SANDIA REPORT
SAND2017-12715
Unlimited Release
November 2017

Emulation Platform for Cyber Analysis of Wireless Communication Network Protocols

Brian Van Leeuwen and John Eldridge

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology and Engineering Solutions of Sandia, LLC.

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@osti.gov
Online ordering:	http://www.osti.gov/scitech

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.gov
Online order:	https://classic.ntis.gov/help/order-methods/

SAND2017-12715
Unlimited Release
November 2017

Emulation Platform for Cyber Analysis of Wireless Communication Network Protocols

Brian Van Leeuwen (9315) and John Eldridge (5832)

Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS0813

Abstract

Wireless networking and mobile communications is increasing around the world and in all sectors of our lives. With increasing use, the density and complexity of the systems increase with more base stations and advanced protocols to enable higher data throughputs. The security of data transported over wireless networks must also evolve with the advances in technologies enabling more capable wireless networks. However, means for analysis of the effectiveness of security approaches and implementations used on wireless networks are lacking. More specifically a capability to analyze the lower-layer protocols (i.e., Link and Physical layers) is a major challenge. An analysis approach that incorporates protocol implementations without the need for RF emissions is necessary. In this research paper several emulation tools and custom extensions that enable an analysis platform to perform cyber security analysis of lower layer wireless networks is presented. A use case of a published exploit in the 802.11 (i.e., WiFi) protocol family is provided to demonstrate the effectiveness of the described emulation platform.

ACKNOWLEDGMENTS

The authors would like to acknowledge that this work was funded by the U.S. Department of Energy under Sandia's Laboratory-Directed Research and Development (LDRD) Program.

We also acknowledge the contribution of Benjamin Dumas who participated in the Sandia National Laboratories Summer Intern Program during the summer of 2017.

CONTENTS

FIGURES

TABLES

NOMENCLATURE

AP	(WiFi) Access Point
C2	Command and Control
CORE	Common Open Research Emulator
CPU	Central Processing Unit
DES	Discrete Event Simulation
DHCP	Dynamic Host Configuration Protocol
DMA	Direct Memory Access
DNS	Domain Name System
DOT&E	Director, Operational Test and Evaluation
DSP	Digital Signal Processor or Digital Signal Processing
DSSS	Direct-Sequence Spread Spectrum
EMANE	Extendable Mobile Ad-hoc Network Emulator
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HITL	Hardware-in-the-Loop
IoT	Internet of Things
IOT&E	Initial Operational Test and Evaluation
IP	Internet Protocol
I/Q	In-phase/Quadrature (signal components)
JTRS	Joint Tactical Radio System
LxC	Linux Container
NEM	Network Emulation Modules (EMANE modules)
MAC	Medium Access Control
M&S	Modeling and Simulation
OFDM	Orthogonal Frequency-Division Multiplexing
OTA	Over-the-Air Channel
PAN	Personal Area Network
QEMU	Quick EMUlator
RF	Radio Frequency
SDN	Software Defined Networking
SDR	Software Defined Radio
SNL	Sandia National Laboratories
Sora	Microsoft Research Software Radio
SVN	Software Virtual Network
OS	Operating System
SITL	System-in-the-Loop
SRW	Soldier Radio Waveform
TDMA	Time Division Multiple Access
VM	Virtual Machine
WAN	Wide Area Network
WARP	Wireless Open-Access Research Platform
Wi-Fi	Wireless-Fidelity (802.11 Specification wireless network)
WIN-T	Warfighter Information Network -- Tactical

1. INTRODUCTION

Wireless connectivity is integral to mobile communications for many emergency response, military and defense systems, for smartphones, and low-cost Internet of Things (IoT) deployments. IEEE 802.11 standards (i.e., WiFi) are ubiquitously used for wireless connectivity in wireless local area networking for home computer networks and for Internet access at service businesses and hotels.

Wireless systems may utilize peer-to-peer connectivity where two or more nodes form an ad-hoc network requiring no infrastructure. These systems support dynamic routing that provides multi-hop connectivity if two nodes do not have direct connectivity. Other wireless systems may be infrastructure based where nodes communicate through access points (AP) or network gateways that can provide Internet connectivity, other wired network connectivity, or bridging to other wireless nodes. Additionally, networks with combinations of multi-hop connectivity with AP(s) are also possible.

A major concern of any wireless connectivity is security. When compared to wired connectivity, wireless connectivity allows more access to the data channel. Wired systems require that an intruder physically access data cables or penetrate through an outward facing firewall. With wireless system access, an intruder only needs to be within range of the wireless network, and range can be increased with the use of gain antennas. The ease of access to wireless communication channels requires security approaches and mechanisms to secure data transmissions, thus necessitating the need for secure protocols and data encryption. Additionally, security approaches and mechanisms must accommodate more sophisticated wireless channel access techniques and control signaling.

A major challenge for developers of wireless security technologies is evaluating their efficacy on systems operating in realistic environments. Wireless communication is subject to intermittent connectivity because of signal strength variations, channel interference, and device mobility. Wireless channel disturbances should be included in the evaluation of security technologies used in wireless systems. Evaluating the security approaches of wireless system in detail requires evaluation on live systems, a high-fidelity testbed, high-fidelity system emulation or high-fidelity system simulation. Each approach has pros and cons:

Live system or operational testbed – Assessing security approaches and cyber-attacks on a live system can cause outages of that system and is not advised. Operational testbeds typically incur significant cost and lengthy creation times. The testbed will require radio frequency (RF) spectrum licensing for licensed spectrum systems. Wireless channel effects are non-deterministic and very difficult to control in a real environment.

High-fidelity system emulation – This approach offers a broad range of capability for wireless system security analysis. Actual system protocol implementations can be used in an emulated model. This approach can use various options for wireless channel representation such as an RF channel emulator or an event simulation model.

A major benefit of system emulation when compared to operational system testing is that in an operational system experiment each new system configuration or scenario would have to be set up in order to support the next scenario. Network emulation tools greatly reduce the resources needed to solve this problem. If a user is able to set up dozens of nodes in a matter of minutes, the focus can be given to the attack and defense scenario, resources can be saved, and the scenario can even be packaged up and sent to another researcher for collaboration. All of these things are what make wireless emulation a very attractive capability.

High-fidelity system simulation – This approach typically is based on a discrete event simulation (DES) model. The simulated system is comprised of behavioral models that provide a representation of system performance. Evaluating effectiveness of security protocol implementations against cyber-attack is not possible since actual implementations are not used in the simulations.

For the class of wireless cyber security analysis presented in this research paper, high-fidelity system emulation is the preferred analysis approach. System emulation use has proven effective for performing cyber security analysis of wired computer network systems [1]. Emulation tools used to create wired computer network systems are capable of producing models of systems that use actual protocols and software implementations, including node operating system (OS), applications, and services necessary for the desired cyber analysis. The analyst can construct the experiment to provide the necessary fidelity in the system components that have the greatest impact on the desired cyber analysis. In some cases, the cyber analysis requires fidelity at scale of the system under study. Emulation allows numerous virtual machines to be hosted on a single hardware server; thus a large-scale system experiment can be created with significantly less hardware than that needed for a live system experiment [1].

An example of current approaches for assessing wireless security is described in a wireless penetration testing guide [2]. The guide describes how to create a wireless testing lab using off the shelf hardware and open source software. The guide describes approaches that cause actual implementations of the 802.11 wireless LAN (WLAN) protocols to be compromised or broken. A major requirement of the approaches described in this guide is the reliance on actual software implementations and in cases actual hardware. A major drawback of the approach is that every evaluation requires the utilization of RF spectrum to provide connectivity between 802.11 radios and/or access points. In the case of 802.11, the RF spectrum is typically in the license-free ISM bands, where the radio limits it effective radiated RF power to that specified by the governing authority and a license is not required [3]. Communication systems that utilize license free spectrum are only a subset of wireless systems requiring a comprehensive analysis of its security implementation and susceptibility to cyber-attack. Also, in experimental settings care must be taken to analyze systems with many transmitters so that RF emissions from each transmitter do not interfere with the systems overall operation.

Our contribution:

Many cyber effects take advantage of specific low-level system artifacts within the wireless network's Link and Physical layer protocols. This requires that the experimental environment faithfully reproduce these artifacts.

Assessing the actual wireless communication protocols and implementations is more challenging. In our research we examine the capabilities of specific network communication emulation tools to model wireless communication systems. More specifically, we identify the capabilities and limitations to represent the wireless specific protocols and implementations at the Link layer and below. Our objectives are:

1. An emulation capability that can faithfully represent wireless Link layer protocols without including specific radio hardware or software defined radio (SDR) hardware. This objective addresses the capability to perform cyber security analysis on Link layer control messaging without actual RF transmissions between radios.
2. An analysis capability that includes visibility of the wireless network protocols using a network analyzer (i.e., Wireshark). This capability provides an analyst visibility into the wireless network control messages and the ability to assess injections of control messages. This analysis capability can be done without actual RF transmissions.
3. An analysis capability to explore cyber-attacks on wireless network protocols and assess mitigations without requiring actual radio systems and field testing. This should include an analyses capability for wireless network authentication and encryption implementations.

