality while reducing interference to other users and/or extending battery life; and (iii) *adaptive modulation and coding* to trade-off throughput and communication reliability and adapt to channel conditions (e.g., pathloss and interference).

- *Quality of service (QoS) provisioning* to provide QoS policies according to profiles implemented through medium access control, throttling, admission control, scheduling, and error control techniques (e.g., ARQ and FEC). This allows both coarse-grained and fine-grained QoS policies to be defined in the network.

- *Wireless handoff* to efficiently manage the Layer 2 transition of a client between APs (access points)

- *Client steering* optimizes the association of all the mobile stations in a wireless network area by directing a client to connect to specific APs based on signal strength and current usage.

- *Adaptive routing* to allow a distributed controller, with its global view of the network, to dynamically switch between existing proactive and reactive routing protocols, and novel software-defined routing protocols, depending on the network conditions and the application constraints.

- *Self healing network* to allow the controller to deploy fault management applications based upon self-healing mechanisms.

- *Cross-layer control* to allow joint optimization of parameters, algorithms, and protocols at all layers of the protocol stack.

To support a broad range of applications, our approach is to extend a generalized model of SDN derived from the OpenFlow specifications [7]. These extensions include support for wireless ports and channels as well as the events and counters specific to wireless networks. We also define various abstractions and their interfaces corresponding to a radio physical port, which is in line with the design principles for SDR. Our model enables SDN controllers to configure, query, and control IEEE 802.11 Access Points (APs) with respect to a wide range of wireless events and also allows fine-grained control of processing abilites of SDRs, which is independent of protocols being implemented.

To validate the approach, we have implemented the model as an extension of the OpenFlow protocol, with a corresponding software implementation in the CPqD SoftSwitch software data plane [1]. We refer to our extension of this model and our initial implementation in IEEE 802.11 APs as *ÆtherFlow* and the implementation of programmatic control of SDRs through GNU Radio [3] framework as *CrossFlow*. GNU Radio provides a modular and open-source digital signal processing environment for SDRs. The modules of GNU Radio are written in C++ and are tied together through a Python wrapper to implement applications. We host GNU Radio on a Universal Software Radio Peripheral (USRP) N210 device from Ettus Research and also run CPqd SoftSwitch software [1] in a separate module as a switch agent.

We tested *ÆtherFlow* by developing a wireless mobility application that supports Layer 2 handoff of mobile stations between IEEE 802.11 access points. The resulting system performs on par with a traditional switching method. For testing *CrossFlow*, we develop three proof-of-concept applications, *frequency hopping adaptive modulation* and *cognitive radio* and validate its performance.

Our contributions can be summarized as follows.

- The extension of the generic SDN model to provide explicit support for wireless radio interfaces and wireless access points.
- An implementation of this extension based on the OpenFlow protocol and the CPqD SoftSwitch.
- An implementation of a controller application using ÆtherFlow framework and experiments to demonstrate the viability of SDN-controlled access points for efficient wireless handoff.
- Implementation of proof-of-concept applications using the CrossFlow framework for design validation.

2. BACKGROUND

2.1 Software Defined Networking

Network reconfigurability is a major challenge in the networking industry. The explosion of mobile devices and cloud services have necessiated the need for on-demand installation of services and reconfiguration of flow rules according to changing traffic patterns. In addition, network elements like routers and switches have their own unique interfaces and as such management of network components is a source of concern for network operators. As network grows, this complexity increases exponentially and rolling out new services becomes a tedious and complicated proces.

Software Defined Networking (SDN) is an architecture which addresses these challenges by decoupling the control and forwarding functions. This enforces abstraction of underlying implementation and enables applications or network services to be developed using the abstractions as shown in Figure 2.1. This simple and elegant design also provides applications a centralized view of the network. As a result, it has sparked tremendous research interest in providing a scalable, secure and programmatic approach towards the challenges discussed above. While SDN is a revolutionary approach, it is still mainly geared towards wired networks. The wireless networks have not recieved much attention in this regard. Through ÆtherFlow, we provide a protocol independent approach for bringing wireless into SDN model. In this thesis, we go a step further and provide a mechanism for dynamic radio resource management to obtain true network visibility in a heterogeneous network.

2.2 OpenFlow

OpenFlow is an open standard, that began as research project, which allows researchers to experiment with innovative protocols on network switches, without the

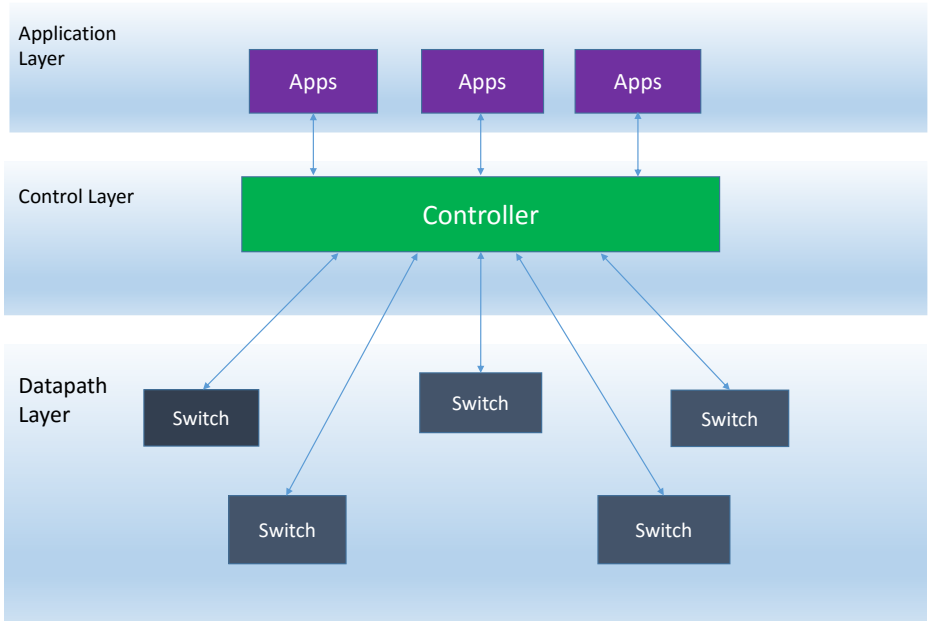

Figure 2.1: Overview of Software Defined Networking (SDN) architecture.

requirement of exposure to thier internal implementations. OpenFlow[16] builds upon the control and data plane abstractions envisioned in the SDN framework, to provide a well defined communication protocol between the two planes as well as a flow table abstraction. Each entry in a flowtable is composed of a number of fields for a packet to match upon, along with a set of associated instructions; each instruction involves performing some actions on a packet or modification of the pipeline processing in the form of allowing packets to be sent to other tables for processing. The OpenFlow communication protocol allows the centralized controller to interact with the switches so that the controller can add, delete or modify the flow table entries to perform certain processing actions. The protocol runs over TCP/TLS (Transport Layer Security) connection so that the communication remains secured and prevent unwanted intrusion which can compromise the security of the entire network. In order for a switch to understand controller's commands, the switch vendors implement

