FANTASY: Fully Automatic Network Emulation Architecture with Cross-Layer Support

Ismet Aktaş, Hendrik vom Lehn, Christoph Habets, Florian Schmidt, Klaus Wehrle
Chair of Communication and Distributed Systems,
RWTH Aachen University, Germany
{aktas,vomlehn,habets,schmidt,wehrle}@comsys.rwth-aachen.de

ABSTRACT

Testing and evaluating real-world wireless and mobile systems is very difficult. The volatile nature of the wireless medium and mobility complicates their evaluation. The access to system information hindered by the operating system further increases the evaluation of a real-world system. In contrast, a simulator allows to easily set up complex wireless and mobile scenarios, log protocol variables of interest and to repeat the whole test easily if desired. Developers of real-world systems also want to perform tests with the simplicity and convenience of a simulation without loosing the ability to execute arbitrary networking software in its genuine environment (an operating system).
In this paper, we present FANTASY, a new network emulation architecture that allows the fully automated setup and execution of an experiment, enables the convenient access to system information and the collection of test results. With the integration of the cross-layer architecture CRAWLER, we demonstrate that we are able to monitor parameters across protocol layers and to evaluate network emulation scenarios where cross-layer optimization is involved.

1. INTRODUCTION

Developers of software for wireless networks have to cope with a number of difficulties. The wireless medium and the distributed nature of the systems complicate the testing of such software. One typical problem is that test setups come with high requirements in terms of hardware and space. Another limitation is that the repeated execution of a test under the same conditions is not possible. While implementations have to be able to cope with the rapidly and unpredictably changing behavior of the wireless channels, during testing, it is beneficial to precisely control the environment to evaluate the influence of different parameters on the system behavior. Including mobility in tests is cumbersome and comes at the cost of human interaction [10] or highly sophisticated test setups [5]. Another problem in this context is that it can be difficult to access relevant system information that is required to evaluate a system. Due to the fact that a number of different systems are involved in such a test, the setup followed by an execution of a test and the collection of results rapidly get quite complex.

Network simulation tools are especially built to solve the aforementioned problems, but have their own disadvantages: Regular simulation tools only allow a very limited use of existing software code. This leads to the problem that the system under test has to be implemented twice – once for the use in a network simulator and a second time for use in real systems. This means that only the system concept can be tested, not the actual implementation. Furthermore, simulation models are often too abstract and do not take into account important effects that are caused by an operating system, such as scheduling.

Network emulation [9] combines simulation and real world testing to benefit from both worlds. However, this is only partially possible in such a combined setup. For example, the real systems that form part of the network emulation setup are not as transparently accessible as the simulated parts. Furthermore, network emulation still requires complex setup for the testbed machines.

In this paper we present FANTASY, a new network emulation architecture that facilitates the evaluation of wireless network software. Our main contribution is a centrally controlled system that allows the fully automated setup and execution of an experiment, enables the convenient access to system information, and automates the collection of test results. The target audience of FANTASY are developers of wireless network software that want to perform tests with the convenience of a network simulation, but nevertheless require the use of real implementations and full operating systems for the system under test. We alleviate the problem of complicated access to relevant system information by integrating the CRAWLER [1] architecture into FANTASY, which provides a unified interface for system information access. Furthermore, since CRAWLER's original focus is on facilitating cross-layer communication, FANTASY is especially suited as a rapid prototyping and testing tool for the design of cross-layer optimizations. We will show one example of a cross-layer optimization that was implemented and tested with FANTASY in Section 4.

The remainder of this paper is organized as follows: Section 2 presents a system overview, introduces the components that FANTASY is composed of and highlights our design goals. In Section 3, we describe our architecture in more detail and explain how we achieve the fully automatic setup and execution of experiments. We evaluate FANTASY in Sec-

Simutools 2012, March 19-23, Desenzano del Garda, Italy

DOI 10.4108/icst.simutools.2012.247759

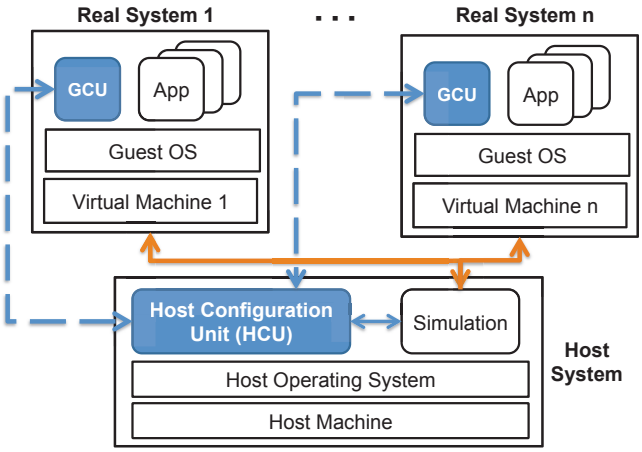


Figure 1: Conceptual view of FANTASY.