2. BACKGROUND

An objective of the research is the ability to apply wireless network emulation capabilities to a broad range of wireless systems including those used in advanced military systems, point-to-point command and control systems, and general use wireless network systems. Specific examples include:

Military radio systems: Joint Tactical Radio System (JTRS) Soldier Radio Waveform (SRW) which is an Internet Protocol (IP) based waveform that can interoperate with other IP based networks. The system provides a seamless network interface with existing defense network infrastructures, such as the Warfighter Information Network -- Tactical (WIN-T). WIN-T is the Army's tactical network backbone providing the satellite and terrestrial communications network that enable soldiers to send and receive information. The SRW has been assessed while under cyber-attacks. Details are in DOT&E's classified annex to the Nett Warrior IOT&E report dated May 2015 [4].

Point-to-point command and control (C2) systems: A common tool is an unmanned aerial vehicle, commonly known as a drone. Drones often include multiple types of links to enable their C2; often a WiFi links is included to enable C2 of the drone.

General use wireless networks: Include personal area networks (PAN) (e.g., Bluetooth, ZigBee), WLANs (e.g., 802.11 (WiFi)), and wide area networks (WAN) (e.g., cellular).

Our objective is to identify and develop a cyber security analysis platform that can be applied to any of the above noted classes of systems. The approach for assessing the cyber security posture of a system is similar, whether it is a military radio system or an open source, general use wireless system. We focus on a general use wireless LAN capability, specifically the 802.11 protocols, in our system descriptions throughout this research paper. These protocol standards, tools to assess them, and open source software implementations of them are generally available. These features facilitate the construction of a cyber security analysis platform.

Research environments aimed at evaluating wireless systems need to instantiate these networks in such a manner to perform repeatable experiments with a level of system complexity and realism to produce meaningful research results. Reference [5] identifies several experimental characteristics that enable research to include: repeatability, experimental control, communication layer 1-4 realism, the ability to run real applications, configurability, manageability, and scalability. Many cyber effects take advantage of specific low-level system artifacts within the wireless protocols. This requires that the experimental environment faithfully reproduce these artifacts.

3. EMULATION TOOLS FOR ASSESSING MOBILE AND WIRELESS NETWORKED SYSTEMS

Tools are available for performing networked information systems analysis. They have a primary use in assessing system performance, system architecture design, and in protocol implementation development. Modeling and simulation (M&S) tools provide the most convenient approach for assessing the operational characteristics and performance of wireless network systems. M&S is especially convenient when the number of wireless nodes become large. However, with wireless systems, simulation models pose significant challenges in representing complex waveform details and modeling RF channels that represent fading and interference. Additionally, many wireless systems incorporate node mobility that must be characterized and represented in mobility models. Simulation models incorporate significant abstractions in lower layer wireless protocols. For the type of cyber security analysis discussed in this report, simulation models do not support the fidelity and realism to enable analysis of cyber security questions. Additionally, the analyses described in this report are not significantly impacted by RF channel conditions as long as nodes have wireless connectivity.

To analyze cyber security protocols or other wireless system software components, the actual software implementation is desired for the component under study. Thus, the software that operates in fielded systems at the Link layer should be the same or nearly the same software that the analysis platform utilizes.

Emulation tools typically differentiate themselves from M&S tools in that they include some real software component from the actual system under study. The emulation tool may be a hybrid tool in that it includes a simulation component along with a real software component. In the case of a hybrid simulation and emulation, the simulation component is typically required to run in real-time.

3.1 Hybrid Emulation and Simulation Tools

Several tools that began as behavioral simulation tools have been extended to support emulation within an experiment. The tools typically call the feature a System-in-the-Loop (SITL) experiment since an emulation sub-system is included in the simulation analysis. These experiments typically interface to the emulation through a packet interface and require the packet to be parsed so that data fields can be integrated into the simulation part of the experiment.

3.1.1 Riverbed OPNET Modeler with System in the Loop (SITL)

Riverbed Modeler (i.e., OPNET) [6] is a discrete event simulation (DES) environment for performing network system analysis. Modeler includes very detailed wired and wireless protocol models that are very effective at identifying protocol standard interactions and system performance. With SITL, these models can interact with physical hardware as a unified system. Thus simulated parts of a system can affect the physical hardware and, likewise, the physical hardware can affect the simulation. The emulation and simulation components are distinguished at the node level and thus do not enable the cyber security analysis of emulated wireless protocol implementations.

3.1.2 EXata Network Emulation Software

EXata Network Emulation Software uses software virtual network (SVN) to represent the network, devices, and protocols [7]. The SVN can interoperate with real devices with its hardware-in-the-loop capabilities (HITL). The HITL and SVN components are distinguished at the node level and thus do not enable the cyber security analysis of the actual wireless protocol implementations unless they are on the real HITL radio.

3.1.3 ns-3

ns-3 is a DES intended for research and educational purposes [8]. Ns-3 has an emulation mode that keeps the simulation time aligned with the actual hardware device time or real-time. Ns-3 can use its *Tap or Emu NetDevice* to allow a "real" host to participate in an ns-3 simulation or to enable the simulation to drive real hardware. An ns-3 simulation may be constructed with any combination of simulated, *Emu, or Tap* devices [8].

Hybrid emulation and simulation approaches may provide a reasonable option for system architectures by using real and simulated nodes, or they may offer an analysis solution to evaluate actual upper layer protocols and applications with simulated wireless lower layers. However, this approach is limited in merging real wireless lower layers with simulated system components.

3.2 Emulation Tools

Emulation tools can be effective for evaluating wireless system operation. More specifically, since emulation models use, in most cases, the actual software implementation, an analysis of the system should realistically behave as the real system assuming that the hardware hosting the emulation is not introducing errors or inconstancies. Several tools for creating a wireless analysis capability with emulation are described below.

3.2.1 Click Modular Router

Click is an open source routing layer abstraction that provides the capability to integrate various Link layer functions such as those required for wireless networking [9]. Click is comprised of flexible modular packet processing elements that support many functions necessary to process packets. Click has integration with *libpcap*, support for *Tun/Tap* devices, and can run as a user process or within the Linux kernel as a module [10]. The Click router is extensible and can be used to perform actions like traffic shaping; packet filtering, dropping and insertion; and header rewriting. Click has an extensive library of elements supporting various types of packet processing. This library enables easy creation of new router configurations by selecting elements and the connections among them.

3.2.2 Software Defined Radio (SDR)

SDRs are comprised of programmable components, analog-to-digital converters, and an RF front end, and they implement the functions associated with a hardware radio in software on a computer workstation or embedded system based on field-programmable gate array (FPGA) or DSP. SDRs implement much of the radio functionality, such as mixers, filters, and modulators/demodulators, in programmable components, thus providing capability for reprogramming. The programming provides a capability to modify both Link layer and Physical layer protocols. SDRs perform the digital signal processing in the programmable component or the general purpose processor. The programming can change on the fly and thus can easily change wireless communication protocols or the waveform, the wireless Physical Layer or RF encoding of the protocol data. Due to timing limitations within general purpose processors, many SDRs use FPGAs to implement waveforms. Most SDRs with FPGAs and the necessary RF hardware are expensive, thus, construction of large scale testbeds also becomes more expensive. However, if the focus of the analysis has to do with Physical layer protocols and implementations, SDR has proven to be an effective tool.

Our research efforts initially focused on two SDR implementations. We recognize that SDRs are an effective tool for wireless system evaluation and analysis, but they have limitations when it comes to system level analysis. More specifically SDRs can quickly limit the node scale of experiments. In our research an objective was to assess wireless systems at scale. SDRs remain an important part of cyber analysis of wireless systems and are effective at answering certain classes of questions. Our research includes other tools that can be used independent of SDRs or along with SDRs. Descriptions of the tools which best met our objectives and supported development of an extensible platform that lends itself to integration with other developer tools are provided. The tools also run on Linux platforms which simplify integration with existing emulation tools and environments. Our evaluation of SDR solutions considered two SDRs.

3.2.2.1 Wireless Open-Access Research Platform (WARP)

WARP consists of an FPGA implementation and RF hardware to implement Link and Physical layer communications blocks. The FPGA-based processing boards and A/D converters are coupled to wide-band radios [11]. Algorithm implementations for 802.11 protocols are available including a flexible OFDM Physical layer.

3.2.2.2 Microsoft Research Software Radio (Sora)

Sora provides flexibility to wireless researchers to explore protocols and implementations of wireless Link and Physical layers [12]. The major differentiating factor with Sora is its approach of performing the digital signal processing on a multi-core PC running a general purpose OS (i.e., Windows). The Sora SDR uses a custom PC interface board that demodulates an RF signal and produces baseband (I/Q) signals that are initially stored in on-board memory within the interface board. Direct memory accesses (DMA) transfer the on-board memory to the PC memory for processing. Thus, much of Sora's signal processing code is executed on general purpose processors and is compiled with standard PC software compilers. This SDR provides a foundation for researching wireless protocols; however, it lacks diversity in available wireless implementations, including limited support for the 802.11 specifications. A further limit of this SDR is that it is a Windows based tool and thus lacks some interoperability with development tools that run on Linux platforms.

