A SOFTWARE DEFINED RADIO MODELING FRAMEWORK FOR ENABLING COGNITIVE APPLICATION

Angelo Sapello (University of Delaware, Newark, DE, USA); Constantin Serban (Applied Communication Sciences, Basking Ridge, NJ, USA); Adarsh Sethi (University of Delaware, Newark, DE, USA); C. Jason Chiang (Applied Communication Sciences, Basking Ridge, NJ, USA); Kimberly Moeltner (CERDEC, Aberdeen, MD, USA)

ABSTRACT

The rapid advances in Software Defined Radio technologies over the past decade encouraged their transition from lab to the field deployment, and shifted the usage from disparate communication devices to the creation of large resilient wireless networks capable of adapting to highly variable environment conditions in a cognitive manner. Simulation models capable of accurately representing the behavior and properties of SDR devices operating in a networked environment thus become a prerequisite for both the evaluation of such networks as well as for providing a development platform for creating new cognitive capabilities.

This paper describes our work in creating a network simulation model framework for software defined radios that takes into account some of the unique behaviors and requirements of software defined radios not previously seen in purely hardware devices. Factors such as large and variable communication delays between software modules, as well as continuous tuning and environment awareness functionality essential to SDR demand a different modeling approach as well as novel techniques that enable accurate scale testing.

1. INTRODUCTION

Software defined radios represent one of the major advances in the area of digital communication, promising to alleviate spectrum shortages by enabling agile frequency reuse, and providing a cost-effective way to implement multiple waveform capability using a single hardware platform. One of the most attractive benefits of software defined radios is their ability to enable the creation of resilient wireless networks capable of adapting to highly variable environment conditions, and at the same time maintaining network transport capabilities in a cognitive manner. While the ultimate performance of the radio transceivers and their networking capabilities are evaluated through carefully designed field tests, highly detailed and accurate simulation models of such radios are essential for the creation and validation of cognitive algorithms capable of turning disparate radio terminals into a coherent network. Simulation models enable: i) thorough testing of the radio capabilities in situations not easily reproducible in field tests, ii) scalability of tests where large network studies are otherwise difficult to carry out as they would require a large number of radio equipment and a large amount of time, and iii) use of resources virtually, such as spectrum bands, instead of physically, without any administrative and regulatory constraint.

In this context, the creation of a simulation model capable of providing an experimentation platform for cognitive adaption functions under realistic load patterns and environmental conditions is both unique and challenging in several aspects. First, software defined radios may exhibit large communication delays between the software-defined link layer and the radio front end (hardware), a delay not present in hardware radios and often not accounted for in simulation models. This delay, introduced in the software domain and dependent on the traffic load and software waveform configuration, prolongs the time needed to process packets and increases the time needed to sense and respond to the channel conditions. This significantly affects the performance of radios in a shared environment. Second, software defined radios are often highly configurable across a wide range of parameters, capable of dynamically changing their settings during operations in an independent, radio-by-radio manner. Traditional network models, with roots in hardware devices, do not enable such tunability and composability and thus limit their applicability with respect to developing cognitive adaptation strategies. And last, traditional network models were not tested with real application data feeds from live applications. However, the software-in-the-loop simulation capability is crucial for testing and evaluating a software-defined radio network using real applications to more accurately gauge the effect of complex cognitive algorithms for the software-defined radio networks.

This paper describes our work in creating a simulation model framework for software defined radios designed to

addresses these unique characteristics. The creation of such a model framework is challenging due to several factors. First, timing issues unique to the software-defined radios have not been fully considered in existing network simulation models. While delays can be easily dealt with in general by reflecting them in event occurrence times, an accurate representation of communication timing requires a faithful representation of the software radio and the computational cost of its software functions.

Other aspects, such as the capability to satisfy arbitrary requests for information necessary to perform environment awareness functionality requires a new methodology for modeling to be both efficient, (i.e. not performing continuous computations,) and to accurately capture the effects of the computational delays. This becomes even more challenging when the parameters of individual radios can be changed dynamically, at runtime, rendering common modeling approaches such as global propagation tables unusable.

Finally, creating a model for simulator-in-the-loop in which simulations may have a long-term evolution required us to bring together a number of modeling techniques, such as obstacle modeling, interference modeling, and multichannel modeling, in a single simulation model using a dynamic control capability. These independent pieces have to fit together to cooperate with one another during a single simulation run.

The simulation model framework described in this paper was built for ns-2 (Network Simulator,) a free, open-source, and widely used network simulation platform. In regards to referent software defined radio, we used the GNU Radio [9], a free and open-source software development toolkit available under the GNU General Public License. The model framework was designed such that it can be easily applied to other software-defined radios. Our extensive software defined radio model consists of a MAC layer, a PHY layer, and a wireless channel. The MAC layer includes a model for a channel coder which can be changed during runtime. The new wireless channel model allows not only the PHY layer to operate on multiple channels, but also creates a new division of labor of modeling tasks in which errors, obstacles and distance-based power attenuation are handled in the channel rather than in the PHY layer, as in standard ns-2 wireless channel model. This new division of labor caused us to introduce a new modeling concept called "channel end-point" to handle spatially dependent computations.

