Abstract—Researchers who evaluate networking protocols and applications can benefit significantly from the availability of flexible, open source tools. Network emulation has been one of the tools of choice for conducting experiments on commodity hardware, and for wireless MANET environments, in the absence of a possibly large number of radio devices. We present an easy-to-use system based on a combination of two such open source emulation tools. We describe integrating the Common Open Research Emulator (CORE), providing virtualization controlled by a graphical user interface, with the Extendable Mobile Ad-hoc Network Emulator (EMANE) framework that provides for more detailed radio models and scenarios. We discuss the benefits of integrating the two tools, along with challenges of this approach and how they were addressed. We address the performance experienced when using these more detailed link- and physical-layer models. Finally, we conclude with open issues and future directions.

I. INTRODUCTION

Network emulation is an attractive, cost-effective tool to have in the network researcher’s toolbox. Managing a multitude of virtual nodes and networks can be easier and less time-consuming than configuring the equivalent physical equipment in a laboratory testbed. For wireless networks, radio hardware may be unavailable or costly to acquire, and then deploy and use in a controlled, repeatable environment. Furthermore, emulation tools that have open source licensing can be easier to acquire, share, and collaborate with. Access to the full source code offers unparalleled flexibility and room for customization.

This paper considers experiences with integrating two such complementary network emulation tools, the Common Open Research Emulator (CORE) [1], [2] and the Extendable Mobile Ad-hoc Network Emulator (EMANE) [3]. CORE focuses on emulating layers 3 and above (network, transport, session, application) while EMANE is concerned with layers 1 and 2 (physical and data link). Together, the tools provide an easy-to-use graphical user interface for designing and configuring virtual networks, consisting of lightweight virtual machines interconnected with pluggable MAC and PHY layer models.

CORE provides a graphical user interface and Python framework for building virtual networks. The GUI presents a blank canvas where the user can click and place routers and end hosts, or networking devices such as hubs and switches, and link them together to form networks (see Figure 1). A wireless LAN (WLAN) cloud tool is available for adding wireless radio interfaces to nodes. Under-the-hood, after pressing a start button to launch the emulation, each node is represented with a lightweight virtual machine. Protocols and applications can run in these nodes without modification. Virtual interfaces are installed into these machines, which are then tied together using Linux bridging. These live-running networks can be connected to real networks and systems. Basic network effects such as delay, loss, and bandwidth restrictions can be applied to these networks using Linux tc (traffic control) netem (network emulation) framework.

For basic wireless emulation, all interfaces that are joined to the same WLAN network are attached to a Linux bridge, with Ethernet bridging tables (ebtables) firewall rules inserted to block or unblock connectivity between pairs of nodes. This provides a basic on/off connectivity required to emulate a MANET network by dragging nodes in and out of range of each other on the canvas or via a mobility script.

In reality, of course, the effects of communicating wirelessly are more complex than on and off connectivity. CORE provides a plug-in architecture to interface with more detailed link modeling. Because CORE is using virtualization and the Linux network stack, with unmodified user protocols and user applications, the system faithfully represents networking layers 3 and above, but the link and physical layers are greatly simplified. EMANE provides exactly the pluggable framework for emulating link and physical-layer wireless networks.

EMANE is a modular framework consisting of processes and shared libraries configured through XML files. A single radio in EMANE is represented by a network emulation module, or NEM, built by coupling a MAC/link-layer module with a physical-layer module. An over-the-air (OTA) manager delivers packets between NEMs. An event system provides for events such as location and pathloss events, and shims may optionally be inserted between components.

EMANE is flexible enough to run with one NEM per physical platform or several NEMs on one physical platform, or mixed numbers of NEMs and platforms. Physical Ethernet interfaces (EMANE’s raw transport using libpcap) or virtual interfaces (virtual transport using TUN/TAP devices) can be used for delivering packets to the NEMs. This paper discusses installing the TUN/TAP virtual interfaces into CORE network namespace nodes.

This paper is organized as follows: related work is consid-
ered in the next section, followed by motivation for integrating the two systems. Next, the integration is described along with challenges and how they were addressed. The performance of using these more detailed link- and physical-layer models is presented, followed by conclusions and future work.