3.2.3 EMANE and CORE

The Extendable Mobile Ad-hoc Network Emulator (EMANE) is an open source framework for modeling wireless networked systems in real time [13, 14]. Common Open Research Emulator (CORE) is an open source emulation tool that features a graphical user interface and allows users to drag and drop nodes to create network topologies. Both EMANE and CORE are maintained by the U.S. Naval Research Laboratory (NRL). EMANE focuses on modeling the Physical and Link layers of the network stack so that system applications can be subject to the types of constraints that would be present in a real-world system. These constraints include bandwidth limits, interference, antenna profile effects, and more. While lending itself to use in simple as well as complex network architectures, EMANE aims to present an interface so that other tools (CORE, ns-3, Qualnet, etc.) might be integrated with it.

Emulated radios in EMANE are configured through plugins described in corresponding XML files. These plugins are then placed in EMANE modules called Network Emulation Modules (NEMs) that communicate via the OTA channel with multicast. Each NEM in the emulation has its own identification number that functions as its MAC address during routing. However, all EMANE instances using the same OTA channel will attempt to process every message. In the EMANE OTA channel model, a basic flow of message processing works as follows:

1. The OTA message is received,
2. A "receive power test" is performed:
   - *Tx Power + Tx Antenna Gain + Rx Antenna Gain - Pathloss > Receiver Sensitivity*
   - If the above condition holds, the message is tested further. Otherwise, the message is considered random noise.
3. The message is deemed in-band or out-of-band:
   - The registration ID of the message must match the NEM's Physical layer registration ID, the "sub ID" of the packet must match the NEM's "sub ID," and the frequency of the signal must match one of the NEM's "frequencies of interest."
4. If all tests have passed, the message is considered valid for the given NEM.

Note that many of the equations' parameters, including path loss, antenna gain, and transmission delay are configured within the XML files and based on various supported models. Other parameters such as interference are determined on a case-by-case basis based on things such as packet collisions. EMANE lends itself to as much or as little customization as the user is interested in performing.

EMANE is able to use two different types of "transports", where "transport" refers to the component that connects the emulation to the running application. Raw Transport uses the libpcap library and physically transports the message, while Virtual Transport uses TUN/TAP devices for virtual message delivery. During transportation, transport messages are sent via UDP packets and "events" are sent via multicast messages. "Events" in EMANE are the way that the environmental effects are communicated to the nodes. This allows for location, path loss, antenna profiles, and other changes to actually influence the emulation as they would in the real world.

When distributing EMANE across multiple machines and networks, the user is not limited to placing each NEM on a separate physical machine; EMANE can run easily inside Linux Containers (LxC). The key is making sure that each NEM maintains independence in its network stack while processing messages. It is important to note that EMANE runs entirely in application space; a virtual interface is installed for communicating with the kernel when necessary. See Figure 1 for a diagram of the EMANE architecture.

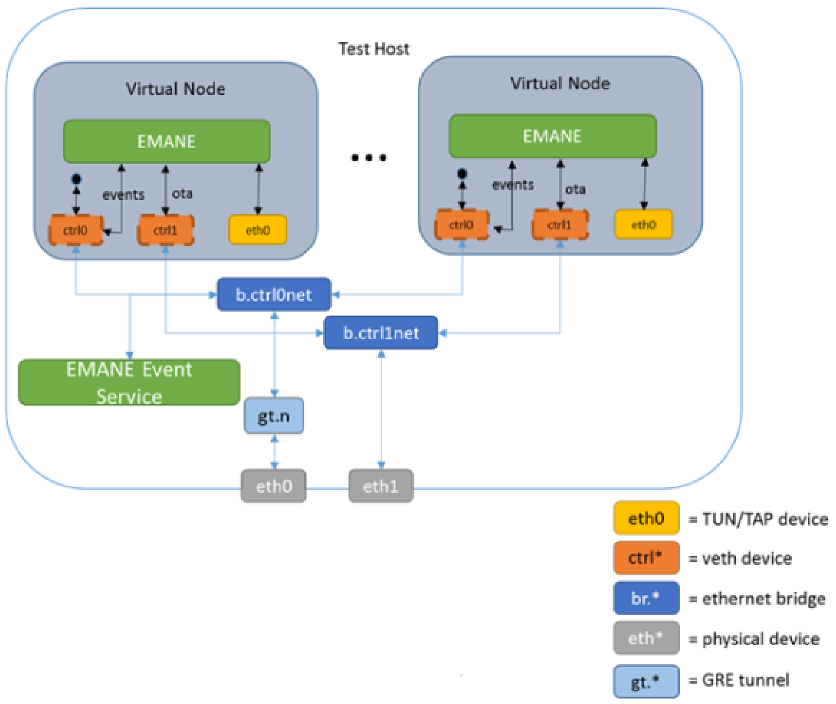


Figure 1: EMANE Architecture on Multiple Hosts [14].

CORE features a graphical user interface (GUI) that allows users to drag and drop nodes to create network topologies. Afterwards, the user is automatically able to run the emulation, ssh into nodes, and gather statistics on the network. This GUI is what makes CORE so attractive for the network emulation toolbox. CORE, in itself, does not create high-fidelity scenarios. CORE contains a daemon that will collect and maintain information about running emulations. This allows the user to switch between emulations and emulate complex networks without much trouble. The daemon listens via TCP port 4038 for incoming API messages that are used to create, delete, and change session information.

CORE focuses on emulating the application, transport, and network layers of the TCP/IP protocol stack while allowing the host OS (i.e., Linux) to control many of the finer details of bandwidth, delay, and data loss. Like EMANE, CORE is able to run multiple nodes on one machine as well as run nodes distributed across physical or virtual machines. Using a real network interface or virtual tunneled interface, CORE is even able to spread across multiple subnets while maintaining all information in a single GUI. Linux bridging is used to join CORE nodes together, and *ebtables* firewall rules are used to control connections between nodes. These components together allow CORE to maintain separate network stacks for each node as if they were independent machines.

3.2.4 GNU Radio

GNU Radio, an open source development toolkit, provides a combination of ready built signal processing blocks and tools to develop new or unique processing blocks. The tool is well suited to implement software radios, and in particular it has good, ready-built support for the 802.11 lower-level protocol processing. GNU Radio supports both RF channel emulation or it can be used with readily-available external RF hardware for over air signal emission. Models developed with GNU Radio are realized as executable code that can be readily integrated into an emulation platform.

3.2.5 Mininet-WiFi

Mininet-WiFi is a network emulation tool that extends the Mininet emulator to include wireless network capability [15]. Mininet-WiFi adds virtual WiFi stations and access points (APs) to the Mininet emulator. Through a script or command line interface, the user is able to construct network topologies in various orientations. Underlying capabilities that are available in the Mininet emulator are still available in the Mininet-WiFi extension. Mininet-WiFi uses Linux OS network namespaces to separate nodes and host processes and connects nodes through the use of virtual Ethernet pairs in a single OS. Network traffic can be shaped, scheduled, policed, or dropped using Linux traffic control. Mobility scripts can be added and run to move nodes in and out of range of each other or move a node between various APs.

Mininet-WiFi is based on *mac80211_hwsim* [16], and it makes available the protocols described in Section 4.1 (i.e., Table 1) for cyber security analysis experiments. Thus, Mininet-WiFi should provide a good platform basis for a security analysis of 802.11 wireless protocols. Mininet-WiFi uses Linux namespaces to isolate operating system resources.

4. METHODOLGY AND TOOL EXTENSIONS FOR ASSESSING MOBILE AND WIRELESS SYSTEMS

Our stated objective is creating a wireless protocol analysis platform to enable detailed cyber security analysis of wireless systems. Our focus area is the lower protocol layers that actually implement the wireless communication system functionality. Furthermore, our objective is to perform analysis without emitting actual RF signals. We also would like to perform realistic analysis for systems of increasing scale. Our evaluation of the tools introduced in the previous section resulted in identifying the CORE and EMANE combination and the Mininet-WiFi emulation tools as the most useful tools for our stated objective.

4.1 EMANE and CORE Integration

An effective integration would be for EMANE to emulate the Physical and MAC layers while allowing CORE to emulate the higher layers and maintain network stack isolation. EMANE and CORE can be integrated through the built-in "WLAN" node and this node is configured via an EMANE plugin. The result is an emulation where EMANE is in control of the network interfaces and emulation while CORE is in control of process isolation and the network topology. The user can get the high-fidelity calculations performed by EMANE as well as the GUI that comes with CORE.

When EMANE is successfully installed, CORE automatically detects and lists possible EMANE plugins such as "802.11abg" or "RF pipe". Afterwards, the configuration XML can be edited within CORE if the user is interested in a more specific emulation scenario. Within CORE, each EMANE plugin is represented as a small Python object. CORE must query this object, and it is able to display the configuration options, create the resulting EMANE XML files, and start the custom emulation. If the user wishes to integrate a new EMANE plugin, all they would need to do is write a Python object for CORE to query (the existing ones can be viewed as examples).

Once CORE gets all configuration data, the emulation starts in multiple stages. EMANE and CORE communicate via a TAP interface, and EMANE must know about each TAP interface before the emulation is started so that it can create the interface and push it into to the CORE namespace. This requires EMANE to start before CORE in the emulation process. CORE starts by showing EMANE each requested interface, starts EMANE with all configuration options, and pushes each NEM's TAP interface into the CORE namespace before starting the entire emulation. This process can become time intensive in large scenarios, so the NRL recommends that the main CORE instance run on a physical machine (as opposed to a virtual machine).