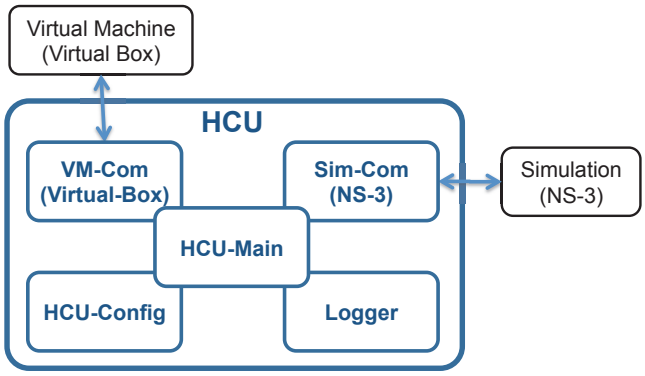


Figure 2: Overview over the Host Configuration Unit (HCU), its interfaces, and its subcomponents.

tion 4 and discuss related work in Section 5. Finally, we describe future improvements in Section 6, before concluding the paper in Section 7.

2. DESIGN OVERVIEW

The main goal of FANTASY is to simplify the process of testing wireless network software. Through a combination of suited emulation components and the support of fully automatic experiment setup and execution, developers shall be able to perform experiments with real software prototypes as easily as with a network simulation. This section provides an overview over the overall design of FANTASY and the used components.

In order to enable developers to test arbitrary networking software in its native environment (an operating system), but nevertheless minimize hardware requirements, we opted for the use of virtual machines that execute the systems under test. Because of its widespread use and easy configuration, we have chosen VirtualBox as virtualization software.

With FANTASY, those parts of an emulated setup that do not run in virtual machines are simulated using the ns-3 simulation software [18]. ns-3 is well suited for this task since it contains detailed models of the MAC layer [2, 17] and already comes with support for network emulation. Part of this is a real-time scheduler that runs the simulation in real time, which is required in order to allow the exchange of networks packets.

To connect the simulation with the virtual machines, FANTASY utilizes two components that have been developed as part of the SliceTime project [21]. For the emulation of Ethernet devices, a tap device which connects to ns-3 using UDP datagrams is created in the system that runs inside the virtual machine. A wireless emulation driver is used for the emulation of wireless network connections. This device driver creates a virtual network device which provides the same interfaces as a real 802.11 wireless network card, including the wireless extensions [23]. Both devices have in common that they provide interfaces to the guest operating system, but forward all sent and received frames to corresponding models inside the ns-3 network simulation.

An advantage of network simulations that is usually lost in case of network emulation [9] is the convenient access to relevant information in the network stack. Through the incorporation of the CRAWLER architecture, FANTASY supports easy access to such information of the systems that run inside the virtual machines. Furthermore, this combination allows to conveniently evaluate the effects of cross-layer optimizations using real-world systems. The only restriction regarding the software that runs inside the virtual machines is that they are constrained to Linux systems, as the wireless emulation driver and CRAWLER have been developed for Linux.

By using the aforementioned components, FANTASY allows the emulation of diverse scenarios on a single computer. However, the overall setup and execution of an experiment is still quite complex when compared to the ease of a pure network simulation. To further simplify these processes, we developed two components that allow to control the whole setup from a central place. The host configuration unit (HCU) is running on the host computer that accommodates the network simulation and the virtual machines. It instantiates the virtual machines, starts the network simulation and is in charge of the overall setup. As part of the systems that are running inside the virtual machines, the guest configuration unit (GCU) waits for commands from the HCU. It loads the wireless emulation driver, configures CRAWLER, starts processes that are part of the experiment and allows the collection of test results to a central place.

Figure 1 gives an overview of the overall setup that is used by FANTASY. The functionality of HCU and GCU are described in more detail in the following section.

3. ARCHITECTURAL DETAILS

This section provides more detailed information on how HCU and GCU are used to control whole experiments in an automated fashion.

3.1 Host Configuration Unit (HCU)

The Host Configuration Unit (HCU) is executed on the host system and controls the entire experiment. It consists of five subcomponents as shown in Figure 2. While designing the HCU we kept it modular to group functionality for usability and maintainability reasons. Moreover, the modularity provides us with exchangeable modules such as the VM-Com subcomponent that is tailored for Virtual Box but can simply be exchanged for another virtual machine soft-

ware. This holds also for the Sim-Com subcomponent that is tailored for ns-3 but can also be exchanged for another simulator. The HCU-Main subcomponent is interconnected with all subcomponents and controls them.

The initial step to conduct a test setup is done by writing a configuration file. The configuration includes the required parameters for the simulation as well as instructions that are performed on the virtual machines. The configuration file is read and executed by the HCU-Config subcomponent. After the experiment, the logged values for all parameters specified in the configuration are collected from all virtual machines and are given back to the Logger subcomponent. In the following we describe how such values can be configured in order to monitor them and how to setup up a complete experiment.

3.1.1 Configuration

Typically when testing is involved, the experimenter has to adjust parameters to evaluate the effects. Sometimes this also requires to repeat the test several times to have stable and credible experimentation results. It is desirable that both of these steps are very simple to achieve. Therefore, the aim of our configuration is to support a very automated, customizable and easy-to-use testing environment.

In our configuration a whole test setup with different settings is described in a configuration file. The configuration contains everything necessary to automate diverse test settings of the simulation as well as application and cross-layer settings in the VM. Repeatability of the test setup is given by re-executing the configuration file which can also be configured. A configuration example is shown in Listing 1. Here, the configuration is subdivided into sections by using the keywords [ns3], [vm], [schedule] and [logger].