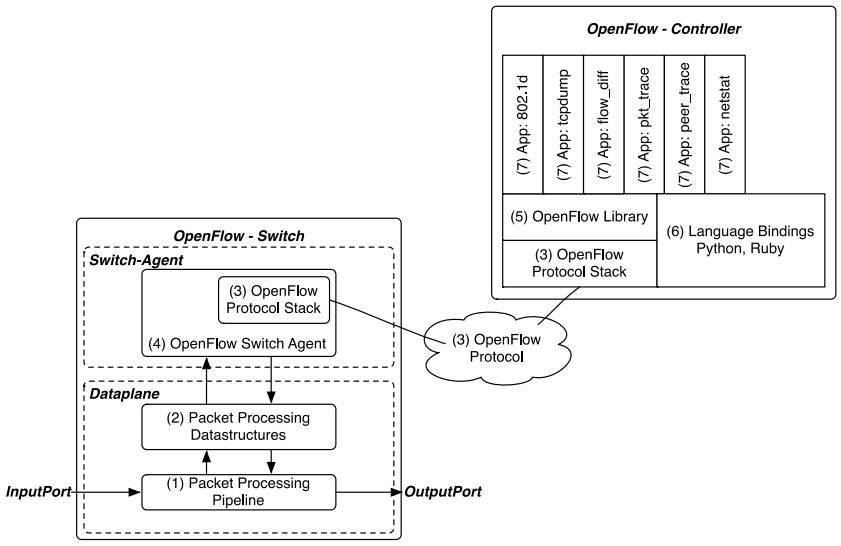

Figure 2.2: SDN ecosystem with OpenFlow protocol.
[Reprinted from [2]]

a switch agent and in this manner the implementation detail is abstracted out from the controller's point of view. This abstraction paradigm helps in development of sophisticated applications which can leverage the OpenFlow primitivies, to configure the data plane as well as listen for specific events, thereby enabling an asynchronous programmming model. This OpenFlow enabled SDN ecosystem opens the door for various network innovations and in this thesis, we leverage this model to implement a programmable control plane for radio resource management using Software Defined Radio (SDR). An overview of the current SDN ecosystem is presented in Figure 2.2.

2.3 Software Defined Radio(SDR)

Most of the wireless protocols in use today are implemented in hardware. With the ever increasing number of protocols to be supported and their diverse requirements, it has become apparent that a programmable environment for hardware is

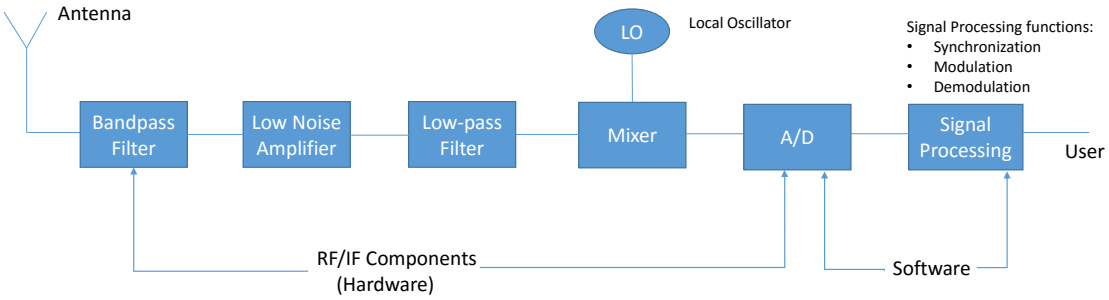

Figure 2.3: Components of Software Defined Radio (SDR).

the need of the hour. This requirement gave rise to the concept of Software Defined Radio (SDR). SDR is a communications system in which various hardware-centric features such as filters, modulators/demodulators and other signal processing blocks are implemented in software, rather than hardware, as shown in Figure 2.3. This is a powerful concept as this design offers high flexibility and runtime reconfiguration. This methodology also has the advantage that the radio can be configured to support various physical-layer protocols based on software, eliminating the need of custom, inflexible, and expensive hardware implementations. This property of reconfigurability is an important feature for various dynamic systems utilizing cognitive radio functionalities.As a result there is a significant interest among researchers and industry alike to make SDR a reality for network operators and end users. In this thesis, we use the programmable feature of SDR to define new abstractions and expose interfaces so that a network of radios can be controlled using SDN principle.

2.4 GNU Radio Framework

GNU Radio [3] is a free and open-source framework that provides signal processing functionality to implement SDRs. The main constituents of the framework are basic blocks which perform distinct signal processing functions. GNU Radio provides great leverage to compose these blocks to synthesize new radio functionality on a general purpose hardware. But the framework alone is not suitable for developing applications to control a network of SDRs. This is because each block exposes its own set of interfaces which does not scale with increasing numbers of radios in the network. In this thesis, we provide uniform interfaces to control and manage these processing block abstractions, so that an application developer does not need to handle every block's unique interface characteristics.

3. RELATED WORK

3.1 Extension of Wireless LAN

The interest for extending WLAN capabilities has been a community goal for a long time, but traditional methods have certain constraints. For example, the approaches reported in [17, 20] require modifications to the mobile clients (referred to as *mobile stations* in the Wi-Fi standard), which makes those approaches hard to deploy and test.

A recent technical report by the Open Networking Foundation (ONF) [10] identified the challenges of mobile networks, such as scalability, management, flexibility and cost, and provided a brief discussion of how SDN solutions can address these issues in few specific scenarios. A working group of Open Networking Foundation, Wireless & Mobile Working Group (WMWG), has been focusing on devising new SDN architecture for wireless use cases of different types [11]. However, to the best of our knowledge no concrete solutions were proposed by either ONF or WMWG up to now.

Several previous works presented systems that use OpenFlow extensions to achieve specific goals in wireless networks. In particular, OpenRoad [24, 26, 25] proposes to use the OpenFlow framework as a research platform for Wi-Fi and Wi-MAX systems. The platform supports slicing and virtualization of network resources, allowing different experimental services to run at the same time. SoftCell [14] focused on LTE networks and proposed to integrate SDN framework into the LTE core network architecture. The objective of ÆtherFlow is to design data plane control interfaces for wireless ports, which is different from these projects.

Other attempts to apply SDN to IEEE 802.11 networks include Odin [22] and OpenS-

DWN [18]. They provide certain wireless interface control and configuration capabilities to the SDN controller. In these solutions, virtual access points and associated device contexts are created for each individual mobile device, and move across access points when the client handoff occurs. Such type of framework can handle user mobility gracefully, but results in overhead in terms of both computational load and traffic load during handoff, especially in the settings with a large number of clients and high user mobility. ÆtherFlow offers a set of interfaces that costs less but still supports a wide variety of wireless applications.

In contrast to the existing works, ÆtherFlow provides a principled and general definition of wireless abstractions within an existing SDN framework. Our approach only requires incremental modifications to the existing SDN network elements.

3.2 Programmatic Wireless Dataplane

The idea of providing a programmable wireless data plane has been implemented in [4] and [5]. Both these papers provide modular blocks and focus on real time guarantees for processing signals. But they do not provide any logical interface to control a network of such programmable devices. We choose GNU Radio in our design as it provides unlimited flexibility. Gudipati et al. [12] deals with centralized control of devices but focuses mainly on LTE networks. Our work is orthogonal to these works as we provide a mechanism for centralized control while making the exposed interfaces protocol independent. The combination of SDRs and SDN has been introduced for various functionality in [8, 21], [15], [9] and [6]. [15] deals with creation of testbed for LTE technologies while [8, 21] focuses mainly on integration of SDN and SDR for 4G/5G technology. [13] describes a SDR model for management of interference in dense heterogeneous networks while [9] developed a jamming architecture using SDN and SDR principles. [6] provides a blueprint for

LTE self-organizing networks(SONs) using SDN and SDR principles. These papers provide distinct solutions for various scenarios but do not provide a generic framework for handling various protocols in a principled manner.

4. OPENFLOW EXTENSIONS

In our previous work, we derived a generalized SDN abstractions model, called TinyNBI [7], from the OpenFlow specifications [16]. In TinyNBI, the OpenFlow data plane is composed of several elements. The data plane elements and their structural relationships are depicted as UML diagram in Figure 4.1. TinyNBI model provides a clean low-level interpretation of the core OpenFlow abstractions and supports development of higher layer abstractions through refinement or extension. The model is primarily based on the notion of resources shared across a data plane.

In the TinyNBI model, each component exposes four types of interfaces: capabilities, configuration, statistics and events. These interfaces and their information flow directions are conceptually depicted in Figure 4.2.

Capabilities. Not every switch provides the same level of support for e.g., matching on protocol fields. Each component in the model must provide a facility that allows a controller to discover the set of operations supported by the device.

Configuration. Each OpenFlow data model element has some configurable parameters that can be modified during switch operation. The configuration interfaces are used to modify state and behavior of the data model element.