CORE and EMANE are able to share environment events that are generated. The user is required to edit the CORE configuration file to let CORE know which tool should be in charge of these events, and CORE will be able to subscribe to EMANE events or publish its events to EMANE. By default, CORE is in charge of generated events.

Currently, the following four different MAC implementations are implemented within the EMANE suite of tools: RF Pipe, 802.11a/b/g, Time Division Multiple Access (TDMA), Soldier

Radio Waveform (SRW), and other limited release models. The RF Pipe model provides a means to emulate a variety of waveforms from a behavioral model approach. The TDMA radio model implements a generic TDMA scheme that supports TDMA schedule distribution. Neither the RF Pipe nor TDMA models provide the necessary protocol implementation fidelity to enable cyber security analysis, but they are very useful for performance analysis of wireless network systems [17].

EMANE includes detailed models of various 802.11 protocols and features. More specifically it supports:

- 802.11b (DSSS rates: 1, 2, 5.5 and 11 Mbps),
- 802.11a/g (OFDM rates: 6, 9, 12, 18, 24, 36, 48 and 54 Mbps),
- 802.11b/g (DSSS and OFDM rates).

However, the 802.11a/b/g models lack several features of the full standard implementation. EMANE implements the effects of the missing features as behavior components in the model. As an example, for 802.11 unicast transmissions, EMANE does not emulate the transmission of control messages (RTS/CTS) but rather models the control message (RTS/CTS) behavior without actually transmitting the control messages. Additionally, it models 802.11 wireless frame retries rather than actually transmitting the re-transmission of the data message [17].

Cyber security analysis of 802.11 protocols in EMANE requires that the types of questions to be answered by an EMANE emulation must consider the model's implementation faithfulness to the protocol standard. If parts of the 802.11 protocol are not implemented, emulation models can be extended to implement the necessary features. The EMANE emulation models are fully open source and appear to be well documented. However, any modifications to protocol models should be tested and verified for correctness.

In the case of the EMANE SRW, significant detail has been incorporated into the Link layer functionality of the model. Initial analysis with the model indicates that the fidelity supports cyber security experiments. It appears that the various EMANE SRW wireless communication mechanisms are implemented with the necessary fidelity so that attempts to disrupt or manipulate the mechanisms will provide results similar to that of an actual system. This has not been proven since the researchers do not have access to real SRW radios for testing. An example scenario consisting of multiple SRW radios is described in Appendix A.

With EMANE emulation, source nodes create data messages that pass through the various radio protocols. As necessary, each radio protocol appends or removes headers until it reaches the Physical layer model abstraction. Metadata is also appended to support intra-layer communication as data messages travel between protocol communication layers. Ultimately, the data message is encapsulated in a multicast packet and transported over the OTA. The packets in the OTA transport the data from one node to another and also carry metadata about the transmission. Our cyber security analysis includes cases of injecting crafted data frames into a node's radio receiver. The crafted, injected packet may be a control or data packet with the objective of disrupting the receiver protocols if no security mechanism is in place to detect and prevent the reception of crafted packets. Our developments include capability to inject crafted packets into the emulated data stream as they are transported over the EMANE OTA channel.

A cyber security analysis that focuses on the wireless Link and Physical layers is most accurate when the study utilizes actual implementations of the protocol under study. Actual implementations are not always available or are hosted on difficult to obtain hardware. For the case analysis of the 802.11 wireless protocols, a common implementation is a Linux wireless device driver [19]. More specifically, consider the mac80211/SoftMac which supports most features provided by various wireless network interface cards (NICs) [19]. Previous emulation development of the 802.11 wireless capability on Linux systems created a kernel module that performs emulation to test Linux drivers; the modules is named *mac80211_hwsim* [18]. Part of the work in creating the wireless emulator for Linux was the development of a wireless channel mechanism to provide transport of wireless frames from source and destination nodes. The wireless medium emulator that resulted from this work is called *Wmediumd* [18]. A major benefit of this capability is the full feature set of 802.11 protocol implementations including those used in wireless networks with APs. Table 1 provides a list of the wireless frame types.

Table 1: WLAN Frame Types

Type of Frame	Frame Sub-Type
Management Frames	Authentication De-authentication Association Request Association Response Re-association Request Re-association Response Disassociation Beacon Probe Request Probe Response
Control Frames	Request to Send (RTS) Clear to Send (CTS) Acknowledgement (ACK)
Data Frames	Data

Cyber security analysis of wireless systems must consider that manipulation of any of the frames noted in Table 1 can lead to a compromised network [20]. The compromise may impact availability via a denial of service attack or confidentiality and/or integrity with an attack on the transported data. In an effort to protect the data from attack or eavesdropping, authentication and encryption mechanisms were added to the 802.11 functionality. The following protocols for data encryption are available:

- Wired Equivalent Privacy (WEP)
- WiFi Protected Access (WPA)
- WiFi Protected Access v2 (WPA2)

For cyber security analysis, these protocols and their implementations should be included in the emulation experiment.

4.2 SDR Implementations

Wireless communication systems implement the lower layers of the communication protocol by a combination of means to include software, firmware, and hardware. Cyber security effects can emerge from each of these means and may be more or less dependent on the specific implementation. In general, it is more practical to extract software implementation methods from a wireless design for emulation on a general purpose compute platform rather than an FPGA hardware description language (HDL) or a unique hardware implementation. Our initial work on creating an 802.11 wireless emulation capability followed this approach for emulation of the MAC protocol operation. The early working premise was to utilize existing and available 802.11 software implementations and to repackage them with well-defined data passing interfaces for emulating the MAC Layer functions. Both software defined radio (SDR) projects Sora and WARP openly provide software and hardware bases for 802.11 wireless processing. Incorporation of either into a general purpose emulation proved challenging.

4.2.1 Wireless Open Access Research Platform (WARP)

WARP is a scalable and extensible programmable SDR platform designed for prototyping advanced wireless networks. The platform is general purpose and suitable for a variety of RF designs. One such design is an 802.11 reference architecture for which all firmware and software are available for review and reuse. At the core of the WARP platform is a field programmable gate array (FPGA), analog-to-digital and digital-to-analog converters, and RF gain and driver components. Figure 2 presents the block architecture of the WARP 802.11 reference design.

The reference design segments the 802.11 protocol processing into embedded microcontroller software and FGPA HDL code. The microcontrollers, which perform the bulk of the 802.11 MAC protocol processing, are implemented as a soft-core processors entirely in the general-purpose memory and logic fabric of Xilinx FPGAs. The platform specifically utilizes Xilinx's MicroBlaze soft microprocessor cores. The MAC layer protocol functions are written as embedded C software that runs directly on the microprocessor without an operating system. The reference design splits the protocol processing and embedded C software across two microprocessors to handle the protocol's real-time requirements and processing loads. Data passing and event signals between the microprocessors and the underlying FPGA code are through common direct memory transfer addresses within the FPGA's memory space.

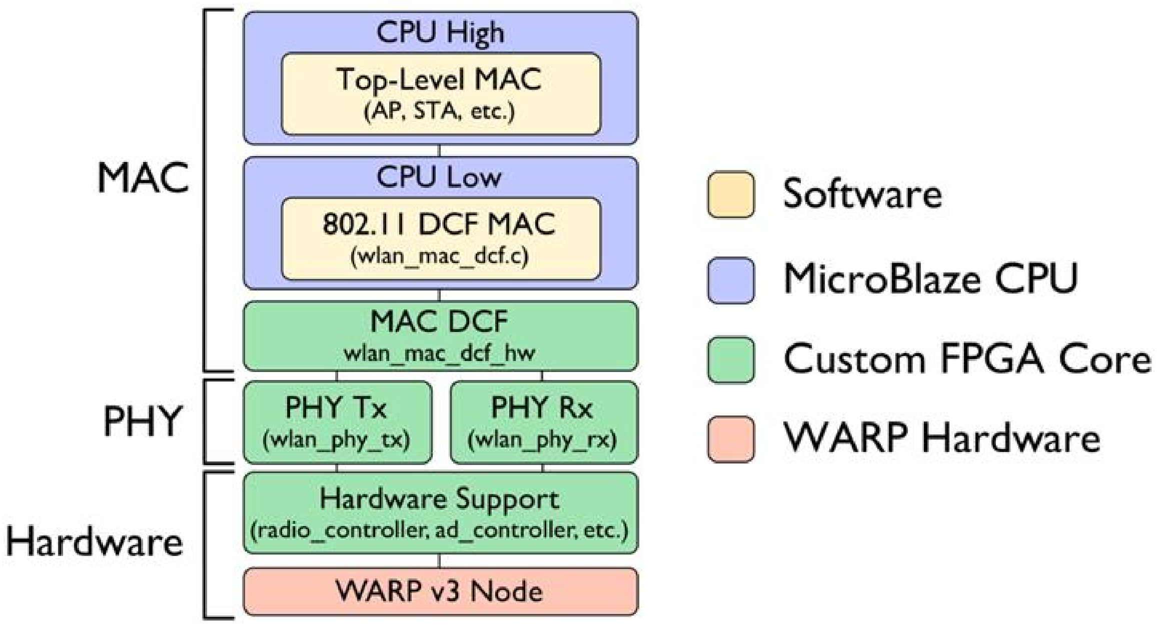


Figure 2: WARP 802.11 system architecture and design [21].

