A Dynamic Network Scenario Emulation Tool

Daniel Herrscher, Kurt Rothermel

University of Stuttgart
Institute of Parallel and Distributed High-Performance Systems (IPVR)
Breitwiesenstr. 20-22
70565 Stuttgart, Germany
Phone: +49-711-7816-233 / Fax: -424
E-mail: herrscher@informatik.uni-stuttgart.de

Abstract -- **Comparative performance measurements of distributed applications and network protocols require the availability of appropriate network environments. Network emulation approaches offer a flexible way to mimic the properties of a variety of networks. Existing emulation tools work either with centralized real-time simulation components, limiting the scenario size and maximum traffic, or focus on the emulation of some network properties at a single point. We propose a tool for the realistic emulation of network links, and show how several emulated links can be combined to reproduce a comprehensive network model. In addition to that, the model can include changing network properties, e.g. emerging from mobile communication partners. This facilitates the distributed emulation of a comprehensive, dynamic network scenario to support repeatable performance measurements.**

I. Introduction

During the development of new distributed applications and protocols, it is essential to analyze the impact of various network environments on their performance. While mathematical analysis and simulations are commonly used in early stages of development, measurements have to endorse the theoretical results as soon as implementations become available. Since the target network scenarios are most likely not available at this stage, there is need for synthetic, configurable environments to conduct comparative performance measurements within them. These environments have to emulate a particular network as closely as possible. To facilitate the emulation of mobile networks, the environment also has to support dynamic changes of the respective network parameters.

A crucial part of any emulation facility is an emulation tool that affects network traffic in a specific way. In this paper, we first state the requirements an emulation tool has to meet. Then, we show how existing approaches address these issues: Centralized emulation tools can work with dynamic scenarios, but constitute a bottleneck to the emulated system and thus limit the scenario in size and bandwidth. Several emulation tools running on a number of hosts to set up a scenario support higher bandwidths, but lack of a central, dynamic model. As a trade-off between centralized and distributed emulation, we propose an emulation tool that can run on several nodes forming a comprehensive scenario, and is controlled in real-time by a central dynamic network model. We indicate how our tool works in the communication stack of the operating system, introducing the specified network parameters in a realistic way. Finally, we show how updates in the central network model can be propagated to the respective emulator instances efficiently.

II. Goals and Requirements

To be able to describe the requirements an emulation tool has to meet, we first introduce the structure of emulation systems in general and define some basic terms.

The purpose of any network emulation approach is to mimic the behavior of a specific network scenario in order to analyze the impact of this scenario on software communicating over the network (see Fig. 1). Therefore, every emulation approach needs a *network model* where the desired network and its properties are defined. The *network emulation facility* is software, hardware, or a combination of both that provides network services to the *software under test (SUT)*. The SUT can be applications or proto-

cols on various layers. The emulation facility has to provide interfaces *(entry points, EP)* between emulated network and SUT, representing different locations within the network. The emulation facility can offer services on different layers, we call this the *emulation abstraction layer*. The usability of an emulation tool for a specific purpose is highly dependent on the emulation abstraction layer it offers to the SUT. We identified three emulation abstraction layers that are used by emulation approaches:

- *Transport Layer Emulation* aims at the reproduction of certain process communication channel characteristics, e.g. the performance of a TCP channel. Emulation on this abstraction layer can be used to measure the performance impacts of channel characteristics on applications.
- *Network Layer Emulation* mimics the end-to-end behavior of a network connecting hosts, e.g. packet delays, congestion losses, etc. In addition to the applications of transport layer emulation, it can also be used for the evaluation of transport protocols.
- *Link Layer Emulation* concentrates on the emulation of single network links, e.g. limited bandwidth, frame delay, etc. In addition to the applications of transport and network layer emulation, emulation on link layer facilitates the evaluation of network layer protocols also.

