RcUBe: Real-Time Reconfigurable Radio Framework with Self-Optimization Capabilities

Emrecan Demirors,* George Sklivanitis,† Tommaso Melodia,* and Stella N. Batalama†
*Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115.
E-mail:{edemirors, melodia}@ece.neu.edu
†Department of Electrical Engineering, The State University of New York at Buffalo, Buffalo, NY 14260.
E-mail:{gsklivan, batalama}@buffalo.edu

***Abstract*—Existing commercial wireless systems are mostly hardware-based, and rely on closed and inflexible designs and architectures. Moreover, despite recent significant algorithmic developments in cross-layer network adaptation and resource allocation, existing network architectures are unable to incorporate most of these advancements. While software-defined radio (SDR) was envisioned as a new paradigm promising radical runtime adaptation through all layers of the networking protocol stack, the reality of the state-of-the-art in wireless networking practice is far from having fulfilled such promise of fast and intelligent reconfigurability and adaptability. Networking research based on the "software-defined radio" paradigm has suffered almost invariably from the lack of adequate and coherently designed abstractions to (i) define networking protocols and their cross-layer interactions across all layers of the protocol stack; (ii) define decision-making algorithms to control such interactions.**

To address this need, we introduce RcUBe (Real-time Reconfigurable Radio), a novel architectural radio framework based on abstractions that offer real-time reconfigurability and optimization capabilities at the PHY, MAC, and network layers of the protocol stack. Unlike state-of-the-art solutions, RcUBe offers a structured methodology at variable levels of abstraction to accommodate implementations of a wide range of network architectures and protocols and complex decision-making in a modular, platform-independent way. RcUBe provides these features through a design structured into four distinct, but interacting planes, namely *decision*, *control*, *data*, and *register* plane. The broad capabilities of the proposed framework are demonstrated on a network level software-defined radio setup through a range of experiments where RcUBe is used to implement various reconfigurable functionalities of a wireless system at the PHY, MAC, and network layer.

I. Introduction

As of today, existing wireless networking standards are inherently *hardware-based*. The majority of deployed networks are neither capable of keeping up with the quickly evolving pace of wireless standards, nor of adopting application-specific protocol designs, mainly due to their limited programmability and reconfigurability [1], [2]. As a consequence, practical deployment of new protocols or of new standards is prohibitive in terms of both cost and time. Moreover, existing architectures can incorporate significant algorithmic developments in cross-layer network adaptation and resource allocation [3]–[6] only through ad hoc tweaks.

Evidently, algorithmic developments and architectural and protocol innovation have been following parallel paths, with little cross-fertilization between the two communities. In recent work [7]–[11], the importance and need for joint reconfiguration at different layers of the protocol stack (cross-layer approach) has been highlighted and demonstrated. However, it is unclear how these developments can be incorporated in existing and future network architectures. At the same time, application-specific (e.g., QoS-constrained applications and cognitive radio (CR)/white-space networking) and context-specific (e.g., video communications) protocol designs require more flexible and reconfigurable wireless infrastructures. Reconfigurability is becoming a crucial feature for future wireless networking technologies.

978-1-4673-7331-9/15/$31.00

Yet, ever since the early days when the term software-defined radio (SDR) was introduced, researchers have lived on the promise of advanced wireless networking technologies with radical runtime reconfiguration across wide spectrum bands and full flexibility in terms of defining networking functionalities across multiple layers in software. SDRs are indeed a helpful tool when it comes to prototyping and experimenting with new concepts at the physical layer. Regrettably, however, the reality of the state-of-the-art in wireless networking practice is far from having fulfilled the promise of fast and intelligent reconfigurability and adaptability at all layers of the networking protocol stack. Significant research and development efforts in the last few years have been directed at creating experimental infrastructures based on SDRs. However, networking research based on the "software-defined radio" paradigm has suffered almost invariably from the lack of adequate and coherently designed abstractions to (i) define, with a structured methodology, networking protocols and their cross-layer interactions across all layers of the protocol stack; (ii) define decision-making algorithms to control such interactions. Thus, even essential networking functionalities such as medium access and routing often need to be implemented "from scratch" for every individual SDR-based project. Consequently, a new set of well-conceived abstractions to allow defining cross-layer protocols and complex decision-making appears to be an undeniable need.

To fill this gap, in this paper, we introduce *R*eal-time *R*econfigurable *R*adio (RcUBe), a novel architectural radio framework that offers self-optimization and real-time reconfiguration and adaptation capabilities at the PHY, MAC, and network layers of the networking protocol stack. RcUBe offers a structured methodology at variable levels of abstraction to accommodate implementations of a wide range of network architectures and protocols and complex decision-making in a modular, platform-independent way. For this purpose, RcUBe exploits the idea of decomposing protocols into primitive building blocks. The design of RcUBe is modular, in the sense that it preserves a layered protocol stack; and is compatible

with TCP/IP (although integration with TCP/IP is out of the scope of this paper).

RcUBe divides the architecture of a network node into four distinct, but interacting planes, i.e., *decision*, *control*, *data*, and *register* plane, each in charge of a different group of functionalities. In contrast to existing frameworks (e.g., based on GNU Radio [1], Wireless MAC Processors [2], or Openradio [12]), RcUBe introduces for the first time a decision plane that allows defining in a flexible way user-defined decision algorithms to perform (in real-time if needed) protocol optimization, adaptation, and reconfiguration. RcUBe separates the decision-making mechanism from execution of the protocol stack, therefore enabling definition and reconfiguration of the decision logic on-the-fly without influencing the on-going protocol execution logic. Moreover, this separation provides the capability to (i) apply decisions that are taken locally or at other devices, and (ii) create decisions to be applied at other network devices. Hence, RcUBe can be leveraged to provide a framework for networks with either centralized or distributed control. RcUBe encloses a set of abstractions that spans the three lowest layers of the protocol stack, i.e., PHY, MAC, and network, in a cross-layer fashion. Based on these abstractions, RcUBe proposes a protocol execution logic, where data-processing functionalities are decoupled from their control logic. In other words, the protocol stack is decomposed into data and control planes, where the control plane controls the sequence of data processing functionalities that will be executed in the data plane. Thanks to this decoupling, RcUBe enables to define and reconfigure protocols just by changing the sequence of execution of these data processing functionalities. In addition, RcUBe introduces a register plane that plays the role of a shared, common-among-all-planes, storage space that maintains the state variables of protocols handling functionalities at different layers of the protocol stack.