Statistics. OpenFlow switches gather statistical information via counters such as the numbers of bytes received and transferred over a port. Statistics interfaces provide the controller with access to the current state and values of these counters.

Events. The events interface reports to the controller certain types of events that

*Reprinted with permission from "ÆtherFlow: Principled Wireless Support in SDN" published in CoolSDN '15 [23], and "CrossFlow: A Cross-layer Architecture for SDR using SDN principles" published in IEEE NFV-SDN '15 ©2015 IEEE [19].

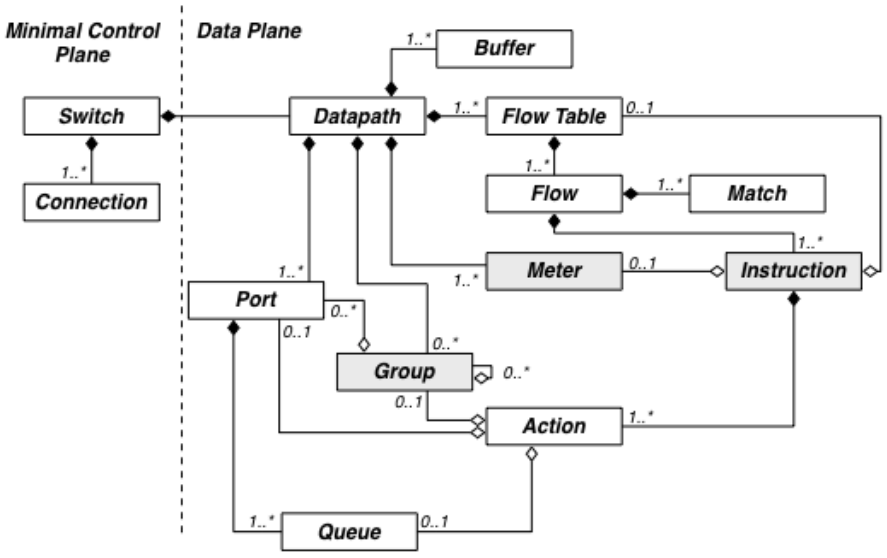

Figure 4.1: A UML diagram of OpenFlow data model. Each box represents a data plane element and the lines show the dependencies relationship among the elements. [Reprinted from [2]]

occur during switch operation.

4.1 ÆtherFlow Data Plane Abstractions

In order to support wireless networks, we refine the notion of ports from the TinyNBI SDN model. The SDN model already has a distinction between physical and logical ports. A *physical port* corresponds to an actual interface (e.g., Ethernet card), whereas a *logical port* is typically defined by software. Logical ports are often used for protocol tunneling and link aggregation.

To support wireless SDN controllers we introduce new types of both physical and logical ports. ÆtherFlow introduces wireless physical port corresponding to an IEEE 802.11 (commonly known as WiFi) radio interface. This allows controllers to query and configure the physical device over which packets are sent and received. Because a single 802.11 radio interface can support multiple simultaneous wireless

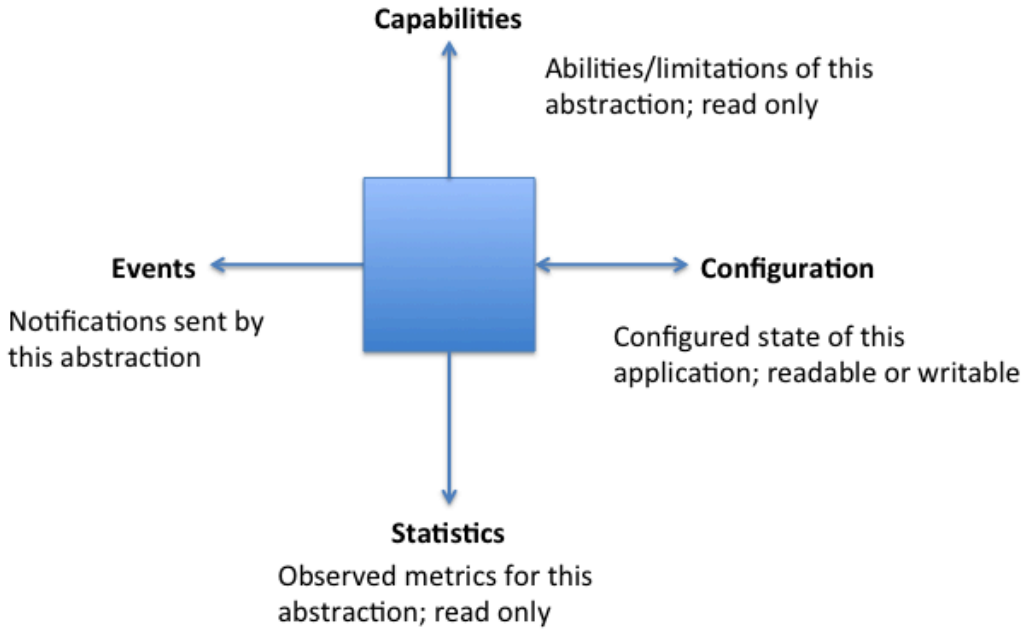

Figure 4.2: OpenFlow interfaces abstraction.

access points (APs), ÆtherFlow also introduces wireless logical port. Each wireless logical port is associated with its underlying physical port.

For packet processing, whenever a packet from a wireless AP is processed, its metadata records its *input port* as the logical port for the AP and its *input physical port* as the physical port the AP is created on. The frames received on the wireless interface are adapted into regular Ethernet frames for pipeline processing, meaning that we do not have to define any new protocol matching features for 802.11 MAC frame fields. This also allows an existing SDN implementation to compose wireless logical ports into link aggregation ports or various forms of tunnels.

The new data plane elements (wireless ports) defined in ÆtherFlow expose to the controller a set of control interfaces, categorized in the same way as the TinyNBI model, which are described as below.

Capabilities. ÆtherFlow allows the controller to query and obtain the capabilities of the radio interfaces of an AP. The supported capabilities information for wireless physical port includes (i) IEEE 802.11 version; (ii) channels; (iii) transmission power; (iv) encryption and key management methods; (v) maximum number of APs supported. ÆtherFlow does not define capabilities interface for wireless logical port.
Configuration. An OpenFlow controller can use ÆtherFlow messages to create or remove AP and dynamically (re)configure the following properties of an AP:

- Wireless physical port: (i) IEEE 802.11 version; (ii) channel; (iii) transmission power.
- Wireless logical port: (i) SSID; (ii) BSSID; (iii) encryption and key management method.

In addition, ÆtherFlow allows the controller to change the state of mobile stations associated with it, e.g., drop a station. Any new configuration to an AP is immediately applied. The configuration interfaces provide a high degree of programmability to applications that require these parameters to be adjusted during network operation.
Events. An SDN controller can receive MAC layer events related to a mobile station. ÆtherFlow currently supports the following types of events for wireless logical port: (i) probe; (ii) authentication; (iii) deauthentication; (iv) association; (v) reassociation; (vi) disassociation; (vii) authorization. ÆtherFlow does not define any event interface for wireless physical port.
The events occur when AP receives the corresponding 802.11 management frames. With these events reported, the controller can keep track of the 802.11 state of all the mobile stations communicating with the APs under control.
Statistics. An SDN controller can query the statistics of each physical wireless port and its associated logical ports. For a wireless logical port the following types

of statistics are supported: (i) number of packets sent and received; (ii) number of bytes sent and received; (iii) number of retries; (iv) number of retry failures; (v) current signal strength of a station; (vi) average signal strength of a station; (vii) connection duration of a station. For wireless physical port the set of supported statistics is identical to that supported by the OpenFlow protocol.

