Interface connecting the INET simulation framework with the real world

Michael Tüxen
Münster University of Applied Sciences
Fachbereich Elektrotechnik und Informatik
Stegerwaldstrasse 39
D-48565 Steinfurt, Germany
tuexen@fh-muenster.de

Irene Rüngeler
Münster University of Applied Sciences
Fachbereich Elektrotechnik und Informatik
Stegerwaldstrasse 39
D-48565 Steinfurt, Germany
i.ruengeler@fh-muenster.de

Erwin P. Rathgeb
University of Duisburg-Essen
Institute for Experimental Mathematics
Ellernstrasse 29
D-45326 Essen, Germany
erwin.rathgeb@iem.uni-due.de

ABSTRACT

The INET framework for the widely used OMNeT++ simulation environment supports discrete event simulation for IP-based networks.

During the development of a simulation model for the new IETF transport protocol SCTP (Stream Control Transmission Protocol), INET was extended to also support external interfaces. These interfaces allow to set up hybrid scenarios where simulated nodes communicate with real external IP-based nodes.

This paper will first give a short introduction to the OMNeT++ simulation environment and the INET framework. Then the requirements for the external interfaces will be discussed and some implementation aspects will be described. Hybrid scenarios offer a whole range of potential applications which will also be presented briefly. The performance of this technique is crucial for its applicability. Therefore, several test setups are evaluated to verify the feasibility of this approach.

Categories and Subject Descriptors

I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis; I.6.6 [**Simulation and Modeling**]: Simulation Output Analysis; D.2.12 [**Software Engineering**]: Interoperability—*Data mapping*; D.4.8 [**Operating Systems**]: Performance—*Simulation*

General Terms

Protocol simulation, Network emulation

Keywords

SCTP, Omnet++, INET, ExtInterface

SIMUTools 2008, March 3-7, 2008 Marseille, France

1. INTRODUCTION

Our groups have been actively involved in the standardization of the new IETF Transport protocol SCTP (Stream Control Transmission Protocol [13]) from the start and have developed the portable open source SCTP implementation SCTPLIB [10] together with an industry partner. To be able to evaluate this feature-rich and complex protocol and to develop new features and extensions, a fully featured, standards conformant simulation model of SCTP was required. In INET, models for the standard protocols of the TCP/IP protocol family are already provided. Since standard conformance was one of the major goals for the SCTP model, it was highly desirable to be able to use the real traffic traces, the test suite implementations for testing real SCTP implementations and of course the readily available real SCTP implementations for debugging, testing and validating the SCTP model. As a consequence, we extended the INET framework to provide external interfaces for communication between simulated nodes and real nodes over a standard IP stack in real-time. The hybrid models combining simulation and real measurements have several possible applications as will be discussed later on. Of course, real-time performance of the simulation environment is the most critical issue for the applicability of the hybrid scenarios.

This paper will introduce the concept, realization and evaluation of the external interfaces and is structured as follows: In Section 2 the simulation environment OMNeT++ and the INET framework in general are described. After motivating the usefulness of simulations interacting with the real world, the requirements for the implementation of the external interfaces for INET are described in Section 3 together with some implementation aspects. Potential applications of the hybrid approach are discussed in Section 4. A performance evaluation of the hybrid approach in selected scenarios is provided in Section 5 in order to prove the feasibility of the concept. After a brief discussion of the limitations of the solution in Section 6 conclusions and a short outlook on future work are given in Section 7.

2. THE SIMULATION ENVIRONMENT

2.1 An Outline of OMNeT++

OMNeT++ [3] is an open source discrete event simulation environment with a modular component based architec-

ture. Types of components are channels (described by the parameters delay, bit error rate and data rate), network definitions, simple and compound modules. The components can be assembled into more complex modules via connected gates. Networks are the result of combined module types that communicate through messages. One message can be encapsulated in another one, thus being able to simulate the transmission of information via layered protocol stacks.

A powerful GUI is implemented that helps to follow the simulation process. Each packet is animated and its contents can be examined just by double-clicking on it. Furthermore debug output can be analyzed for each module individually.

