Device Driver-enabled Wireless Network Emulation

Elias Weingärtner, Hendrik vom Lehn, and Klaus Wehrle
Communication and Distributed Systems
RWTH Aachen University
{weingaertner, vomlehn, wehrle} @ comsys.rwth-aachen.de

ABSTRACT

Testing and evaluating the performance of actual software for wireless networks is difficult. Real-world wireless testbeds are costly and cumbersome to maintain. Measurement studies are complicated by many uncontrollable environmental influences, particularly the wireless channel. Network simulations on the contrary allow the convenient analysis of wireless networks with a maximum level of controllability; however they typically do not allow the execution of arbitrary and unmodified wireless communication software inside the simulation environment.

In this paper, we present a new network emulation architecture for the evaluation of wireless communication software. By bridging the gap between simulation and wireless software using a custom device driver, our framework enables arbitrary and unmodified wireless communication software to be evaluated in a fully simulated network. In accordance to this architecture we present a new 802.11 emulation framework based on ns-3 that allows the investigation of arbitrary Wi-Fi software for Linux. It eases both the development and the performance analysis of present and future Wi-Fi software.

Keywords

Network Simulation, Network Emulation, Wireless Software

1. INTRODUCTION

Both the development process as well as performance evaluations of software for wireless networks are often challenging and sometimes even painful. There are two prominent requirements in this regard. First , it is often vital to repeat an evaluation or an experiment multiple times in a deterministic fashion. Second, a major necessity is the aptitude of investigating real-world software. Ideally, a respective methodology or performance evaluation tool allows for the analysis of the wireless software in its genuine context, which usually is an entire operating system on a real machine.

SIMUTOOLS 2011, March 21-25, Barcelona, Spain

DOI 10.4108/icst.simutools.2011.245543

According to the development of new software for wireless networks researchers typically rely on *network simulations* or real-world *testbed deployments*. **Network simulations** are the primary evaluation technique of choice for to the analysis of wireless networks: Contemporary network simulators, such as OMNeT++ [30], JiST/SWANS [6], ns-2 [21] or ns-3 [24] facilitate the convenient investigation of wireless networks. In fact, these simulators provide models for a plethora of wireless network technologies, ranging from models of 802.11 [4,16,25] and its different sub-standards to cellular networking technologies like LTE [20] or sensor networks operating on IEEE 802.15.4 [9]. While all simulation tools inherit a very high degree of controllabiliy and scalability from the underlying concept of discrete event-based network simulation, one aspect disqualifies them as the one and only evaluation technology for wireless networks: With rare and specific exceptions [18], network simulators do not allow the execution of arbitrary wireless networking software in the simulation environment. This issue is critical for different reasons: First, this hinders studies of legacy wireless software for which no source code is available. Second, network simulation frameworks require the wireless software under investigation to be adapted to the simulation environment, which is mostly work intensive and often costly. In addition, simulation environments strongly differ from operating systems forming the original context in which wireless networking software is executed. Hence, it is difficult to uphold the original behavior of networking software if its ported to a simulation environment.

Real-world software deployments and testbeds naturally allow the investigation of genuine wireless software and hardware. Such testbeds are expensive both in terms of hardware costs and maintenance. Moreover, measurements with real-world wireless systems typically also suffer from many uncontrollable environmental conditions. For example the radio propagation on the wireless channel is prone to unpredictable interferences with other radio devices. In addition, node mobility usually originates from vehicle or human movement, which is difficult to reproduce. Hence repeating real-world measurements under equal conditions is hard if not impossible in most cases.

Network emulation [10] enables the analysis of real-world network systems and software in a fully controlled environment. For this purpose, these real-world communications systems pass their traffic through a network simulation that reproduces the behavior of an interconnecting network. Depending on the scenario's requirements, the form of the simulation may vary between a mere imitation of packet

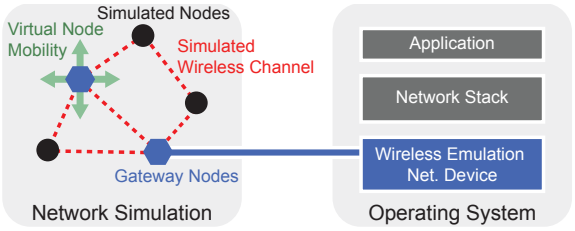


Figure 1: Conceptual overview of our wireless emulation approach: An instance of an entire operating system is integrated with the network simulation using a custom wireless network device driver. Traffic between the simulation and the device driver is exchanged at so-called gateway nodes. The network simulation models the wireless channel, other fully simulated nodes as well as potentially virtual node movement.

propagation characteristics and a provision of a fully simulated network consisting of simulated hosts, simulated protocol stacks and simulated network applications.

In this paper, we revisit the concept of network emulation for the evaluation of wireless networking software. Specifically, the contributions of this paper are the following:

- We introduce a new architecture for the emulation of wireless networks (Section 2). We integrate the wireless software by providing it with a virtual wireless network interface that behaves like a real wireless networking card, but instead handles the transmission and reception of data using a network simulation. This way, any real-world wireless networking software as well as routing and transport protocol implementations can be investigated inside a fully simulation-controlled environment. The simulation models the MAC and PHY layers, the wireless channel, potential node movement as well as other fully simulated nodes.
- We present an elaborate discussion of a new 802.11 emulation framework for ns-3 that we designed corresponding to this architecture (Section 3). Our framework allows any networking software for Linux making use of 802.11 wireless networks to be evaluated within ns-3 simulation scenarios of any kind.
- The evaluation in Section 4 shows that our framework accurately integrates the ns-3 Wi-Fi models with real-world networking software for Linux at the MAC layer, both according to latency and network bandwidth.

We discuss important related work in Section 6. The paper concludes with final remarks in Section 7.