4.2 ÆtherFlow Messages

To implement ÆtherFlow in the framework of OpenFlow, we use experimenter messages provided in OpenFlow protocol to carry ÆtherFlow messages. In the current version, nine messages are defined in ÆtherFlow:

- Event report message – notify controller of events.
- Logical port statistics request/reply – request and reply of current statistics from a logical port.
- Physical port configuration request – modify the configuration of a physical port.
- Logical port configuration request – modify the configuration of a logical port.
- Physical port capabilities request/reply – request and reply of capabilities of a physical port.
- Drop station – force a mobile station to disassociate.
- Error message – customize error reporting for wireless.

4.3 CrossFlow Data Plane Abstractions

We extend the data model proposed in [7] to create an abstraction model for the CrossFlow framework, which is displayed in Figure 4.3. We build upon the *radio*

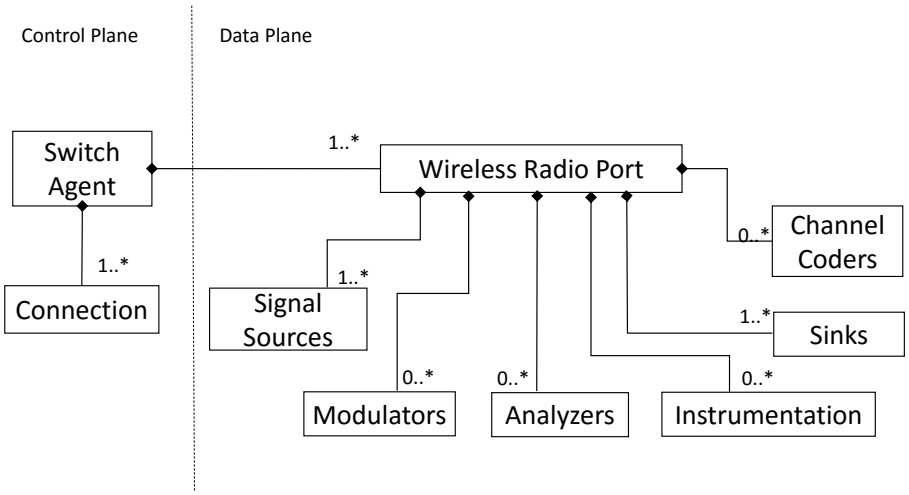

Figure 4.3: UML diagram of CrossFlow model.

physical port concept proposed in [23] to create a new layer of abstractions, which we refer to as the *wireless radio port* in this paper. This layer of abstractions exhibits a composition or *has-a* relationship with the *wireless radio port* abstraction (i.e., "the wireless radio port *has a* sink, modulator, or channel coder). This means that the blocks of this layer are the objects or members that comprise the *wireless radio port*. These blocks are derived from the most commonly used processing blocks in GNU Radio [3]. This abstract wireless radio port model serves the following design vision:

- It allows visibility into the signal processing blocks from an application point of view, without going into implementation details.
- It allows for the development of an event driven framework for radio operation.
- It allows composition of blocks to implement new functionality, as this decision is handled by the higher *radio physical port* abstraction. The application simply specifies the blocks to be connected for a specific wireless port instance and the internal framework handles the implementation.

In our current design, we focus on the first point of changing and quering the parameters of blocks at runtime. We assume that the number of blocks is fixed and the blocks can be connected in a consistent manner. In order to change parameters, the application needs to send ⟨*command,value*⟩ tuple in a message. For query and receive event responses, it registers for events for each block and during an event, appropriate callbacks are invoked. This ability to register for events is important so that a centralized controller can receive events asyncronously and implement a reactive model for its operations.

One of the main requirements of the CrossFlow model is that each abstraction should implement four types of interfaces as proposed in [7], namely: capabilities, configuration, statistics and events. The interface model for CrossFlow provides the interfaces for a wireless radio port abstraction with only two processing blocks, *Sink* and *Modulators*. The Sink abstraction allows the controller to manage the signal sinks which can be a USRP device, file or a socket, while the Modulators abstraction allows management of modulation schemes (e.g., BPSK, QPSK, and 8PSK).

The interfaces for *Sink* and *Modulators* are categorized as follows:

Sink.

- **Capabilities**: The interface allows the controller to query the capabilities of sinks such as: (i) Type of sink (USRP, socket, etc.); (ii) Channels supported; (iii) Center Frequency; and (iv) IP address.

- **Configuration**: The interface allows the controller to configure properties of signal sinks such as: (i) Gain; (ii) Frequency, and (iii) Sample rate.

- **Statistics**: The interface allows the controller to gather statistics for sinks such as: (i) Received Signal Strength Indicator (RSSI) and (ii) Temperature on-board.

- **Events**: The interface allows the controller to take decisions based upon events in a sink such as: (i) Low or high RSSI and (ii) Low or high on-board temperature.

Modulators.

- **Capabilities**: The interface allows the controller to query the properties of the modulator block such as: (i) Modulations supported; (ii) Current samples/symbol; and (iii) Gray code.
- **Configuration**: The interface allows the controller to configure properties of the modulator block such as: (i) Choice of modulation scheme (e.g. BPSK, QPSK and 8PSK); (ii) Samples/symbol; and (iii) Use of a Gray code.
- **Statistics**: The interface allows the controller to gather statistics for the modulator block such as: (i) Signal to Noise Ratio (SNR) and (ii) Bit Error Rate (BER).
- **Events**: The interface allows the controller to take decisions based upon events in the modulator block such as: (i) Low or high SNR and (ii) Low or high BER.

4.4 CrossFlow Message Extensions

CrossFlow uses SDN design principles to control a network of configurable SDRs. As such, to enable control plane interactions between the SDN controller and the SDR, we had two options: either we could have implemented our own control protocol to enable their interactions or extend the existing OpenFlow [16] framework. This is because OpenFlow does not natively support wireless features. In order to enable a cleaner implementation, we decided to extend OpenFlow by using experimenter messages within the OpenFlow protocol, similar to ÆtherFlow. Experimenter messages are a part of the standard OpenFlow protocol which provides a mechanism for

vendors to include propriety information within the protcol. This provides us with two advantages:

- We do not need to implement a new protocol for control and data plane interactions.
- As we are using experimenter messages to carry CrossFlow messages, the SDN controller does not need to perform special handling for these messages. This enables the controller to remain independent of the underlying devices and hence it can handle both wired and wireless devices.

In the current version, we define three messages in CrossFlow:

- Configuration message request - Request for modification of parameters like gain, frequency, SNR threshold and modulation scheme.
- Statistics message request - Request for statistics such as SNR, BER and RSSI.
- Event message response - Response for events like SNR below threshold and low BER.

5. ÆTHERFLOW IMPLEMENTATION

5.1 Implementation on an AP

To validate our design and to demonstrate the viability of the ÆtherFlow framework as a platform for the development and deployment of intelligent wireless SDN applications, we implemented and deployed ÆtherFlow on a commercially available access point.

We chose the access point TP-LINK WR1034ND v2 as the hardware platform for our implementation. This AP has five 100Mbps Ethernet ports and one 3-antenna radio interface, supporting protocols IEEE 802.11b/g/n. We replaced the firmware of the AP with OpenWRT 14.07 Barrier Breaker. OpenWRT is an open source Linux distribution designed for network embedded systems. Network utilities are integrated in OpenWRT and are optimized in size to fit in embedded environments which usually do not have as much resources as general purpose computer systems. In OpenWRT, when the radio interface is set up as an access point, its data plane is managed by Linux kernel and its control plane is run in the user space by daemon `hostapd`.

The native OpenWRT system does not support SDN. To make our access point an SDN switch, we used the open source CPqD SoftSwitch (`ofsoftswitch`), which implements an OpenFlow v1.3 pipeline and switch agent that can be deployed on OpenWRT system.