RcUBe has a fundamentally platform-independent design. In its current instantiation, it is implemented on a linux-based host PC and tested with the Universal Software Radio Peripheral (USRP) platform, a commercial, low-cost RF front-end. C++ and Python languages, supported by the GNU Radio framework, are used for PHY layer implementations, while the overall architecture is implemented in Python. Even though RcUBe takes advantage of the native modularity and flexibility of GNU Radio at the PHY Layer, it goes beyond GNU Radio by offering a structured methodology to target cross-layer protocol implementations, as well as reconfigurations across multiple layers of the protocol stack (i.e., beyond protocol parameters). Through these design choices, RcUBe offers easier programmability compared to [1], [2], [13].

We demonstrate the broad capabilities of the proposed framework through a variety of experiments on a network-level SDR setup where RcUBe is used to implement various functionalities of a wireless system at the PHY, MAC, and network layer that are optimized either independently or in a cross-layer fashion. In particular, we use RcUBe to demonstrate (i) ability to support reconfigurability at the PHY layer alone by implementing adaptive modulation schemes, (ii) ability to support reconfigurable schemes at the MAC layer alone by implementing a switching/adaptive process among different MAC protocols, (iii) ability to support reconfigurability in a cross-layer fashion by implementing joint optimal routing and spectrum allocation that requires real-time cross-layer optimization interactions.

The rest of the paper is organized as follows. In Section II, we briefly review related work. In Section III, we describe the general architecture of RcUBe and elaborate on each of its individual components. In Section IV, we discuss some key architectural considerations. In Section V, we discuss a variety of design implementation scenarios that validate the proposed framework and present experimental results. Conclusions are drawn in Section VI.

II. Background

Early Efforts. Wireless protocol stacks are traditionally implemented in hardware with a static architecture, which offers little or no room for reconfigurability. Early works, such as MultiMAC [14], FreeMAC [15], and Soft-TDMAC [16], adopted the idea of developing overlay software modules on commercial wireless cards. These works aimed to reach configurability by hacking existing drivers and exploiting firmware configuration registers. However, this approach can offer only limited flexibility due to inherited card specifications.

Software-Defined Radios. Another proposal to address reconfigurability of existing hardware promoted the use of dedicated wireless reconfigurable radios. SDRs such as USRPs that work along with the GNU Radio framework are a good example of such platforms. GNU Radio offers flexibility thanks to its software-based processing and modular design features. However, in existing SDRs, flexibility is mainly limited to the PHY layer. SORA [17] offers flexibility as well as high performance through a multi-core processing architecture, though the software complexity in reconfiguring the protocol stack is significant. On the other hand, platforms like AirBlue [18] and WARP transfer most of the processing functions to Field-Programmable Gate Arrays (FPGAs) to improve delay performance, yet they miss the abstractions to reconfigure the protocol stack. AirBlue also has a modular architecture that enables modifications at the MAC layer.

Reconfigurable MACs. Another family of recent proposals is based on the idea of decomposing wireless MAC protocols into core functional blocks that can be composed through a high-level language to enable reconfigurability [1], [2], [12], [13]. In [1], the architecture is built on USRP and the MAC functionalities are split between the host PC processing and the FPGA processing units. Hence, time-critical MAC functionalities are run on FPGA for better performance and timing control, while other control functionalities are kept in the host PC. In [13], all MAC primitive blocks are implemented on a FPGA and a PowerPC CPU is employed as the main processing unit of the architecture. In [2], the proposed architecture is built on a resource-constrained commodity WLAN card. Unlike previous work, [2] proposes the implementation of an abstract execution machine on the card which is based on conditions and events and controls the execution of primitive building blocks. The blocks are created by decomposing basic MAC protocols and defining basic MAC functions (i.e., actions). MAC protocols can be flexibly defined as extended finite state machines (FSMs) implemented on commodity wireless cards, thus enabling real-time reprogrammability, without modifying the hardware. In a follow-up work [19], extensions are proposed to reduce the run-time reconfiguration delay, and a control logic is proposed

to handle (i.e., load, move, and activate) MAC programs. The work in [12] proposes a programmable dataplane architecture for a single network node that decomposes wireless protocols into separate processing and decision (in our design similar to control plane) planes that can work on any multicore DSP processor.

Software-Defined Networks. At the network layer, OpenFlow [20] has been the primary proposal for testing experimental routing protocols on production wired networks. OpenFlow is inspired by the internal structure of an Ethernet Switch and consists of three main components, i.e., (i) an internal flow table where each flow is matched with its corresponding action, (ii) a secure channel connecting a switch with an outer controller, and (iii) the OpenFlow Protocol that offers a standard for communication between an outer controller and a switch. OpenRoads or OpenFlow Wireless [21], [22] are recent wireless extensions of OpenFlow.

III. RcUBe Architecture

RcUBe is structured into four planes, i.e., *decision*, *control*, *data*, and *register* plane, as illustrated in Fig. 1. The decision plane consists of user-defined decision algorithms that provide the control logic for real-time protocol optimization, adaptation, and reconfiguration. The required logic for routing, MAC protocol execution and data plane management is stored in the control plane. Data processing takes place in the data plane, while the register plane stores and manages access to system parameters and environmental information. As a result, RcUBe architecture decouples the decision-making mechanism from execution of the protocol stack, while at the same time separates data-processing functionalities from their control logic. In this section, each plane and their interactions are discussed in detail.

A. Decision Plane

The *decision plane* is the component of RcUBe that provides decisions (in real-time if needed) for the other planes based on user-defined decision algorithms. The decision plane is optional upon design, i.e., it can be enabled/disabled for different implementations. It is interfaced with the control and data planes through the register plane, which acts as a carrier for decisions. This process is discussed in Section III-D.

Decision Engine. The *decision engine* contains a set of user-defined decision algorithms, each associated with a sensitivity list. Each item in the sensitivity list is used to describe conditions under which the corresponding algorithm is invoked. Decision algorithms may span from simple threshold-based binary decisions, to complicated solvers for convex optimization problems (e.g., dynamic resource allocation, distributed power control). Decision algorithms also include routing algorithms that provide routing decisions for the data plane. These decisions are stored as routing tables in the register plane.