2. SYSTEM ARCHITECTURE

Figure 1 shows a high-level view of our architecture for the emulation of wireless networks. Corresponding to the underlying concept of network emulation [10], our framework consists of two main components: The first is a host operating system (OS) that executes the wireless software to be investigated. We consider the wireless software to be any program or service that makes use of a wireless networking device. The second component is a discrete event-based network simulation, which models a virtual wireless

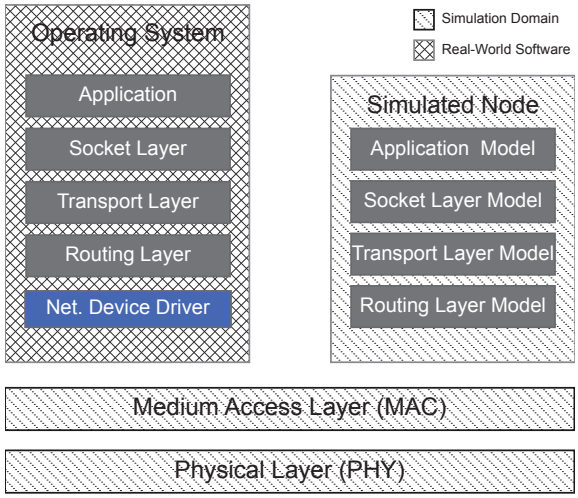


Figure 2: Stack organization in a device-driver enabled emulation scenario. We integrate the OS with the network simulation at the MAC layer. The network simulation models the MAC and the PHY layers; for fully simulated nodes it provides the entire protocol stack.

network containing both simulated nodes and so-called gateway nodes. The gateway nodes connect the simulation domain with the real-world software prototypes.

2.1 Real-world wireless network software

Our architecture integrates entire instances of operating systems executing the wireless software under investigation into the emulation set-up. The most important cornerstone in our architecture is a special device driver providing the wireless software with MAC-layer connectivity to the simulated wireless network. In the following we explain why these design decisions were made.

First, the major design requirement for our emulation architecture is to enable the incorporation of arbitrary wireless software into an emulation scenario. Any software making use of wireless communication, for instance ad-hoc routing protocol daemons, VoIP applications or legacy operating system applications, should be able to be included into the wireless emulation set-up. For this reason we need to provide the wireless software with its genuine environment that is of course the operating system context for which the software was developed for. Second, we generally assume that modifying the software for the inclusion into an emulation framework is not possible. Hence, we generally do not require source code to be available. This is true for many commercial applications.

Interfacing the wireless software with the simulation is generally possible at different levels of the protocol stack. One option would be to provide an alternative socket layer for the wireless software to link against, for example like in EmuSocket [3]. However, this constrains one to rely on TCP/IP for all means of wireless communication, and thus, investigating custom routing or transport protocol implementations becomes impossible. Similar problems hold for the interception at the IP layer (e.g. [28]), which require the wireless software to use IP for communication.

Figure 2 displays how we address this problem. The common language of all nodes in the simulation is the protocol used for wireless communication at the MAC layer. All MAC and PHY layer behavior is therefore modeled by the network simulator.

Our architecture employs a custom device driver behaving like a real-world wireless networking card to embed arbitrary software into a wireless emulation scenario. Besides sending and receiving data from other nodes, it also implements device specific actions such as scanning for access points, depending on the emulated wireless communication technology. In effect, any protocol stack or application that is capable of accessing network interfaces can be transparently used for wireless network emulation. This neither requires source code changes nor any other additional effort such as recompilations or relinking the code.

2.2 Network Simulation

The overall task of the network simulation is to model a wireless network, consisting of so-called *gateway nodes* and optional fully simulated nodes. The gateway nodes are stand-ins for the real-world software inside the simulation's virtual network topology.

The core functionality of the *gateway nodes* is to relay traffic originating at real-world wireless software over the simulated wireless channel. To enable the communication between gateway nodes and other nodes in the simulation, the gateway nodes only implement the physical and medium access control layers of the simulation.

Besides incurring the pure communication actions for the software prototypes, the gateway nodes also implement other functionalities usually carried out by wireless communication hardware. One example is reading Received Signal Strength Indicator (RSSI) values. In the real world, RSSI values indicate the signal strength associated with received packets. Typically such values are exported to the operating system and the software using an interface at the device driver. For the emulation case the RSSI values are the outcome of simulation models. In order to enable real-world software to access such "simulated" environment parameters, the driver bridges important parameters and properties of the wireless simulation model with the API of the host operating system. Similarly, the gateway nodes map other commands to corresponding actions in the simulation, for instance a request to scan for access points. Hence our architecture not only emulates the wireless communication characteristics but also the operating system interface of respective typical networking hardware.

It is also noteworthy that a major reason to rely on an event-based network simulator as an emulation engine is its capability of modeling additional environmental behavior in a deterministic way. Most notably this concept allows the simulator to implement *virtual mobility* support. Network emulation with virtual mobility allows one to investigate deterministically how real-software prototypes and their performance are affected by influences due to node movement. Similarly, the network simulation may also implement *simulated nodes*, for instance access points in the context of 802.11 networks or simulated hosts forming an arbitrary background network. This enables emulation scenarios to scale up to a larger degree in terms of node count or emulations containing network components that are not available otherwise.

2.3 Message Exchange

A crucial part of every device driver-enabled wireless emulation framework is the message exchange between the gateway nodes and device drivers that are associated with them. We assume that the network simulation and the wireless emulation device driver are typically executed on separate installations of an operating system. For example, if two physical machines are used, one might host the network simulation while the other runs the emulation device driver and the wireless software. The use of virtual machines of course makes it is also possible to run the network simulation as well as multiple OS installations with the driver on one physical computer.

The first important requirement regarding the communication between both components is *low latency*. As the execution of the operating system hosting the driver and the network simulation is usually not tightly integrated, the message exchange scheme directly influences the communication delay perceived by the wireless software. Hence, a low messaging latency is vital to avoid potentially significant performance loss regarding round trip times and end-to-end throughput between a gateway node and another node in the simulation. A second challenge is the adequate reproduction of *buffering* behavior. While network simulations mostly assume unlimited transmission queues, the capacity of transmission buffers found in real-world network devices is strictly limited. In order to obtain realistic performance measurements, for instance regarding the throughput measured on the emulated network device, we also need to emulate buffer capacity.

According to communication between the device driver and the gateway node, the following forms of message exchange between driver and simulation can be differentiated in a device-driver enabled wireless emulation tool-chain:

- **Driver (un-)registration**
  We define the wireless simulation to be the "master" component at which the device drivers may register and unregister at any point in time. This implies that the network simulation that models the environment for the gateway nodes has always to be instantiated prior to the wireless device drivers.