The simulation can be run at different speed rates. Every movement of a message can be inspected by stepping through the simulation. The velocities *Run* and *Fast* provide a normal and less detailed animation than the *Step* mode whereas in the *Express* mode the displayed information is updated only in long intervals.

2.2 The INET Framework

Since OMNeT++ is a very versatile tool there are a great number of ready-made simulation models provided for download. One of those is the INET framework [2].

The INET framework consists of a variety of protocol implementations, among them are TCP, UDP, ICMP, IP, PPP, Ethernet, and some routing protocols. In addition a lot of protocol independent modules like *RoutingTables*, *Routers*, *Switches*, and *Hubs* are available. They are all simple modules and can be combined to form compound modules and networks.

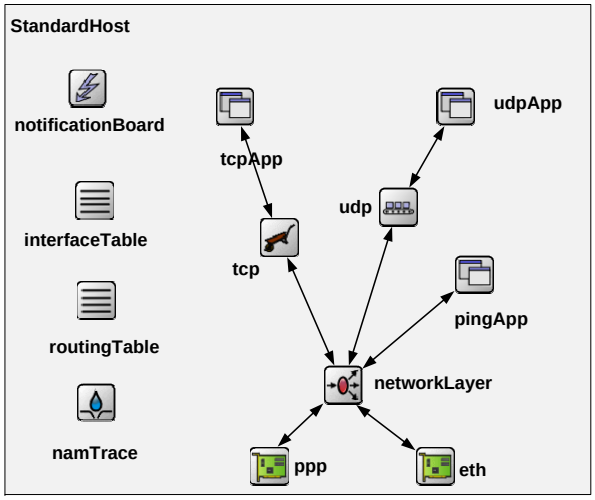

Figure 1: Compound Module StandardHost

One of those compound modules for instance is the *StandardHost* (Figure 1) which consists of a complete IP stack with PPP or Ethernet interfaces, a network layer, a Ping application, TCP or UDP as transport layer and corresponding applications. We have complemented this host with the transport protocol SCTP, a suitable application, a dump module and external interfaces (see Figure 2). For more information on the implementation of SCTP in the INET framework see [12].

Another important feature of INET is the ability to use real network addresses and do the routing according to rules derived from routing tables. Although a FlatNetworkConfigurator can be used to automatically distribute addresses among the hosts of a network, we prefer to set up routing tables where we can configure routes to other hosts or networks. Thus we can determine the way a message takes through a network. An example will be shown in Section 5.2.

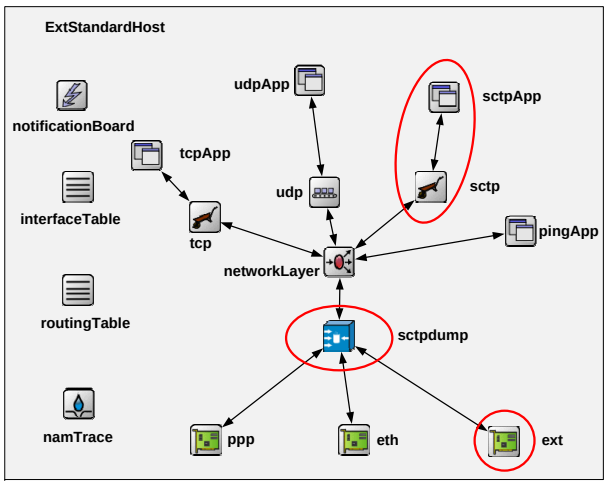

Figure 2: Compound Module ExtStandardHost

3. EXTERNAL INTERFACE

During the testing and debugging of the SCTP simulation module it seemed to be very attractive to be able to use existing tools like the Wireshark packet analyzer for analyzing the message transfers and to test the interoperability with available real implementations. Existing implementations of SCTP test suites, like the ETSI conformance test suites, could also be used to test the simulation model.

Therefore, we implemented a network interface module for the INET simulation model which allows communication between the simulated nodes and real nodes connected to the host running the simulation via an IP-based network. This capability proved to be very helpful during analysis and testing of the SCTP simulation model.