II. RELATED WORK

The CORE GUI is derived from the open-source IMUNES tool [4], adding, most notably, support for Linux virtualization, Python, and wireless networks. IMUNES leverages FreeBSD jails [5] for virtualization and an in-kernel netgraph packet-forwarding system. A full spectrum of virtualization techniques is available in a Linux environment, ranging from emulating all of the system hardware, with tools such as VirtualBox, VMware, and QEMU [6]–[8], to varying levels of kernel isolation, such as Xen, user-mode Linux (UML), Kernel-based Virtual Machine (KVM), OpenVZ containers, and Linux containers (LXC) [9]–[12].

Linux network namespaces is synonymous with LXC; LXC includes additional management tools, cgroups for resource isolation, and is more focused on persistent containers defined using configuration files. CORE provides Python modules that use network namespaces to create non-persistent nodes having a shared filesystem. This is the virtualization technique considered in this paper.

EMANE is a complete redesign of Naval Research Laboratory’s (NRL) MNE [13] and MANE [14] emulation tools. With MNE, one physical machine is used for each network node, all connected to a control back-channel, and iptables rules are used to block the MAC addresses of nodes that may be out of range. MANE uses a more centralized approach, where each network node (client “test node”) is connected to a MANE server and servers may be interconnected. This emulator uses a range model to determine the connectivity matrix between nodes, and performs packet dropping. A packet treatment component provides a virtual radio interface at the test nodes and limits the bandwidth used by each node. EMANE represents the next-generation network emulation framework, designed with lessons learned from these previous MNE and MANE developments. It can provide a virtual interface or capture packets from real interfaces, forward data traffic to a centralized packet treatment server or have decentralized packet treatment at the interfaces. The pluggable MAC and PHY models provides an architecture for developers to create their own radio models.

The ns-2 simulator had an emulation extension [15] which could be run in real-time mode. The ns-3 simulator [16] has an emulation mode with a real-time scheduler, and some work has been done to run CORE in conjunction with underlying ns-3 simulated wireless networks. Such integration shares similarities to CORE and EMANE integration but is not addressed in this paper.

Related commercial offerings include QualNet’s Exata [17] and OPNET’s SITL [18]. The clear distinction between these is that CORE and EMANE are freely available and modifiable open source tools.

III. BENEFITS OF INTEGRATION

Perhaps the most obvious motivation for integrating EMANE with CORE is to provide users with more choice. Some users will want access to all of the radio models available with EMANE, to more closely emulate the networking effects of those radios. Both tools run on a variety of hardware systems; memory and CPU processing capabilities may vary greatly. For some sizes of emulation and levels of network utilization, users will be able to dial up or down the level of complexity to fit their hardware resources.

It is also desirable to reuse existing mobility scripting formats without having to re-implement or convert them. Existing scenarios can be run in the virtual environment provided by one machine or on a rack of EMANE machines. EMANE has event generators that understand the EEL (emulation event log) format, MITRE mobility format, and an NRL script schema format [19]. When CORE is run with EMANE radios, mobility and connectivity can be controlled using the generators that read from those types of script files.

EMANE uses a multicast over-the-air channel for exchanging packets between NEMs. NEMs can therefore be distributed across several physical machines. This allows wireless networks in CORE to be distributed. For wired networks, CORE builds GRE tunnels between emulation machines, controlled by a single GUI.

EMANE users can benefit from the GUI provided by CORE for working with networks. The user sees the logical layout of the network on screen. Usually, EMANE is configured through XML files. CORE automates the process of generating these XML files and offers a dialog-based configuration for model parameters. Additionally, CORE contains other niceties such as automatic assignment of interfaces and IP addresses, auto-configuration of networking services such as routing protocols, the ability to generate traffic flows by clicking on nodes, and a start/stop button for actually launching the emulation.

Once the emulated network is running, CORE can provide the EMANE user with run-time interaction. By pointing at a node on the canvas and double-clicking it, a window pops up with shell access. Packet capture (tcpdump) and routing console (vtysh) access are one click away. CORE provides
Fig. 2. Example packet flow through CORE and EMANE

a set of widgets for visualizing the network. An adjacency
widget displays OSPF router adjacencies, while a throughput
widget displays the real-time bandwidth (kbps) for links. Observer widgets allow the user to mouse over a node to view
information like running processes, routing tables, firewall
rules, and more. There is a run tool for easily running a
command on a selection of nodes and a traceroute tool for
viewing network paths. Finally, the user is able to drag nodes
around the canvas to generate EMANE events that affect the
connectivity.