The decision engine is capable of executing multiple decision algorithms in parallel. Algorithms can be executed in a synchronous, or asynchronous fashion, as described in detail in Section IV. Decision algorithms can (i) modify a parameter in a protocol, (e.g., value of the minimum contention window (CW)), (ii) trigger switching among different modes within a protocol, (e.g., modulation scheme), (iii) enable switching among different protocols altogether. Thus, decisions are not constrained to be intra-protocol parameter modifications but

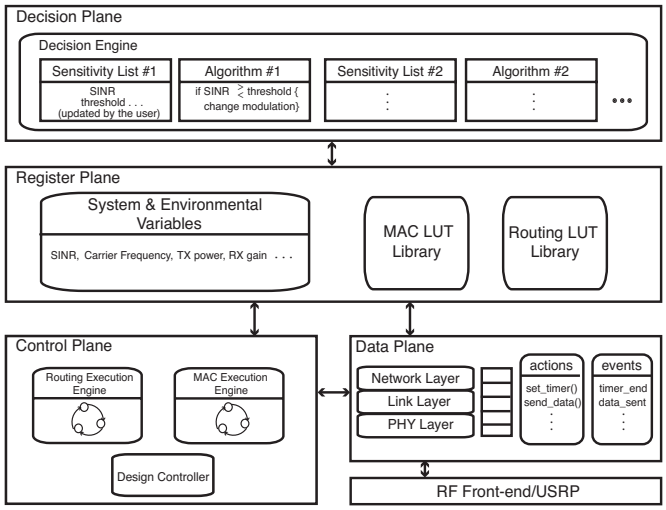


Fig. 1: RcUBe Architecture.

can also result in inter-protocol alterations. The decisions are not applied to the system by the decision engines. After decision making, the decision engine writes the decisions in the register plane. Therefore, results of specific decision algorithms can be accessed by the data and control planes and adopted in their execution logics. This will be further discussed in Sections III-B and III-C. The execution logic of the decision engine can be illustrated through the following toy examples:

(i) PHY Layer Example. Consider a decision algorithm that is defined to decide the modulation scheme of the PHY layer based on the currently perceived signal-to-interference-plus-noise ratio (SINR) value. SINR values, obtained by the data plane, and user-defined threshold value/values, stored in the register plane, are included in the sensitivity list and monitored by the decision plane. When the measured SINR value decreases below the threshold level, the optimization decision engine triggers the execution of the decision algorithm to select the appropriate modulation scheme. A parameter corresponding to the selected modulation-scheme is then written in the register plane and thus becomes accessible by the data plane. Consequently, this parameter will be accessed by the data plane to adapt execution at the PHY layer accordingly.

(ii) MAC Layer Example. Consider a network with centralized control. The central node (access point or base station) runs a decision algorithm that adaptively selects a specific MAC protocol and imposes it to the client nodes through control packets, based on the offered load to the network. For example, when the offered load is above a certain value, Time-Division Multiple Access (TDMA) based protocols like the TRaffic-Adaptive Medium Access (TRAMA) protocol are selected, since they offer better energy efficiency and channel utilization compared to Carrier Sense Multiple Access (CSMA) based algorithms. The decision process follows similar execution to the previous example, and at the last step the decision is written to a parameter that indicates MAC protocol selection. The design controller at the control plane, which will be discussed in Section III-B, detects the parameter change and loads the newly selected MAC protocol from the MAC look-up-table (LUT) library of the register plane to the MAC execution engine of the control plane.

(iii) Network Layer Example. In the case of a routing algorithm, two flag variables are defined in the register plane. These variables can be thought of as implementing a message passing technique between the control and the decision plane. One of them is used in the decision algorithm sensitivity list to invoke its execution. The second one is used to update the control plane upon completion of the execution. As a result, a routing decision is taken and is written to the routing tables that are hosted by the register plane.

B. Control Plane

The *control plane* is designed to handle the control logic of data processing, which takes place in the data plane. The control plane decides the sequence of data processing functionalities that will be executed in the data plane. This is mainly done based on two execution engines. The two engines contain and execute a FSM that implement the logic of the MAC and of the network-layer protocol execution, respectively. The logic executed in the execution engines is managed by the design controller, as shown in Fig. 1.

Execution Engines. Execution engines are designed to execute FSMs that define MAC and routing protocols. The FSMs are designed as extended finite state machines. Each FSM contains symbolic-states and an extended state transition definition. Each state transition is designed as a triplet of *events*, *conditions*, and *actions*. A transition between two states occurs if either *events* take place or *conditions* are fulfilled or both hold together. As a result of one of the above, the corresponding *actions* are invoked. *Actions* are mainly designed to trigger a data processing functionality in the data plane, while *events* may occur as a result of data processing actions. *Conditions* are controlled via the register plane, where system parameters and environmental variables are stored. To simplify the definition of MAC and routing protocols, the state transitions of FSMs are defined by LUTs. These tables are stored in the register plane and create a library of different protocols. LUTs are loaded on the execution engines to be transformed into FSMs for execution. The design idea of using FSM structures for MAC protocols is explored in [2]. Here, it is reformulated at a different level of abstraction to fit our design and includes both MAC and routing protocols.

FSMs enable reconfiguration for MAC layer by orchestrating the sequence of data processing functionalities without changing the data plane itself. Furthermore, they facilitate handling of time-critical deadlines that are crucial for the MAC protocols. Regarding the routing layer design, FSMs control functionalities that take place on the data plane such as header processing and packet forwarding. Each routing protocol has a generic FSM design, which outsources the routing decision process to the decision plane and controls it through the register plane with flag variables as discussed in Section III-A. As a result, the generic routing FSM design provides flexibility by supporting both packet and flow-based routing as will be further discussed in Section IV. Extra delays introduced by outsourcing the packet processing (e.g., header formation) to a programmable router or an external controller may also be avoided. This feature is crucial for protocol designs that follow packet-based routing decisions.

Design Controller. This component is responsible for control and management of the process of loading the routing and MAC layer protocols into the execution engines. Specific protocols are defined by the user as LUTs in the register plane. The design controller also controls switching between different protocols (pre-defined in the libraries at the register plane) on-the-fly based on the decisions taken at the decision plane. The design controller monitors parameters in the register plane that identify, which protocols need to be executed by the execution engines. These parameters can either be updated in real time by decisions taken in the decision plane, or offline by the system designer.

C. Data Plane

The *data plane* is where data/control queues are handled and data processing takes place. It also contains two libraries of pre-defined primitive building blocks, *actions* and *events*, and interacts with the RF front-end/USRP. Briefly, data coming from upper layers are received at the network layer, and after header formation at the network and MAC layer, are transformed into digital waveforms at the PHY layer, which is implemented in GNU Radio. Subsequently, digital waveforms are converted into analog waveforms by the digital-to-analog converter (DAC) in the RF front-end.