An ÆtherFlow data plane extension is then implemented in `ofsoftswitch`. The extension adds in wireless physical and logical ports that are mapped to the radio interface and access points managed by `hostapd`. To establish communication between `ofsoftswitch` and `hostapd`, `hostapd` is also modified to enable control of

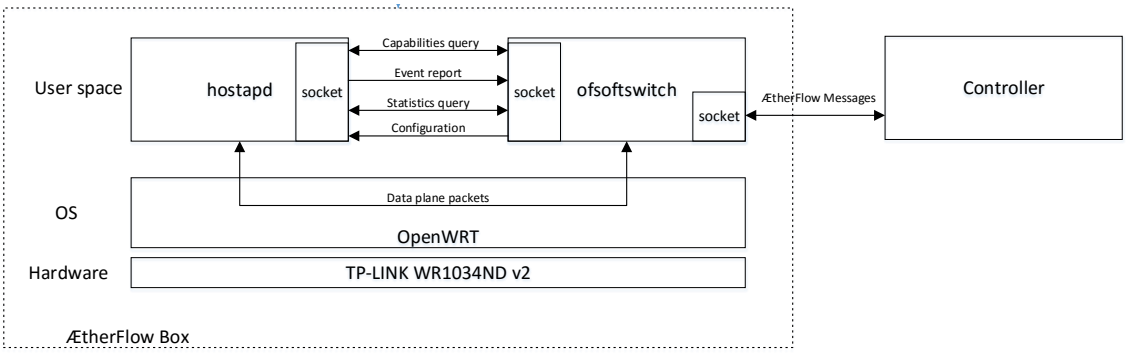

Figure 5.1: Implementation diagram of ÆtherFlow.

AP from `ofsoftswitch` and event reporting from AP to `ofsoftswitch`. The two processes communicate via a Unix socket in the OpenWRT system. An overview of this implementation is depicted in Figure 5.1.

Whenever an event related to a mobile station is triggered in `hostapd`, the event summary is sent to `ofsoftswitch`, which forwards it to the controller using the event port message. Whenever a statistics request from the controller is received by `ofsoftswitch`, the request is forwarded to `hostapd`, and the statistics data is sent to `ofsoftswitch` and then sent to the controller with a statistics reply message. Similar behavior occurs for capability queries and configuration updates.

5.2 ÆtherFlow Applications

The design of ÆtherFlow extends the capability of OpenFlow to wireless (specifically IEEE 802.11) interfaces in a natural way. ÆtherFlow enables applications to control both wireline switches and wireless access points. As a result, network applications that used to require different protocols and cooperation of software from different vendors can now be implemented easily using the ÆtherFlow framework.

We use a Layer 2 fast handoff application to demonstrate the flexibility and new functionality offered by the ÆtherFlow framework. This application aims to facilitate the process of mobile station handoff within the same subnet during which a device's Layer 3 address is not changed.

A typical Layer 2 fast handoff application runs in three phases. The first phase is handoff prediction. The controller collects signal strength information of the mobile stations by requesting statistics of all mobile stations associated with APs under its control. At the same time, it receives the probe signal strength of the mobile stations measured by other APs from the probe event reports. By keeping these data updated in a timely fashion, the controller may predict that a handoff is about to happen, e.g. when the mobile station's signal strength to its associated AP gradually weakens while the signal strength to another AP gradually strengthens.

The second phase of the Layer 2 fast handoff application is multicasting. When a handoff prediction of a mobile client is made, the controller multicasts all the packets with the client as destination to both its current associated AP and the predicted AP. The action is completed by modifying the flow entries of the switches in the network. Multicast guarantees that the client can receive packet immediately after it reassociates with the new AP, thus minimizing the packet loss during the handoff.

The third phase is flow redirection. After the multicasting phase, if the client as-

sociates with a new AP, the multicast is stopped and all the following packets to the client will be redirected to the new AP. If the prediction is wrong and a handoff did not occur within a certain timeout period, multicast is stopped and all the following packets will be forwarded to the original AP that the client is associated to. ÆtherFlow makes the decision possible with event report interface that provides client association event report to the controller.

Other than Layer 2 handoff application, wireless network applications such as client steering, user-based QoS control, etc. can also be easily implemented using ÆtherFlow framework.

6. CROSSFLOW IMPLEMENTATION

6.1 Illustrative CrossFlow Implementation

In this section, we describe our implementation of *adaptive modulation*, *frequency hopping* and *cognitive radio* applications using the CrossFlow framework. For illustration, we implement our model on a USRP N210 embedded SDR from Ettus Research. The N210 provides a Zynq 7020 All Programmable SoC, which combines a dual ARM Cortex-A9 processor and FPGA on the same device. We use the CPqD Softswitch [1] (`ofsoftswitch`) software as the switch agent in the SDN model. Its main functionality is to enable communication between GNU Radio and the python based Ryu SDN controller. As described in previous sections, the applications will send messages to the processing blocks, e.g., to configure them. The `ofsoftswitch` then forwards this request to a centralized `CrossFlow Hub` inside the GNU Radio domain.

There are four main components (blocks) in the illustrative CrossFlow module, as given in Figure 6.1, that we implement in GNU Radio, namely, the `CrossFlow Hub`, the Modulation Controller (Mod Controller for brevity), and the USRP Controller.

- The `CrossFlow Hub` is the interface between the Mod and USRP controllers in GNU Radio and the Ryu SDN controller. The `CrossFlow Hub` and the Ryu SDN controller communicate via Socket PDU. The `CrossFlow Hub` is responsible for receiving commands from `ofsoftswitch` (or any other compliant interface), interpreting the commands, and forwarding the commands to the appropriate controller block (i.e., either the USRP controller or Mod controller in our implementation). It is also responsible for receiving information from different controller blocks and sending information to the Ryu SDN controller.

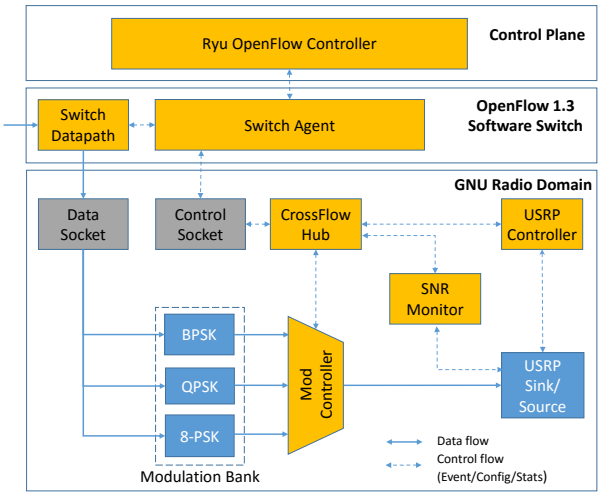

Figure 6.1: Transmitter implementation diagram of CrossFlow with two processing blocks: Sink and Modulators.

The `CrossFlow Hub` has in/out ports to send commands and receive information to/from the GNU radio controller blocks. It also has in/out PDU ports for interfacing with Socket PDU.

- The Mod Controller is one of the main controllers in the design. It is responsible for receiving commands from the `CrossFlow Hub`, and selecting the appropriate modulation scheme from the modulation bank. For illustration, we include three modulation schemes in our modulation bank (BPSK, QPSK, and 8PSK); however, thanks to the modular design, we can easily add more schemes. The Mod Controller can also feedback information to the Ryu SDN controller about the modulation scheme that is currently in use and the number of modulation schemes available in the modulation bank.

- The `SNR Monitor` is responsible for monitoring the SNR level and generating an event in case the SNR level falls below a certain threshold, which can be configured by the application. Currently the framework uses the existing SNR probe

of GNU Radio, which supports four M-PSK SNR estimators. This monitoring block is also responsible for relaying the SNR statistics back to `CrossFlow Hub` in response to a SNR statistics query generated by the application.

- The USRP Controller is another controller block in the CrossFlow module. It is responsible for controlling different RF parameters of the USRP Transmitter/Receiver based on commands from the `CrossFlow Hub`. In our proof-of-concept implementation, we control the carrier frequency and the power of the signal. It can also feedback information to the `CrossFlow Hub` about the current RSSI, temperature, SNR, carrier frequency, power, etc.