- **Exchange of data frames**
  The main type of message exchange is the transmission of data frames from the gateway node to the emulation device driver. Similarly, data packets delivered to the driver need to be transferred to the gateway node where they are injected into the simulated wireless network.

- **Status update notifications**
  The wireless driver needs to provide statistics and status information such as RSSI values. As this information is only available at the wireless communication stack of the gateway node, a data exchange mechanism for status information needs to be established.

- **Network hardware configuration and commands**
  As the wireless software might invoke certain commands typically carried out by the wireless networking hardware, the messaging scheme needs to forward them to the gateway node.

As we later discuss in Section 3.3, an actual implementation of the data exchange between driver and real-world simulation might also require the proactive transfer of information, e.g. to enable the timely access to status information.

2.4 Scalability

Like with any network emulation framework an important aspect is the degree of achievable scalability. More specifically, the question is how many emulated hosts can be modeled by the simulation (*simulation scalability*) and how many real systems can be attached to it (*emulation scalability*).

The *emulation scalability* is heavily dependent on the actual implementation of the gateway nodes and the message exchange with the device drivers. One important factor is the accumulated traffic between all gateway nodes and the the corresponding driver instances. It has to be kept within the bounds of available communication resources of the computer executing the simulation. In addition, each gateway node requires state information that is in the same magnitude as the one required by a simulated host. From our experience with our 802.11 implementation of device driver-enabled wireless network emulation (discussed in Section 3), however, we have learned that emulation scalability is not really an issue. Early experiments had shown that our framework is easily capable of handling a couple of dozens of driver instances at the same time.

Instead, the bigger limitation by far is *simulation scalability*. Given the computational complexity of wireless channel models and the detailed simulation of the MAC layer, the resource demands of wireless simulations grow fast with the number of simulated hosts and gateway nodes. As the network simulation needs to operate in real-time to be used for network emulation this hinders the set-up of large-scale wireless network emulation scenarios. Throughout the rest of this paper we accept this constraint and only focus on network emulation scenarios that employ real-time capable simulations. However, we have developed a network emulation platform called SliceTime [32] that eliminates this constraint. We also show in [32] how device driver-enabled wireless network emulation in conjunction with SliceTime can be used for large-scale 802.11 network emulation scenarios.

3. AN 802.11 EMULATION FRAMEWORK

Corresponding to the architecture presented in the previous section, we now describe the implementation of a device driver-enabled wireless emulation framework for 802.11 (Wi-Fi) networks. Although we focus on 802.11 throughout the rest of this paper, the concept of device driver-enabled wireless network emulation proposed in Section 2 may also applied to different communication technologies like Bluetooth or Zigbee.

Our 802.11 wireless emulation framework encompasses the following components:

- We rely on ns-3 for the simulation of the 802.11 network. We extended the 802.11 model of ns-3 with an implementation of an **802.11 gateway node** to enable the 802.11 model to be used for network emulation. In order to support further typical features of 802.11 such as scanning for access points (APs), only minor changes had to be applied to the model itself.
- We implemented a custom **Wi-Fi device driver** for Linux as loadable kernel module. It provides all functionality of a common wireless network device and supports the Linux Wireless Extensions. Thus any protocol implementation and Linux application can seamlessly be incorporated into a Wireless emulation set-up.
- We designed a lightweight and flexible **message exchange** protocol to integrate the functionalities of the driver and the gateway nodes.

We now describe important aspects regarding the implementation of the individual components.

3.1 Extensions of ns-3

We chose ns-3 as underlying network simulator because of its good support for data exchange with real systems. It already contains a real-time scheduler, which is a requirement to exchange network packets with external systems. Furthermore, the internal representation of network packets is the same as in real networks. Hence no explicit conversion is required when packets are exchanged with real systems.

3.1.1 The ns-3 802.11 model

The 802.11 network model used in ns-3 originates from the Wi-Fi model of an earlier discrete event-based simulation tool named *Yet Another Network Simulator (YANS)* [19]. Recent versions of ns-3 include detailed MAC layer and PHY layer simulation models for 802.11a and 802.11b networks [23]. ns-3 supports the simulation of infrastructure as well as the investigation of ad hoc scenarios. The ad hoc network implementation, however, is not complete, as it currently only sends the data frames themselves and does not contain management operations. Still, it can be used for the simulation of ad hoc networks, even though the behavior is not fully compliant with real 802.11 networks. The complete Wi-Fi model consists of several classes which form a stack of sub-layers.

3.1.2 The 802.11 gateway node

The 802.11 gateway node bridges the logic of the ns-3 Wi-Fi model with the Wi-Fi device driver. Figure 3 illustrates the module composition that forms a gateway node in our 802.11 framework. Due to the clean design of ns-3, we were able to implement the simulation part of our emulation framework by just adding two essential classes: The `WifiEmuBridge` module and the `WifiEmuComm` adapter. It centrally manages the data exchange between multiple gateway nodes and associated Wi-Fi device drivers.

The `WifiEmuBridge` module is the central cornerstone of our gateway node implementation. In order to enable the wireless software to send data over the simulated 802.11 network, it receives raw data frames via the `WifiEmuComm` adapter that originate from the Wi-Fi device driver (cf. Figure 2). In a similar fashion 802.11 data frames received on the simulated Wi-Fi channel are relayed to the driver using our `WifiEmuComm` adapter.