GNU Radio is an open-source software that offers a plethora of C++ digital signal processing blocks. Interaction between the blocks is implemented with Python, a scripting language that offers a higher-level of abstraction. GNU Radio also provides the capability for designing new custom signal processing blocks that can enhance the GNU Radio libraries and be re-used in various PHY layer implementations. A collection of connected blocks with a particular data flow from a *source* towards a *sink* is called *flowgraph*.

The data plane has three main design features that separate it from traditional systems. First, *RcUBe decomposes protocols into primitive building blocks*. In this way, the data plane has a modular and flexible structure and can be reconfigured just by changing the order of execution of these primitive blocks. Primitive building blocks include *actions* and *events*. *Actions* are functional blocks that perform data processing. They can be classified based on (i) transmit/receive data and control packet (e.g., ACK, CTS, RTS) functionalities, including processing of packets with headers and selecting corresponding PHY flowgraphs as subsystems, (ii) timing functionalities, (iii) routing functionalities (e.g., forwarding), (iv) data transfer functionalities between different layers. On the other hand, *events* are primitive building blocks that are responsible for indicating the results of data processing functionalities. Second, as briefly discussed in Section III-B, *RcUBe separates data processing from its control logic*. This means that the control plane only decides the execution sequence of data processing functionalities with its execution engines, while data processing itself takes place in the data plane. For instance, suppose the control plane decides to send an RTS packet to a neighbor node by invoking the *action* named *"send_RTS()"*. Then, the data plane acquires the next hop information from the routing table stored in the register plane. Accordingly, it forms the RTS packet structure at the MAC layer and transmits it through the PHY layer by using the corresponding GNU Radio flowgraph. Thanks to this separation, RcUBe is also able to define different protocols just by changing the sequence of data processing actions. Finally, *the parameters of protocols at different layers are stored as variables in the register plane*. Therefore, any

parameter of any protocol at any layer can be reconfigured on-the-fly through the register plane. Protocol parameters (e.g., inter-frame spacing, modulation, size of minimum CW) can be "intrinsically" accessed by the decision plane as a result of a decision algorithm, or by the data plane according to specific data acquisitions, or "extrinsically" by the protocol designer.

RcUBe architecture allows PHY layer processing to take advantage of the open source GNU Radio framework developed for USRP. Thanks to the granularity of GNU Radio, RcUBe framework can enable reconfigurability in three main ways. First, according to the PHY layer preferences of the MAC protocol, the control plane can switch between pre-defined flowgraphs. This is accomplished by taking advantage of an embedded capability of GNU Radio that allows *locking* the flowgraph, performing the reconfiguration and then restarting it by *unlocking* the flowgraph. The only drawback introduced by this process is a small (non-quantifiable) delay that is introduced. Second, the parameters of each block that form flowgraphs can be updated in runtime (without any extra delay) through the register plane based on the decision engine. Finally, users can simply connect different blocks to implement new PHY protocols.

D. Register Plane

The *register plane* is responsible for saving both environmental variables/readings (e.g., SINR) and state variables at the different layers (e.g., modulation, transmit power). It acts as a reference plane for the other planes (i.e., decision, data, control), and it can be updated by both the decision plane (based on results from executed algorithms) and data plane according to environmental sensed data or incoming information from neighboring nodes. The register plane also hosts two libraries of LUTs that are designed to store custom MAC and routing protocols. Therefore, users can easily create their own cross-layer protocol libraries, thus enabling fast prototyping. Moreover, the control plane may use these libraries to switch between protocols on-the-fly based on decisions taken at the decision plane. The register plane also hosts neighbor and routing tables, which may be accessed and updated by the decision and data planes based on specific protocol designs.

IV. Architectural Considerations

Asynchronous or Synchronous Decisions. In RcUBe a decision algorithm can be either executed synchronously, i.e., triggered by another plane, or asynchronously. In particular, for synchronous execution, a flag variable should be defined into the algorithm's sensitivity list in the decision plane. After execution of the corresponding decision algorithm, a different flag variable is enabled and correspondingly raise an event to inform the control plane about completion of the execution. However, sometimes the synchronous approach may not be desirable for trading the protocol stack's performance off with the delays associated with execution of the decision algorithm. In these cases, decision algorithms can be executed asynchronously without waiting for an acknowledgement from the decision plane. In this way, the rest of the system does not need to wait for the optimized parameters obtained by the decision algorithm to execute the protocol stack. Therefore, one can either choose to use asynchronous decisions to meet hard deadlines in protocol stack execution and operate with non-optimized parameters, or wait for optimized parameters at the price of extra delay.

Centralized and Distributed Control. RcUBe can be used in networks with both centralized and distributed control. In Section V, we will discuss the implementation of a complex protocol in a network with distributed control. For networks with centralized control, the following toy example may be explanatory. Consider a cluster-based network with TDMA scheduling. Here, all decisions, e.g., number of time slots to be allocated to each user, are taken by the cluster head and forwarded to cluster members through control packets. At cluster members, RcUBe can be configured with a disabled decision plane. In this way, children nodes can only decode received control packets and write enclosed information (number of slots assigned to them for the next session) to their register plane.

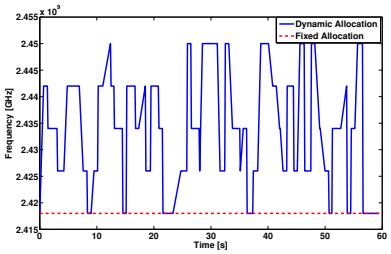
Flow/Packet Based Routing. RcUBe can perform both flow-based and packet-based routing. For the flow-based case, the routing FSM decides if a received packet belongs to an existing flow or if it is the first packet of a new flow by checking the routing tables that are placed in the register plane. If it is the first packet of a new flow then the routing FSM triggers the routing protocol execution in the decision plane. As a result a routing decision is made and a new flow is added in the routing table. Otherwise, routing FSM uses the routing data that already exist in the routing table. For the packet-based case, where routing decisions are taken on a packet-by-packet basis, the routing FSM triggers the routing protocol execution each and every time a packet is received. In this case, routing decisions are not stored in the routing tables.

Delay and Timing. In its current instantiation, RcUBe mainly runs on a host PC, and leverages the flexibility of the software-defined physical layer, therefore providing broad reconfigurability since it is not strictly limited to the capabilities of selected commodity WLAN cards. Instead, it exploits the PHY layer capabilities of the GNU Radio framework and takes advantage of its modular and flexible structure. However, pure software implementations of the protocol stack come at the expense of higher processing latency and coarser control on the timing of operations [1], [23]. RcUBe may overcome such limitations thanks to its fundamentally platform-independent design and the flexibility of its software-hardware implementation boundaries. Specifically, to address and resolve these limitations, we plan to offload/migrate software processing from the host PC to FPGA. The ultimate objective will be to combine the flexibility offered by high-level programming abstractions with the processing efficiency of hardware implementations.