For the purpose of developing a general purpose 802.11 emulation environment, we were only interested in and worked with the embedded C code as designed for the soft-core microprocessors. This code implements the upper-level of the MAC protocol. The approach was to start with this C code base and to modify it for execution as a standalone program on a general purpose compute node.

We explored two means of utilizing the C code. One was to take the code as written and then modify it for execution on a general purpose computer. The other method was to execute the code exactly as written within a QEMU virtual machine emulation of the MicroBlaze soft processor. Since the software is structured as embedded code and tailored for direct execution on the processor, it contains a number of constructs that we would need to alter. In particular, the code is partitioned for operation across two separate processors utilizing a common memory space between the processors. Modifying the code would have required the merging of the two code bases into a single executable and to then replace the processor-to-processor data passing sections. There were also a few sections of the code that were heavily dependent upon the MicroBlaze and FPGA design which would need to be changed. Execution of the code within a QEMU processor emulator will run in that environment with fewer changes. However, it would have also required us to craft QEMU emulator additions to move data into and out of the QEMU environment for integration with other aspects of the overall emulation.

WARP provides an SDR based analysis tool that is effective at meeting the timing constraints of wireless protocol implementations. It has several specific protocol implementations available that can be accessed and modified by the developers. However, it requires implementation in the FPGA. The next section describes Sora that attempts to mitigate the use of FPGAs and run the protocol on a multi-core general purpose processor.

4.2.2 Microsoft Research Software Radio (Sora)

Microsoft researchers crafted the Sora SDR code to execute as an application on a general purpose computer using a minimum of off-board hardware support for timing critical functions and final RF signal generation. Sora's architectural design seeks to make it easier for software developers to experiment with SDR in a general purpose computing environment. The majority of the Sora 802.11, and in particular the MAC layer, processing occurs within the application's user space under Windows. Sora partitions the application into user and kernel space portions where the kernel portion of Sora implements a wireless network driver for Windows. The kernel portion of the code also manages CPU core utilization. Sora requires execution on a multicore CPU and sets minimum processor capability requirements. Since the SDR computational loads are heavy and the process timings are critical, Sora allocates CPU core(s) to exclusive use for 802.11 SDR processing. Further, Sora is explicitly written for execution under MS Windows and as 32-bit code. Both the exclusive allocation of CPU cores for SDR processing and the MS Windows specific nature of the code made it difficult to extract code sections for specific 802.11 processes for inclusion into a more generic emulation environment.

A limitation of Sora was the lack of available protocol implementations. At the time of this research only a limited 802.11 protocol implementation was available. None of the AP services were available. Since our research has an objective of leveraging existing code and actual implementations in the wireless platform Sora had limited utility.

4.2.2.1 Colombo

An independent Microsoft research group based in Cambridge, UK has taken the Sora application code, modified it, and released the Colombo SDK [22, 23] for use as a continuous time simulator. Colombo decouples the Sora 802.11 MAC layer application code from its dependencies upon and interaction with Sora kernel level code and the associated RF processing hardware. In effect Colombo, takes a first step in converting the Sora SDR application into a user space only application, suitable for execution entirely on a general purpose compute platform. The Colombo project extracted from Sora the relevant software modules necessary for 802.11 MAC layer processing. The project then wraps these modules in a small simulation layer that handles MAC layer packet and event processing. In practical 802.11 systems, packet and event processing occur in real time and within tightly controlled timing constraints. While Colombo accurately simulates that packet and event processing, it does so under non-real time conditions. It maintains an internal time reference. At each simulation event, Colombo increments the time reference by a fixed time step and checks for additional processing events. It is possible to alter the incremental time step, however the time step must be maintained smaller than the minimum timing constraints within the 802.11 protocol. In operation, the time step is typically 10 microseconds or less which results in relatively long simulation runs for even a few seconds of simulation time. Long simulation runs of minutes or hours of simulated time are computationally expensive and may be difficult to accommodate in larger simulation/ emulation experiments.

SNL researchers modified the Colombo SDK to include programming interfaces for passing emulated/simulated network traffic into and out of Colombo for 802.11 upper-level MAC layer processing. The architectural goal of the changes being the evaluation of feasibility for creating a complete set of interconnected emulation modules where each module would handle portions of the 802.11 processing for the various communication layers within the protocol. The programming interfaces are implemented as network sockets for operation across a collection of network connected compute nodes.

4.3 GNU Radio Integration

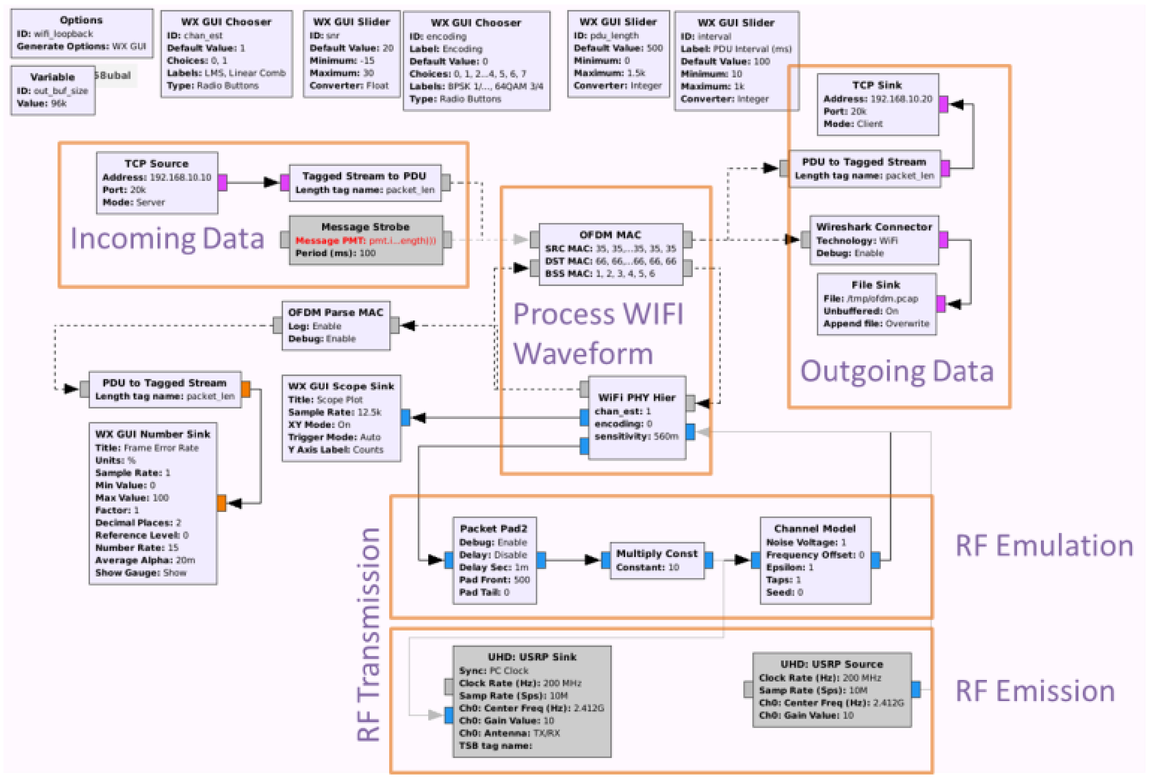
A protocol module for 802.11 lower-level processing was modeled within GNU Radio. In Figure 3 an annotated screen capture of the model design as built with the GNU Radio companion graphical design tool is illustrated. The GNU Radio model establishes two TCP socket interfaces:

1. Accepting 802.11 MAC packet data into the model, and
2. Returning processed packets out of the model.

Packet data enters the model through the blocks label as “Incoming Data.” The model then passes data to the “Process WIFI Waveform” blocks where it appends link level addresses for the source, destination, and intermediate transceivers. These blocks also partition the data across carrier sub-channels and encode the sub-channel data for selected physical layer encoding. After physical layer encoding, the model will pass the data through an RF medium in the blocks labeled as “RF Transmission.” Two options are available at this point:

1. Fully emulate the RF channel within the model so as to avoid all over-the-air transmissions and the hardware cost associated with those emissions. With this option, it would be possible to add model artifacts that could interfere or interact with the RF signal.
2. The second option is to place RF transmitting equipment in the loop to interface the model to a real RF transceiver(s) outside of the emulation model.

After the packet data transits the RF Transmission path, the model decodes the packet data by returning it to the blocks for WiFi Waveform processing. Packet data is extracted from the MAC layer encoding then passed back to the originating application through the outgoing TCP socket implemented in the blocks labeled “Outgoing Data.”