During the instantiation of the gateway node, the `WifiEmuBridge` module uses callbacks to register at the different sublayers of the ns-3 802.11 model. This is required to gather status information and statistics from lower layers during an emulation run. For example, the `WifiEmuBridge` obtains RSSI values from the WifiPhy component. Table 1 lists all the 802.11 status values that are supported by our

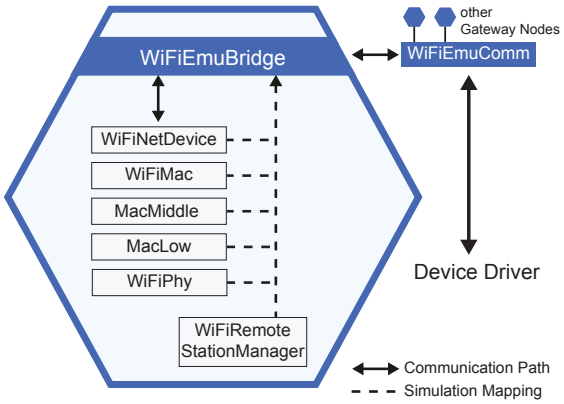


Figure 3: Structure of our ns-3 gateway node implementation: The WifiEmuBridge integrates the data transfer with the ns-3 Wi-Fi stack and maps simulation properties and actions to the model. The actual data communication with the device driver is carried out by one singleton object.

802.11 property	Description
RSSI	Received Signal Strength Indicator
Operation Mode	802.11 Infrastructure, Monitor or Ad Hoc mode
PHY standard	802.11 standard in use: a,b
Data Rate	The current data rate of the the interface, eg. 54Mbit
SSID	The SSID of the access point the gateway node is currently associated to
BSSID	The Basic Service Set Identifier of the network the gateway node currently belongs to
Channel	The number of the 802.11 channel currently used

Table 1: 802.11 status values and statistics supported by our Wi-Fi emulation framework. They are either accessible to the Wi-Fi software via RadioTap headers or the common API defined by the Linux Wireless Extensions.

Wi-Fi emulation framework. All of these are made accessible to the Wifi software via the Wi-Fi emulation device driver, either by implementing respective Linux Wireless Extension calls or using RadioTap headers.

The `WifiEmuComm` adapter centrally manages the data exchange between multiple gateway nodes and associated Wi-Fi device drivers. It is implemented as a singleton object, which is instantiated only once for the entire emulation scenario. If data frames are received from a Wi-Fi emulation device driver, `WifiEmuComm` dispatches them to the corresponding gateway node using a identifier sent along with the data frame. This design decision was made mainly for the reason of decreasing the complexity of the gateway node implementation. A second helper class encapsulates the low-level communication. This enables alternative implementations of the message exchange mechanism between `WifiEmuComm` and the device driver.

We further emphasize that our Wi-Fi emulation extensions only require minor changes to the 802.11 model, such as the addition of a few callbacks to access 802.11 status values. One important extension is scanning support; it is required to get common Wi-Fi software such as `iwlist` working. In order to enable scanning in a Wi-Fi emulation scenario, we extended an early prototype by Gustavo Carneiro [7] and incorporated it into our implementation.

3.2 Wi-Fi emulation device driver

The network driver which is part of our 802.11 emulation framework is implemented for the Linux operating system. The open nature of Linux makes it suitable to form the basis of a device-driver enabled wireless emulation tool-chain, as all parts of the system can be easily inspected and modified. However, from a conceptual point of view additional device drivers could be also implemented in an analogous way for any operating system providing general support for network communication.

Since the main goal of this driver is to represent the simulated wireless network card, it has to interact with the Linux network stack exactly like a driver of a regular wireless network card. Hence it has to make use of the interfaces Linux provides to access wireless network cards, whereby 802.11 wireless network cards are handled by Linux through the same kind of interface as Ethernet network cards. This interface [31] works as follows: During initialization or when hardware is found, a network card driver registers itself at the networking subsystem, providing a list of function pointers. These functions are called later during the execution by the networking subsystem to pass data which has to be sent, to retrieve statistics or to start and stop the network card. In turn, the network driver can call functions of the networking subsystem to start and stop its sending queue or to transfer received packets.

While this general network card interface already allows the driver to exchange network packets with the networking subsystem, it does not support any wireless network card specific features. For this purpose, the so-called *wireless extensions* [33] (wext) are added on top of this interface. Through a number of additional pointers to functions provided by the network driver, the Linux kernel can set or get additional parameters or retrieve statistics of the wireless network card. Starting from Linux version 2.6.22, a new interface called *cfg80211* [8] can be used instead of the wireless extensions. Our implementation, however, makes use of the classic wireless extensions, as they provide all necessary features and are available in previous as well as current Linux kernel versions.

3.3 Message Exchange

In order to integrate the 802.11 model and our gateway node implementation of ns-3 with the emulation Wi-Fi device driver, we have implemented a lightweight messaging interface that supports all communication primitives discussed in Section 2.3. In order to fulfill the low latency requirement, we need to keep the delays caused by message processing as low as possible. For this reason we developed a straightforward lightweight UDP protocol that embeds both data frames as well as status information in binary form.

This enables a rather efficient conversion of data structures using static typecasts in contrast to protocols that would introduce a far higher messaging complexity, for example protocols based on XML messages.

To provide efficient access to statistics and status information such as RSSI and BSSID, the gateway nodes push changes of this information to the device driver using our messaging interface. This decision was made for performance reasons, as the wireless interface of Linux splits access to 802.11 status messages into a series of system calls. By pushing all status information to the driver, all such requests can be answered locally without further interactions with the gateway nodes. In contrast to that, a pure polling approach would require a much higher amount of interaction between the emulation Wi-Fi driver and the network simulation and thus would introduce a higher messaging overhead.

In addition, we also equipped the driver with a *virtual transmission buffer* to emulate the limited capacity of sending queues found in real 802.11 network cards. If this feature is enabled, the device driver counts the number of bytes transferred to the gateway node. After the gateway node has sent the data on the simulated Wi-Fi channel, it instructs the driver to subtract the number of transmitted bytes from the counter again. Hence, the counter amounts to the number of bytes that are currently waiting to be sent. If this counter exceeds a certain configurable threshold, the virtual buffer is full and the emulation device driver blocks the network stack from sending new data frames.

4. EVALUATION

We now evaluate the accuracy of our driver-enabled 802.11 emulation framework regarding throughput and end-to-end latency. Later in this section, we also investigate the timing behavior of our driver-based integration of the ns-3 Wi-Fi model with Linux more precisely. All emulation runs were performed on a Dell Optiplex 960 machine, equipped with a 3 GHz Quad Core CPU, 8 GB of main memory and a 320 GB hard disk.