This paper begins in Section 2 with an overview of our goals and the concrete devices modeled in our framework. Section 3 provides a comparison between a software defined radio architecture and a NS-2 simulation model, outlining the gaps and differences between the two. Section 3 then continues with a detailed description of the objects in our framework, indicating their interfaces and differences from the standard simulation models for hardware radios. Section 4 presents an evaluation of a GNU radio simulation model created using this framework, providing a parametric study of the model as well as fidelity and scalability data. We conclude the paper by discussing how the GNU Radio specific model could be adapted to model other software-defined radios.

2. SDR MODEL GOALS AND REFERENCE DEVICES

The main goal of this work was to create a modeling framework capable of accommodating different instances of software defined radios. The models developed under this framework are intended to: a) evaluate the impact of different functions of the software defined radios with respect to the performance of the radios in a networked environment subject to realistic environmental conditions; and b) support the development and evaluation of different runtime adaptation strategies that require reconfigurations of different components of the radio, at individual nodes across the network.

The models developed under this framework were intended to be used as follows:

a) as pure simulation deployment, where the network application and control traffic are purely synthetic, and the adaptation algorithms are entirely abstracted as finite state machine simulation modules;
b) as hybrid Software In The Loop (SITL) simulation deployment, where actual computer hosts are virtually attached to the simulation model (one per simulated node,) and can inject/receive real (IP) traffic into/from the simulation. In this usage mode, cognitive adaptation engines reside in the computer hosts where they observe and control the behavior of the simulated nodes during the course of the simulation. Details on SITL and hybrid deployments can be found in [1] and [2]. [3] also discuss how to perform network management functions in such a network.

The initial model developed under this framework was a model of a multi-channel GNU-based packet radio used for platform communication in tactical ad-hoc networks, with models of other Software Defined Radio waveforms such as WIN-T LAW and SRW soon to follow. The description in this paper is limited to the GNU radio model.

GNU Radio is a popular, open source software radio platform operating on a general purpose processor (GPP,) capable of offering an unprecedented level of re-configurability using common and accessible programming languages and paradigms. As such, it offers an ideal platform for developing cognitive algorithms where re-configurability goes significantly beyond a choice of a few presets. We used USRP II as the front-end hardware for the GNU Radio model, using a configuration more precisely described in [8] and [10].

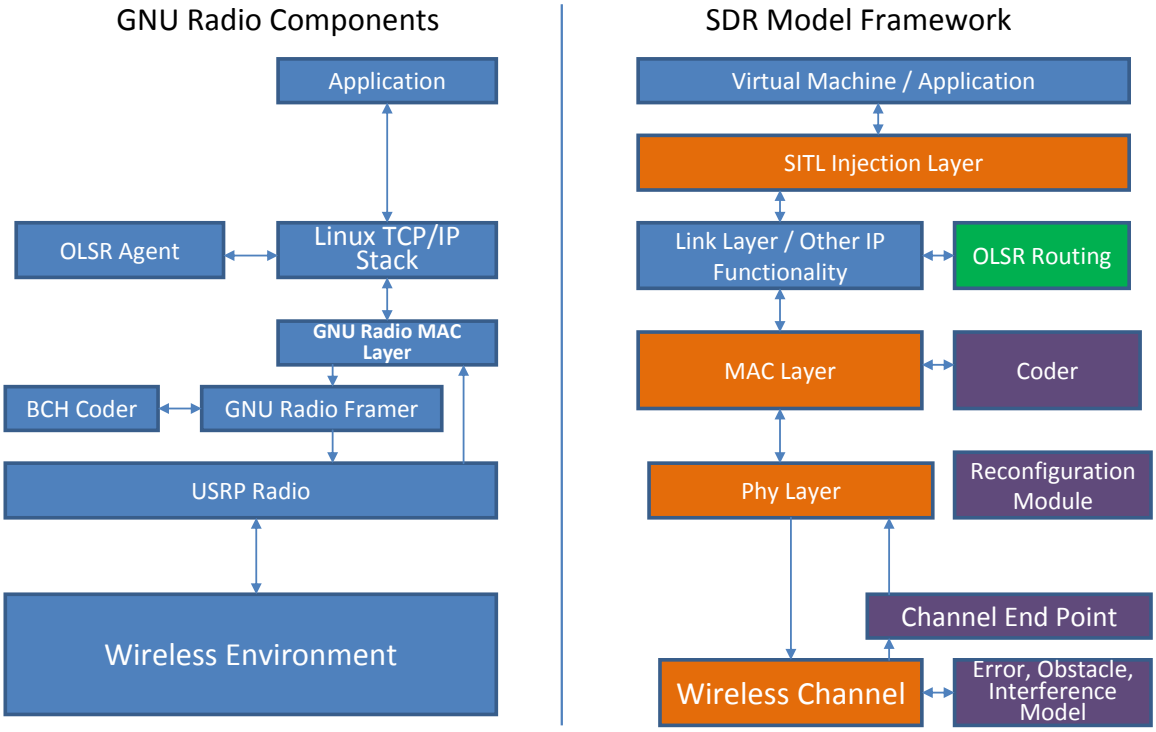

Figure 1 GNU Radio components vs. NS-2 node framework

The SDR Modeling Framework was created using the ns-2 network simulator, where various existing simulated functions such as the link layer and wireless channel could be easily adapted to an SDR framework, due to the permissive license and open-source nature.