Combining CORE and EMANE also brings lightweight vir-
tualization to the EMANE world. While EMANE can provide
multiple NEMs on one machine, CORE can offer multiple
network stacks that the NEMs can be connected to, and a
process space for isolating user processes. Wireless MANET
networks can be expanded to include virtual nodes that are
wired into the topology. Real nodes (with virtual interfaces)
can be intermingled with virtual nodes, and the network can
be connected to other wired networking equipment.

CORE networks do not need to be created from a GUI or
have a GUI attached while running. The CORE framework
includes Python modules that can be imported by Python
scripts for an advanced scripting environment. Additionally,
the EMANE event service has Python bindings, so the same
Python scripts can publish or subscribe to EMANE events.

IV. INTEGRATION AND ITS CHALLENGES

Integrating CORE and EMANE creates a co-emulation
environment where interaction needs to be managed and
coordinated. Areas that can overlap and require attention to
avoid conflicts include configuration and startup dependencies,
maintaining consistent state information, and clock synchro-
nization.

At a high level, an integrated CORE/EMANE emulation
consists of:

- Network topology and node configuration managed by
  CORE.
- Emulated nodes with independent network stacks man-
  aged by CORE.
- Network interfaces controlled by EMANE.
- Network emulation controlled by EMANE.

An example packet flow between a sender and receiver in
this environment is illustrated in Figure 2.

Certain challenges were experienced when integrating
CORE and EMANE to work together; they are described here.
Location information, for example, exists in both domains.
The location of the node on the CORE canvas and the
geographic (latitude, longitude, altitude) location given by the
last location event for a NEM need to be synchronized. The
question must be answered: who controls the node position?
Through the use of the EMANE event service library Python
binding, CORE was given the ability to subscribe to EMANE
events or to publish them. The user must choose to move
nodes via a mobility script (scripted events update the node
position on the canvas) or from the CORE canvas (CORE
publishes location events) when nodes are dragged about using
the mouse. The CORE canvas initially had only a Cartesian
(x, y) coordinate system, so a geographic reference point
and pixels-to-meters scale needed to be added in order to
convert to latitude and longitude. As the CORE canvas is two-
dimensional, altitude is not supported, so it is discarded when
received and set to a fixed value when generating a location
event.

EMANE is a modular framework, and different models
having different configuration parameters may be available
on a system. Another challenge is how to present the con-
figuration options for these models to the user in a modular
fashion. When a new model is developed, we do not want
to have to make changes to the GUI (i.e. program a new
dialog box) to support that model. This problem was solved
using configuration messaging such that the GUI can query the
backend for available models and possible configuration values
for each model. The WLAN wireless cloud can be configured to use one of these models. An EMANE model is represented in the backend by a small Python object. This object keeps a list of parameters and their text captions, possible values, and default values that will be presented to the user, and code for writing those configured parameters out to an XML file before starting EMANE. When the GUI receives the parameter list, it dynamically generates a dialog box with text entries and drop-down lists and their appropriate captions. The list of models to load is contained in a core.conf configuration file. When a new model is introduced, the developer needs to write one of these small (about 100 lines) Python classes and add the model’s class name to the list in the CORE configuration file.

The timing involved with setting up the virtual devices presented some challenges. The TUN/TAP virtual device has two sides, one in user-space and the other in kernel-space. The user-space side is a socket held by a process that can be read from or written to, and to the kernel the device appears as another network interface having its own addresses and routes. The device can be “pushed” into a namespace, at which point it disappears from the host machine and appears only within the namespace. The EMANE user-space transport process must open the socket to the device before it is installed into the namespace. The easiest way to do this is to supply the EMANE process with a name and allow it to create and open the device. CORE normally creates a node and its associated interfaces when it receives Node and Link API messages from the GUI. In the case of EMANE, however, to include all of the interfaces in one XML file, the backend must wait until it receives all node and link information.

The concept of run-time stages was introduced in CORE to address this. Instead of simply having an edit (stopped) and execute (started) mode, CORE was extended to have several stages, including configuration and instantiation stages. The GUI signals the backend via a new Event message. The GUI starts sending node and link information during the configuration stage, and when it is done describing all of the nodes, signals an instantiation stage. Now having the knowledge of all nodes and interfaces, it is at this point that the EMANE XML files can be generated, the EMANE transport daemon can be started, and the CORE backend can install the TAP devices into their namespace nodes.