In section [ns3] simulation related settings are listed. Line 3 indicates which ns-3 version is used. In line 7 the ns-3 configuration, already preconfigured, is loaded. The parameters of the ns-3 configuration can be re-adjusted within the configuration by assigning values to parameters with the use of tuples as shown in line 16.

Virtual machine related settings are listed in section [vm]. For example, the number of emulated nodes is set on line 20 and their VM image is loaded from a path given on line 23.

The main part of the configuration is the [schedule] section which determines when to execute which instruction. For example, in line 28 indicated with iperf, we used two tuples. We have introduced a three tuple notation `(<vm>, <time>, <command>)`; the first index indicates the VM, the second index the relative time in seconds. Note that the experimentation starting time between all VMs is synchronized, i.e., all VMs are set once to a common time zero. After the simulation has started, the HCU receives a signal and itself sends a signal to all VMs setting them to the common time zero. The third index gives instruction about what should be done on that particular VM. If we come back to the example at line 28, an iperf server is started on VM 2 at time 4 and an iperf client on VM 2 at time 5. Similarly, at line 29 a monitoring application is started on VM 1 at time 5 that uses the CRAWLER shared library to monitor the congestion window (CWND) of TCP. The outcome of the monitored parameter is stored in a log file. The same holds for line 30 where the frame error rate (FER) is observed.

To deliver the logged values from the GCU within the VMs to a central place, the [Logger] section is used. For example, line 34 indicates which values should be delivered and line 36 indicates where these values should be stored. Results for this particular configuration will be presented in Section 4.

```
1 [ns3]
2 # Path to the ns-3 folder
3 ns3_path: /home/crosslayer/ns-3.7-slicetime
4
5 # Path to the ns-3 configuration file;
6 # leave empty to use default configuration
7 ns3_config: /ns_3/usecase2.cc
8
9 # Offset in seconds to start the HCU after simulation
10 hcu_start_offset: 0.5
11
12 # Simulation duration in seconds
13 ns3_duration: 50
14
15 # (Parameter,value)-pairs to adjust ns-3 config
16 ns3_param_value: [("sim_node_count",1),("protocol","
       TCP"),("wlan","a")]
17
18 [vm]
19 # Number of VMs to be started
20 vm_machines: 2
21
22 # Template file for the VMs
23 vm_templ: ./virtual_machines/templates/default.vdi
24
25 [schedule]
26 wifi:[([1,2], 0.7, "load_wifi_emu")]
27 crawler:[(1, 1, "load_crawler tcpLayer CLKernelModule
       /src/test/enabletcp.ko")]
28 iperf:[(2, 4, "iperf -s -p 5001"), (1, 5, "iperf -c
       192.168.1.2 -t 30 -y c -p 5001 -x CMSV -i 1 &>
       iperf.log")]
29 tcp_cwnd:[(1, 5, "crawlerapp monitorapp transport.tcp
       .tcp_out_5001.cwnd &> /home/crosslayer/gcu/
       tcp_cwnd.log")]
30 fer:[(1, 1.5, "crawlerapp monitorapp wemu0.
       wireless_stats.qual.fer &> /home/crosslayer/gcu/
       fer.log")]
31
32 [logger]
33 # what should be logged
34 log:[(1,"iperf.log"),(1,"tcp_cwnd.log"),(1,"fer.log")
       ,(1,"gcu.log")]
35 # Path to where the log files should be kept
36 log_path: ./logs
```

Listing 1: A simple configuration file in FANTASY.

So far, we have only seen a fixed test setup, a single experiment without changing parameters. What happens if we want to have a slightly different test setup? To relieve the user from having to write an additional configuration file for only minor changes we distinguish between main and custom configuration files. The main configuration file describes the major test setup while the custom configuration only includes the differences. The custom configuration overwrites the respective values of the main configuration. For example, if we want to use a different simulation configuration as given in the main configuration shown in line 7 of Listing 1, we have to add only a single modified line in the custom configuration to the respective section ([ns3]), e.g., `ns3_config:./ns_3/usecaseXX.cc`, that overwrites the main configuration.

However, when many similar tests with only minor changes are supposed to be conducted, the process of creating custom configuration can be very cumbersome. To alleviate this process, we have implemented a configuration generator which is explained next.

3.1.2 Configuration Generator

The configuration generator is an interactive tool that helps to create several custom configurations. For each of the parameters needed by a main configuration, the configuration generator asks for assignments. If a default value is present for a parameter it can be kept by just pressing enter and moving on to the next parameter. We have introduced a list notation in brackets that allows to assign several values to a variable. The following example assigns several values to the variable `ns3_param_value`:

```
("sim_node_count", [0,1,2,3,4,5,6,7,8,9,10,11,12]),
    ("protocol",["UDP","TCP"]),("wlan",["a","b"])
```

For each of the parameters within these tuples one value is assigned from the list. In this particular example, the permutation of all these assignments allows to conduct 52 tests, all of which can be started by issuing one command that, for ease of use, is given to the user when the configuration generator finished successfully. This is a shortened example that we also use later in the evaluation Section 4.2. The only difference is that we configured more simulated nodes (ranging till 20 nodes), more runs (10) and one additional assignment to another variable (TCP and UDP traffic between emulated nodes) which leads to 1680 different test runs for generating the experimentation results as presented in Section 4.2. As a result, with the help of the configuration generator, custom configurations are automatically generated and then used to run automated tests without any further interaction from the experimenter.