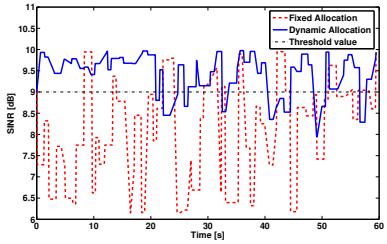
V. Using RcUBe

In this section, we discuss practical implementation scenarios that demonstrate the broad capabilities of RcUBe. We deployed an indoor SDR testbed of 6 nodes (Fig. 3). Each one consists of a Linux-PC, hosting the RcUBe framework, connected via Gigabit Ethernet (GigE) with a USRP $\mathrm{N}-210$.

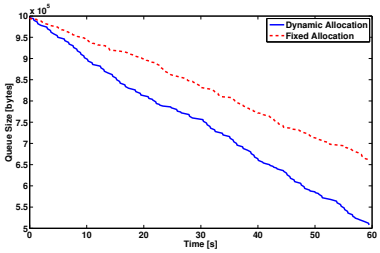
1) Scenario 1 – PHY-level reconfiguration: We define a session between N1 and N2 to demonstrate PHY layer reconfiguration. The nodes perform dynamic spectrum allocation and modulation scheme selection (i.e., Binary-Phase-Shift-Keying (BPSK) or Quadrature-Phase-Shift-Keying (QPSK)) according to SINR estimates. Specifically, the node will first



(a) Carrier frequency for dynamic and fixed spectrum allocation.



(b) SINR for dynamic and fixed spectrum allocation.



(c) Queue size for dynamic and fixed spectrum allocation.

Fig. 2: Testbed results for the 2-node setup in Scenario 1.

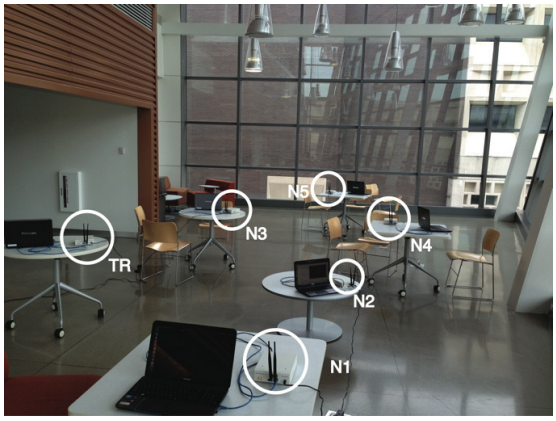


Fig. 3: Indoor testbed deployment.

decide on the carrier frequency with the highest SINR estimate and accordingly on the modulation scheme for that channel with the objective to maximize the data rate.

An Orthogonal Frequency Division Multiplexing (OFDM) scheme with 200 subcarriers is used at the PHY layer. Each OFDM subcarrier is modulated either with BPSK or QPSK. If the SINR estimate of the link is lower than a pre-defined SINR threshold of 9 dB, the algorithm selects BPSK, otherwise it selects QPSK. We define a common control channel (CCC), at a center frequency of 2.41 GHz and a data channel (DC) with five sub-channels at center frequencies of 2.418, 2.426, 2.434, 2.442, and 2.450 GHz respectively, all with a bandwidth of 1 MHz. The CCC is used by both nodes to exchange control messages and coordinate with the selected carrier frequency of DC. For this experiment, as there is no other source of interference, SINR estimates are synthetically generated to trigger dynamic spectrum allocation.

An IEEE 802.11-like MAC is implemented in the control plane as a FSM. Based on the MAC design, prior to every packet transmission, a decision algorithm that selects the carrier frequency and modulation is executed at the decision plane. The decision algorithm is triggered by an action invoked by the control plane. Upon completion of the execution, the decisions are stored at the register plane, thus becoming accessible to the data plane. At the same time an event that indicates the end of the decision algorithm is raised as discussed in Section IV. Packet transmission is driven by the control plane and carried out in the data plane by adopting the decisions on carrier frequency and modulation.

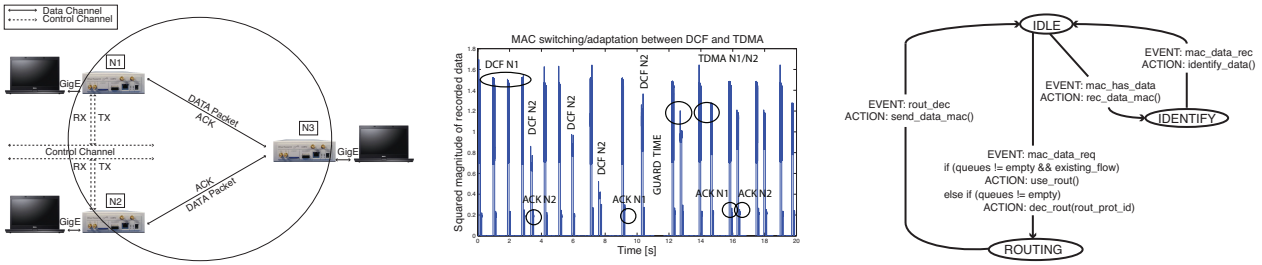
Results and Discussion. Figure 2(a) depicts dynamic and fixed frequency allocation results, while Fig. 2(b) illustrates the corresponding SINR estimates. It is clear that dynamic spectrum allocation operates on a higher SINR when compared to fixed spectrum allocation. As a result, dynamic spectrum allocation can obtain higher transmission rates as it can select higher-order modulation schemes (i.e., QPSK). This selection also affects the session queues (Fig. 2(c)), where the queue size decreases significantly faster for dynamic spectrum allocation.

In this set of experiments, we build an adaptive PHY SDR testbed by simply defining a decision algorithm and exploiting the inherent capability of RcUBe to interface with GNU Radio, instead of implementing "from scratch" the whole protocol execution logic. As a result, the flexibility of the RcUBe framework enables rapid implementation and performance evaluation of *adaptive PHY layer* algorithms.

2) Scenario 2 – MAC-level reconfiguration: Four wireless nodes, N1, N2, N3, and TR, a trace node are used to implement switching between Distributed Coordination Function (DCF) with binary exponential backoff and TDMA. Gaussian-Minimum-Shift-Keying (GMSK) modulation scheme is used by N1, N2, and N3. Data communication is established at 2.42 GHz and two different sessions are defined between N1-N3 and N2-N3. N1 and N2 initially contend for spectrum access. An acknowledgement (ACK) packet is sent in the same carrier frequency by N3 to notify successful reception of the data packets. If ACK fails, a node performs re-transmissions, thus the size of the CW is doubled. A total number of six unacknowledged data packets triggers a control message exchange between the two nodes and consequent MAC switching.