Although our illustrative implementation only has two controllers (one facilitating abstraction of the USRP Sink/RF implementation and the other facilitating abstraction of the adaptive modulation implementation), additional controllers can be easily added to support new applications, functionalities, and abstractions.

6.2 Example Applications

6.2.1 Frequency Hopping Application

Frequency hopping is a technique of transmitting radio signals by spreading the signal over a sequence of changing frequencies. It has tremendous application in military as it is used against jamming and for protecting against unauthorized eavesdropping. For implementation, the receiver of the signal must be aware of the sequence of frequencies so that it can tune into the appropriate channel. This requires synchronization between the transmitter and the receiver. In CrossFlow, we implement this application easily as only the controller needs to be aware about the predetermined sequence. This sequence can even be dynamic according to the channel conditions and policies.

6.2.2 Adaptive Modulation Application

Adpative Modulation is a technique where the modulation is changed according to the conditions of the channel. There are various estimators which are used for obtaining channel quality. These can be Signal-to-noise ratio (SNR), Bit error rate (BER) and other environment specific estimators. For illustration, we assume a fixed sequence for changing the modulation schemes every 5 seconds.

6.2.3 Cognitive Radio Application

We build upon the frequency hopping application mentioned above to construct a cognitive radio application. Cognitive radio is a type of radio in which the device is aware of its environment and can dynamically change its operating parameters like transmission power, frequency, gain etc in response to changing environmental conditions. In CrossFlow, we implement an application that can configure a radio device to switch channels based upon a low SNR event measured by the device.

7. VALIDATION

7.1 ÆtherFlow Validation

We use the ÆtherFlow implementation described in 5 to evaluate the performance and demonstrate the viability of the ÆtherFlow approach. Our results demonstrate that ÆtherFlow framework allows SDN applications to efficiently and dynamically configure wireless networks without loss of performance.

7.1.1 Experiment Setup

Our experiment uses a simple network topology, shown in Figure 7.1. It consists of two access points (AP1, AP2), a Layer 2 switch, a wireline traffic generator and a wireless 802.11 mobile station (STA). We use an OpenFlow enabled Layer 2 switch and ÆtherFlow enabled APs as described in Section 5. The mobile station has a single WiFi radio interface.

In our experiment, both APs and the traffic generator are connected to the switch through Ethernet. All the three boxes are connected to an OpenFlow controller through a separate control plane subnet that is not displayed in the figure. The two APs are located at a certain distance and have overlapping coverage areas. Both APs are configured with the same SSID and use open authentication.

7.1.2 Layer 2 Handoff Application

Our Layer 2 handoff application accords with what we described in Section 5.2. The investigation of good predictors and predictive models for handoff is beyond the scope of this paper. In our implementation, the controller application always predicts that the handoff of STA from AP1 to AP2 will occur seven seconds after the experiment starts. The time period is selected solely for the purpose of this experiment and does

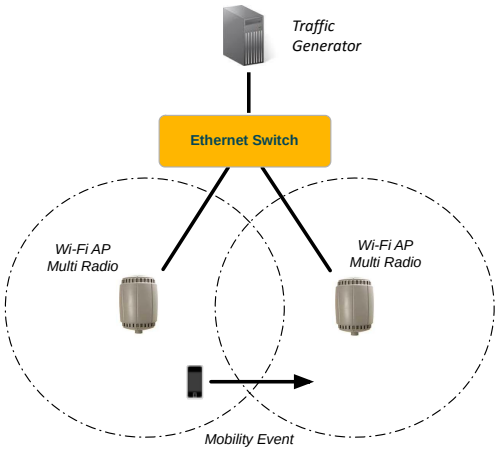

Figure 7.1: ÆtherFlow Experiment network topology.
[Reprinted from [2]]

not apply for general cases.

After seven seconds, the controller starts to multicast packets going to STA to both AP1 and AP2 by sending FlowMod messages to both APs and the switch. After STA associates with AP2, the controller configures the switch to stop multicasting and forward packets to only AP2. If the predicted handoff did not happen 15 seconds after the prediction, the controller reverts the multicast and forwards packets to only AP1.

7.1.3 Experiment Procedure and Results

In each round of experiment, the mobile station is initially associated with AP1. Both the traffic generator and the mobile station are assigned static IP addresses within the same subnet. Before the experiment starts, a UDP `iperf` session with bandwidth of 9Mbps is initiated from the traffic generator to STA. After experiment

starts, STA moves from coverage of AP1 to coverage of AP2, which forces the client to handoff from AP1 to AP2. We move STA in a controlled manner such that the handoff happens at seven seconds after the experiment starts. This time is selected such that the handoff happens almost immediately after the controller application initiates multicasting. Throughput and packet loss rate during each round of test is measured by iperf with an interval of 0.5s. In each round of experiment, one of the following configurations is used:

- **Bridge configuration** uses neither OpenFlow nor ÆtherFlow. Instead, the Layer 2 switch and the two APs use the Linux built-in learning bridge to forward packets. This is the traditional way of configuring a Layer 2 network with two access points and one switch.

- **ÆtherFlow configuration with prediction** enables ÆtherFlow on the APs and the switch, and the handoff is managed by the Layer 2 handoff application (described above) running on the ÆtherFlow controller using the Ryu controller framework.

- **ÆtherFlow configuration without prediction** enables ÆtherFlow on the APs and the switch, and the handoff is managed by a similar Layer 2 handoff application as given above without enabling prediction.

Fifteen rounds of experiments are conducted on each of the three configurations given above. In a single round of experiment, the mobile station is considered to be in handoff process during an interval after time $t = 7$s if its average throughput during the interval is less than 8 Mbps. By this criteria we can determine the handoff duration of STA in each round of experiment. Our results, depicted in Figure 7.4, indicate that the average handoff duration of ÆtherFlow configuration

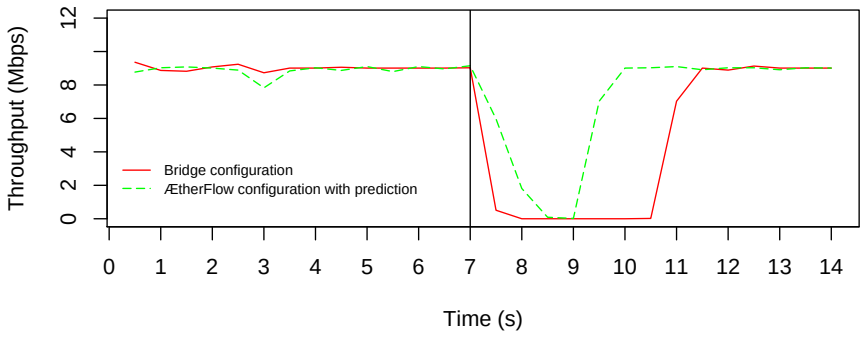

Figure 7.2: Comparison of throughput for ÆtherFlow with prediction and the baseline configuration.

with prediction across the fifteen rounds of experiments is **2.53**s, which is lower than that of bridge configuration **3.8**s and ÆtherFlow configration without prediction **3.17**s.

We compare the traffic throughput and packet loss rate of the experiments which have median handoff duration in ÆtherFlow with prediction and bridge configurations (experiment 3 for bridge configuration and experiment 4 for ÆtherFlow configuration). The plots are shown in Figure 7.2 and Figure 7.3. These plots demonstrate that in terms of both throughput and loss rate, the ÆtherFlow configuration recovers from handoff faster than the bridge configuration.

The experiment results show that even with the overhead induced by SDN data plane processing, the performance of Layer 2 handoff application based on ÆtherFlow is better that of Linux kernel bridge configuration.

