Enabling Dynamic Reconfigurability of SDRs Using SDN Principles

Prithviraj Shome*, Jalil Modares†, Nicholas Mastronarde†, and Alex Sprintson*
*Department of Electrical and Computer Engineering, Texas A&M University
†Department of Electrical Engineering, University at Buffalo

***Abstract*—Dynamic reconfiguration and network programmability are active research areas. State of the art solutions use the Software Defined Networking (SDN) paradigm to provide basic data plane abstractions and programming interfaces for control and management of these abstractions. This approach provides the benefit of control and data plane separation, but it is mainly limited to wired networks. Currently, SDN technologies do not provide appropriate abstractions to support constantly evolving wireless protocols. On the other hand, the Software Defined Radio (SDR) paradigm enables complex signal processing functionality to be implemented efficiently in software, instead of on specialized hardware. However, SDR does not cater to the demand for adaptive radio network management with respect to changing channel conditions and policies. To address this, we present CrossFlow, a principled approach for application development in radio networks. CrossFlow defines several fundamental radio port abstractions and an interface to manipulate them. The framework provides a flexible and modular cross-layer architecture using the principles of SDR and a mechanism for centralized control using the principles of SDN. The main feature of the CrossFlow architecture is that it provides a protocol independent framework for application development in wireless radio networks. We validate our design using proof-of-concept applications, namely, adaptive modulation, frequency hopping, and cognitive radio. Our results indicate that our framework is efficient, flexible, and can be used for a variety of applications.**

I. Introduction

With 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 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 SDRs. Since SDRs can be reconfigured to provide a wide variety of radio functionalities, it would be highly desirable to have a consistent interface to expose the SDR's functional modules to the application developer. As modules can be added, removed or changed any time, such an interface framework must be able to adapt to these changes, report events to the application, and allow control of various constituent modules while hiding their complexity from the application developer. This level of abstraction is necessary because, as the network grows and becomes more heterogeneous, it is impossible for the application developer to keep track of low-level details. 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.

In order to provide abstractions taking into account the above considerations, we use the concept of Software Defined Networking (SDN). SDN defines abstractions that represent data plane components along with the interface to control and manage these abstractions. As a result, these primitives (including asynchronous callback function event reporting) enable an application developer to obtain a logically centralized view of the network. The application developer can then dynamically adjust rules to reflect changing network conditions and requirements.

In this paper, we aim to integrate SDR and SDN to provide a principled approach for developing a consistent interface to manage underlying abstractions of SDRs. We build upon the abstract model presented in our previous paper [12], where we described a monolithic architecture for *wireless radio port* abstraction. In the current paper, we go beyond that and broaden the design space to provide a modular design, which is in line with the design principle of SDR. This also enables an integration of both wired and wireless networks which can be managed in a programmatic manner, thereby enabling development of key applications catering to a heterogeneous network. We call our platform CrossFlow. Some network applications that we envision can leverage CrossFlow include, but are not limited to, the following:

- *Physical layer adaptation* including:
  1) *frequency hopping* to resist narrowband interference and prevent unauthorized interception;
  2) *transmission power control* to maintain a target link quality while reducing interference to other users and/or extending battery life;
  3) *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.
- *Adaptive routing* to allow a CrossFlow 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 CrossFlow 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.

We use the generalized model of SDN introduced in [5] as a template for defining the abstractions and their features discussed above. We also build upon the concept of *wireless radio ports* as discussed in [14]. This abstraction is composed of a number of smaller abstractions, one for each processing block, so that fine-grained control of the processing capabilities of a radio device is provided to application developers without exposing its intricate details. This enables manipulation of critical physical, data link (including the medium access control sublayer), and network layer properties through various well defined interfaces. Thus using the architecture of CrossFlow, we can build applications across all layers of the network stack.

For validation purposes, we use the popular GNU Radio [2], which provides a modular open-source Digital Signal Processing (DSP) software framework for SDRs. The modules of GNU Radio are written in C++ and provide a mechanism to connect and manage data between them. A Python wrapper ties these blocks together 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] as a separate module. CPqd SoftSwitch serves as a switch agent interacting between the SDN controller and GNU Radio modules. This is done through message extensions which we will discuss in subsequent sections. We also develop three proof-of-concept applications to validate our design principles: *frequency hopping*, *adaptive modulation*, and *cognitive radio*.

Our contributions can be summarized as follows.
- We propose a framework that provides a uniform and consistent view of SDRs, so that a network of SDRs can be managed in an efficient manner.
- We extend the SDN model with message extensions to provide support for wireless radio interfaces.
- We provide sample applications using the framework for validation.

The rest of the paper is organized as follows. In Section II we review the related work done in this area. Section III describes the CrossFlow architecture with its SDN extensions. Section IV describes a proof-of-concept implementation of three applications using our framework. Section V concludes the paper and discusses future work.