Options
ID: wifi_loopback
Generate Options: WX GUI

Variable
ID: out_buf_size
Value: 96k

WX GUI Chooser
ID: chan_est
Default Value: 1
Choices: 0, 1
Labels: LMS, Linear Comb
Type: Radio Buttons

WX GUI Slider
ID: snr
Default Value: 20
Minimum: -15
Maximum: 30
Converter: Float

WX GUI Chooser
ID: encoding
Label: Encoding
Default Value: 0
Choices: 0, 1, 2...4, 5, 6, 7
Labels: BPSK 1/..., 64QAM 3/4
Type: Radio Buttons

WX GUI Slider
ID: pdu_length
Default Value: 500
Minimum: 0
Maximum: 1.5k
Converter: Integer

WX GUI Slider
ID: interval
Label: PDU Interval (ms)
Default Value: 100
Minimum: 10
Maximum: 1k
Converter: Integer

TCP Source
Address: 192.168.10.10
Port: 20k
Mode: Server

Tagged Stream to PDU
Length tag name: packet_len

Message Strobe
Message PMT: pmt.i...ength())
Period (ms): 100

OFDM MAC
SRC MAC: 35, 35,...35, 35, 35
DST MAC: 66, 66,...66, 66, 66
BSS MAC: 1, 2, 3, 4, 5, 6

TCP Sink
Address: 192.168.10.20
Port: 20k
Mode: Client

PDU to Tagged Stream
Length tag name: packet_len

Wireshark Connector
Technology: WiFi
Debug: Enable

File Sink
File: /tmp/ofdm.pcap
Unbuffered: On
Append file: Overwrite

OFDM Parse MAC
Log: Enable
Debug: Enable

PDU to Tagged Stream
Length tag name: packet_len

WX GUI Scope Sink
Title: Scope Plot
Sample Rate: 12.5k
XY Mode: On
Trigger Mode: Auto
Y Axis Label: Counts

WiFi PHY Hier
chan_est: 1
encoding: 0
sensitivity: 560m

WX GUI Number Sink
Title: Frame Error Rate
Units: %
Sample Rate: 1
Min Value: 0
Max Value: 100
Factor: 1
Decimal Places: 2
Reference Level: 0
Number Rate: 15
Average Alpha: 20m
Show Gauge: Show

Packet Pad2
Debug: Enable
Delay: Disable
Delay Sec: 1m
Pad Front: 500
Pad Tail: 0

Multiply Const
Constant: 10

Channel Model
Noise Voltage: 1
Frequency Offset: 0
Epsilon: 1
Taps: 1
Seed: 0

UHD: USRP Sink
Sync: PC Clock
Clock Rate (Hz): 200 MHz
Samp Rate (Sps): 10M
Ch0: Center Freq (Hz): 2.412G
Ch0: Gain Value: 10
Ch0: Antenna: TX/RX
TSB tag name:

UHD: USRP Source
Clock Rate (Hz): 200 MHz
Samp Rate (Sps): 10M
Ch0: Center Freq (Hz): 2.412G
Ch0: Gain Value: 10

Figure 3: GNU Radio model of lower-level 802.11 processing.

4.3.1 GNU Radio and Colombo Integration

In the course of our research it was possible to craft a piecewise emulation where network data was processed through the full 802.11 MAC and physical protocol communication layers. This was accomplished with a combination of custom applications, the modified Colombo software, the GNU Radio model, and the Wireshark network utility. Two custom applications were the source and sink of user data within the emulation. One of these applications created user source data that it then passed to the modified Colombo software. The Colombo software accepted that data and processed it through the 802.11 upper-level MAC protocols emulation. The Colombo software then passed the processed data to the GNU Radio model which processed the data through an emulation of the 802.11 lower-level MAC protocols, the physical layer waveform encoding, and an RF channel model. At that point GNU radio passed the packet data to a second application that collected the packet data and stored it to a file for offline analysis.

In the experiment each stage of the emulation ran within its own operating system process. In total four processes were necessary. Two processes ran the source and sink applications. One ran the Colombo executable and another ran the GNU Radio model executable. Data passing between these operating system executables were through general TCP/IP socket interfaces. Hence, it was possible to distribute the emulation across multiple compute nodes connected by a general purpose network.

This experiment demonstrated basic principles of the emulation approach. However, applying this demonstration to cyber security experiments would require significant further development. The demonstration as crafted passed user data in a simplex fashion; data moved only from source to sink. Work would be required to develop duplex transmission paths in each of the separate emulation applications. Further, the modified Colombo application utilizes a fixed emulation time step which would likely present problems in trying to interface real user applications with the joint emulation. Depending on the user application, it is likely that variable time-step information would need to be passed to the Colombo code for proper end-to-end user application emulation. Further work would also be required in developing a framework for emulation configuration across the various processes and compute nodes. Depending on desired emulation experimental results, it may be possible to utilize this demonstration for some cyber security investigations, but further work would be required to develop it into a more robust tool.

4.4 Mininet-Wifi Extension Module

Mininet-WiFi's use of Linux *namespaces* provides a means to isolate the network stack. Each *namespace* is created with its own virtual network interface, its application, and associated virtual network. Mininet-WiFi experiments are limited in scale to those experiments that can run on a single host. For our research platform, increasing experiment scale beyond that which can run on a single host is necessary.

To address emulation scaling with Mininet-WiFi, an extension module was developed to enable a single wireless emulation experiment to span Mininet-WiFi instances on differing compute platforms. The basic function of the extension module is to bridge WiFi packets between two emulation instances on differing compute platforms. This extension is depicted in Figure 4.

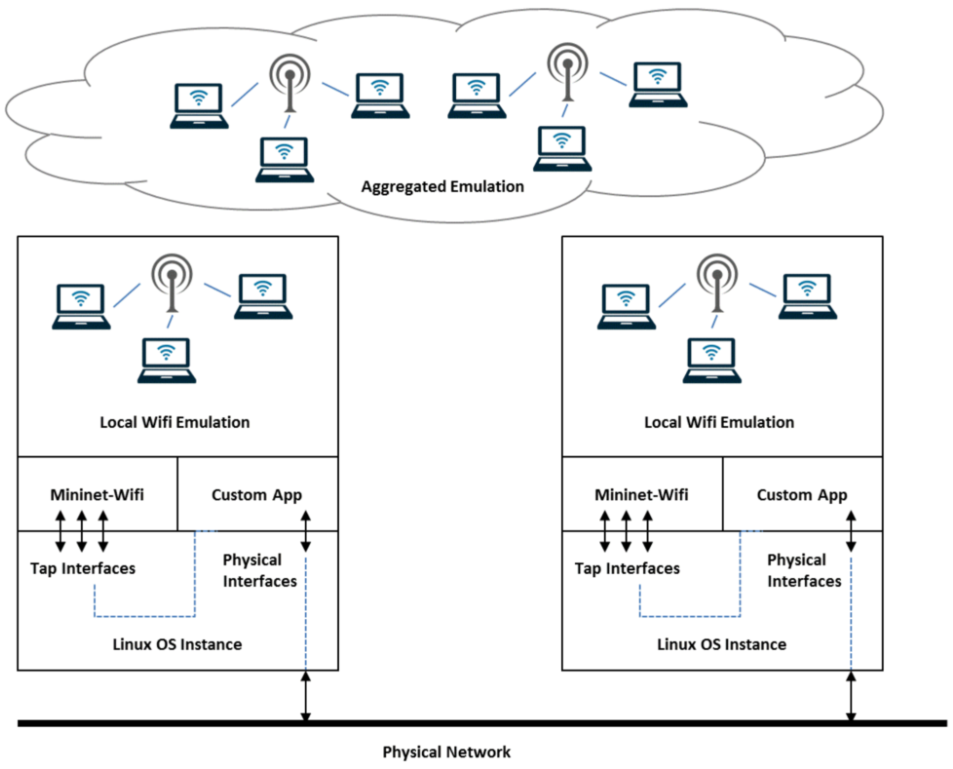


Figure 4: Mininet-WiFi multi-host extension [24].

5. Demonstration Assessment Experiment

Numerous experiments have been executed with the wireless analysis platform described in this research report. Analysis has been performed on the SRW system using the combination EMANE and CORE capability. This was possible because of the detailed SRW model. Analysis experiments were also performed with Mininet-WiFi capability upon 802.11 protocols including the associated encryption protocols. Since the 802.11 protocol is commonly used around the globe, we selected it for a demonstration experiment.

An EMANE and CORE experiment that includes multiple SRW radios is described in Appendix A. The scenario includes both single mode and dual mode radios that bridge communication between multiple islands of communication.

One demonstration experiment illustrates the capability to perform a wireless cyber-attack experiment without the need for radios (e.g., SDR). The experiment can be performed on a single host system and analysis of data frames can be performed with Wireshark. The demonstration attack is a well-known attack with descriptions published in many resources [25]. The demonstration illustrates the realism of the analysis platform. The attack is based on a published exploit of the Wired Equivalent Privacy (WEP) encryption. Publications of this attack provide very specific details on each step of the attack and the expected system response [25]. In this demonstration the published attack description will be our ground truth. The demonstration experiment performed as expected and included the following steps.

1. Set up and configure an AP and multiple user nodes. The communication will be via wireless networking. In Mininet-WiFi configure the AP to use a WEP security mode.
2. Use wireless sniffing / troubleshooting tools to monitor wireless traffic. In this experiment *airmon-ng* is used. *Airmon-ng* reports traffic as expected.
3. Use *airodump-ng* to obtain traffic dump files from transmissions taking place.
4. The tools used to compromise the encryption in this attack require a sufficiently large amount of data to be transmitted. Use *iperf* to produce this network traffic. This step includes capturing ARP and injecting these packets back into the emulated network.
5. Use *aircrack-ng* to discover the WEP key. *Aircrack-ng* uses the data packets stored in the dump files.

For this demonstration a large number of data packets are needed. The experiment runs at real-time and took approximately three minutes to discover the encryption key. The time for *aircrack-ng* to discover the key is non-deterministic, thus key discovery times vary. The experiment executed as expected and was done with only a workstation with a general purpose processor and running a Linux OS.