3.2 Guest Configuration Unit (GCU)

The HCU gives instructions on what should be done when a certain time arrives such as starting the simulation or starting a real application on a VM. However, these instructions have to be received and performed on the VMs by a counterpart. This is done by the Guest Configuration Unit (GCU). Through the GCU, the HCU has the power to fully control the VMs. Within each VM, the GCU is executed when the system is started. This entails that the experimenter must install the GCU software on the VM template which the HCU clones before starting the experiment.

After the VM is booted and the GCU started, it performs some initials tasks and then waits for further instructions from the HCU. Figure 3 shows the main responsibilities of the GCU. These responsibilities include loading applications that have been determined in the configuration (like iperf in Listing 1). The GCU also loads CRAWLER [1] which allows passive monitoring and active manipulation of protocol and system variables within the VM, as well as the WiFi emulation driver [21] which couples the VM with the ns-3 simulation.

In a simulation, accessing protocol variables is relatively easy. Most network simulators are designed with ease of access and observation of protocol or system variables in mind. However, in a real system, access to many such variables is restricted. This is mainly due to the protocol stack being integrated into the operating system which only provides few limited interfaces because of security concerns. To facilitate access to protocol and system variables in a real machine, we use the CRAWLER framework. FANTASY integrates support in a way that hides the details of the CRAWLER implementations and relieves the developer from directly configuring the framework. We do this by providing a wrapper

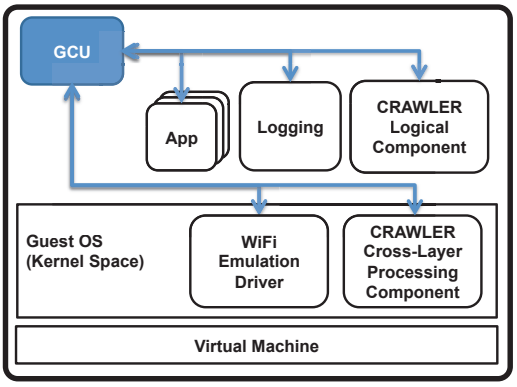


Figure 3: Overview over the tasks and responsibilities controlled by the guest configuration unit (GCU).

called monitorapp that translates FANTASY logging instructions into CRAWLER monitoring instructions (cf. Listing 1, lines 29 and 30).

Since CRAWLER originated as a cross-layer optimization framework, it is of course also possible to use it for this purpose. Developers who want to test cross-layer optimizations in a VM can do so without any further changes to the standard FANTASY setup. The configuration to define such optimizations is syntactically very similar to FANTASY's configurations and therefore poses little additional learning effort. Detailed discussion of the setup of cross-layer optimizations with CRAWLER is, however, beyond the scope of this paper, and are explained in [1].

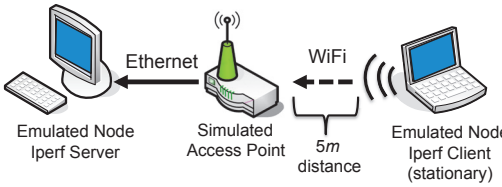
3.3 Implementation

FANTASY is a combination of many different tools and libraries. Of those, two of the vital pieces, CRAWLER and the WiFi emulation driver tightly integrate into the Linux kernel. This means that, while the developer is free to choose a distribution, they are restricted to Linux as operating system. For our tests, we used Ubuntu distributions. All components of the HCU and GCU are implemented in Python [19].

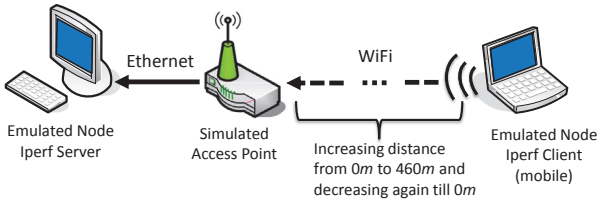
The communication channel between the HCU and the GCU is realized through a separate virtual network in which the HCU provides IP addresses to the GCUs via DHCP. As a communication protocol between the HCU and GCU we used the Python Remote Objects (Pyro) [6] package for Python [19], which runs a daemon inside each VM allowing the HCU to connect to it. Thus, the HCU is able to call functions of an instance of the GCU as if they were local objects.

4. EVALUATION

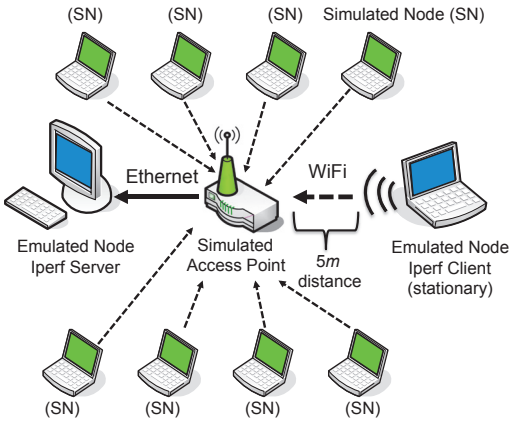
With our evaluation we show different test cases where we emphasize different features of FANTASY. In each test, we show a subset of these features. The highlighted features are: (i) comparability of results between real world tests and FANTASY (ii) cross-layer support for emulated nodes that allows us to passively monitor protocol and system information as well as accessing them actively; (iii) repeatability of experiments; (iv) support of mobile wireless scenarios; (v) automation and rapid testing capabilities. We will also give some insight into the scalability of FANTASY.