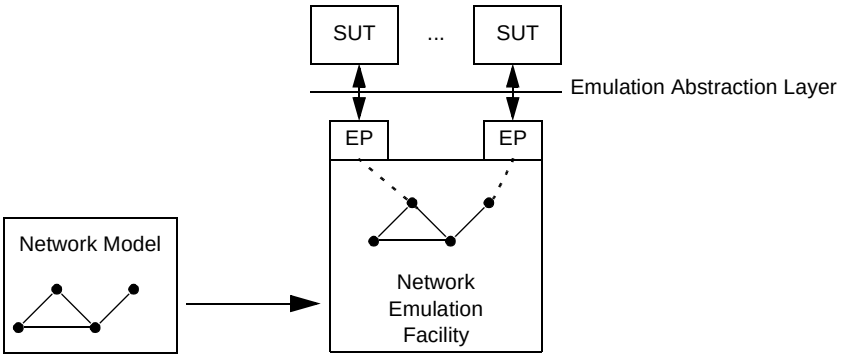

Figure 1: Basic Structure of a Network Emulation System.

The primary goal of every network emulation approach is to achieve the maximum possible conformance to the behavior of a real network. Every aspect of the emulated part of a network has to be considered to achieve this, including physical effects, hardware design, and protocol issues. It is evident that realistic emulation on a higher abstraction layer is more difficult to achieve than on a lower layer, because more factors have to be considered. E.g., network layer emulation has to mimic the effects of network layer issues (dynamic routing, queueing etc.) in order to be realistic. In contrast, approaches that focus on the realistic emulation of links can use the actual implementations of network layer protocols to create these effects. In general, lower emulation abstraction layers will lead to more realistic results. We experienced that emulation on link layer is the lowest possible emulation abstraction that is feasible without using special hardware.

With respect to this goal, we are able to state the requirements a network emulation system should meet:

- *Link Layer Abstraction:* To provide the highest possible realism, and to be able to include network layer protocols as SUT, it is necessary that a network emulation facility provides services on link layer. Therefore, the remaining of this paper focuses on network emulation on link layer.
- *Emulation Parameters:* Network emulation on link layer has to mimic the behavior of an actual network link as closely as possible. Every parameter affecting performance has to be considered. These parameters are [8]: Bandwidth limitation (including frame serialization delay emerging from a limited bandwidth), propagation delay (usually fixed), medium access delay (may vary from frame to frame), and frame loss (corruption loss).
- *Transparency:* The connection of the SUT and the emulated network through the respective entry points has to be totally transparent to the SUT. This facilitates the SUT to be evaluated in its original, unmodified form.
- *Dynamic Model:* Especially in mobile wireless networks, network parameters are subject of per-

manent change. Because they depend on a variety of physical effects, parameters like the connectivity, loss characteristics, throughput etc. are far from constant. But also in wired networks, link parameters can change (physical link failure, dialup connections etc.). In order to be capable of emulating these effects, the network model has to include dynamic changes. As a result, the emulation facility has to provide the possibility of changing its settings while in service.

- *Minimize Side Effects:* Feeding network traffic into an emulation facility creates some inevitable overhead (an additional delay). This overhead is a network property that is not part of the specified scenario and therefore has to be minimized. The parameter updates necessary for the emulation of dynamic networks could also introduce unwanted effects, especially if they happen frequently. Thus, the parameter update mechanism has to be designed to work efficiently, without disturbing the emulation process.

III. Related Work

Since the simulation of complete network scenarios has been addressed in great detail already [3], it stands to reason that there are several efforts to reuse existing simulators for emulation purposes. Simulators can work with complex network models, and their simulator core can compute the effects specified network properties have on network traffic traversing the model. This already makes up the main part of an emulation facility. Usually, network simulators work with models on link abstraction layer. There are two basic problems to be solved to facilitate the linkage of a simulator core with the SUT. First, common network simulators work with discrete event schedulers. Events are processed as soon as the effects of all prior events are evaluated. To interact properly with the SUT, the scheduler must be modified to be synchronized with real-time. Second, an interface for packet capture and generation has to be provided (the entry points). These issues have already been addressed for the most common network simulators [7, 10]. The real-time requirement appears to become a problem when moving to higher bandwidths. To some extent, this can be addressed by parallelizing the discrete event simulator [16]. Other similar approaches explicitly state that they aim at the emulation of low bandwidth links only [5, 11].