4.1 802.11 Throughput

We first investigate the throughput between two hosts in an 802.11 emulation scenario. Both nodes communicate with each other over a simulated Wi-Fi channel modeled by ns-3. For this experiment, we used a Xen [5] hypervisor with two virtual machines (VM) hosting Linux and the emulation driver as well as one VM that executed the Wi-Fi emulation framework based on ns-3. Figure 4 compares the throughput for both emulated 802.11a and 802.11b with real-world 802.11 measurements in infrastructure mode. The real-world measurements were obtained on a plain meadow (low interference) using two MacBooks running Linux and a Linksys WRT610N wireless router that serves as access point. The `TCP_STREAM` and `UDP_STREAM` tests of netperf [12] were used to measure the throughput for both the emulated as well-as the real-world 802.11 networks. The upper bounds are taken from [2].

For both 802.11 sub-standards and investigated transport protocols, our Wi-Fi emulation framework produces realistic throughput measurements in the right magnitude. Regarding 802.11a the throughput obtained using the emulation is slightly higher than the one measured in the real-world. It is the other way around for 802.11b: the measurements taken in the real-world outperform those of the emulated scenario. Such discrepancies according to the 802.11 throughput are well-known and not a specific property of our 802.11 emulation framework. For example, the measurements presented in [2, 11] show that the achieved throughput in 802.11 networks may be strongly influenced by the Wi-Fi hardware used.

4.2 802.11 Round Trip Times

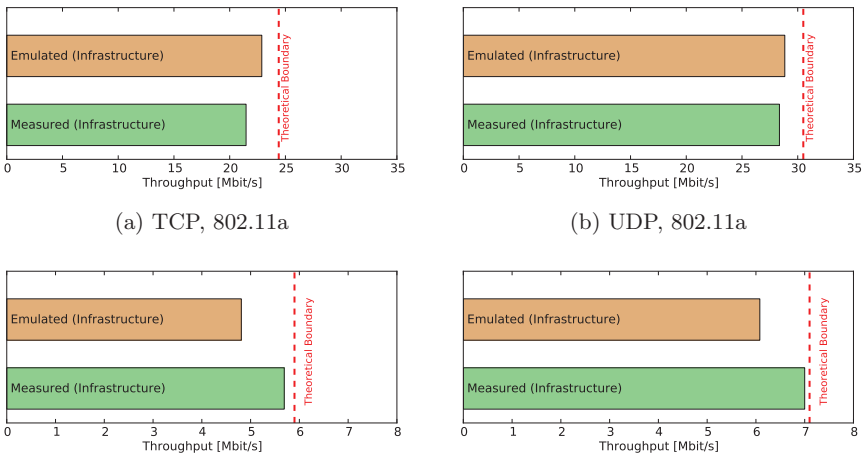
Analogous to the throughput measurements we now compare the round trip time (RTT) between two hosts in an emulated and real-world scenario. Figure 5(a) and 5(b) display the RTT distributions for 802.11a and 802.11b. Most notably the round trip times taken using our emulation framework are constantly lower than the RTTs measured in reality. We regard this difference as natural disparity caused by the abstractions of the ns-3 802.11 simulation model from the real-world behavior of Wi-Fi. The level of abstraction of the ns-3 Wi-Fi model for good reason increases at lower layers, as it is the case for most wireless simulation models. For instance, the Yans channel model [19] only approximates the typical delays of the 802.11 channel access. Other sources of delay, for example those imposed by the design of Wi-Fi hardware are also not reflected by the 802.11 model, as their implications on performance evaluations of network protocols and applications are mostly irrelevant. The potential delay differences between a Wi-Fi emulation set-up and a corresponding real-world scenario can be easily compensated, e.g. by introducing additional static or random delays in the 802.11 channel model.

4.3 Timing Analysis

We now investigate the timing behavior of our 802.11 emulation framework in more detail. Figure 6 breaks down the round trip times between a simulated host and a Linux VM into the individual communication actions between the Linux device driver and the ns-3 802.11 models. The boxplots visualize the absolute delay distributions of the individual communication actions. The color bar at the bottom of the plot shows how the average delays of the individual communication actions accumulate the total round trip time.

By far the largest fraction of the RTT is constituted by the ns-3 Wi-Fi simulation (denoted by `Simulated Wifi`), which also contains the time for accessing and data transmission on the simulated 802.11 channel. According to the communication actions introduced by our 802.11 emulation framework, `Simulation TX` accounts for the largest part of the delay overhead. It encompasses all delays caused by message processing inside the *WiFiEmuBridge* component when a packet is relayed over the simulated wireless channel over the gateway node. Altogether, the delay overhead caused by this and the other communication actions introduced by our 802.11 emulation framework amounts to less than a third of the overall communication delay. This fraction is certainly not negligible, however we have previously shown in Section 4.2 that our framework constantly achieves lower RTTs than a comparable real-world 802.11 deployment.

We conclude that the timing behavior of our driver-enabled 802.11 emulation framework is well suited for the analysis of Wi-Fi software, especially because the measured RTTs are constantly well below the reference measurements taken in a low-interference real-world deployment.



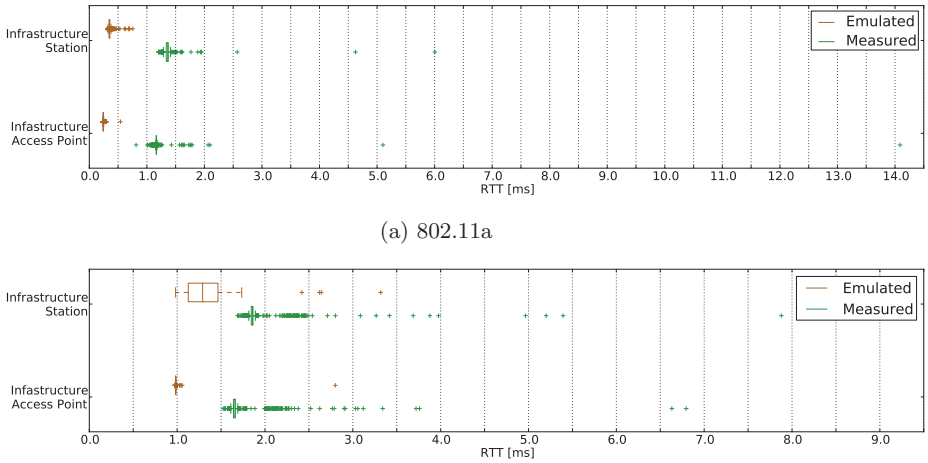
(a) TCP, 802.11a

(b) UDP, 802.11a

(c) TCP, 802.11b

(d) UDP, 802.11b

Figure 4: Throughput of emulated 802.11 compared with real-world measurements. Our 802.11 emulation framework reaches realistic throughput performance for both UDP and TCP.