(a) Test case 1: comparison of real world tests and emulated results with FANTASY



(b) Test case 2: monitoring several parameters across several protocol layers of a mobile emulated node



(c) Test case 3: testing the scalability by adding simulated nodes until the simulation's real-time capability is lost.

Figure 4: Topology of the three test case scenarios that were used for evaluation.

4.1 Demonstrating Areas of Application

In this section we give two examples to demonstrate the areas of application for FANTASY. In the first example we have conducted real world tests and we compare them with the emulation results achieved with FANTASY. This also highlights the integrated cross-layer support. In the second example, we showcase simulated mobility as well as monitoring of parameters of several protocols in the emulated node's Linux kernel. In both test cases, the physical layer and channel are simulated with ns-3's channel model. We used the two-ray ground propagation loss model that is part of ns-3 [17] for our simulated wireless channel.

4.1.1 *Test Case 1: Comparability, Cross-Layer Optimization Support*

We present a test case where we changed TCP's congestion control algorithm depending upon the underlying network conditions. The motivation for this optimization is as follows: TCP CUBIC [11] is the standard congestion control algorithm in the Linux kernel since 2.6.19 due to its superior performance and fairness properties under most network conditions. However, TCP Westwood [16], specifically developed for wireless communications (such as in WiFi), provides better throughput in changing network conditions with high loss rates. The idea is to specify a cross-layer optimization that, based on the observed network conditions, switches between different congestion control algorithms at runtime without reinitializing the TCP connection. How cross-layer optimizations can be realized with CRAWLER is explained in our prior work [1] and is not within the scope of this paper.

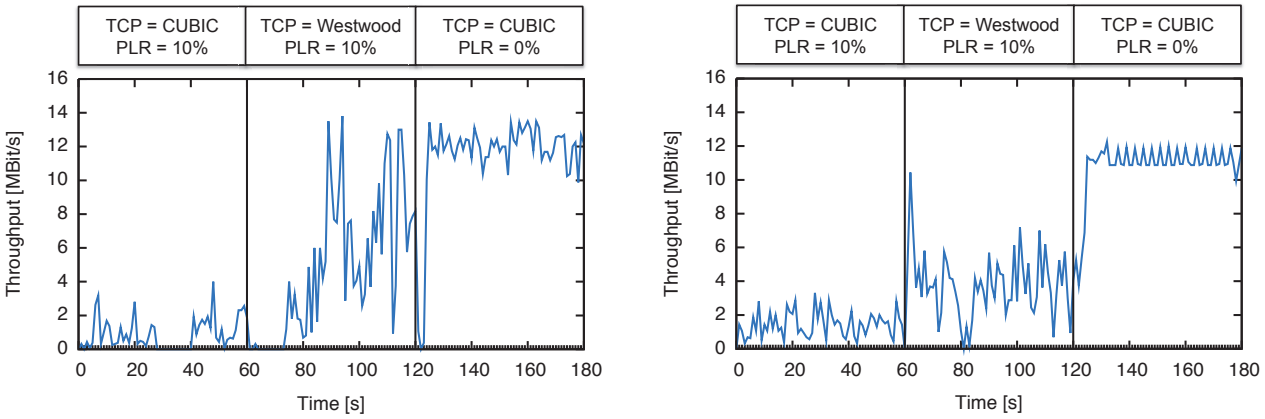
In the real world testbed, our test setup consists of two PCs and one 802.11g access point. One PC runs the cross-layer optimization configured in CRAWLER, and is connected to the access point via WiFi. On this PC we use iperf [20] to create TCP traffic, and netem [12] to create different packet loss rates (PLRs). The other PC serves as the destination for iperf traffic; it is connected to the access point via Ethernet. In FANTASY, the two PCs are emulated virtual machine nodes that use the emulated network drivers to connect to a simulation setup. This simulation simulates an 802.11a access point since 802.11g functionality is not given in the used ns-3 version; but 802.11a should show the same behavior. The test setup is depicted in Figure 4(a).

Figure 5 shows the results of our experiments. For the first 120 seconds, we have set the PLR to 10% which is a very significant PLR for TCP. For the first 60 seconds, the optimization is not active, as depicted by the low TCP throughput achieved during this time. The optimization is activated at 60 seconds which triggers the switch from CUBIC to Westwood and subsequently improves the throughput. At 120 seconds, we set the PLR to 0%, so that TCP switches back to CUBIC and thus achieves a consistently higher throughput. Although the results obviously are not perfectly similar, they show the same tendency. The differences can be mainly attributed to channel effects; the emulated setup uses a simple path loss model and does not account for small-scale fading or interference from other nearby machines that use a wireless connection, but are not part of the test setup.