II. Background and Related Work

Software Defined Networking. Network reconfigurability is a major challenge in the networking industry. The explosion of mobile devices and cloud services have necessitated 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 application developers. As the network grows, this complexity increases exponentially and rolling out new services becomes a tedious and complicated process.

Software Defined Networking (SDN) is an architecture which tries to address 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. 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. Through our previous work, ÆtherFlow [14], we tried to provide a protocol independent approach for bringing wireless into the SDN model. In this paper, we go a step further and try to provide a mechanism for dynamic radio resource management to obtain true network visibility in a heterogeneous network.

GNU Radio framework. GNU Radio [2] 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 enables the composition of these blocks to synthesize new radio functionality on general purpose hardware, but it 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 paper, 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.

Aside from GNU Radio, the idea of providing a programmable wireless data plane has been implemented in [4] and [3]. Both these papers provide modular blocks and focus on real time guarantees for processing signals. But like GNU Radio, they do not provide any logical interface to control a network of such programmable devices. We choose GNU Radio in our design because it is flexible, open-source, and widely used. The paper [8] deals with centralized control of devices but it focuses mainly on LTE networks. Our paper is orthogonal to these works as we provide a mechanism

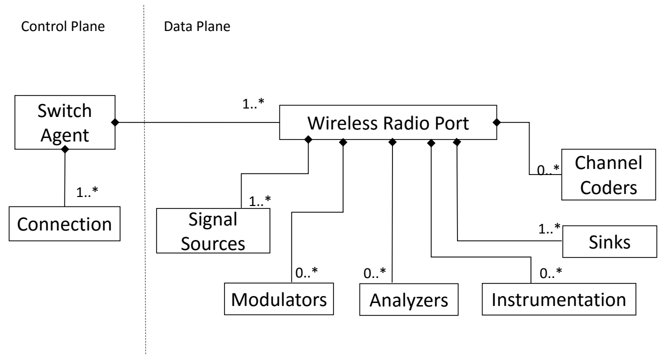

Fig. 1: Abstraction model of CrossFlow

for centralized control while making the exposed interfaces protocol independent. The combination of SDRs and SDN has recently been used in a variety of contexts [6], [13], [10], [7]. In [10], SDR and SDR are used to create a testbed for LTE technologies while [6], [13] focus on integration of SDN and SDR for 4G/5G technology. In [9], an SDR model for management of interference in dense heterogeneous networks is proposed while [7] developed a jamming architecture 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.

III. CrossFlow Architecture and Design

In this section, we motivate and describe the architecture and design of CrossFlow. In Section III-A, we introduce the proposed data plane abstractions. Then, in Section III-B, we describe how we extend the OpenFlow protocol to accommodate CrossFlow messages.

A. Data Plane abstractions

We extend the data model proposed in [5] to create an abstraction model for the CrossFlow framework, which is displayed in Figure 1. We build upon the *radio physical port* concept proposed in [14] 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 [2]. 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 this paper, we focus on the first two bullets. Specifically, we develop an abstract interface to enable event-driven dynamic configuration of signal processing blocks at run-time. We assume that the number of blocks is fixed and the blocks can be connected in a consistent manner. From an application point of view, the application creates instances of the requisite blocks. 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. One of the main requirements of the CrossFlow model is that each abstraction should implement four types of interfaces as proposed in both [5] and [14], namely: capabilities, configuration, statistics and events. Thus far, CrossFlow provides the interfaces for a wireless radio port abstraction with two processing blocks: *Sink* and *Modulators*. However, we plan to extend it to include the other processing blocks shown in Figure 1. The Sink abstraction allows the SDN 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 SDN 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 SDN controller to configure properties of signal sinks such as: (i) Gain; (ii) Frequency, and (iii) Sample rate.
- **Statistics**: The interface allows the SDN controller to gather statistics for sinks such as: (i) Received Signal Strength Indicator (RSSI) and (ii) Temperature on-board.
- **Events**: The interface allows the SDN 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 SDN controller to query the properties of the modulator block such as: (i) Modulations supported; (ii) Current samples/symbol; and (iii) Use of a Gray code indicator.
- **Configuration**: The interface allows the SDN controller to configure properties of the modulator block such as: (i) Choice of modulation scheme (e.g. BPSK, QPSK and 8PSK); (ii) Sample/symbol; and (ii) Use of a Gray code.
- **Statistics**: The interface allows the SDN 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 SDN controller to take decisions based upon events in the modulator block such

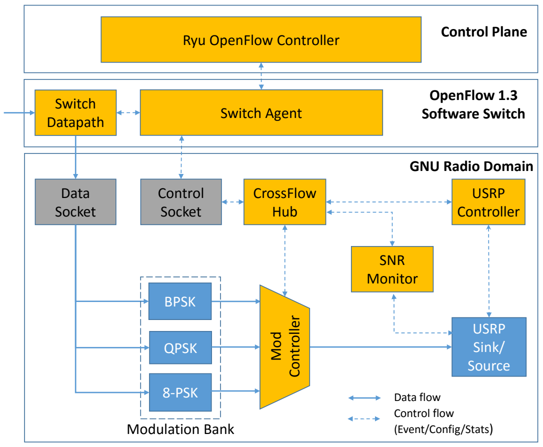

Fig. 2: Transmitter implementation diagram of CrossFlow with two processing blocks: Sink and Modulators

as: (i) Low or high SNR and (ii) Low or high BER.