Figure 1 shows a side-by-side comparison between the components of a GNU Radio, and the components of the NS-2 SDR framework. A GNU Radio, depicted in the left hand side, consists of a MAC layer module which handles packets from a TAP interface and transmits them via the GNU Radio *framer*. Frames are subsequently encoded and the set of symbols are then sent to the USRP radio frontend through an Ethernet connection. On the receiving path, the symbols received from the USRP frontend are passed up to the GNU radio software where several smaller channels are separated from a single set of channels. Once frames are decoded, they are sent to the application through the network TAP interface.

The right hand side of Figure 1 shows the corresponding modules in the ns-2 SDR Framework. The blue boxes represent the modules existing in the original ns-2 wireless framework; the orange boxes represent the ns-2 components that were modified to incorporate SDR functionality; the purple boxes represent the components that we added to the framework; and finally the green box (routing) represents a component that already existed, but was not integrated in a wireless ns-2 model, and thus it needed to be integrated into the framework.

3. MODELING FRAMEWORK

The right-hand side of Figure 1 shows the main components of the ns-2 SDR Framework. Applications (either traffic producers/consumers or SDR cognitive engines) can be either simulated applications, ns-2 models, or real applications executing in their virtual machines. At the top, the framework provides a software in the loop (SITL) injection layer, where a virtual machine can be attached to the SDR simulated model. This layer accepts packets from a virtual machine and injects them into the model at the appropriate node. Next, a stock ns-2 IP layer block and Link Layer block handles the IP and link layer functionality and provides routing protocol interactions, (noting here that ns-2 separates the MAC layer from the link layer). An SDR Framework MAC layer sits below the IP Link Layer module. The MAC layer creates transmission frames and schedules transmissions. Besides, it interacts with the coder for modeling interleaving and error correction, and also with the physical layer for carrier sensing functionality. The physical layer module handles the simulation of communication delays in an SDR, and the delays between the software modules and the physical front-end in an SDR

device. The physical layer sits above the wireless channel module. The wireless channel interacts with an error model, obstacle model and interference model to simulate the behavior of a real wireless environment. Finally, a reconfiguration module interacts with all other modules in order to change the functionality of the other modules (either at the parameter level or by entirely swapping a module) at runtime, and also collects parameter states in the other modules to provide a standard interface for SDR state observation.

Before diving into the details of each of the modules and their SDR specific functionality, a few remarks are made here. First, while models exist for ns-2 for most of the functionality described herein, much of it was designed for a very narrow scope, i.e. specific hardware devices, and was never fully integrated into ns-2. One shortcoming of the existing wireless model in ns-2 is that some of the modeled objects carry out tasks they would not actually carry out in the real world. As an example, one might expect to find code related to error modeling in the wireless channel code, however for the 802.11 model, error modeling is done in the physical layer code. This is done for reasons that will be explained later. This, however, required us to rework some of these interfaces. Thus, a significant portion of our effort was to re-design and modify components to provide a framework that would incorporate all of the features previously described at once. Furthermore, since many of the features had to be reconfigurable at runtime, as in an actual SDR radio with cognitive capabilities, we had to design interfaces and re-design some of the modules to support capability not provided in a typical network simulation. The remainder of this section describes each component of the model in detail.

3.1 The MAC Module

This module is responsible for simulating the media access control functionality (MAC) in the Software Defined Radio. Since the MAC functionality can be significantly different for different radios, we will not cover the specifics of the MAC layer, but will instead focus on the interfaces to the upper and lower layers.

On the transmitting side packets are received from the interface queue (a standard ns-2 object mirroring driver queues in network interfaces). The MAC module then decides whether and when it is ready to transmit, possibly by asking the PHY module for *carrier signal strength detection*. It may then loop by rescheduling the packet for transmission until it is ready to transmit (either by carrier sense detection as in CSMA, or slot interval as in TDMA), but should eventually either drop the packet (if it cannot transmit) or forward the packet to the PHY module. Before transmission, the MAC module emulates the computation responsible for frame formation. After the MAC frame is computed, the packet is also passed to the Coder module for specific error correction encoding and interleaving (if deemed to exist in the referent system), as well as channel coding. To support multichannel radio designs, the MAC module is also responsible for setting the transmission frequency of the PHY module before handing off the packet. The MAC module then informs the interface queue that it will be ready to access the next packet after the current packet is transmitted.

On the receiving path, upon receiving a coded frame from the PHY module, the MAC module first decides whether the MAC object is ready to receive a packet (computationally, and in the right state) or drop the packet. Subsequently, the MAC module schedules the decoding of the packet in iterative stages. At each stage, the Coder module decodes a portion of the packet and verifies that it had been decoded successfully, and drops the packet if any decoding error occurs. Also, the access code is checked for errors in a specialized routine, reflecting whether the access code is protected or not by channel coding. For modeling multi-channel SDRs, the MAC module also maintains per-channel state information (i.e. multiple packets being received simultaneously.) The MAC module is also responsible for registering with the MultiChannel module about the channels (frequencies) on which it wishes to listen.