This test case demonstrates the following features: We have gained comparable results between real world tests and our architecture. Furthermore, we showed an example of executing a cross-layer optimization and continuously monitoring a protocol parameter. As FANTASY also supports mobility and monitoring of diverse parameters across protocol layers and system components, we want to give an additional example where we demonstrate these features.

4.1.2 *Test Case 2: Monitoring, Mobility, Repeatability*

For our second test case (cf. Figure 4(b)), we use an emulated setup similar to the previous one. However, we use a standard TCP setup without any cross-layer optimizations. Instead, we introduce mobility by simulating a constant speed movement. The node that is connected to the AP via WiFi starts at a distance of 0m to the access point and moves to a distance of 460m at a speed of 1m/s. Upon reaching that point, it moves back to 0m at the same speed. To



(a) Testbed measurements

(b) Emulated measurements with FANTASY

Figure 5: Comparison of throughput measurements for a cross-layer optimization scenario. TCP's congestion control algorithm is changed based on the changing packet loss ratio (PLR) and the received signal strength indicator (RSSI).

showcase the monitoring capabilities of FANTASY, we logged several key metrics throughout the experiment. The setup is the result of the configuration shown in Listing 1 and discussed in Section 3.1.1.

The monitored parameters are shown in Figure 6. From the application layer we logged the throughput measurements gained by iperf, from the transport layer the congestion window (CWND), and from the WiFi emulation device driver the frame error rate (FER). We chose these parameters for two reasons. First, they are good candidates to demonstrate the effects of mobility and packet loss. Second, we show that we are able to monitor different layer or system components within an emulated node.

As can be seen in Figure 6, the throughput is reduced with increasing distance. The step pattern of the throughput curve shows the effect of rate adaptation. At around 400s, the distance becomes too large for any meaningful communication: virtually all frames are dropped due to errors, and no TCP packets are received any more. After the node reaches its maximum distance and slowly returns back to the AP, the communication recommences at around 550s, and throughput gradually increases afterwards.

To investigate whether we can produce repeatable test setups with FANTASY, we ran this scenario several times with identical simulation settings. As can be seen in Figure 7, the throughput is almost exactly the same over all runs. Other measured parameters were also highly similar to each other.

With this test case, we demonstrate FANTASY's capabilities to model mobile wireless scenarios. We also give an example of monitoring and logging of system and protocol parameters inside a virtual machine, and we use this to show how FANTASY produces very similar results over several runs with identical simulator setups.

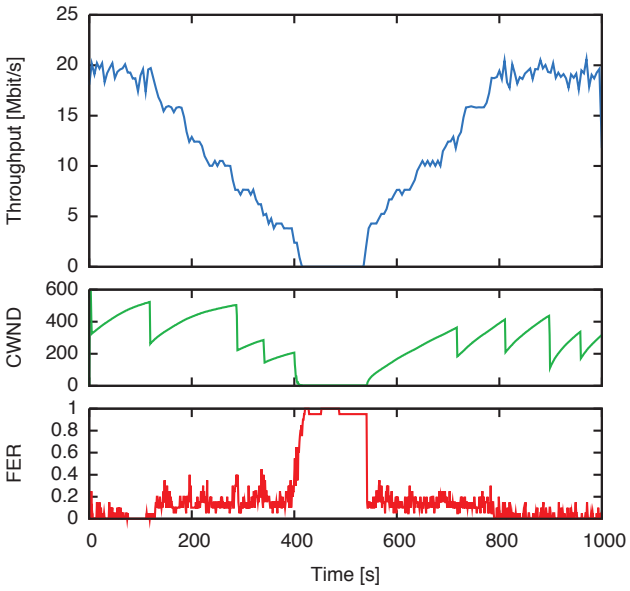


Figure 6: Monitoring of three metrics (TCP throughput, TCP congestion window, and frame error rate) at an emulated node over time. The node starts at a distance of 0m to the access point, moves away at a constant speed up to a distance of 460m, and turns back until it reaches 0m again.

4.2 Test case 3: Scalability, Automation, Rapid Testing

If the simulation has too many events to process, it is not able to follow the real time demands. In such a case, simulation overload introduces artifacts that strongly distort the results. Therefore, FANTASY stops the experimentation in such a case. We therefore want to determine how complex the simulation can get before simulation overload occurs. This of course strongly depends on the performance of the computer running the simulator and the virtual machines. Evidently, a stronger machine allows more complex simulations and more emulated nodes. For our tests, we used a Dell OptiPlex 960 with 4GB Ram and Intel Core2 Quad CPU Q9400, each processor running at 2.66GHz. Ubuntu 10.04 was installed on an external USB 2TB hard disc.

The test setup is very similar to the previous ones. The emulated WiFi node is kept at a fixed distance of 5m and sends netperf [14] traffic to the emulated Ethernet node. With each run, we add an additional simulated node to the

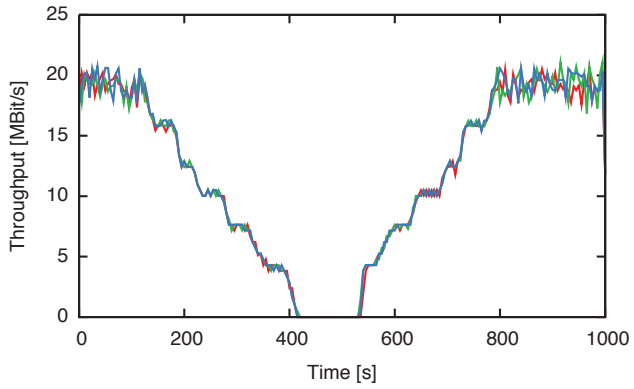


Figure 7: Monitored TCP throughput from three simulation runs with the same parameters. Repeated runs produce highly similar results.