7.2 CrossFlow Validation

We use the CrossFlow implementation described in 6 to provide a proof-of-concept validation of our design. We demonstrate the viability of CrossFlow which allows

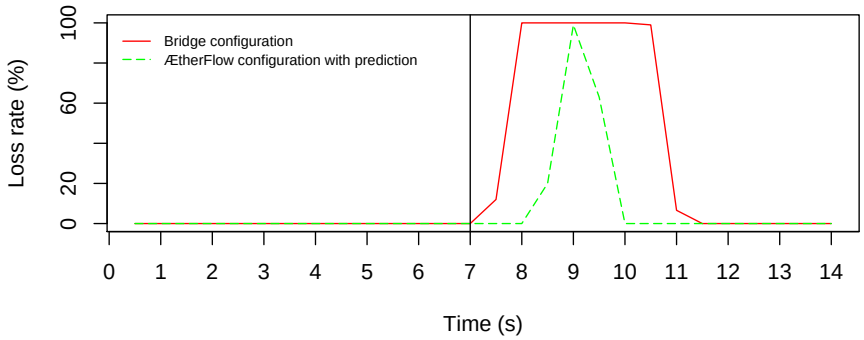

Figure 7.3: Comparison of packet loss rate for ÆtherFlow with prediction and the baseline configuration.

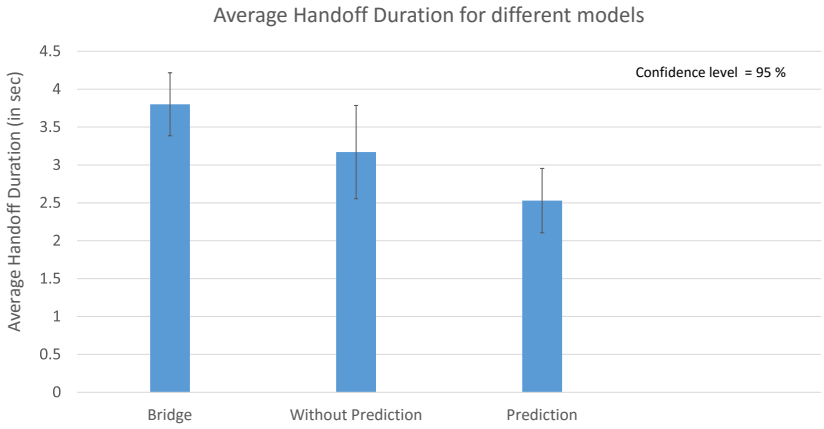

Figure 7.4: Comparison of average handoff duration for ÆtherFlow with different models and standard error.

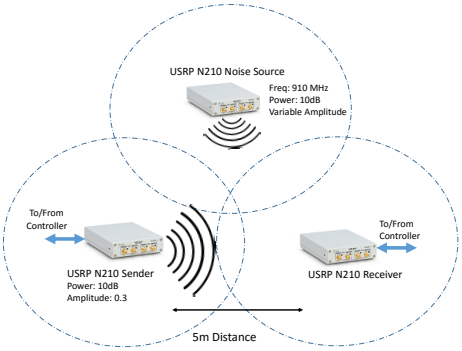

Figure 7.5: Setup for cognitive radio application in CrossFlow

Table 7.1: Variation of SNR and PER with increasing Noise Amplitude Factor keeping a fixed data rate of 1Mbps.

Noise Amplitude Factor	Packet Error Rate	SNR (in dB)
0	0.15%	5.8553
0.09	6.34%	-0.2983
0.14	19.89%	-0.9483

SDN applications to change the radio physical properties of wireless radio upon various qualifying parameters like channel conditions.

7.2.1 Frequency Hopping Application Implementation

In this implementation, the controller simply issues *GNU-CONFIG-FREQ* command with the desired frequency and pushes this configuration to the device. As shown in Figure 6.1, the `ofsoftswitch` receives this command and forwards it to the GNU Radio domain. The centralized `CrossFlow Hub` inside the GNU Radio domain processes this request and issues appropriate commands to the USRP Controller, which ultimately signals the USRP block to tune into the requested frequency. The application uses a pre-determined sequence of 910, 915 and 920MHz frequencies which

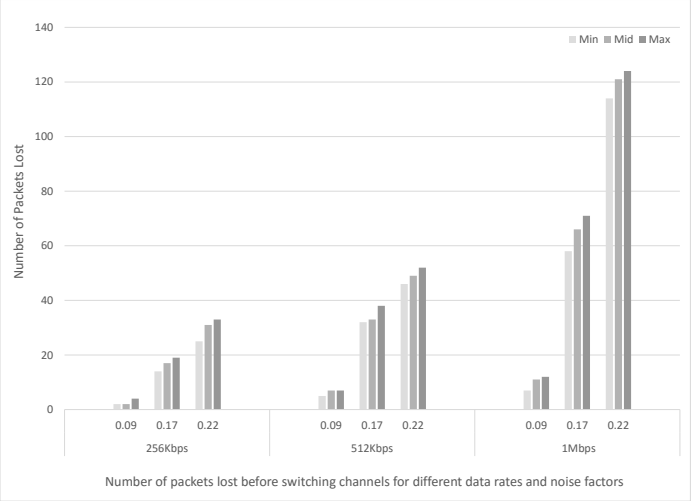

Figure 7.6: Range of packets loss while changing channels by keeping a fixed data rate and a varying noise factor across each experiment

changes every 5 seconds along with a fixed BPSK modulation scheme.

7.2.2 Adaptive Modulation Application Implementation

Similar to the *frequency hopping* application, the Ryu SDN controller issues the *GNU-CONFIG-MOD* command with the appropriate modulation scheme (BPSK, QPSK, or 8PSK) and forwards the request to the device. The request ultimately reaches the Mod Controller, which is a multiplexer block as shown in Figure 6.1, that selects the requested modulation scheme. The application requests the use of a modulation scheme from among BPSK, QPSK and 8PSK modulations every 5 seconds with a fixed carrier frequency of 910 MHz.

7.2.3 Cognitive Radio Application Implementation

As stated in Section 6, we build upon the frequency hopping application to implement a cognitive radio application. Figure 7.5 shows the experimental setup, where we

have three USRP N210 devices that act as sender, receiver and noise source. The sender is set at a 10 dB power level and 0.3 transmission amplitude factor, while the noise source is set at 10 dB power with a variable amplitude factor. Note that the amplitude factor is simply a constant that is multiplied to the transmitted signal to adapt the effective transmission power. The sender and receiver are at a distance of 5 meters apart and the sender begins transmission at 910 MHz carrier frequency with 1 Mbps data rate and packet length of 50 Bytes. The noise source on the other hand, sends high frequency pulses with varying amplitude factors at 910 MHz frequency. We refer to the noise source's transmission amplitude factor as the noise amplitude factor (NF). When SNR falls below a specified threshold, a low SNR event is triggered by the `SNR Monitor` block and the event summary is sent to `ofsoftswitch` through `CrossFlow Hub`. This request is then forwarded to the Ryu SDN controller using the event response message. The application, upon receiving this message, sends a GNU-CONFIG-FREQ command so that the device changes the channel to the requested frequency. The sequence of actions involved in changing the channel is similar to the one mentioned in the previous section for *frequency hopping*.

Using this setup, we conduct two tests: one to measure the effect of the NF on the receiver's packet error rate (PER) and SNR at the 910 MHz carrier frequency (both measured over 1,000,000 packets transmitted at 1 Mbps with NFs of 0.0, 0.09, 0.14), and another to measure how quickly the cognitive radio can trigger and respond to a low SNR event (with NFs 0.09, 0.17 and 0.22, and data rates 256 Kbps, 512 Kbps and 1 Mbps). Table 7.1 shows the PER and SNR values obtained in the first experiment. As expected, the PER increases and SNR decreases with increasing NFs. In the second experiment, which demonstrates a simple cognitive radio application, the transmitter and receiver pair switch to a new carrier frequency (915 MHz) when the instantaneous received SNR falls below the pre-defined 6 dB threshold. In Figure 7.6,

we show the number of packets that are lost over the course of time required for the SNR to be sensed below the 6 dB threshold, for the receiver to generate the low SNR event, and for the Ryu SDN controller to respond by issuing the GNU-CONFIG-FREQ command, and finally for the transmitter to switch frequencies. We repeat this experiment three times for each combination of data rate and noise factor, and plot each measurement in Figure 7.6 as a separate bar.