The MAC module is responsible for modeling certain delays specific to SDR devices, usually not encountered in hardware devices, and normally not present in network simulator models:

- Computation and buffering delays encountered as a result of digital signal processing
- Computational and buffering delays encountered as result of carrier sensing

While the existence of processing delays is expected in a Software Defined Radio, their effect in multi-channel settings and at high data rates must be modeled to take into account the computational capability of the modeled processors. Least expected are the effects of computational delays and data buffering for carrier sensing. These delays, ranging anywhere from micro-seconds to hundreds of milliseconds, may have a large impact on carrier sense based MAC and derived protocols since they provide a false view of the channel state. This means that by the time the channel is sensed to be free by the MAC protocol, another radio may already have occupied the channel and the resulting transmission will cause a collision, significantly reducing the performance of the radio.

The MAC module provides a large number of parameters which can be changed at runtime. These include CSMA specific parameters (such as back off limits, post transmission yield times, carrier sense threshold, etc.) as well as less specific parameters that may be modified at

runtime, such as the number of channels, channel width and spacing, data rates, etc.

3.2 The Coder Module

This module simulates the encoding and decoding of packets with error-correcting codes [4] [6]. A special *dummy* coder object is provided to simulate a radio without channel coding functionality (direct translation of source bits to symbols). .

The Coder module is highly configurable object that can be configured with a different coder number, block length, coding rate, and coding data. The coding data is a set of points on the curve representing the input error rate versus output error rate for the code that needs to be modeled. The coder performs linear interpolation between these points, and an error rate prior to the first point is assumed to always result in a proper decoding, while an error rate after the last point is assumed to always result in a decoding error.

When encoding a packet, the encoder stores the coder type in the meta-header of the packet. (This header is a simulation construct that does not contribute to the simulated transmission time of the packet.) It also transforms the size of the packet to reflect the encoding ratio and possible padding.

To decode a packet, the decoder first checks whether the coder type is correct assuming that decoding would not succeed if the wrong coder was used. It then restores the original size of the packet for the higher layer objects. Finally it checks the error header to determine the errors and their distribution accumulated during transmission. Based on the number of errors, it computes the input error rate and looks up the corresponding output error rate. It then randomly generates the number of output errors using a Bernoulli distribution. If the number of output errors is greater than zero, the coder determines that the packet was uncorrectable; otherwise it returns that the packet was corrected. For the dummy coder, its encoder portion simply marks the packet as un-encoded. The decoder portion of the dummy coder checks the error header of the packet and returns an uncorrectable error if any errors are present.

The Coder module is also responsible for modeling the computational and buffering details introduced by various coding schemes (stream- or block- oriented.)

Implementation-wise, different channel coders can be active in a simulation at the same time, each initialized with different coding data. This mechanism allows for the channel coder to be easily changed at runtime by simply loading another set of coding data. This is important since many cognitive radios actually have multiple channel coding algorithms and adapt to the wireless environment.

3.3 The PHY Module

This module simulates the behavior of the software-defined radio front end (the hardware portion of the radio,) as well as its communication path between the front end and processor. As is the case of an SDR, this is a thin module that acts as the go-between the MAC module and the Wireless Channel module. This module is responsible for computing the delays in transmission, receive, and observed signal strength (carrier sense functionality) paths, caused by the digital communication path between the radio fron end and the processor.

On the sending side, the PHY module requests the MultiChannel object associated with the current transmission frequency and width from the Wireless Channel module, and it forwards the packet to the channel. On the receiving side, it schedules the packet to be received by the MAC module after a certain communication delay, and adds the packet to the received packet list. This received packet list allows the PHY module to compute the signal strength detected on the channel. The communication delay is also used for adding the packet to the receiving packet list and removing the packet from the list. This delay, similar to the one discussed in the previous section, simulates the fact that the real software always receives dated information from the hardware. In the case of the GNU Radio and USRP combination, the delay is due to the Ethernet connection between the processor computer and the USRP. Even with low level driver communication, buffers on either side of the connection create small but non-negligible communication delays between the software and the hardware.

The PHY module contains a number of reconfigurable parameters. The bits per symbol, samples per symbol, and sample frequency, etc., can be changed at runtime. Similarly, the communication delay between the hardware and the software, transmission power, radio system loss factor, and transmission frequency can also be changed at runtime.

3.4 The Wireless Channel and ChannelEndPoint Modules

The Wireless Channel module simulates the behavior of a wireless channel. A global instance of this object maintains a list of the currently simulated channels. It provides an interface to start and stop listening to a channel. For performance reasons and in order to support dynamic behavior, the Wireless module creates a channel lazily, when the first listener requests it and destroys the channel when the last listener stops listening.

The Wireless Channel module has an associated frequency, width and noise floor and maintains a list of listening PHY modules. On the downstream side, it

connects to the Error and Propagation Modules. The list of listening PHY instances are connected each through a unique ChannelEndPoint instance. This is necessary since the channel itself exists everywhere and errors and propagation delays are dependent on location. These ChannelEndPoint instances represent the channel at the location of the associated PHY instance.

On the sending side, the Wireless Channel module first records information (location, transmission power, node id) of the sending node. It duplicates the packet (over the air receipt) and schedules it to be received by each ChannelEndPoint module instance after the appropriate propagation time.

In order to support large scale scenarios, the Wireless Channel module has an optimized implementation that uses a distance table to selectively forward the packet to receivers in a fixed range of the sender. Another important modeling detail is that the sender receives its own packet from the channel: this is to provide accurate modeling of signal strength at the receiver, self interference effects in a full duplex mode, as well as the possibility of modeling STAR (simultaneous transmit and receive) for a device as well as for cognitive tuning of the noise-level thresholds.