(a) 802.11a

(b) 802.11b

Figure 5: Round trip times of 802.11 measured in a real experiment and in an emulated network. (whisker length: 1.5 IQR)

5. APPLICATION

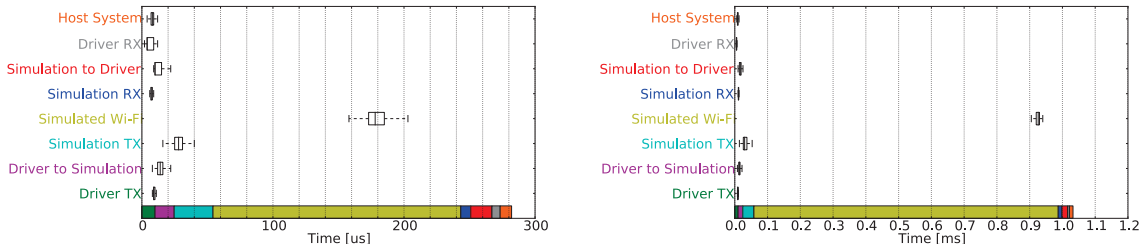
A main motivation behind the concept of device-driver enabled wireless network emulation and our corresponding 802.11 emulation framework is to enable the easy investigation of arbitrary wireless networking software in a fully simulated network. In this section we describe our Wi-Fi emulation framework from a user's perspective and show how we address this goal.

5.1 Network Simulation

In order to use an existing ns-3 Wi-Fi simulation scenario for network emulation only a few lines have to be added to the simulation program (see Figure 7): As in any standard network emulation set-up, we first instruct ns-3 to use its real-time scheduler to pin the execution of events to wall clock time. We also need to switch on the calculation of checksums for all packets to enable the communication with real-world hosts. In order to prepare the interaction with the device driver, one simply needs to instantiate a `WifiEmuBridge` and install it onto a simulated node (node 0 in this example), which forms the gateway node. Anything else in the simulation stays untouched, and of course, any feature or model of ns-3 may be used in conjunction with our 802.11 Wi-Fi extensions.

5.2 Device Driver

Figure 8 illustrates how the Linux device driver provides a virtual 802.11 networking device serving as entry point to the simulated network. First, two `insmod` commands are



(a) 802.11a (b) 802.11b

Figure 6: Timing analysis of our device-driver enabled 802.11 framework: The largest amount of a RTT is caused by the ns-3 Wi-Fi model and not by the interaction between device driver and gateway node. The color bar shows the accumulated average delay of the individual communication actions.

```
GlobalValue::Bind ("SimulatorImplementationType",
  StringValue ("ns3::RealtimeSimulatorImpl"));
GlobalValue::Bind ("ChecksumEnabled",
  BooleanValue (true));

WifiEmuBridgeHelper wbridge;
wbridge.SetAttribute("ClientId", IntegerValue(42));
wbridge.Install(c.Get(0), staDevice.Get(0));
```

Figure 7: Any ns-3 Wi-Fi simulation can be easily turned into an 802.11 emulation scenario using few lines of code

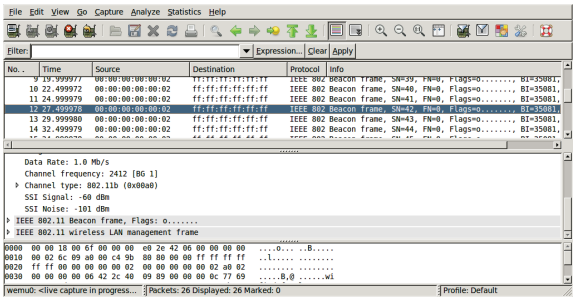
```
root@wifi-test2:~/wifi-emu-kern# insmod ./wifi-emu.ko client_id=42
&& insmod ./wifi-emu-udp.ko peer_addr=192.168.1.2

root@wifi-test2:~/wifi-emu-kern# iwconfig wemu0
wemu0   IEEE 802.11b  ESSID:"wifi-b"  Nickname:"wifi-emu"
        Mode:Master  Frequency:2.447 MHz
        Access Point: 00:00:00:00:00:02 Bit Rate:11 Mb/s
        Link Quality=45/100  Signal level=-56 dBm
        Noise level=-101 dBm
        Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
        Tx excessive retries:0  Invalid misc:0   Missed beacon:0

root@wifi-test2:~/wifi-emu-kern# iwconfig wemu0 mode Monitor
root@wifi-test2:~/wifi-emu-kern# ifconfig wemu0 up
```

Figure 8: Terminal output showing how to load and configure the wireless emulation driver for use in Monitor mode.

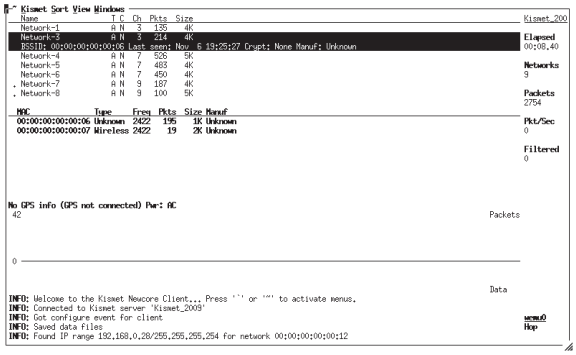
used to load and to initialize the emulation Wi-Fi device driver. The only parameters needed to instantiate the driver are the remote location of the network simulation and the ID of the gateway node to which the driver is associated. If the device driver is running, the output of `iwconfig` shows that the network device and its properties resemble a wireless networking card. Finally we configure the interface to operate in the 802.11 monitor mode, which instructs the gateway node to relay any frame received on the MAC layer to the the `wemu0` interface. The interface acts like a real 802.11 network card and supports the Linux wireless extensions.