8. CONCLUSION AND FUTURE WORK

In this paper we presented two SDN frameworks ÆtherFlow and CrossFlow. These frameworks aim to bring the wireless networks into the SDN fold using a principled approach and provide greater flexibility and programmability in wireless networks. They provide new abstractions and extensions to demonstate a protocol independent architecture and provide proof-of-concept implementations to showcase flexible network management.

ÆtherFlow includes the ability to handle wireless packets using an OpenFlow data path, remotely configure access points, query mobile station capabilities and statistics, and report mobile station events.

To validate our ideas, we implemented an ÆtherFlow switch and adapted an existing OpenFlow controller to work with our extensions of the OpenFlow protocol. We experimented with an SDN-based mobile handoff application, and found that our design slightly outperforms an optimized non-SDN application. We note this is a proof-of-concept experiment designed to show that useful SDN applications can be written against the ÆtherFlow extensions to OpenFlow.

As a general wireless SDN framework, the ÆtherFlow model can also be immediately leveraged to support a number of different applications, or can easily be extended to support them. In addition, similar extension approaches can be used on systems other than IEEE 802.11, such as WiMAX or cellular networks, which is a promising direction for the evolution of SDN. We leave this as our future work.

On the other hand, the CrossFlow framework allows flexible and real-time configuration of software defined radio interfaces from a network controller application. It allows a controller application to be written without worrying about the internal

details of implementations. In order to validate our approach, we implemented *frequency hopping*, *adaptive modulation* and *cognitive radio* applications. This shows that our design is viable and can extended to introduce new capabilities.

One of the challenges that we need to consider is the issue of latency between controller and SDR framework. The issue can be mitigated, by the introduction of distributed control module in SDR. The distributed control module will allow devices to take local decisions while the centralized controller is responsible for introducing policies and global management, thereby ensuring reduced latency.

The CrossFlow framework can also be extended to allow controller to create GNU radio blocks and manipulate inter-connections between GNU radio blocks. It requires to design new API on switch agent and can be implemented by combining the methodology provided by GNU Radio. In GNU radio, each block has an input and output port and the application needs to specify the connecting ports in order to connect the blocks. Only ports which are similar, i.e, ports which operate on same types of data(message or stream), can connect to each other. The data type supported by a block can be obtained by sending capability messages. The decision whether two ports are compatible can be left to the application.

Our results indicate that while current SDN protocols support the development of very intelligent wireline network management applications, ÆtherFlow and CrossFlow are significant steps in bringing that same level of programmability to wireless networks.

REFERENCES

[1] CPqD OpenFlow 1.3 Software Switch. http://cpqd.github.io/ofsoftswitch13/. [Online; Accessed: 2014-09-30].

[2] Flowgrammable. http://flowgrammable.org/. [Online; Accessed: 2014-09-01].

[3] GNU Radio. http://gnuradio.org/redmine/projects/gnuradio/wiki. [Online; Accessed: 2015-09-30].

[4] M. Bansal, J. Mehlman, S. Katti, and P. Levis. Openradio:A Programmable Wireless Dataplane. In *Proc. HotSDN, 2012*, 2012.

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

[6] C. Ramirez-Perez and V. Ramos. SDN meets SDR in Self-Organizing Networks: Fitting the Pieces of Network Management. *IEEE Communications Magazine*, 54:48–57, 2016.

[7] C. J. Casey, A. Sutton, and A. Sprintson. TinyNBI: Distilling an API from Essential OpenFlow Abstractions. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 37–42, New York, NY, USA, 2014. ACM.

[8] H.-H. Cho, C.-F. Lai, T.K. Shih, and H.-C. Chao. Integration of SDR and SDN for 5G. *Access, IEEE*, 2:1196–1204, 2014.

[9] C. Corbett, J. Uher, J. Cook, and A. Dalton. Countering Intelligent Jamming with Full Protocol Stack Agility. *Security & Privacy, IEEE*, 12(2):44–50, 2014.

[10] Open Networking Foundation. Openflow-enabled mobile and wireless networks. Technical report, September 2013. https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-wireless-mobile.pdf [Online; Accessed: 2014-08-26].

[11] Open Networking Foundation. Wireless & mobile working group. Technical report, January 2014. https://www.opennetworking.org/images/stories/downloads/working-groups/charter-wireless-mobile.pdf [Online; Accessed: 2014-02-01].

[12] A. Gudipati, D. Perry, L. E. Li, and S. Katti. SoftRAN: Software Defined Radio Access Network. In *Proceedings of the second workshop on Hot topics in software defined networks, ser. HotSDN '13, 2013*, 2013.

[13] R. Gupta, B. Bachmann, A. Kruppe, R. Ford, S. Rangan, N. Kundargi, A. Ekbal, K. Rathi, A. Asadi, V. Mancuso, et al. LabVIEW based Software-Defined Physical/MAC layer architecture for prototyping dense LTE Networks. In *SDR WInnComm*, 2015.

[14] X. Jin, L.E. Li, L. Vanbever, and J. Rexford. SoftCell: Scalable and Flexible Cellular Core Network Architecture. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '13)*, pages 163–174, 2015.

[15] V. Mancuso, C. Vitale, R. Gupta, K. Rathi, and A. Morelli. A prototyping methodology for SDN-controlled LTE using SDR. In *ETSI workshop on Reconfigurable Radio Systems - Status and novel Standards (ETSI RSS Workshop 2014)*, 2014.

[16] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling Innovation in Campus

Networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[17] R. Murty, J. Padhye, A. Wolman, and M. Welsh. Dyson: an architecture for extensible wireless lans. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference (USENIXATC '10)*, pages 15–15, 2010.

[18] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S.Schmid, and A.Feldman. OpenSDWN: Programmatic Control Over Home and Enterprise WiFi. In *1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR '15)*, 2015.

[19] P. Shome, M. Yan, J. M. Najafabadi, N. Mastronarde, and A. Sprintson. CrossFlow: A Cross-layer Architecture for SDR Using SDN Principles. In *Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (IEEE NFV-SDN),Nov. 2015)*, 2015.

[20] V. Shrivastava, N. Ahmed, S. Rayanchu, S. Banerjee, S. Keshav, K. Papagiannaki, and A. Mishra. CENTAUR: Realizing the Full Potential of Centralized WLANs Through a Hybrid Data Path. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking (MobiCom '09)*, pages 297–308, 2009.

[21] S. Sun, M. Kadoch, L. Gong, and B. Rong. Integrating network function virtualization with SDR and SDN for 4G/5G networks. *Network, IEEE*, 29(3):54–59, May 2015.

[22] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao. Towards Programmable Enterprise WLANs with Odin. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*, pages 115–120, 2012.

[23] M. Yan, J. Casey, P. Shome, A. Sprintson, and A. Sutton. Aetherflow: Principled wireless support in SDN. In *Proceedings of the ICNP 2015 Workshop on Control, Cooperation, and Applications in SDN protocols (CoolSDN '15)*, 2015.

[24] K.-K. Yap, M. Kobayashi, R. Sherwood, N. Handigol, Te.-Y. Huang, M. Chan, and N. McKeown. OpenRoads: Empowering Research in Mobile Networks. *ACM SIGCOMM Computer Communication review*, 40(1):125–126, January 2010.

[25] K.-K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, and N. McKeown. The Stanford OpenRoads Deployment. In *Proceedings of the 4th ACM International Workshop on Experimental Evaluation and Characterization (WINTECH '09)*, pages 59–66, 2009.

[26] K.-K. Yap, R. Sherwood, M. Kobayashi, T.-Y. Huang, M. Chan, N. Handigol, N. McKeown, and G. Parulkar. Blueprint for Introducing Innovation into Wireless Mobile Networks. In *Proceedings of the second ACM SIGCOMM workshop on Virtualized Infrastructure Systems and Architectures (VISA '10)*, pages 25–32, 2010.