The ChannelEndPoint module deals with the reception of packets in the channel. Upon receiving a packet, this module uses the sender information in the packet header and the receiver information associated with the ChannelEndPoint to compute the received signal strength. It then adds the received signal strength to the current sensed signal strength at the ChannelEndPoint and adds the packet to the receiving packet list to later remove the packets signal strength from the sensed signal strength. This is a separate list from the PHY module instances packet list and is used by the Error module to get the instantaneous signal power. It also updates the error information for all other packets being received on the channel to indicate that a signal change has occurred. Finally, it forwards the packet to the PHY modules to be further processed.

3.5 The Error Module

This module simulates the introduction of errors into a packet as a result of channel conditions. The current model has just a single function, called *poi* (an acronym for *point of interest*) which is called whenever the channel conditions change for a particular packet. When called on a packet, it computes the signal to noise ratio for the packet (note that at this point the packet has been duplicated and is specific to a particular receiving PHY module instance.) It then computes the number of received symbols since the last point of interest and uses the previously recorded signal to noise ratio to compute the number of errors introduced in that interval, as well as record other properties of interest for the signal for the given interval. The number of errors is a randomly generated number based on a Bernoulli distribution clamped between zero and the number of symbols transmitted in the interval with a mean of:

```
symbols * 0.5 * erfc(sqrt(sinr))
```

This calculation simulates the number of errors introduced by a binary symmetric channel with hard decision decoding. Note the separation between channel coding functionality and the error module, faithfully representing the separation of concept existing in real SDRs.

3.6 The Reconfiguration Module

The reconfiguration module enables a separate application to reconfiguration simulation parameters for a given simulated SDR node, or monitor simulation statistics, i.e. observable parameters, while the simulator is running. The goal of this module is to offer support for executing a cognitive engines that is capable of observing and adapting the behavior of the simulated SDR node. As previously mentioned, these applications can be either real applications, deployed in VMs associated with the simulation through SITL technology, or simulated applications modeled as part of the simulation.

The Reconfiguration module interacts with all the other modules in the NS-2 SDR framework, setting and collecting their properties, and providing formal interfaces for accessing the information. Currently there are two basic interfaces:

i) an interface allows an application to issue TCL commands to NS-2's TCL interpreter as if executed in a script. This works since the modified NS-2 objects support runtime configuration and statistic collection via a TCL command interface.

ii) a second reconfiguration interface allows a (VM deployed) application to issue formal SNMP requests (get and set) to the VMs which are then transformed , delayed, rescheduled, and redirected to the other modules in the framework.simulator by appending the node id to the request. The response is then sent back to the VM.

The interfaces to the reconfiguration module are one of the most important contributions to the NS-2 SDR framework, since they enable testing and development of cognitive adaptation algorithms for SDRs. Note that such functionality is not available in standard network models for two reasons. First, hardware radios are assumed to be uniform throughout a network, thus parameters affecting their behavior are generally configured and modified globally, not on a per node basis. Our framework provides local parameters that are changed or read by *co-located*

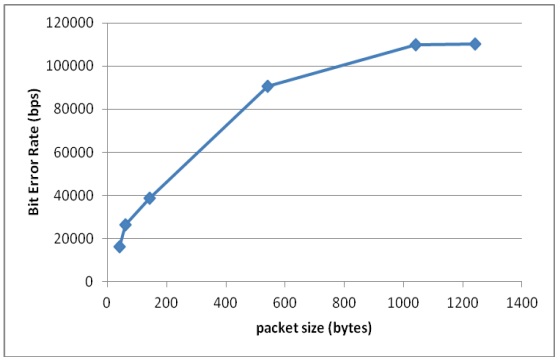

Figure 2 Packet size vs. bit error rate for the GNU Radio Model

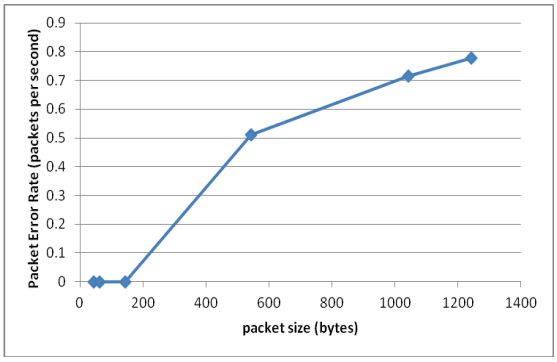

Figure 3 Packet size vs. packet error rate for the GNU Radio Model

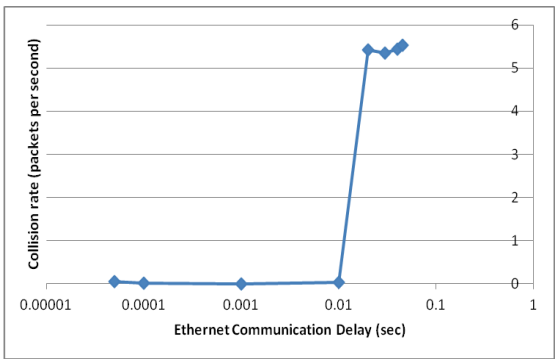

Figure 4 Hardware and software communication delay vs. packet collision rate for GNU Radio Model