MAC protocols are implemented as FSMs that are based on pre-defined common primitive building blocks. These blocks may be used for any 802.11-like MAC protocol implementation. The LUTs for DCF and TDMA are loaded to the register plane. MAC switching is performed on-the-fly based on the decision taken by the algorithm assigned to the decision engine at the decision plane. The decision engine monitors the MAC switching metric (i.e., number of ACK fails) included in the engine's sensitivity list and stored at the register plane. A MAC switching operation is performed as soon as the user-defined condition (i.e., number of ACK fails ≥ 6) is violated.

Results and Discussion. Figure 4(a) depicts the experimental setup and Fig. 4(b) the trace data acquired from the TR node. Adaptation between the two MAC protocols is



(a) MAC-adaptive experimental 3-node setup. (b) Recorded snapshot for Scenario 2. (c) Generic FSM for routing protocols.

Fig. 4: Testbed deployment, experimental results for Scenario 2, and generic routing FSM from left to right.

conducted according to a user-defined decision metric. The power levels of the nodes N1, N2, and N3 are distinct and separable after proper positioning of the TR node. Initially, N1 and N2 are contending for the spectrum following a DCF MAC protocol. However, an ACK that is not received (though transmitted) by N2 triggers the preset user-defined condition for our decision algorithm and nodes N1 and N2 switch to TDMA. The guard time depicted in Fig. 4(b) allows transition between the two MAC protocols and helps TDMA synchronization.

Contrary to existing works, RcUBe introduces a decision making mechanism for MAC-level reconfiguration and offers a structured methodology that reduces implementation complexity and time in adaptive MAC-layer scenarios. In particular, the implementation of Scenario 2 was realized through less than ten lines of code.

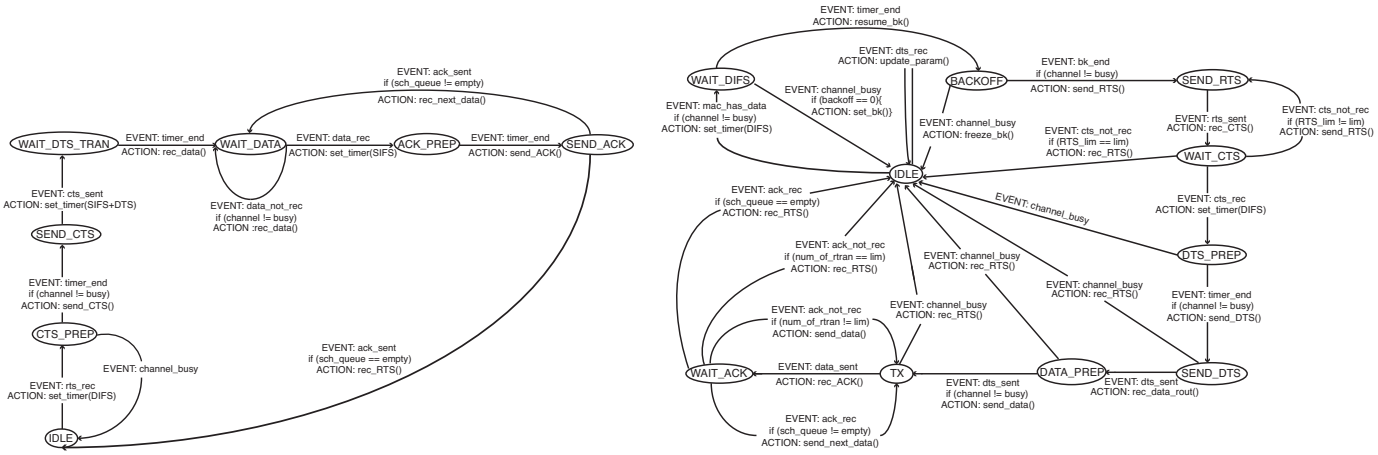
3) Scenario 3 – Network Scenario/Cross-layer Approach: In this scenario, we consider a network-level deployment of 5 nodes. We demonstrate a cross-layer algorithm implementation that performs dynamic spectrum allocation jointly with routing decisions. ROSA [8] was mainly selected because is based on a cross-layer architecture with complex interactions and employs a decentralized optimization algorithm. Moreover, it continuously relies on real-time decisions and reconfigurations as it adapts its PHY, MAC, and routing behavior according to time-varying traffic demands, network topology, and interference profile. Hence, the ROSA implementation serves as a good example for showcasing the advantages and capabilities of RcUBe's structured methodology in a network-level setup.

Definition of the Algorithm. ROSA is a CR algorithm that aims to maximize the network throughput by a cross-layer approach, through joint optimal routing, dynamic spectrum allocation, scheduling, transmit power control, and modulation scheme selection. Nodes contend for spectrum access and exchange local control information on the CCC, while they use DC's equally-sized discrete frequency bands for data and ACK packets. Each node can acquire local information via explicit exchange of control messages (by overhearing control packets exchanged on the CCC) or via spectrum sensing. Each backlogged node is able to acquire spectrum and queuing information from its neighbors and can accordingly decide on the next hop and spectrum band to be used for transmission. Apart from RTS and CTS, a control packet, named Data Transmission reServation (DTS) is defined to update neighbor nodes about spectrum reservation and transmit power.

ROSA is based on a back-pressure principle and performs joint routing, dynamic spectrum allocation, and waveform selection (i.e., modulation scheme). It is triggered whenever a backlogged node senses the CCC to be idle. Each node, source or intermediate, maintains a separate queue for each session that is defined by a fixed node-destination pair. As a first step of the algorithm, for each session, each node determines a set of feasible next hops, i.e., nodes that are closer to the destination than itself. Accordingly, for each possible link the algorithm (i) solves a link-capacity maximization problem to select optimal spectrum and power allocation; (ii) identifies the session with maximum differential backlog for that link. Then, for each possible link it calculates a spectrum-utility function metric, which is the mathematical product of the maximum capacity and the maximum differential backlog. The algorithm then selects the combination of session/link that maximizes the spectrum utility function. As a result, spectrum allocation, next hop routing and power allocation decisions are taken. Moreover, for each selected link and session, the algorithm decides a modulation scheme based on the estimated SINR of the selected channel at the transmitter node. Finally, each node selects a CW size based on the value of the calculated spectrum-utility function metric. In this way, the CW size determines the priority of the node being scheduled for transmission as compared to neighboring competing nodes. In simple terms, if a node has a larger differential backlog as well as more spectrum resources, it has a higher probability to be scheduled for transmission.