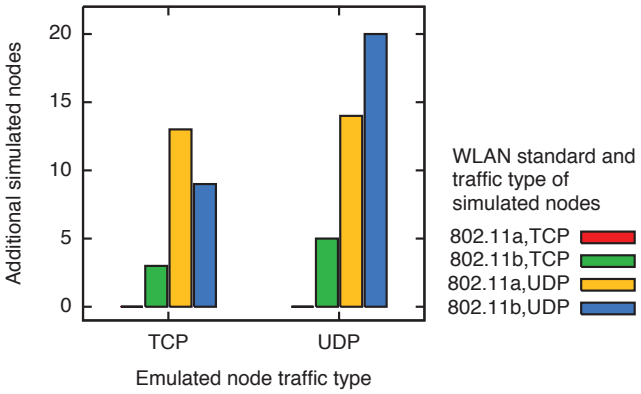


Figure 8: Number of nodes that could be simulated using FANTASY before simulation overload occurred.

simulation setup. We continued this until the simulation experienced overload and canceled the evaluation. Figure 4(c) shows the set for this test case. We placed the simulated nodes equidistantly around the access point. The cross-traffic created by the simulated nodes always adds up to 8 Mbit/s, so that, for example, with four nodes, each simulated node sends with 2 Mbit/s. This way, the channel is always kept roughly equally busy from traffic by the simulated nodes, with netperf using up the remaining capacity.

Apart from the number of simulated nodes ranging from 0 to 20, we varied three settings for this evaluation setup: (a) we used either TCP or UDP for the netperf traffic between the emulated nodes, (b) we used either TCP or UDP for the traffic between the simulated nodes, and (c) we used either 802.11a or 802.11b as the WLAN standard. We also conducted each test 10 times, resulting in 1680 experimental runs. (How these diverse tests were generated has been shown in Section 3.1.2.) The reasons for these different tests were all related to simulator performance. UDP produces less overhead than TCP due to the less complex protocol. 802.11b produces less load on the simulator than 802.11a because of the lower maximum speed of the wireless network (11 Mbit/s and 54 Mbit/s, respectively).

The results of our tests are summarized in Figure 8. In general, the results confirmed our expectations. UDP traffic between the emulated nodes meant we could add more simulated nodes before the simulation overloaded (compare same-colored bars with each other) compared to TCP traffic between them. Likewise, UDP cross-traffic allowed us to add more nodes than TCP cross-traffic (compare first pair of bars in each set with second pair). 802.11b allowed us to add more nodes than 802.11a (compare first bar with second, and third with fourth in each set). The exception from these rules are 802.11a with TCP cross-traffic, in which both experiments overloaded after adding a single cross-traffic node. The setup with just the emulated nodes running netperf completed successfully for both TCP and UDP traffic.

In this section, we have demonstrated that we are able to set up a mix of emulated and simulated nodes. The possible complexity of the network depends on the used protocols and used MAC layer models (and the used hardware to run the network emulation setup). We will discuss concepts to improve the performance and allow larger emulated networks in Section 6.

5. RELATED WORK

Since network simulation and emulation play an important role in protocol evaluation, there are different projects that offer functionality similar to FANTASY.

Arguably the most elaborate of these is *Emulab* [8], a project that aims at combining network simulation, network emulation and real networks to create a complex testbed. From the first implementation at the *University of Utah*, several additional Emulab testbeds at different locations were spawned, demonstrating the concept's success. The original Emulab consists of several hundred PCs with additional hardware connecting them. Later additions to the testbed include stationary wireless nodes, as well as *Mobile Emulab* [13] which uses mobile robots to carry sensor nodes in a room dedicated to this purpose. Clearly, this setup allows testing in wireless networks with a lot of realism and support of repeatability, but access to existing testbeds is limited and costs for the setup of a new testbed are high. With emulation, Emulab supports another way to perform tests in wireless networks. However, the emulation features for mobile nodes are more restricted and simple than the functionality that FANTASY offers through the incorporation of ns-3.

An approach similar to FANTASY is given by the *Network Experiment Programming Interface (NEPI)* [15] which tries to offer a single user interface for testing with many different tools. For example, NEPI can use *ns-3*, *PlanetLab*, *Emulab* and *ORBIT* in its tests to create diverse setups and topologies. Since NEPI tries to offer an uniform interface for this diverse set of tools, the interface of NEPI itself is already quite complex. If this combination of different tools is not required, it is therefore easier to learn the use of a single tool such as FANTASY.

Another similar concept is given with *Ad-hoc Routing/Emulation* [7] where a cross-layer implementation is tested on virtual machines. These virtual machines represent robots that have to navigate through a virtual world created by a simulation. However, the virtual machines do not interact with the simulation but only get status information about the virtual world therefrom. This approach targets a rather special case of cross-layer testing as it was designed specifically to test the robots' navigation. Using this setup for general cross-layer testing is therefore not possible.