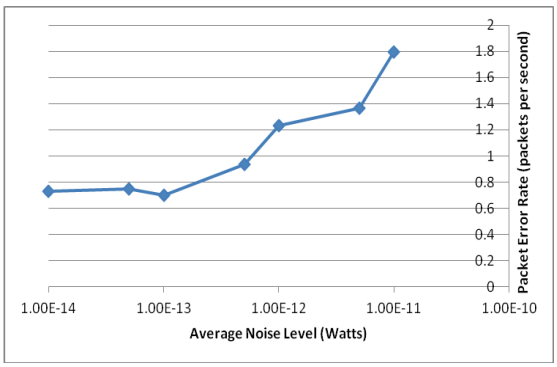

Figure 5 Average channel noise level vs. packet error rate for the GNU Radio Model

software. Second, in simulation models, parameters are commonly assumed to be constant for the duration of a simulation: accordingly, also configurable, such parameters are fixed per each simulation run, therefore not changeable on demand by adaptation engines. Our framework provides a safe mechanism for changing and reading parameters in an asynchronous manner, and taking into account the access cycle provided in a referent SDR device.

4. EVALUATION RESULTS

4.1. SDR Model Framework Parameter Study

In order to demonstrate the necessity of properly modeling key features of a referent SDR radio in the proposed NS-2 SDR Framework, this section shows how the model behavior changes with respect to these parameters. In the first experiment we vary the packet size of the CBR traffic in the simulated network and measure its effect on both bit error rate and packet error rate. In the second experiment we show how the packet error rate changes with respect to the average channel noise level. In the final experiment for this section we show how the packet collision rate is affected by changes to the communication delay between the hardware and software layers of the SDR model.

In our first experiment we created an 8-node simulation generating CBR traffic at an average rate of ten packets per second between each pair of nodes. The bit error rate was set to 120kbps and the ECC used was BCH 21/31+1 parity [5]. All other parameters are according to measured values from the actual SDR testbed. Measurements are from the perspective of a single node and the results are shown in Figure 2. As we increased the size of the packets we notice that the error rate in bits per second increases approaching a limit of the channel capacity. Considering the very noisy conditions of this experiment, this is the expected behavior.

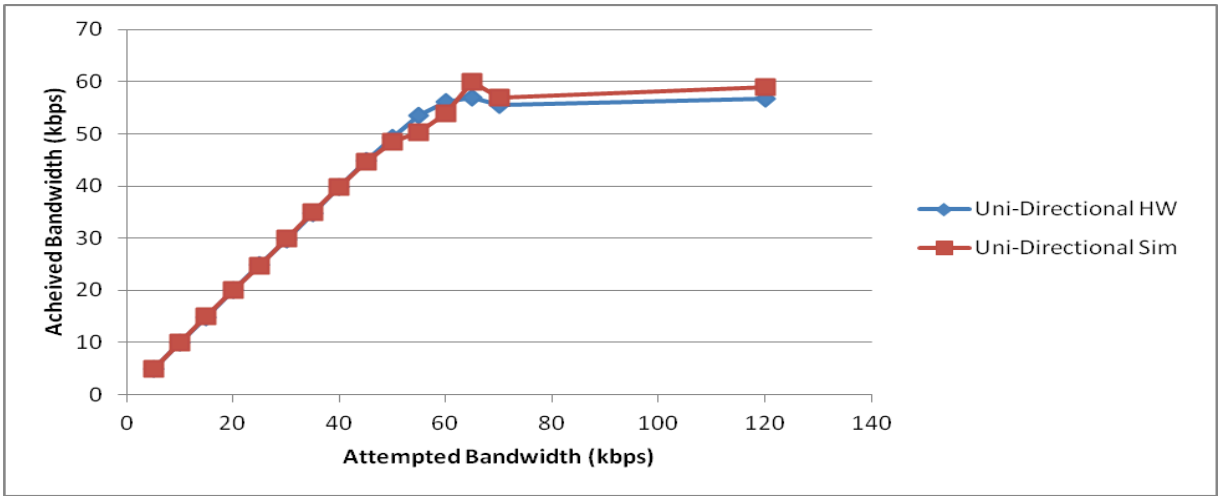

Figure 6 Attempted bandwidth vs. achieved bandwidth for a single uni-directional traffic flow on an GNU Radio/USRPII network and its equivalent simulation model.

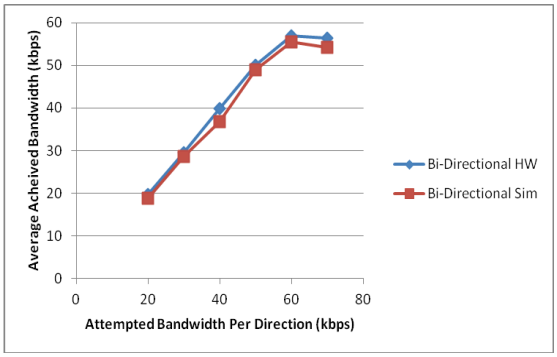

Figure 7 Attempted bandwidth vs. achieved bandwidth for a single bi-directional traffic flow on a GNU Radio net and its equivalent simulation model