There are other platform and installation issues with integration. A user interested in running both CORE and EMANE has the added burden of reading both sets of documentation for installation and configuration. Each emulator tool has its own dependencies and platforms for which it has been tested and for which packages have been built. For example, EMANE currently has packages for Fedora 12 and 13, Ubuntu 10.04, Windows, and Mac OS X; CORE meanwhile has packages for Fedora 12, 13, and 14, Ubuntu 10.04 and 10.10, and FreeBSD 8.x. As one can tell from these overlapping sets of operating systems, there is one common operating system (Linux) and a couple of common distributions (Fedora and Ubuntu). Also both tools are under active development, and changes to one tool and upgrading one version may break compatibility with the other. These issues are addressed by attempting to provide clear documentation on what platforms and versions supported.

V. PERFORMANCE

This section begins to explore some of the performance issues with using this integrated emulation system to represent a MANET with several nodes participating in the same broadcast wireless network. This is not a validation of the system but the results of some initial measurements to quantify the cost of adding some layer 1-2 fidelity. These tests highlight the following points: (1) the kernel to user-space copy incurs additional latency; (2) care should be taken as user applications contend for the same CPU resources as the emulator; (3) the data rate and number of nodes that can be achieved are sensitive to the chosen radio model and its parameters. Further work is needed to more carefully understand and validate the layer 1-2 fidelity.

Here each node is virtual, represented by a network namespace under Linux, instantiated by CORE. Every node behaves as a router, forwarding packets it receives towards destinations for which it has routes. To simplify things, the nodes here are stationary (not mobile) and arranged in a worst case chain of routers. Test are performed end-to-end, so each packet must traverse every hop in the chain.

The system used for testing was a Dell Precision T1500 having a 2.80GHz Intel Core i7 860 CPU with 1 MB L2 and 8 MB L3 caches and four processor cores (plus Hyper-Threading), and 8 GB of 1333 MHz DDR3 RAM and a 250 GB disk. This desktop PC was running 64-bit Ubuntu 10.10 Linux on a 2.6.35-30-generic Linux kernel. The CORE 4.2 (pre-release) and EMANE 0.7.1 software was installed. The CORE GUI was not used here; these tests were scripted using CORE’s Python modules.

We tested each of the following types of networks:

- bridged
- bridged with netem effects (54M rate limit)
- EMANE Bypass model
- EMANE RF-PIPE model, with and without a set bitrate (54M), and with pathloss restricted connectivity

The bridged network tests uses veth pair devices joined to a Linux Ethernet bridge device on the host; this is the default wireless LAN in CORE, with basic on/off connectivity governed by ebtables (MAC firewall) rules. The netem test network adds basic network effects using the NetEm “network emulation” framework [20] available in the Linux kernel. Here, we use netem to limit the bandwidth to 54 Mbps. We did not add a link delay for easier comparison of the average latency ping test, and because we are not testing the RF-PIPE model’s transmission and propagation delays.

The EMANE Bypass model is the simplest possible EMANE model that passes through any packet that it receives. Performing measurements using this model gives us an idea of the impact of copying packets between kernel and user space. The EMANE Virtual Transport uses the TUN/TAP as described previously, to get packets from the virtual nodes.
Note that even though the static routes are used to forward packets in a chain of routers (at layer 3), every NEM in the bypass model receives every packet (no layer 2 connectivity information is used.)

The RF-PIPE model from EMANE is a generic radio effects MAC model, which is coupled with the Universal PHY layer. The RF-PIPE MAC offers jitter and delay effects, a promiscuous setting, and flow control. The Universal PHY provides common functionality (for use with other MACs), including pathloss calculation, received power calculation, and noise processing. See Figure 3 for possible RF-PIPE parameters. RF-PIPE flow control was not used, and real-time scheduling was enabled for these tests. For the “rfpipe54” test, the bandwidth was fixed at 54,000 kbps. The default parameters were used elsewhere.

An important distinction between the Linux bridging and EMANE tests are whether or not the packets from other radios are received. For Linux bridging, the ebtables MAC filtering prevents nodes that are not linked together from receiving packets. The Bypass model broadcasts every packet to every radio. The RF-PIPE model normally uses pathloss and noise calculations, based on node position, to determine packet reception. Here, we did not set node positions or script mobility patterns, so two of the RF-PIPE tests involve every radio receiving every packet. For the last RF-PIPE test named “pathloss”, we explicitly generate EMANE pathloss events to arrange the nodes in a chain, where a given node only receives packets from the node immediately to its left or right.