Implementation with RcUBe. The control plane contains the libraries of the routing and MAC protocols illustrated in Fig. 4(c) and Fig. 5, respectively. Since the algorithm needs per-packet routing decisions, the routing protocol FSM triggers a routing decision algorithm in the decision plane for each packet traversing the node. Moreover, it also invokes *actions* to form and handle network layer headers, in coordination with the MAC-layer FSM, based on routing information contained in the routing table in the register plane. Therefore, the routing FSM prepares packets to be sent to the MAC layer or identifies received packets from the MAC layer. The FSM design of the MAC execution engine is shown in Fig. 5 with two subfigures depicting receiver and transmitter path that are connected to each other through the shared state *IDLE*. The resulting FSM is formed based on common primitive building blocks that can be used for any 802.11-like MAC protocol.

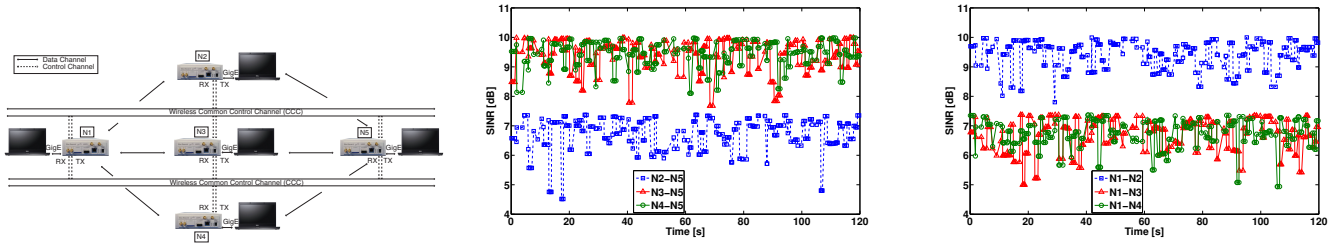
The decision plane hosts and executes the ROSA decision algorithm. To trigger ROSA's execution, the decision engine monitors an enable parameter (*"decision_enable"*) that is defined in its sensitivity list. This flag-type parameter is raised



(a) Receiver path of the FSM.

(b) Transmitter path of the FSM.

Fig. 5: Sample FSM architecture taking place in the MAC execution engine at the control plane. State transitions are based on the triplet (events, conditions, actions).



(a) Schematic overview of the 5-node testbed setup.

(b) Maximum SINR values for links [N2-N5], [N3-N5], and [N4-N5] after dynamic spectrum allocation.

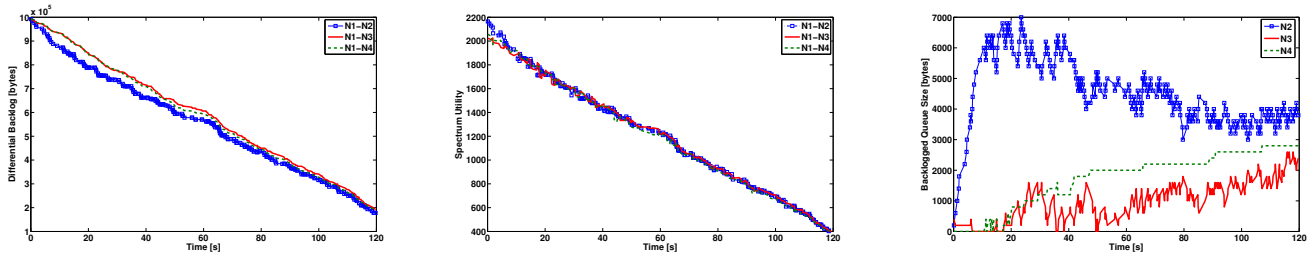
(c) Maximum SINR values for links [N1-N2], [N1-N3], and [N1-N4] after dynamic spectrum allocation.

Fig. 6: Maximum SINR values for links in the 5-node setup with 1-session between [N1-N5].

as a consequence of an action invoked by the control plane (routing execution engine) and stored in the register plane. After execution of the decision algorithm, the updated/optimized parameters are written in the register plane. Moreover, an event is raised to indicate the end of the decision algorithm execution. This event is used as a state-transition trigger in the routing execution engine. This process can be a good example of both synchronous decisions and packet-based routing decisions, as discussed in Section IV. The data plane contains the data processing of PHY, MAC, and routing layers as well as queues (sessions) for data packets. The register plane stores all the system parameters that can be accessed by other planes and also hosts the neighbor table that is predefined for the algorithm, routing table, and LUT libraries for other MAC protocols. Most importantly, the ROSA algorithm implementation can be completed by simply defining a LUT for the MAC protocol and the decision algorithm.

Results and Discussion. In this set of experiments, we define 1-session from N1-N5, active for 120 s. Based on the ROSA algorithm, each node has a pre-defined neighbor table for routing purposes. Nodes N2, N3 and N4 are intermediate nodes, only one hop away from destination node N5. Each node can overhear control packets from neighbors through the CCC. Fig. 6(a) depicts the neighbor connections.

Figure 7(a) shows the differential backlog for each source-intermediate node link. During the first 10 s, N2 gets most of the traffic load from N1 due to higher SINR in this link (Fig. 6(c)). Accordingly, higher spectrum utility is achieved because of the higher channel capacity. Moreover, since link N1-N2 has high SINR values, it uses QPSK modulation, which results in higher transmission rates. Between the 10th and 20th second, as a result of its fast start, N1-N2 reaches a lower differential backlog value compared to the other links, because of backlogged packets at N2. Therefore, the spectrum utility of all three links is balanced, as it can be observed from Fig. 7(b). Eventually, the traffic is equally shared between the intermediate nodes. After the 20th second, each one-hop link has similar spectrum utility, therefore incoming traffic from the source is equally received. However, Fig. 7(a) shows that the differential backlog value of the N1-N2 link reaches the level of others around the 120th second. This can be understood by Fig. 7(c), which depicts the backlog queue sizes of intermediate nodes. High traffic offered by the N1-N2 link and low SINR values at the N2-N5 link (Fig. 6(b)) result in a highly backlogged node N2. However, after the 20th second, as N2 is more likely to be scheduled for transmission due to the narrow contention window selection (which is directly related to the spectrum utility of the link) the number



(a) Differential backlog for links [N1-N2], [N1-N3], and [N1-N4].