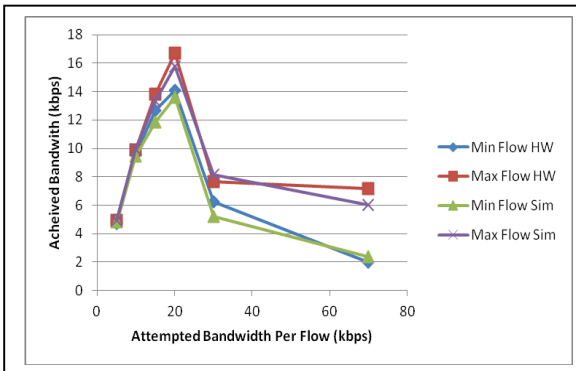

Figure 8 Attempted bandwidth vs. achieved bandwidth for multiple simultaneous traffic flows on a GNU Radio net and its equivalent simulation model

Figure 3 shows the error rate measured in packets per second, in the same settings as above. The results shown in Figure 3 are similar to the first experiment except that we initially see no errors at the packet level as the error correction code was able to correct all the packets at these smaller sizes (simulated error correction BCH 21/31).

An important modeling factor that the NS-2 SDR framework emphasizes is the sensitivity of the performance of the SDR radio to inherent processing, buffering, and communication delays. This experiment demonstrates the sensitivity of the model to adjusting the aggregate hardware and software communication delay. For this experiment we varied this delay and measured the collision rate. The results shown in Figure 4 demonstrate an interesting behavior. It seems that there exists some threshold below which the communicate delay between the hardware and software has a negligible effect on the collision rate but above which the collision rate becomes non-negligible. This clearly demonstrates the need for properly modeling such delays.

4.2. Framework Fidelity

While it is not our goal to prove that the GNU Radio model developed under the NS-2 SDR framework perfectly models our GNU Radio/USRP software hardware combo, in this section we show some results indicating that the model is to the extent measured consistent with reality. We performed three tests: i) in the first we studied the behavior of a single uni-directional traffic flow; ii) in the second we studied the behavior of a single bi-directional traffic flow; and iii) in the

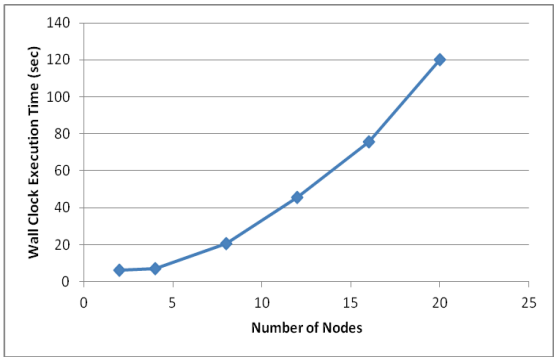

Figure 9 Number of simulated nodes vs. wall clock execution time for a GNU Radio Model simulation

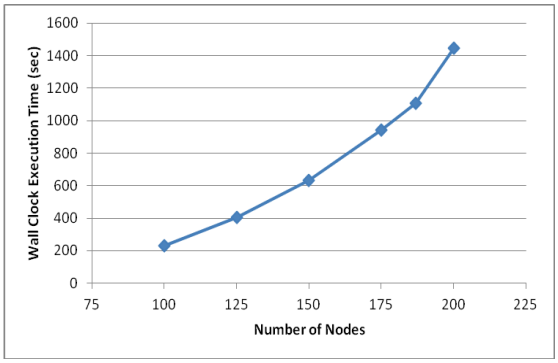

Figure 10 Number of simulated nodes vs. wall clock execution time for a GNU Radio Model simulation with optimization for large scenarios

final test we studied the behavior of multiple simultaneous traffic flows. In all tests we compared the behavior of both the NS-2 SDR model and the GNU Radio/USRPII combo.

In the first experiment we selected a pair of neighboring nodes: we set one node to be a receiver and the other a sender for '*iperf*' traffic. We throttled the bandwidth from 5kbps to 70 kbps in 5kbps increments and then performed a final run at 120kbps. The radios and their model had a 120kbps channel bit rate and used a BCH 21/31+1 parity error correcting code. As seen in Figure 6 both the actual GNU Radio/USRPII as well as its NS-2 SDR model achieved the desired bandwidth until they reached a limit of approximately 60kbps. This is consistent with the actual settings of a 2/3 coding rate and additional overhead due to various protocols in the stack.

The second experiment used a similar setting and similar rates as above, except the flows were established in both directions over the selected link. In Figure 7 we show that the combined bandwidth achieved on the two flows is approximately the same as the achieved bandwidth in the first fidelity experiment. More importantly the simulation results are very close to the results observed in the actual GNU Radio/USRP II.

In the final fidelity experiment we set up three traffic flows between three nodes. One link had a bi-directional traffic flow and the other two links had a uni-directional traffic flow. For this experiment we measured the maximum and minimum flow rates achieved in the network. While the link which had the greatest flow throughput and the least flow throughput varied from run to run, the overall behavior remained the same: one link had a dominant throughput and the remaining links suffered. Figure 8 shows that the throughput of the flows increased until reaching a maximum with a desired throughput of 20kbps per flow then dropped off quickly. This is the result of collision caused by the competition for the channel in the presence of the previously mentioned delays in sensing the carrier. More importantly, it can be observed that the simulation results match fairly well the results obtained from the GNU Radio/USRPII radios, validating our approach where various delays between functional blocks need to be taken into account and faithfully represented.