B. 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 [11] 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.

IV. Proof-of-Concept Implementation

A. 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 SDR from Ettus Research. 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 2,that we implement in GNU Radio, namely, the `CrossFlow Hub`, the Modulation Controller (Mod Controller for brevity), the SNR Monitor 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. 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.

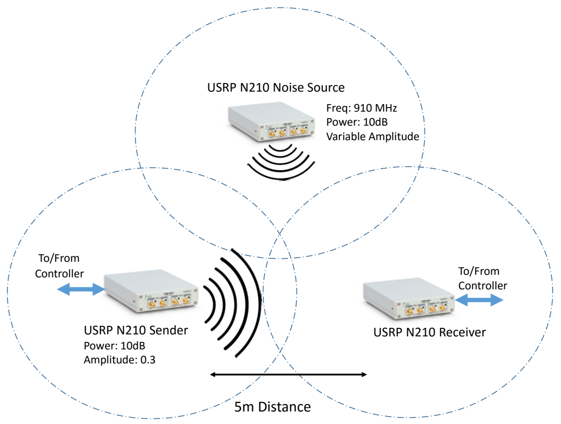

Fig. 3: Setup for cognitive radio application in CrossFlow

TABLE I: 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

- The `USRP Controller` 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 three controllers (one facilitating abstraction of the USRP Sink/RF implementation, the other facilitating abstraction of SNR estimation and another one facilitating abstraction of the adaptive modulation implementation), additional controllers can be easily added to support new applications, functionalities, and abstractions.

B. Example Applications

1) Frequency Hopping Application: Frequency hopping is a technique of transmitting radio signals by spreading the signal over a sequence of changing frequencies. It can be used against jamming and for protecting against unauthorized eavesdropping. In our implementation, the Ryu SDN controller simply issues a *GNU-CONFIG-FREQ* command with the desired frequency and pushes this configuration to the device. As shown in Figure 2, 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

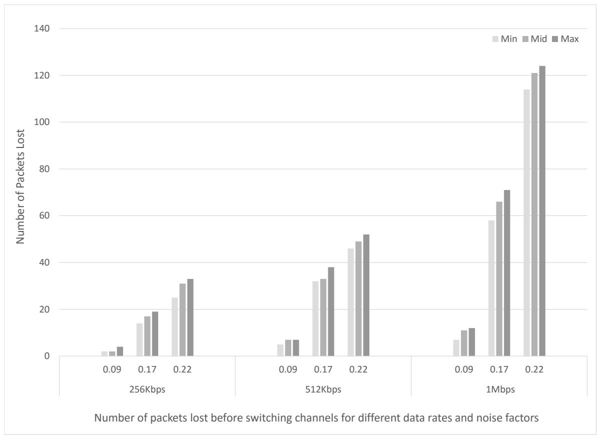

Fig. 4: Range of packets loss while changing channels by keeping a fixed data rate and a varying noise factor across each experiment

into the requested frequency.

2) Adaptive Modulation Application: Adaptive 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. 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 2, that selects the requested modulation scheme.

C. 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. The experimental setup is shown in Figure 3, 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 I 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 4, 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 4 as a separate bar.

V. Conclusion and Future Work

In this paper, we presented a framework for programming a network of software defined radios using software defined networking (SDN) principles. The framework we propose allows adaptive, flexible, and real-time (re)configuration of software defined radio interfaces from a network controller application. It streamlines the development of network applications by hiding the low level internal details of the signal processing pipeline. In order to validate our approach, we also provide three proof-of-concept applications: *frequency hopping*, *adaptive modulation* and *cognitive radio*. This shows that our design is viable and can be extended to introduce new capabilities.

One of the challenges that we need to consider is in-band control of the radio devices. Currently we implemented our design using an out-of-band wired control channel. The CrossFlow framework can easily be extended to enable in-band control of devices and it will be our next design goal. Another area of focus is the 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 other abstract radio blocks and manipulate inter-connections between those radio blocks. It requires design of new APIs 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.

References

[1] CPqD OpenFlow 1.3 Software Switch. http://cpqd.github.io/ofsoftswitch13/.

[2] GNU Radio. http://gnuradio.org/redmine/projects/gnuradio/wiki.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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. 2015.

[10] V. Mancuso, C. Vitale, R. Gupta, K. Rathi, and A. Morelli. A prototyping methodology for SDN-controlled LTE using SDR. 2014.

[11] 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.

[12] 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.

[13] 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.

[14] 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.