(b) Spectrum utility for links [N1-N2], [N1-N3], and [N1-N4].

(c) Backlogged queue sizes of N2, N3, and N4.

Fig. 7: Testbed results for the 5-node setup with 1-session between [N1-N5].

of backlogged packets decreases. Therefore, the N1-N2 link has a higher differential backlog than before, even if it did not get more traffic. It can be concluded that each node was capable of taking decisions and reconfigure PHY (i.e., waveform selection, spectrum allocation), MAC (i.e., size of CW) and network layer parameters (i.e., next hop decisions) in real-time through a decentralized decision algorithm.

The experimental results verify that the structured methodology and design abstractions of RcUBe enable rapid implementation (less than hundred lines of code) of a cross-layer algorithm that is based on complex interactions and on the solution of a decentralized optimization algorithm. Furthermore, RcUBe facilitates for the first time the implementation of a cross-layer algorithm that requires real-time decision making throughout the three lowest layers of the protocol stack.

VI. Conclusions

In this work, we proposed and evaluated experimentally RcUBe, a novel radio framework. RcUBe offers a structured methodology at variable levels of abstraction to accommodate implementations of a wide range of network architectures and protocols and complex decision-making in a modular, platform-independent way. As a result, RcUBe accelerates experimental assessment of future research developments in either one of the three lowest layers or in cross-layer algorithms. The modular architecture of RcUBe provides users with the ability to abstractly define protocols in a high-level description language and therefore to easily implement new algorithms and protocols. RcUBe can thus simplify real-life experimental evaluation and potentially accelerate the transition of research results to the industry.

References

[1] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste, "Enabling MAC protocol implementations on software-defined radios," in *Proc. of USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, 2009.

[2] I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, and F. Gringoli, "Wireless MAC processors: Programming MAC protocols on commodity Hardware," in *Proc. of IEEE INFOCOM*, Mar. 2012.

[3] J. Tang and X. Zhang, "Cross-layer resource allocation over wireless relay networks for quality of service provisioning," *IEEE J. Select. Areas Commun.*, vol. 25, no. 4, pp. 645–656, 2007.

[4] P. Weeraddana, M. Codreanu, M. Latva-aho, and A. Ephremides, "Resource allocation for cross-layer utility maximization in wireless networks," *IEEE Trans. Veh. Commun.*, vol. 60, no. 6, pp. 2790–2809, 2011.

[5] J. Liu, C. H. Xia, N. B. Shroff, and H. D. Sherali, "Distributed cross-layer optimization in wireless networks: A second-order approach," in *Proc. of IEEE INFOCOM*, 2013.

[6] Y. Shi, Y. Hou, J. Liu, and S. Kompella, "Bridging the gap between protocol and physical models for wireless networks," *IEEE Trans. Mobile Comput.*, vol. 12, no. 7, pp. 1404–1416, 2013.

[7] S. Chen and A. M. Wyglinski, "Efficient spectrum utilization via cross-layer optimization in distributed cognitive radio networks," *Comput. Commun.*, vol. 32, no. 18, pp. 1931–1943, Dec. 2009.

[8] L. Ding, T. Melodia, S. Batalama, J. Matyjas, and M. Medley, "Cross-layer routing and dynamic spectrum allocation in cognitive radio ad hoc networks," *IEEE Trans. Veh. Technol.*, vol. 59, no. 4, pp. 1969–1979, May 2010.

[9] Z. Guan, T. Melodia, D. Yuan, and D. A. Pados, "Distributed Spectrum Management and Relay Selection in Interference-Limited Cooperative Wireless Networks," in *Proc. of ACM MobiCom*, Sep. 2011.

[10] Y. Song and J. Xie, "A distributed broadcast protocol in multi-hop cognitive radio ad hoc networks without a common control channel," in *Proc. of IEEE INFOCOM*, 2012.

[11] D. Xue and E. Ekici, "Cross-layer scheduling for cooperative multi-hop cognitive radio networks," *IEEE J. Select. Areas Commun.*, vol. 31, no. 3, pp. 534–543, 2013.

[12] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "Openradio: a programmable wireless dataplane," in *Proc. of Workshop on Hot topics in software defined networks (HotSDN)*, 2012.

[13] J. Ansari, X. Zhang, A. Achtzehn, M. Petrova, and P. Mahonen, "A flexible MAC development framework for cognitive radio systems," in *Proc. of IEEE WCNC*, Mar. 2011.

[14] C. Doerr, M. Neufeld, J. Fifield, T. Weingart, D. Sicker, and D. Grunwald, "MultiMAC - an adaptive MAC framework for dynamic radio networking," in *Proc. of IEEE Conf. on New Frontiers in Dynamic Spectrum Access Networks (DySPAN)*, Nov. 2005.

[15] A. Sharma and E. M. Belding, "FreeMAC: framework for multi-channel MAC development on 802.11 hardware," in *Proc. of ACM Workshop on Programmable routers for extensible services of tomorrow (PRESTO)*, 2008.

[16] P. Djukic and P. Mohapatra, "Soft-TDMAC: A software tdma-based MAC over commodity 802.11 hardware," in *Proc. of IEEE INFOCOM*, Apr. 2009.

[17] K. Tan, H. Liu, J. Zhang, Y. Zhang, J. Fang, and G. M. Voelker, "SORA: high-performance software radio using general-purpose multi-core processors," *Commun. ACM*, vol. 54, no. 1, pp. 99–107, Jan. 2011.

[18] M. C. Ng, K. Fleming, M. Vutukuru, S. Gross, A. Arvind, and H. Balakrishnan, "Airblue: A system for cross-layer wireless protocol development," in *Proc. of ACM/IEEE Symp. on Architectures for Networking and Communications Systems (ANCS)*, Oct. 2010.

[19] G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, and I. Tinnirello, "MAClets: active MAC protocols over hard-coded devices," in *Proc. of ACM CoNEXT*, 2012.

[20] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[21] K.-K. Yap, M. Kobayashi, R. Sherwood, T.-Y. Huang, M. Chan, N. Handigol, and N. McKeown, "Openroads: empowering research in mobile networks," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 125–126, Jan. 2010.

[22] 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 *Proc. of ACM Workshop on Virtualized infrastructure systems and architectures (VISA)*, 2010.

[23] K. Chowdhury and T. Melodia, "Platforms and testbeds for experimental evaluation of cognitive ad hoc networks," *IEEE Communications Magazine*, vol. 48, no. 9, pp. 96–104, Sep. 2010.