While simulator-based approaches aim at the modeling of a whole network scenario within one simulation, providing several entry points for SUT, there are also tools that aim especially at the emulation of a single link. These tools affect network traffic directly within the protocol stack of a node processing traffic. In most cases, this will be the node where the traffic emerges from, but it is also possible to affect the traffic on the receiving node or even on an intermediate node acting as a router or transparent bridge. Various tools have been proposed for this purpose, but only few support link layer emulation.

The original Dummynet tool [15] catches calls from TCP to IP in the FreeBSD stack and can introduce a fixed delay and packet loss. It also offers a FIFO queue with configurable output rate for bandwidth throttling. The current implementation, which is included in FreeBSD distributions, is implemented as part of the firewalling code within the network layer. Therefore, it is no longer bound to a specific transport protocol. Rules specify whether a packet should be affected or not. These rules can be based on device addresses. Therefore, the current version of Dummynet can be viewed as a link layer emulation tool.

A recent approach combines gathering and replay of network parameters to emulate the changing performance in a mobile environment [14]. In the gathering phase, network properties between two hosts are measured and reduced to delay and packet loss. The replay phase uses an emulation tool working between network layer and device driver to introduce the gathered parameters again. While the actual gathering experiment described in the paper measured end-to-end characteristics, the emulation tool used to replay the data is capable of link layer emulation.

Most modern operating systems offer the possibility of limiting the bandwidth of network devices below the network layer (e.g. [2]). Usually, a drop-tail queue with the desired output rate is added above the device driver. While these approaches do introduce delay and packet loss, they appear as a side-effect of the drop-tail queue, and therefore cannot be used for the emulation of propagation delay and corruption loss. Therefore, bandwidth throttling alone cannot be viewed as complete link layer emulation.

There are other emulation tools working within the communication stack, but work on higher layers. Delayline [9] provides transport layer emulation through a custom socket implementation. To make use

of the tool, an application has to be recompiled to call the Delayline sockets. Therefore, the tool is not transparent to applications. Delayline can emulate both propagation delay and packet loss of an unreliable communication channel before passing the packets to the actual sockets provided by the operating system. NISTNet [4] is a tool working within the Linux kernel. Like Dummynet, it can introduce bandwidth limitation, delay, and packet loss, but adds delay variation, packet reordering, and packet duplication. NISTNet aims at network layer emulation and therefore can only be configured on basis of network layer addresses.

Link layer emulation tools working within the protocol stack are designed to run on one or two machines, emulating a link between two network devices. By using up a number of connected nodes running an emulation tool, a comprehensive network scenario can be set up. The Utah Network Testbed [12] uses a single network description based on an OTcl-script (similar to an ns-2 simulation setup script) to define a network model. Several FreeBSD nodes running Dummynet are used to emulate network links. The combination of a central network model and the use of link layer emulation tools makes up a complete network emulation facility working on link layer abstraction. However, the network model does not include the dynamic change of parameters.

We stated the requirements for network emulation facilities above. There is no existing emulation approach meeting all of our requirements: Some emulator tools that emerged from simulators can work with dynamic network models and support proper emulation on link layer abstraction [10,11]. However, because all traffic within the network scenario has to be sent to a central emulation process, the overall traffic is limited to the amount the emulation process can handle. Therefore, emulation with a centralized emulator instance works for scenarios with few nodes and low bandwidths only.

Existing link layer emulation tools do not consider all parameters that affect performance. E.g., Dummynet [15] does not support a variable delay parameter.

The common bandwidth limitation approach in network link emulators is to introduce a drop-tail queue with a specified output rate. An application using a transport protocol with flow control will experience the throughput to match the bandwidth specified at the emulator. However, some algorithms on network layer will not work any more like expected: If a real link is saturated, queueing algorithms like fair queueing [13] will try to give every sender its fair share of the resource. If a link is throttled by a common emulator, however, the additional drop-tail queue would drain the other queues immediately and prevent the effects of fair queueing (see Fig. 2). This means that the emulation process is not transparent to higher layers.