3.1 Simulation – Emulation – Real Network

Analyzing the performance of protocols and their implementations can be done using different approaches. A widely used way of doing this is based on simulations, most of the time discrete event simulations. The advantages are that one can abstract from details which are not relevant, can easily debug the simulation model and can reproduce tests because everything can be run in a deterministic way. One of the problems with this approach is that several parameters which might influence the performance have to be specified when running a simulation. Sometimes it is very hard to provide reasonable values for some of them or even model them appropriately. An example is the CPU time needed for some message handling since this can be influenced by the cache effects of the CPU. Also the impact of having multiple CPU cores is hard to model. In general, it is much easier to analyze the generic protocol performance compared to the performance of a specific protocol implementation using a simulation.

When using real implementations for performance analysis one can emulate the network between the sender and

receiver. This can be done by special nodes running for example the DUMMYNET (see [11]) network emulator of the FreeBSD operating system, which allows to emulate packet loss rates, bandwidth limitations, delays and with some additional tools also packet duplication and corruption. There are also similar tools for the Linux operating systems, for example NIST Net described in [6]. Going one step further one could replace the network emulation by a real network between the nodes. This is done, for example, in a project called PlanetLab, where experiments can use an almost global network which is based on the public Internet. See [7] for more details. This approach not only provides real endpoint behavior but also incorporates all effects of real network scenarios. However, it is hard to reproduce experiments, because of the various impacts on the network which can not be controlled. Real systems are also used by Emulab (see [1] for more details), but they are not arbitrarily distributed, and so parameters like delay and loss can be controlled.

Having the possibility that nodes within a simulation can interact with nodes in a real IP-based network combines the advantages of these different approaches.

The Network Simulator NS-2 (see [4] for more details) has a limited support for interacting with real nodes, as described in chapter 44 of the manual of NS-2. An integration of Emulab and NS-2 is described in [8], including the usage of this technique for distributed simulation.

3.2 Requirements for the external interfaces

The INET framework currently runs on several Unix based operating systems and a variety of Windows operating systems versions. The external interfaces should be supported on all of these systems. Although we are mainly interested in SCTP, it should be possible to use all IP-based protocols like UDP, TCP and SCTP, but also OSPF and other protocols on these interfaces. From the simulation's point of view the interface should look like the already supported interfaces for the Point-to-Point Protocol (PPP) or Ethernet. For being able to use the external interface for multihomed SCTP endpoints multiple external interfaces have to be supported. This is also required to support scenarios where a real sender sends messages to a real receiver through a simulated network.

3.3 Receiving and sending real packets

When an IP packet is received by the host running the simulation for a node being simulated, it must be transformed into an OMNet++ object and injected into the simulated network. The network stack of the host running the simulation should not process these packets. Therefore, the host can not have the IP addresses of the simulated node configured as addresses of one of its real interfaces. This means that using raw sockets is not an appropriate mechanism for receiving these packets. The packet capture library *libpcap*, however, provides an appropriate way of capturing these packets. This library provides an operating system independent way of capturing Ethernet packets and supports all operating systems which are relevant here. It provides capture filters which can be used to make sure that only packets which are sent to the simulation are captured and not the ones sent by the simulation.

Sending packets from the simulation to nodes in the real network can be done by using raw IP sockets. It should be noted that the host sends packets with source addresses not belonging to the host. This is not a problem for the Unix based operating systems but may not be supported by all versions of the Windows operating system. In this case the *libpcap* could also be used to send the packets.

The routing in the real network has to be configured such that the host running the simulation acts like a router which provides access to the network being simulated.

For sending packets, a method of transforming the simulation internal format to the network format has to be implemented for each protocol. This is called a serializer. For receiving packets a method called parser will transform the packet in network format into the simulation internal format. The code is structured in a way that these methods are encapsulated on a per protocol basis.

3.4 Scheduling external and internal events

A discrete event simulation which also takes interactions with the real external world into account has to handle two kinds of events: internal events which have their origin in the simulation and external events which stem from the interface to the external world. Also the simulation time has to be synchronized to the real time. This is possible assuming that there is a speedup in the simulation compared to real-time, i.e. the simulated time runs faster than the real time. As we will show in our experiments, this is a valid assumption when using state-of-the art computer hardware and networks having a limited total packet rate. Time synchronization is basically done by looking at the time of the next scheduled event. If this time is already in the past, this event is processed. Otherwise external events are processed and the simulation is put to sleep until either the next internal event has to be processed or another external event arrives. It is important to note that internal events have to be given a higher priority.