5.3 Wireless Software

Once the ns-3 Wi-Fi simulation is running and the driver has been initialized, any networking software may access the emulated 802.11 network device. Figure 9(a) displays a screen-shot of the Wireshark network protocol analyzer



(a) Wireshark



(b) Kismet

Figure 9: Our 802.11 network emulation framework enables arbitrary unmodified networking software for Linux to be investigated in a Wi-Fi scenario modeled by ns-3.

monitoring an 802.11 infrastructure network modeled by ns-3. Here Wireshark is used to examine the RadioTap header of a packet received from an access point. The parameters shown correspond to the state descriptors of the MAC layer of the gateway nodes.

Figure 9(b) shows an unmodified version of the Kismet [15] wireless network scanner monitoring the simulated Wi-Fi network. The topology contains a number of access points and wireless stations, for which Kismet displays the MAC addresses corresponding to the ns-3 simulation scenario. The main purpose of Kismet is to passively scan for 802.11 stations. It internally makes use of the Linux wireless exten-

sions to implement Wi-Fi scanning and for gathering miscellaneous 802.11 information. The fact that our 802.11 framework is able to execute programs such as Kismet in an entirely simulated context demonstrates its ability to provide an investigation platform for wireless software that requires deep interaction with the wireless network device driver. Hence we expect our 802.11 emulation framework especially to be supportive for the analysis of ad hoc routing protocol implementations or Wi-Fi network management software.

6. RELATED WORK

The common ground across all wireless network emulation approaches is that the wireless channel is replaced by an artificial model which allows for an easier evaluation of wireless effects. Such modifications can be either performed as part of the wireless network hardware or in software.

Approaches performing these modifications in hardware range from attenuated wireless channels [14], over switchable antenna ports [27] to a complete emulation of wireless transmission through direct modification of the wireless signal [13]. The advantage of such approaches is that they make use of regular wireless network cards and therefore provide a very authentic environment to the software under test. However, they require even more hardware as in the case of of a regular wireless network testbed. This makes them costly and difficult to set up.

Performing wireless network emulation in software usually means to completely abandon wireless network hardware and to perform the emulation on a per-packet level. Approaches as the one presented by Noble et al. in [22] or *MobiEmu* [34] use wired local area networks in which they drop or delay packets according to a wireless network model. Others as *NEMAN* [26] or *CORE* [1] do not make use of a complete network, but instead provide virtual network interfaces through which the emulated wireless network is accessed. A disadvantage of such approaches is that they are limited by the fixed set of actions with which they modify the packet flow and the fact that they require a separate software instance for each station of the wireless network.

Wireless network emulation based on discrete event-based network simulation is more flexible in this regard, as it allows the inclusion of fully simulated nodes that are fully implemented as part of the network simulation. Similar to our wireless emulation framework, a few other wireless network emulation systems [17,28,29] are also based on discrete-event simulation. *JiST/MobNet* [17] provides wireless network emulation support based on the JiST network simulator. It provides virtual network interfaces in the system executing the network simulator which allow the transmission of IP packets over the simulated wireless network. This kind of interface, however, has some disadvantages: First of all, the integration at the network layer prevents the investigation of network protocols other than IP. The local instantiation of all network interfaces furthermore limits scalability and can lead to problems with regard to the routing functionality of the host operating system. Moreover, the provided interface does not support any wireless-specific functionality.

A system overcoming the last-mentioned limitation is presented by Seipold in [28]. It is based on ns-2 and provides similar network interfaces which support the Linux wireless extensions. However, it only supports locally executed applications that are limited to transmitting IP packets.

VirtualMesh [29] is an emulation framework based on OMNeT++. Like our wireless network card driver it provides an integration at the MAC layer and can instantiate emulation interfaces on both local and remote machines. It also supports the use of wireless-specific functionality, but provides its own custom interface for that. Hence, it is required to modify all software making use of this feature, for example for tools such as iwconfig or Kismet. By contrast, the wireless network card driver of our emulation framework provides the same interfaces as ordinary wireless network card drivers and hence allows the evaluation of arbitrary wireless network software without any changes to the software.

7. CONCLUSION

In this paper we presented a new architecture for the emulation of wireless networks. We employ a custom device driver that is tightly integrated with the MAC layer of a wireless network simulation. This way we form an emulation environment that not only emulates the wireless communication characteristics, but also the operating system interface. In contrast to other approaches this enables arbitrary unmodified and even potentially closed-sourced networking software to be investigated in a fully simulated wireless environment.

We have presented an in-depth discussion of our 802.11 emulation framework for ns-3 and Linux wireless software that was designed in accordance to this architecture. From our evaluation results we conclude that our framework is well suited for testing and for performance evaluations of wireless software. As we have shown, our 802.11 emulation framework is applicable to unmodified networking applications such as Kismet that make use of the Linux Wireless extensions. This opens up the possibility to apply wireless network emulation for the analysis of other software domains such as ad hoc routing protocols implementing link-layer awareness or 802.11 localization frameworks. We regard the validation of our framework for such kinds of use as future work.

As we believe that our Wi-Fi emulation framework will be useful for a number of researchers and developers, we have made the source code[1] available to the public. Our Wi-Fi framework is already integrated with SliceTime [32], which enables network emulation scenarios that incorporate network simulations of arbitrary complexity.

Acknowledgements

The authors thank Florian Schmidt, James Gross and Suraj Prabhakaran for helpful comments and discussions. This research was partially funded by different DFG grants and the UMIC excellence cluster, DFG EXC 89.

8. REFERENCES

[1] Ahrenholz, J., Danilov, C., Henderson, T., Kim, J., and Works, B. Core: A real-time network emulator. In *Proeedings of the IEEE MILCOM* (2008), pp. 1–7.

[2] Atheros. White paper – 802.11 wireless LAN performance. `http://www.atheros.com/whitepapers/atheros_range_whitepaper.pdf`, 4 2003. (accessed May 23, 2010).