6. FUTURE WORK

As the scalability tightly depends on the power of the test machine, the power needed to be real-time capable rapidly grows with the simulation complexity. In other words, for any amount of computing power, it is possible to create network emulation setups that will overload the simulation and therefore be too complex to be run in real-time. Therefore, we are considering to solve this problem that FANTASY suffers from like all other network emulation frameworks more fundamentally by relieving it from that real-time constraint.

One solution to do so is SliceTime [22], which continuously synchronizes virtual machines and network simulation with each other to combat the time drift to originates from simulation overload. We are currently investigating the complexity of integrating SliceTime with FANTASY. Since SliceTime is tightly integrated with Xen [3, 4] instead of VirtualBox, this will mean a change in the VM software used. Due to the modular design of FANTASY, this will be mainly a question of creating a new VM-Com module (see Section 3.1) for the HCU.

7. CONCLUSIONS

In this paper we presented FANTASY, a new network emulation architecture that allows the fully automated setup and execution of an experiment, enables the convenient access to system information and the collection of test results. With the integration of the cross-layer architecture CRAWLER, we demonstrated that we are able to monitor parameters in a mobile emulated node across protocol layers and to evaluate network emulation scenarios where cross-layer optimization is involved.

Our fully automated emulation architecture is already an essential part for testing of cross-layer optimizations within the CRAWLER project [1]. As we believe that our fully automatic network emulation architecture will be useful to other researchers and developers in the area of wireless networks, we have made the source code[1] available to the public.

8. ACKNOWLEDGMENTS

We would like to thank Raimondas Sasnauskas, Tobias Drüner, Dominik Dennisen, Martin Henze and Elias Weingärtner for many fruitful discussions and valuable feedback. This research was funded in part by the DFG Cluster of Excellence on Ultra High-Speed Mobile Information and Communication (UMIC).

9. REFERENCES

[1] I. Aktas, J. Otten, F. Schmidt, and K. Wehrle. Towards a Flexible and Versatile Cross-Layer-Coordination Architecture. In *Proceedings of the 29th International Conference on Computer Communications (INFOCOM 2010)*, pages 1–5, March 2010.

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

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

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

[4] Citrix Systems, Inc. Xen hypervisor, the powerful open source industry standard for virtualization. `http://www.xen.org/`. (accessed Nov 14, 2011).

[5] P. De, A. Raniwala, S. Sharma, and T. Chiueh. MiNT: a miniaturized network testbed for mobile wireless research. In *Proc. IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM 2005*, volume 4, pages 2731–2742 vol. 4, 2005.

[6] I. de Jong. Pyro 3.x - Python remote objects. `http://www.xs4all.nl/~irmen/pyro3/`. (accessed Nov 14, 2011).

[7] K. Dörnemann. Ad-Hoc Routing/Emulation - Philipps-Universität Marburg - Verteilte Systeme (AG Freisleben). `http://www.uni-marburg.de/fb12/verteilte_systeme/forschung/pastproj/adhoc_routing_emul`. (accessed Nov 14, 2011).

[8] Emulab.Net - Emulab - Network Emulation Testbed Home. `http://www.emulab.net/`. (accessed Nov 14, 2011).

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

[10] R. S. Gray, D. Kotz, C. Newport, N. Dubrovsky, A. Fiske, J. Liu, C. Masone, S. McGrath, and Y. Yuan. Outdoor experimental comparison of four ad hoc routing algorithms. In *Proceedings of the ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, 2004.

[11] S. Ha, I. Rhee, and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, 2008.

[12] S. Hemminger. Netem-emulating real networks in the lab. In *Proc. Linux Conference Australia*, 2005.

[13] D. Johnson, T. Stack, R. Fish, D. M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau. Mobile Emulab: A Robotic Wireless and Sensor Network Testbed. In *Proceedings of the 25th Conference on Computer Communications (IEEE INFOCOM 2006)*, April 2006.

[14] R. Jones, K. Choy, and D. Shield. Netperf. `http://www.netperf.org`. (accessed Nov 14, 2011).

[15] M. Lacage, M. Ferrari, M. Hansen, T. Turletti, and W. Dabbous. NEPI: Using Independent Simulators, Emulators, and Testbeds for Easy Experimentation. In *ACM SIGOPS Operating Systems Review*, pages 60–65, January 2010.

[16] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang. TCP westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proc. MobiCom'01*, pages 287–297, New York, NY, USA, 2001. ACM.

[17] ns-3 Wifi Models. `http://www.nsnam.org/docs/release/3.7/doxygen/group___wifi.html`. (accessed Nov 14, 2011).

[18] ns-3 Website. `http://www.nsnam.org/`. (accessed Nov 14, 2011).

[19] Python Programming Language, Official Website. `http://www.python.org/`. (accessed Nov 14, 2011).

[20] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. Iperf: The TCP/UDP bandwidth measurement tool. `http://iperf.sourceforge.net/`, 2004. (accessed Nov 14, 2011).

[21] E. Weingaertner, H. vom Lehn, and K. Wehrle. Device-driver enabled wireless network emulation. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques (SIMUTools 2011)*. ICST, 3 2011.

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

[23] Wireless Tools for Linux. `http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html\#wext`. (accessed Nov 14, 2011).