6. CONCLUSIONS

Assessing cyber security of wireless network technologies and implementations is best performed with high-realism experimentation. However, experimenting with live wireless network technologies and implementations results in radios emitting RF energy into potential licensed spectrum. This type of experimentation can become costly and time consuming. Capabilities to create experiments that include realistic lower-layer wireless protocols and their interactions without emitting RF spectrum are a significant enhancement for assessing wireless networks. In this research paper, wireless network emulation capabilities and benefits are described. Techniques to employ various tools to evaluate the security posture of a wireless network and assess its operation when subject to cyber-attacks are described. A major benefit of the emulation approach is that it can be done on standard computing platforms without the need for specialized radios. This research paper also provides guidance on the types of experiments and type of emulation capability best suited to assess specific types of wireless protocols and technologies. A demonstration experiment was presented and its results to assess the effectiveness of the wireless emulation platform.

7. BIBLIOGRAPHY

[1] B. Van Leeuwen, V. Urias, J. Eldridge, C. Villamarin and R. Olsberg, "Performing cyber security analysis using a live, virtual, and constructive (LVC) testbed," - MILCOM 2010, San Jose, CA, 2010.

[2] V. Ramachandran, "BackTrack 5 Wireless Penetration Testing," Packt Publishing Ltd., Birmingham B3 2PB, UK. 2011.

[3] "IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical layer (PHY) Specifications," (2012 revision). 5 April 2012.

[4] FY-15 ARMY PROGRAMS – Rifleman Radio, www.dote.osd.mil/pub/reports/FY2015/pdf/army/2015rifleman.pdf

[5] Glen Judd and Peter Steenkiste, "Using Emulation to Understand and Improve Wireless Networks and Applications", Carnegie Mellow University, Pittsburgh, PA, USA.

[6] Steel Central Riverbed Modeler, http://www.riverbed.com/products/ steelcentral/steelcentral-riverbed-modeler.html

[7] Scalable Network Technologies, http://web.scalable-networks.com/

[8] Ns-3 - Emulation Overview: vns-3.11, https://www.nsnam.org/docs/release/3.11/models/html/emulation-overview.html

[9] The Click Modular Router Project, http://read.cs.ucla.edu/click/click

[10] K. W. Parker, "Scaling Mobile Ad Hoc Networks: Alternate Semantics for Local Routing Combined with Leveraging Small Amounts of Global Capacity." Advanced Concepts Lab, Lockheed Martin Report. 2013.

[11] WARP: Wireless Open Access Research Platform, https://warpproject.org/trac

[12] J. Zhang, et.al, "Experimenting Software Radio with the Sora Platform." SIGCOMM'10, 2010, New Delhi, India. ACM.

[13] J. Ahrenholz, "CORE: A real-time network emulator," MILCOM'2008, San Diego, USA, Nov. 2008.

[14] "The extendable mobile ad-hoc network emulator" (EMANE). [Online]. http://www.nrl.navy.mil/itd/ncs/products/emane

[15] R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos and C. E. Rothenberg, "Mininet-WiFi: Emulating software-defined wireless networks," 2015 11th International Conference on Network and Service Management (CNSM), Barcelona, 2015, pp. 384-389.

[16] Linux Wireless: mac80211_hwsim
https://wireless.wiki.kernel.org/en/users/drivers/mac80211_hwsim

[17] EMANE Model - IEEE 802.11abg Model
https://github.com/adjacentlink/emane/wiki/IEEE-802.11abg-Model

[18] A. M. Illán, "Medium and mobility behaviour insertion for 802.11 emulated networks." Thesis, Sept. 2013.

[19] Linux Wireless - About mac80211,
http://linuxwireless.org/en/developers/Documentation/mac80211

[20] Van Leeuwen, B., Torgerson, M., "Difficulties with Authenticating Routing Information in Wireless Ad Hoc Networks." Proceedings of the 2002 IEEE Workshop on Information Assurance, United States Military Academy, West Point, NY. June 2002.

[21] "WARP 802.11 Reference Design: Architecture," [Online]. Available: http://warpproject.org/trac/wiki/802.11/Architecture. [Accessed 11 September 2017].

[22] Colombo SDK project page. https://www.microsoft.com/en-us/download/details.aspx?id=52612

[23] Dinan Gunawardena, Božidar Radunović, "Microsoft Research Colombo SDK – simulating the innards of a wireless MAC", (Microsoft Research, Cambridge, UK), MobiCom 2011, September 19-23, 2011.

[24] Vexels graphics attribution for WiFi subgraphics: Designed by Vexels.com

[25] Sourceforge, "Precision Time Protocol Daemon," https://sourceforge.net/projects/ptpd2/

[26] Mininet-WiFi Emulator for Software Define Networks
http://www.ramonfontes.com/mininet-wifi/

[27] Mininet-WiFi Emulator for Software Define Networks
http://www.ramonfontes.com/mininet-wifi/

[28] Cozybit source code repository for mac80211_hwsim and wmediumd
https://github.com/cozybit/wmediumd

[29] Mininet-WiFi Source Code: https://github.com/intrig-unicamp/mininet-wifi

APPENDIX A: EMANE EXPERIMENT OF SRW RADIO SCENARIO

This experiment includes multiple Soldier Radio Waveform (SRW) models deployed in EMANE. In this six-node test, there are four single-radio nodes and two dual-radio nodes that exist on three separate islands, where an island is a group of nodes that can communicate via the same frequency. One of the islands, referred to as a "Tier 1B" island, is topologically higher than the other two, and it is used to connect the two lower "Tier 1A" islands. During the scenario, each island's nodes use different frequencies to achieve physical layer isolation. Figure A1 illustrates a complete layout of the scenario topology.

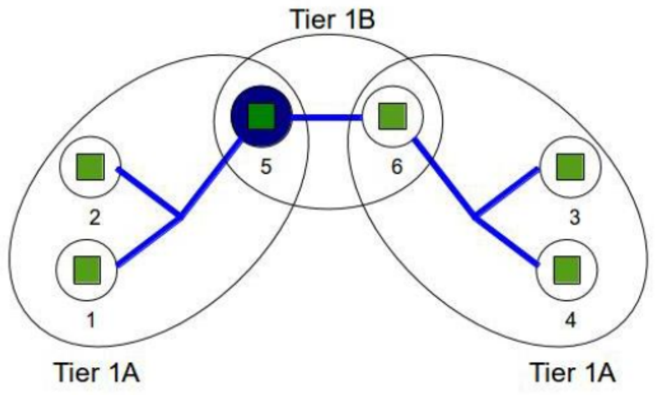


Figure A1: EMANE SRW Scenario Topology.

During the scenario, calls are made, where calls refer to digitized voice data streams. Several of the calls are local and can only traverse their own island. Several other of the calls are global and can travel across islands via the "island head." In Figure A1, Nodes 5 and 6 are the island heads of their Tier 1A islands, and Node 5 is the island head of its Tier 1B island. When physical isolation is emulated, island heads are the only gateway to other islands.

For best performance of the scenario experiment, the user syncs the clocks of all the machines that will be used in the scenario. To guarantee accuracy, time synchronization within 100 milliseconds should be selected. This can be achieved using the Precision Time Protocol tool, *ptpd2* [25]. After downloading and installing the tool, sync the master machine with any slave machines using the following commands:

```
$ cd ~/Emane/demonstrations/05-cnr/
$ sudo su
# chrt 70 ptpd2      –m     –i eth0 # run on one master machine
# chrt 70 ptpd2      –s     –i eth0 # run on all slave machines
```

Next, create the configuration files; note that several files will need to be edited. *elgenerator.xml* and *modelsrwmac.xml* will need to be edited so that *$INSTALLPATH* is the full path to *initial.eel* and *modelsrwpcr.xml* respectively. Next, *$EVENTSERVICEDEVICE* will need to be replaced by the network interface to be used for sending EMANE events. This parameter exists in *eventservice.xml* and *platform-00X.xml*, where "X" represents the node that the current machine represents. Then, $OTADEVICE will need to be replaced by the network interface to be used for sending EMANE Over-The-Air traffic. This parameter exists in *platform-00X.xml* and *radio-00X.xml*. Finally, the parameter, *$LOGPATH*, within *radio-00X.xml* will need to be replaced by a suitable full path to a log file for voice events to be written to. At this point, the scenario can be started.

The user may go ahead and start EMANE along with the events that will be generated for the scenario. Note that from this point forward in the demonstration, each line should be run on each machine one-at-a-time. For example, in the next two lines, line 1 should be run on all six machines before line 2 is run on all six machines.

```
# emane --daemonize --realtime --logfile Logs/emane-n<X> platform-00<X>.xml
# emaneeventservice eventservice.xml
```

Note that the "X" above must be replaced by the node number corresponding to the machine the command is being run on. For example, *platform-001.xml* will be run on Node 1, *platform-002.xml* will be run on Node 2, and so forth. The second command will create path loss across the nodes for the emulation. This must occur for the nodes to be able to find each other.