Due to limitations of the *libpcap* library, the simulation is put to sleep for a fixed small amount of time if no external event is present. This results also in a time granularity for all internal events. Choosing a small value for this granularity led to good results. It should be noted that operating systems have a systems based timer granularity for putting processes to sleep, hence choosing a smaller value for the simulation granularity than the systems granularity does not provide any benefit. For the Mac OS X operating systems, e.g. this granularity is 10 ms.

4. POSSIBLE APPLICATIONS

4.1 Verification of IETF specifications

After having implemented a simulation and tested it, the question arises how the functionality (rather than the performance) of the model can be verified in conjunction with other implementations. For real protocol implementations, the normal way to do this is attending IETF interoperability events to prove that the implementation can interact correctly with those of others. This is what we did with the external interface, when we attended the last two SCTP InterOps in Vancouver and Kyoto.

As the focus was on functionality, performance issues were not crucial. Therefore, it was sufficient to run the simulation in a virtual machine on a notebook computer. After a second external interface was configured, it was even possible to test the behavior of multihomed hosts. The advantage of at-

tending interop tests compared to just testing against a box with any kernel implementation is the interchange of ideas with the developers as sometimes the IETF specifications can be interpreted differently. Hence clarification of some issues was necessary and led to an interoperable solution.

4.2 Testing new features

Thinking of a new feature to implement, for instance handling incoming packets in a different way than before, it is always important to test new ideas. Besides simulating the new feature and testing it, its impact on real systems has to be analyzed. With the external interface the outside behavior, which sometimes differs depending on the operating system used, can be taken into account and the feature can be tested in a real environment.

4.3 Testing protocol behavior that needs a real counterpart

There are aspects of behavior that can only be properly examined and evaluated in a real network environment. The flow control, i.e. the reduction of the advertised receiver window, is hard to model in a simulation as it can depend on the host's resources and on the way in which incoming data is stored and counted. One question could be whether the window is reduced by the payload that arrived or by the amount of memory that is really needed to process the packet. It is, therefore, necessary to test congestion control aspects in a real network or at least against hosts with other implementations before making general assumptions.

4.4 Behavior of middleboxes and routers

What can be done to test a new feature where middleboxes or routers are involved without influencing their code? An easy way is to simulate the middleboxes and connect them via external interfaces to real computers. Thus the feature only has to be implemented on the hosts and the code of the real routers does not have to be altered.

An example of such a situation is the Quick-Start [9] algorithm for TCP and IP that we wanted to adopt for SCTP. Here every router that is on the path has to examine the packet, calculate its free capacities and change several parameters in the IP header. A good way to test the algorithm on real hosts is to implement it in the kernel, but to simulate the behavior of the router. Thus the packets that originate from a real sender are manipulated by the simulated router and send on to the real receiver.

4.5 Generate traces

As mentioned in Section 2.2 we added a dump module similar to the one used by TCP to the StandardHost module (see Figure 2). It was placed between the link layer and the network layer to be able to distinguish between the interfaces the message passed through.

Thus we got an overview of the packets sent to and from the hosts. One drawback of the dump module was that its output was bound to the output of OMNeT++. Therefore we wanted to use the ExtInterface to generate traces that could be read by protocol analyzers, independent from the simulation. Hence our task was to provide a file in the pcap format, where each packet in network format is preceeded by a pcap header and an Ethernet header. As the transformation from the simulation internal format to the network format is done by the serializer, the file entries are formed by writing the headers and the output of the serializer for each packet.

After the simulation the trace can be analyzed by Wireshark ([5]) or any other packet sniffer understanding pcap files. This feature is of great benefit when it comes to analyzing the simulation data.