4.3. Scalability

Scaling an SDR network using actual devices can be both costly and time consuming. Such networks are difficult to maintain and even more difficult to perform controlled experiments on. Therefore, a simulation framework as provided by the NS-2 SDR framework is of utmost importance in understanding the behavior of these networks at greater scales than easily achievable with actual SDR devices.

The importance of a scalable framework is even more poignant in the case of an SDR network model, due to the extra-detailed modeling of various delays occurring in an SDR radio, as well as the re-configurability requirements for cognitive functions. Therefore, we have to apply several techniques aimed at speeding up the model at large scales. First, we created a distance map of each pair of nodes and maintained a sorted list of node distances from each nodes perspective. While this was not a new concept, we made several improvements by using hash tables and exploiting redundancies in calculating distances between nodes. Second, we took advantage of the fact that the NS-2 simulator updates the node movement information in a bulk operation, and we only sorted the distance table once upon node creation or periodically when all node positions were updated in a particular bulk operation.

In our experiments we created a single subnet within a 200 meter radius with fully connected radios and constant bit rate (CBR) traffic between all pairs of nodes. Figure 9

shows the relationship between the number of simulation nodes and the wall clock execution time of an 1800 second simulation. It is important to note that the fully connected traffic graph meant that the traffic grows as a square of the number of nodes therefore we see a quadratic growth in the simulation time.

The optimization results can be seen in Figure 10. For this experiment we deployed the nodes in a sparse network, where the spacing between nodes grows in proportion to their number. The optimized channel thus forwards packets only to nodes which are within interference range of the transmitting nodes. Each test performed 600 seconds of simulation time with a full mesh of CBR traffic flows transmitting one packet every ten seconds. Even with this extremely heavy network load, we see that the simulator performed large scale tests in nearly linear time with relation to the number of nodes.

Although the execution of large scale simulation scenarios may shows a simulation speed that may be smaller than real time, this does not invalidate our objective of executing real application and cognitive algorithms tuning the behavior of each modeled SDR device in real time. This is due to the Time Synch functionality [11], that allows the execution of entire virtual machines, including the software deployed on them, at a speed closely synchronized to that of a simulator, where VAN SITL technology is employed.

5. CONCLUSIONS

This paper presented the NS-2 SDR Framework, a framework written for the NS-2 simulator designed to facilitate the modeling of the networks consisting of SDR devices. Salient features of our framework are a detailed and explicit modeling of time delays in the most important processes encountered in a software defined radio, including but not limited to channel coding, digital transformations, carrier sense, as well as communication and buffering delays encountered between the software and hardware domains. Additionally, a reconfiguration framework was provided, an essential component of any SDR radio capable of providing runtime adaptation in a cognitive manner.

To best of our knowledge, this is the first framework that successfully provided both these capabilities, promoting further development and validation of cognitive strategies, in a high fidelity environment.

10. REFERENCES

[1] P. K. Biswas, C. Serban, A. Poylisher, J. Lee, S. Mau, R. Chadha, and C. J. Chiang, “An Integrated testbed for Virtual Ad Hoc Networks,” in Proc. TRIDENTCOM 2009.

[2] A. Poylisher, C. Serban, J. Lee, T. C. Lu, R. Chadha, and C.Y. J. Chiang. A virtual ad hoc network testbed. In International Journal of Communication Networks and Distributed Systems,Vol. X, 2010.

[3] C. Serban, A. Poylisher,and C. Y. J. Chiang, “Virtual Ad hoc Network Testbeds for Network-aware Applications,” in Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2010), April 2010. [1] Y.M. Gottlieb, C.J. Chiang, R. Chadha, H. Ohel, K. Moeltner, S. Ali, “Policy-controlled dynamic spectrum access in multitiered mobile networks,” in IEEE Military Communications Conference (MILCOM), San Jose, CA USA, Oct. 2010.

[4] A. Hocquenghem, "Codes correcteurs d'erreurs" (in French), in Chiffres, Paris, September 1959.

[5] R. Bose, D. Ray-Chaudhuri, "On A Class of Error Correcting Binary Group Codes", in Information and Control 3, March 1960

[6] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". in IEEE Transactions on Information Theory 13, April 1967

[7] T. Clausen, P. Jacquet, (editors,) “Optimized Link State Routing Protocol (OLSR)", in Network Working Group Request for Comments RFC, October, 2003

[8] F. Ge, C. Chiang, Y. Gottlieb, & R. Chadha, “GNU Radio-based digital communications: computational analysis of a GMSK transceiver,” in IEEE Global Communications Conference (GLOBECOM,) 2011.

[9] GNU Radio Project Page, available at http://gnuradio.org/redmine/projects/gnuradio/wiki , last visited July 2012

[10] C. Serban, F. Ge J.C. Chiang, R. Chadha, K. Moeltner, “A practical platform for cognitive functions in tactical edge networks.” to appear in IEEE Military Communications Conference (MILCOM), Orlando, FL USA, Oct 29-Nov 1. 2012.

[11] F. Sultan, A. Poylisher, C. Serban, J. Lee, R. Chadha, J.C. Chiang, K. Whittaker, “TimeSync: Virtual Time for Scalable, High-Fidelity Hybrid Network Emulation.” to appear in IEEE Military Communications Conference (MILCOM), Orlando, FL USA, Oct 29-Nov 1. 2012.