[1] All source files are available at `http://www.comsys.rwth-aachen.de/research/projects/slicetime`

[3] Avvenuti, M., and Vecchio, A. Application-level network emulation: the emusocket toolkit. *J. Netw. Comput. Appl. 29* (November 2006), 343–360.

[4] Baldo, N., Requena, M., Nunez, J., Portoles, M., Nin, J., Dini, P., and Mangues, J. Validation of the ns-3 IEEE 802.11 model using the EXTREME testbed. In *Proceedings of SIMUTools Conference, 2010* (March 2010).

[5] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. In *Proc. SOSP'03* (Bolton Landing, NY, USA, Oct. 2003), ACM.

[6] Barr, R., Haas, Z. J., and van Renesse, R. JiST: an efficient approach to simulation using virtual machines. *Softw, Pract. Exper 35*, 6 (2005), 539–576.

[7] Carneiro, G. Ns-3: Wifi scanning patch. `http://www.nsnam.org/contributed/ns-3-wifi-scanning.tar.bz2`, 2009. accessed Oct 27, 2010.

[8] cfg80211 – Linux Wireless. `http://wireless.kernel.org/en/developers/Documentation/cfg80211`. accessed May 10, 2010.

[9] Chen, F., and Dressler, F. A Simulation Model of IEEE 802.15.4 in OMNeT++. In *6. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze, Poster Session* (Aachen, Germany, July 2007), pp. 35–38.

[10] Fall, K. R. Network emulation in the Vint/NS simulator. In *4th IEEE Symposium on Computers and Communication* (1999).

[11] Giustiniano, D., Bianchi, G., Scalia, L., and Tinnirello, I. An explanation for unexpected 802.11 outdoor link-level measurement results. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE* (2008), pp. 2432 –2440.

[12] Jones, R., Choy, K., and Shield, D. Netperf. [Online] Available `http://www.netperf.org` December 21, 2009.

[13] Judd, G., and Steenkiste, P. Repeatable and realistic wireless experimentation through physical emulation. *ACM SIGCOMM Computer Communication Review 34*, 1 (2004), 63–68.

[14] Kaba, J. T., and Raichle, D. R. Testbed on a desktop: strategies and techniques to support multi-hop manet routing protocol development. In *Proceedings of ACM MobiHoc 2001* (New York, NY, USA, 2001), ACM, pp. 164–172.

[15] Kershaw, M. Kismet wireless network detector and sniffer. `http://www.kismetwireless.net` (accessed Oct.2010).

[16] Koepke, A., Swigulski, M., Wessel, K., Willkomm, D., Haneveld, P., Parker, T., Visser, O., Lichte, H., and Valentin, S. Simulating wireless and mobile networks in OMNeT++: The MiXiM vision. In *Proc. SIMUTools 2008* (2008), pp. 1–8.

[17] Krop, T., Bredel, M., Hollick, M., and Steinmetz, R. JiST/MobNet: combined simulation, emulation, and real-world testbed for ad hoc networks. In *Proc. WinTECH'07* (New York, NY, USA, 2007), ACM, pp. 27–34.

[18] Lacage, M. Direct Code Execution with ns-3. Talk given during the "Workshop on ns-3", March 15th, 2010, Malaga, Spain. `http://www.nsnam.org/workshops/wns3-2010/code-execution.pdf`, 3 2010. Accessed Februrary 4, 2011.

[19] Lacage, M., and Henderson, T. Yet another network simulator. In *Proceeding from the 2006 workshop on ns-2: the IP network simulator* (2006), ACM, p. 12.

[20] long Qiu, Q., Chen, J., di Ping, L., fei Zhang, Q., and zeng Pan, X. LTE/SAE model and its implementation in ns-2. pp. 299 –303.

[21] McCanne, S., Floyd, S., Fall, K., Varadhan, K., et al. Network simulator ns-2, 1997.

[22] Noble, B., Satyanarayanan, M., Nguyen, G., and Katz, R. Trace-based mobile network emulation. In *Proc. SIGCOMM'97* (1997), ACM New York, NY, USA, pp. 51–61.

[23] Ns-3: Wifi models. `http://www.nsnam.org/doxygen-release/group___wifi.html`. accessed May 9, 2010.

[24] ns-3 Website. `http://www.nsnam.org/` (accessed Oct. 2010.

[25] Papanastasiou, S., Mittag, J., Strom, E. G., and Hartenstein, H. Bridging the gap between physical layer emulation and network simulation. pp. 1 –6.

[26] Pužar, M., and Plagemann, T. NEMAN: A network emulator for mobile ad-hoc networks. Tech. Rep. 321, Department of Informatics, University of Oslo, 3 2005.

[27] Sanghani, S., Brown, T., Bhandare, S., and Doshi, S. EWANT: the emulated wireless ad hoc network testbed. In *Proc. IEEE WCNC 2003* (2003), pp. 1844–1849.

[28] Seipold, T. Emulation of radio access networks to facilitate the development of distributed applications. *JOURNAL OF COMMUNICATIONS 3*, 1 (2008), 1.

[29] Staub, T., Gantenbein, R., and Braun, T. VirtualMesh: an emulation framework for wireless mesh networks in OMNeT++. In *Proc. SIMUTools'09* (Brussels, Belgium, 2009), pp. 1–8.

[30] Varga, A., and Hornig, R. An overview of the OMNeT++ simulation environment. In *Proc. SIMUTools 2008* (Marseille, France, March 2008).

[31] Wehrle, K., Pählke, F., Ritter, H., Müller, D., and Bechler, M. *Linux Networking Architecture – Design and Implementation of Networking Protocols in the Linux Kernel*. Prentice-Hall, 5 2004.

[32] Weingaertner, E., Schmidt, F., vom Lehn, H., Heer, T., and Wehrle, K. Slicetime: A platform for scalable and accurate network emulation. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI '11)* (3 2011), USENIX.

[33] Wireless Tools for Linux. `http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html\#wext`. accessed May 10, 2010.

[34] Zhang, Y., and Li, W. An integrated environment for testing mobile ad-hoc networks. In *Proc. MobiHoc'02* (New York, NY, USA, 2002), ACM, pp. 104–111.