5. EXAMPLE SETUPS

5.1 Performance evaluation

One aspect that always matters is the performance of an implementation. Of course a simulation cannot be as efficient as a real implementation because a lot of additional information is stored, other data structures are used, and performance is normally not the most important issue when designing a simulation model.

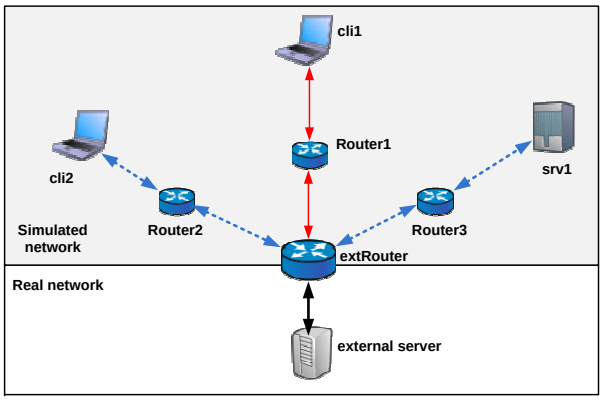

Figure 3: Cli1 sends data via extRouter to a real PC while internal traffic from cli2 to srv1 is passing through extRouter

Nevertheless we wanted to find out how good our simulation was and tested it against a real implementation. We chose the setup in Figure 3, where the simulated parts are marked by a gray background. The external interface can be considered as part of both worlds. Here the client *cli1* is linked via a router with the external interface, which is connected to a real server. The channel between the client and the router is limited to a data rate of 10 Mbit/sec. First we sent 200,000 data chunks of increasing sizes between 10 and 1400 Bytes and measured the throughput. In Figure 4 the red graph with the diamond-shaped symbol shows the result. As a comparison the maximal theoretical throughput is represented by the blue graph with the square-shaped symbol. The figure shows that the simulation is able to fully utilize a link of 10 Mbit/sec.

A second series of measurements was performed to find out whether additional traffic passing through the router would have an impact on the throughput, meaning that the processing of the events from the external router could not keep up with the packets arriving. Therefore *cli2* was to start earlier than *cli1* and had to run longer than the external association. The throughput is shown by the green graph with the triangle-shaped symbol. It is obvious that the internal traffic has no significant influence on the external traffic.

Another goal was to find out how long it takes for a router to process a packet. Therefore the throughput was measured