Next, the user should start the SRW waveform. Once the waveform has started, the call scenario should be ready to run. The calls can be started using the srweelgenerator python script, which parses an Emulation Event Log (EEL) and generates voice events between nodes. These commands will need to be run in a separate terminal, as the previous command should have hung on the previous terminal.

```
# wfctrl --daemonize --logfile Logs/srw-nX radio-00X.xml
# srweelgenerator.py scenario.eel # note that srweelgenerator.py is located in /usr/bin/
```

The packet flow can be viewed in EMANE by capturing packets on two different interfaces. The srw0 interface shows various control messaging queries and responses. The eth0 interface shows the timing packets, EMANE packets, and voice data.

Finally, once the user is ready to stop the scenario, the following command should be run on all machines.

```
# killall –s QUIT wfctrl
# killall emane           # also perform a ctrl+C on the emaneeventservice terminal
```

This command will send the "QUIT" signal to the waveform and stop EMANE.

APPENDIX B: MININET WIFI CONFIGURATION TEST SCRIPT

```
#!/usr/bin/python

'This example creates a simple network topology with 1 AP and 2 stations'

from mininet.net import Mininet
from mininet.node import  Controller, OVSKernelAP
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.link import TCLink

def topology():
    "Create a network."
    net = Mininet(controller=Controller, link=TCLink, accessPoint=OVSKernelAP)

    print "*** Creating nodes"
    sa1 = net.addStation('sa1', passwd='123456789a', encrypt='wep', ip='192.168.221.21',
mac='fa:fb:fc:fb:fa:00')  # encrypt=(wpa,wpa2,wep)
    sa2 = net.addStation('sa2', passwd='123456789a', encrypt='wep', ip='192.168.221.20',
mac='fa:fb:fc:fb:fa:01')  # encrypt=(wpa,wpa2,wep)
    sb1 = net.addStation('sb1', passwd='123456789a', encrypt='wep', ip='192.168.221.10',
mac='fa:fb:fc:fb:fa:02')  # encrypt=(wpa,wpa2,wep)
    sb2 = net.addStation('sb2', passwd='123456789a', encrypt='wep', ip='192.168.221.11',
mac='fa:fb:fc:fb:fa:03')  # encrypt=(wpa,wpa2,wep)
    # encrypt=(wpa,wpa2,wep)
    ap1 = net.addAccessPoint('ap1', ssid="cyber", mode="n", channel="5", passwd='123456789a',
encrypt='wep', mac='fa:fb:fc:fb:fa:0a')
    ap2 = net.addAccessPoint('ap2', ssid="rebyc", mode="n", channel="7", passwd='123456789a',
encrypt='wep')

    c0 = net.addController('c0', controller=Controller, ip='127.0.0.1', port=6633)
    #c1 = net.addController('c1', controller=Controller, ip='127.0.0.1', port=6634)

    print "*** Configuring wifi nodes"
    net.configureWifiNodes()

    print "*** Associating Stations"
    net.addLink(sa1, ap1)
    net.addLink(sa2, ap1)
    net.addLink(sb1, ap2)
    net.addLink(sb2, ap2)
    #net.addLink(sb1, ap1)
    #net.addLink(sb2, ap1)


    print "*** Starting network"
    net.build()
    c0.start()
    #c1.start()
    ap1.start([c0])
    #ap2.start([c0])

    print "*** Running CLI"
    CLI(net)

    print "*** Stopping network"
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    topology()
```

APPENDIX C: SELECT CODE PORTIONS FROM THE CUSTOM MININET WIFI EXTENSION

Libraries:

```
#General:
import os
import SocketServer
import socket
import sys
import signal
import threading
import time
import Queue

from scapy.all import Ether
from scapy.all import TCP
from scapy.all import IP
from scapy.all import Raw
from scapy.all import sendp
from scapy.all import *
from scapy.utils import PcapWriter

def dbg_print(text, level):
    print(text)
```

Select Global Variables

```
#Miscellaneous globals to get the application up and running.

#Include services host and port numbers

gbl_sendHost = '192.168.15.30'
gbl_dataPorts = [40010]    #Accept incoming application data packets. From Host API.

#Queue size upper limit may be set by maxsize = upper_limit.
#If maxsize is not included, then the queue will grow without bound.
#gbl_packet_queue = Queue.Queue(maxsize=2000)
gbl_packet_queue = None

ap_list = []
gbl_dst = None
gbl_sniff_interface = "mon0"
gbl_inject_interface = "ap1-wlan1"
```

Transmission code

```
class SendIt():
    def __init__(self, ip="192.168.15.30", port=40010):
        self.target_ip = ip
        self.target_port = port
        self.target =(self.target_ip, self.target_port)
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    def send(self, data):
        try:
            self.sock.sendto(data, self.target)
        except:
            print("SendIT: could not send data")

def PacketHandler(pkt) :
    global gbl_dst
    if pkt.haslayer(Dot11) :
        #Locally reinject packet to a different interface or process
        #inject(pkt)

        #Send packet across the network to another VM or compute node.
        gbl_dst.send(bytes(pkt))
        if (pkt.type == 2):
            print("type: %d, subtype: %d, len: %d" % (pkt.type, pkt.subtype, len(pkt)) )
            #pkt.show2()
            pass
        if (pkt.type == 0 and pkt.subtype == 8) :
            if pkt.addr2 not in ap_list :
                ap_list.append(pkt.addr2)
                print "AP MAC: %s with SSID: %s " %(pkt.addr2, pkt.info)
    else:
        print("No Dot11")

def main():
    #main application code

    #Start the thread for reception of incoming 802.11 traffic.
    startServers()

    global gbl_dst
    global gbl_sniff_interface
    global gbl_sendHost

    try:
        gbl_dst = SendIt(ip=gbl_sendHost)
    except:
         gbl_dst=None

    try:
        #Continuously monitor for 802.11 traffic on selected interface in main thread
        sniff(iface=gbl_sniff_interface, prn = PacketHandler)
    except:
        print("No sniffing today!")

if(__name__ == "__main__"):
    main()
else:
    print("Not main: %s" % (__name__) )
```

Reception Code

```
def serviceQueue():
    """
    This method provides the functionality processing packets out of the service
    queue for incoming packets. The methods runs within its own application
    thread, but will interacts with the DataHandlerUDP class instance within
    the application's main thread of operation. This method goes into
    an infinite loop to process packets out of the queue.
    """
    global gbl_packet_queue
    global ENDIT
    #Set the ip address and port number for returned traffic
    #packet_return_client = PacketReturn(host=gbl_dataRtnHost, port=gbl_dataRtnPorts[0])
    #dbg_print(str(packet_return_client), 2)
    while(ENDIT != 1):
        data = gbl_packet_queue.get()
        data_to_go = packetProcessor(data)
        gbl_packet_queue.task_done()
        try:
            pass
            #packet_return_client.send(data_to_go)
        except:
            dbg_print("...Error: serviceQueue, packet_return_client.send", 0)
    dbg_print("   Stopping: serviceQueue", 1)

def startQueueServers():
    """
    This method create a new application thread to execute the packet queue
    processing code (method serviceQueue).
    """
    dbg_print("... entered startQueueServers", 1)
    queue_thread = threading.Thread(target=serviceQueue)
    queue_thread.daemon = True
    queue_thread.start()

def inject(packet):
    global gbl_inject_interface
    #packet = Dot11(data)
    if (packet.haslayer(Dot11)):
        print("Dot11")
        sendp(packet, iface=gbl_inject_interface)
    else:
        print("Other")

def packetProcessor(data):
    """
    This method processes the raw UDP data received from the network server.
    It strips off the internal transport header and then converts the rest of
    the data into a scapy packet format for processing.
    @param data: raw input that was received from the UDP server.
    """

    global glb_streams_state

    try:
        packet_data = bytes(data[0:])
        pkt = Dot11(packet_data)
    except:
        dbg_print("...Error: packetProcessor, no pkt", 0)
        dbg_print("data(%d): " % (len(data)), 0)
        return data

    try:
        inject(pkt)
    except:
        print("...Error: Injection failed.")
    return data

class DataHandlerUDP(SocketServer.BaseRequestHandler):
    """
    This class is the UDP callback to accept and process any incoming
    network packets. The packet arrive as UDP data. It then performs a
    quick sanity check on the this data and passes it off to packetProcessor
    to further action.
    """
    def setup(self):
        pass
        #dbg_print "UDP connection received from " + str(self.client_address) +', thread '+
__name__

    def handle(self):
        global gbl_packet_queue

        #dbg_print ("In DataHandlerUDP HANDLE loop")
        remote = self.client_address
        #data, socket = self.request.recv(gbl_maxPktBuf)
        #data, socket = self.request

        try:
            data, socket_value = self.request
            gbl_packet_queue.put(data)
        except:
            dbg_print("...Error: DataHandlerUDP.handle, place data in queue.", 0)
            pass #To cover case of dbg_print removals.
        return
```

DISTRIBUTION

1	MS0509	David Wiegandt	5330
1	MS0503	Joseph Sorroche, Jr.	5321
1	MS0621	Andrea M. Walker	5832
1	MS1027	John M. Eldridge	5832
1	MS0757	Silpan Patel	6613
1	MS0813	Han Wei Lin	9315
1	MS0813	Vincent Urias	9315
1	MS0813	Brian Van Leeuwen	9315
1	MS0899	Technical Library	9536 (electronic copy)
1	MS0359	D. Chavez, LDRD Office	1171

Sandia National Laboratories