For the six different types of test networks, we perform the following tests:

- end-to-end ping, reporting the average latency (in milliseconds)
- end-to-end iperf, reporting the maximum TCP throughput (in bits per second)
- run MGEN UDP traffic flows from end-to-end, at differing rates (bits per second), while measuring the CPU usage times

The results of the latency tests using ping, shown in Figure 4, demonstrate the effect of copying packets from kernel to user-space for processing in EMANE. The number of nodes in the chain of MANET routers was increased from 5 to 30 in increments of 5. The EMANE tests add roughly 1ms of delay time per node, while delay incurred by bridging, where forwarding remains in the kernel, is orders of magnitude less. Further tests should be performed using a configured delay and location; this data only shows possible minimum delays when set to zero and nodes are at the same position.

The TCP throughput measurements reported from the iperf tests are presented in Figure 5. The unrestricted bridged network (without netem) is not shown on this Figure; it achieved iperf speeds ranging from 1.4 Gbps to 224 Mbps. The clear difference between EMANE and bridging is that the maximum bandwidth achieved by iperf decreases as more network hops are added. EMANE and iperf are both user-space processes that must compete for processor scheduling. This test shows that it is important not to overtax the CPU when running more complex radio models.

The CPU usage was measured for an MGEN 512kbps UDP Traffic Flow in the chain of MANET nodes, with results in Figure 6. This shows how the more times a packet needs to be handled in the system, the more CPU cycles will be used. Specifically, the CPU usage growth rate is $O(n^2)$ when all nodes receive all packets and $O(n)$ for the pathloss case where each node only receives...
packets from its immediate neighbors. Again, the user-space processing of packets will limit the amount of CPU resources available for applications in the emulation.

This final set of tests shows results for varying rates of MGEN UDP traffic flows, in Figure 7. A network of 10 MANET routers is used with an end-to-end MGEN UDP traffic flow at varying rates (offered load). Note that the UDP data rate is shown (the input to MGEN), and the actual bit rate including the IP and UDP headers is slightly different. The Bypass model starts to consume the most CPU time as all of the nodes receive all transmitted packets. The use of pathloss events with the RF-PIPE shows less CPU usage, as each node only receives neighboring transmissions. The simplicity of the bridging models can be observed with a low (1-2 percent) CPU usage throughout.

VI. CONCLUSION AND FUTURE DIRECTIONS

This paper built upon the work presented in [1], here concentrating on wireless networking models, and discussing the integration of two open source network emulation tools, CORE and EMANE. The pros and cons of using both systems were discussed and initial performance measurements were presented to characterize the cost of adding some amount of layer 1 and 2 fidelity. Care should be taken as the emulation system adds some delays, and as CPU resources are shared by the emulator and the applications running in the nodes.

Further work needs to be done to characterize the performance and fidelity of this system. The level of fidelity added by EMANE and the validation of the wireless models were not addressed in this paper. A validation of EMANE wireless models versus real radios and validating their use in conjunction with virtualization is left for further study.

Here, emulation for only a single system was considered. With limited CPU resources being the primary constraint for these virtual systems, a future study of the effects of distributing the emulation across multiple physical machines would be worthwhile. Also, performance results using different types of hardware and Linux kernels may be of interest. This paper considered only constant-rate test flows for unicast traffic. Another area for future study would be to characterize the types of missions, services, applications, and data traffic that work well across emulations, physical testbeds, and field experiment environments. The impact of network emulation on routing protocols and applications could be studied.

CORE is being extended to control physical testbed nodes and provide health monitoring facilities for running emulations. It would be interesting to compare EMANE runs on a physical testbed (rack of servers), purely virtual testbed, and combinations of the two, to understand the trade-offs. Built-in health monitoring techniques should aid in understanding system bottlenecks.

Finally, use of common scenario formats across tools, such as the NRL-developed emulation script XML schema, may help with running the same experiment in different ways. Support for this common XML format is also currently under development for CORE.

ACKNOWLEDGMENT

This work was done in cooperation with the NRL Propane Research Group, Code 5522 under the OSD-sponsored Network Communications Capability Program (NCCP).

REFERENCES