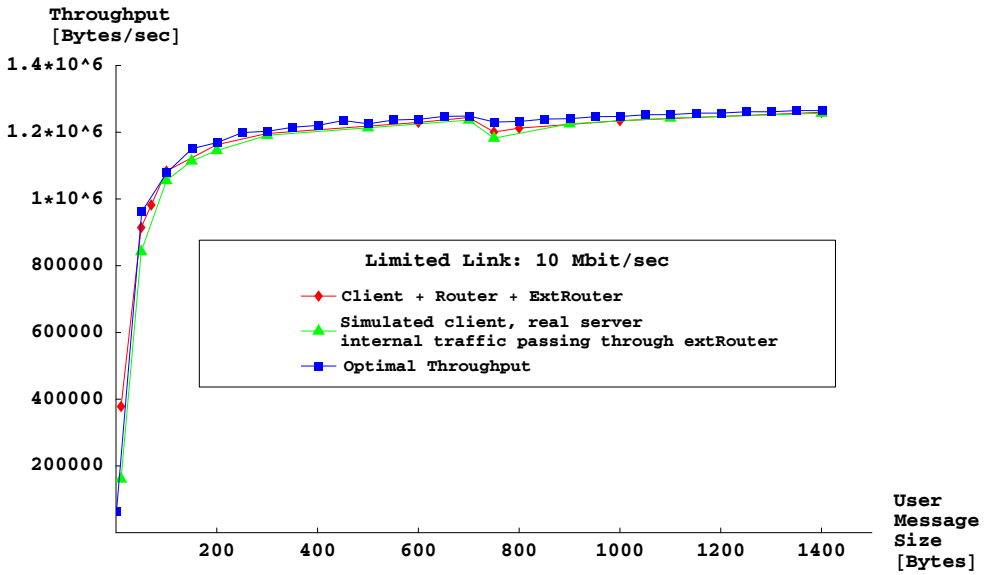

Figure 4: Throughput of the SCTP-association between Cli1 and a real PC

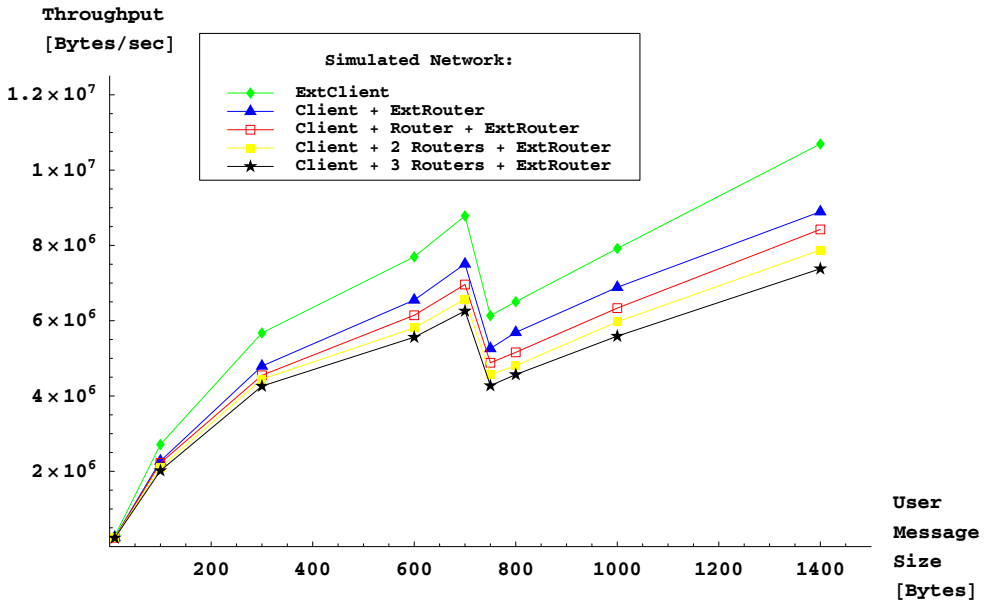

Figure 5: Throughput of the SCTP-association between Cli1 and a real PC with varying number of routers

without internal traffic but with 0, 1, 2, and 3 routers between the client and the external router. The data rate was not limited. From Figure 5 can be concluded that the time a router needs to handle the packets is nearly constant. Subtracting the measured times for the particular user message sizes from one router to another and dividing the difference by the number of packets that passed through the routers, an average process time of about 11 μs per packet can be assumed.

5.2 Traceroute

In Figure 6 the network scenario of an example setup with the corresponding IP addresses is shown. The route the requests have to take are marked with bold arrows. We verified that the output of the *traceroute* command reflects various settings of network parameters in the simulated network, for example configured delays.

5.3 Ping flood

In this example, shown in Figure 7, the simulated parts only consist of routers. That means that two external interfaces are needed that have to be simultaneously serviced by the scheduler. The command we chose to test the performance of the scheduler was *ping* with the f-option. This option causes the source to send packets as fast as they come back or one hundred times per second, whichever is more. We verified that no packets were lost. Some packets arrived after *ping* had been stopped. They were still in queues and

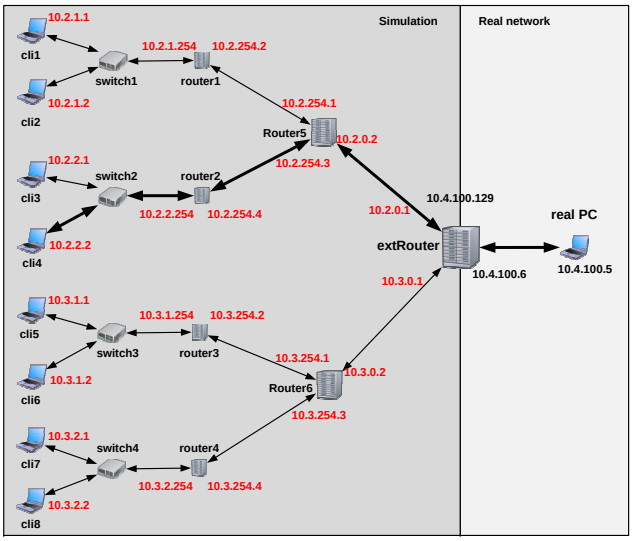

Figure 6: Real PC follows the way to a client in the simulation using *traceroute -n 10.2.2.2*

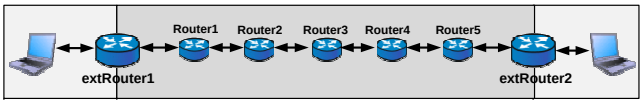

Figure 7: Ping flood is passing from a real PC to a real PC through the simulation

left the simulation after having been processed.

6. LIMITATIONS AND CONSTRAINTS

The usage of *libpcap* for handling the packets received by the simulation and raw sockets for sending packets from the simulation results in some limitations and contraints. First of all, it is not possible for a regular user on a Unix operating system to use this feature because he needs root privileges. However, this is not a serious drawback but might be important when using this technique for education purposes.

The simulation time is handled in a discrete way with a specific granularity. However, experiments we did showed no substantially different results when busy waiting is configured instead. The only difference was the CPU usage. So we recommend to use a simulation timer granularity equal to the operating system timer granularity.

When packets arrive at a very high rate the *libpcap* library might not be capable to handle all these packets. This would result in packet loss. We did not observe this during our experiments. However, it should be taken into account when doing experiments. The simulation tool reports at the end how many packets were dropped. The major limitation is the CPU of the host running the simulation. It is easy to observe the CPU utilization during experiments to make sure that the CPU is not the factor which dominates the results.

7. CONCLUSION AND OUTLOOK

This paper discussed the various benefits of the ability for simulated nodes to interact with real nodes in a network. An implementation of such an interface for the discrete event simulation OMNet++ and the INET framework was described and the performance of this implementation has been analyzed to show that it is possible to do reasonable experiments with such a setup. A complete network can be simulated or only one node. Several applications where the external interface can successfully be used have been described.

The external interface will be integrated into a future version of the INET framework. More protocol parsers and serializers will be implemented. In particular protocols for telephone signaling will be supported. This helps us testing the simulation model of SIGTRAN protocols which are currently being developed.

8. ACKNOWLEDGEMENTS

We would like to thank Andras Varga for supporting us during the development of the external interface, Christian Dankbar for developing a prototype of it, and the anonymous reviewers for their comments.

9. REFERENCES

[1] emulab – total network testbed. *See at: http://www.emulab.net/.*

[2] INET Framework Documentation. *Retrieved from: http://www.omnetpp.org/staticpages/index.php?page=20041019113420757.*

[3] OMNET++ User Manual Version 3.2. *Retrieved from: http://www.omnetpp.org/doc/manual/usman.html.*

[4] The Network Simulator NS-2. *Available at: http://www.isi.edu/nsnam/ns/.*

[5] Wireshark protocol analyzer. *Available at: http://www.wireshark.org.*

[6] M. Carson and D. Santay. NIST Net: a Linux-based network emulation tool. *ACM SIGCOMM Computer Communication Review*, 33(3):111–126, 2003.

[7] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[8] S. Guruprasad, R. Ricci, and J. Lepreau. Integrated network experimentation using simulation and emulation. *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on*, pages 204–212, 2005.

[9] A. Jain and S. Floyd. Quick-Start for TCP and IP. *Work in Progress (Internet-Draft draft-ietf-tsvwg-quickstart-06), August*, 2006.

[10] A. Jungmaier, M. Tüxen, T. Dreibholz, et al. SCTPLIB–an SCTP implementation, 2005.

[11] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM SIGCOMM Computer Communication Review*, 27(1):31–41, 1997.

[12] I. Rüngeler, M. Tüxen, and E. Rathgeb. Integration of SCTP in the OMNeT++ Simulation Environment. *International developer's Workshop on OMNeT++ (OMNeT++ 2008)*, March 2008.

[13] R. Stewart. Stream Control Transmission Protocol. *RFC 4960*, September 2007.