While there are approaches to combine several link layer emulation tools to form a comprehensive network scenario [12], dynamic scenario changes have not yet been considered.

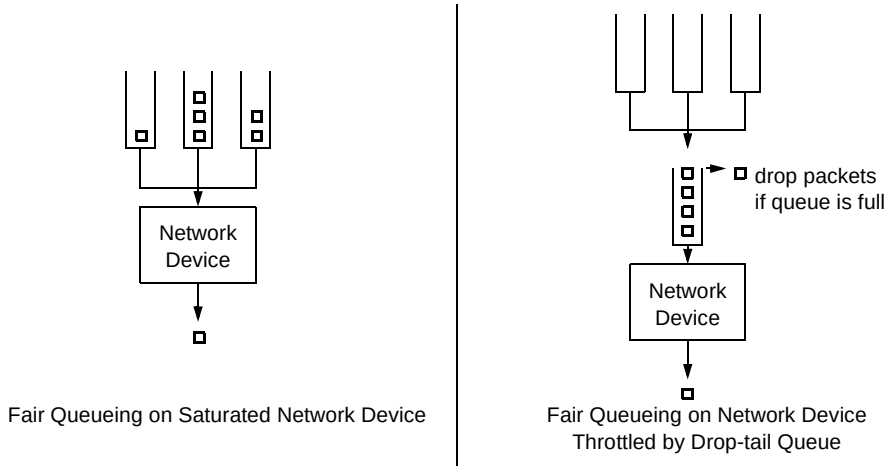

Figure 2: Bandwidth Throttling with Drop-tail Queues Prevents Fair Queueing.

IV. THE NETSHAPER TOOL

In this section, we will present "NETShaper,"[1] a network emulation tool that overcomes the limitations of the approaches mentioned above. NETShaper provides emulation on link layer that is completely transparent from applications and even protocols down to the network layer. Therefore, NETShaper can be used to evaluate e.g. routing protocols in their original form (i.e. without code modifications). NETShaper uses a bandwidth throttling approach that exactly mimics the actual behavior of saturated network devices, and therefore works with queueing algorithms provided by the network layer. Further emulation parameters include fixed delay, variable delay, and frame loss. Due to its modular architecture, it can be inserted and removed without rebooting systems.

First, we will show how our emulation tool fits into to communication stack and why this placement is a good choice for the transparent, realistic emulation of network links. Then, we illustrate how several instances of our tool can be run in appropriate testbed hardware to form a comprehensive network scenario. Finally, we describe how a central, dynamic network model can be used to trigger parameter changes in the emulated scenario automatically.

A. Architecture of NETShaper

NETShaper is implemented as a Linux kernel module. It uses the same programming interface as network device drivers, and is therefore treated like a real network device by the operating system. Several instances of NETShaper can be set up on the same machine. Each instance can be bound to an existing network device. The network address of the real interface can be assigned to the emulator device. This enables the emulator to intercept outgoing network traffic when it is about to be sent to the network device driver (see Fig. 3). NETShaper is managed by a simple configuration utility running in user space that uses IOCTL calls to communicate with the kernel module. Packet interception at this position facilitates transparent, efficient emulation on link layer for the following reasons:

- Since NETShaper works within the kernel, its presence is completely hidden from user space programs. Any user space software using the standard UNIX communication interfaces (sockets, raw sockets) will be affected by the specified network properties.
- The overhead introduced by the additional layer a packet has to pass is marginal. Since we work in kernel space, packet data has not to be touched or copied to different memory areas (like with user space approaches). Communication through interfaces with no NETShaper attached is not affected at all.
- Since NETShaper works below the network layer, new transport and network protocols can be analyzed using emulated links without changes. This enables us to conduct measurements for some of the most promising areas of network research (e.g. multicast, geocast, routing in ad-hoc networks).

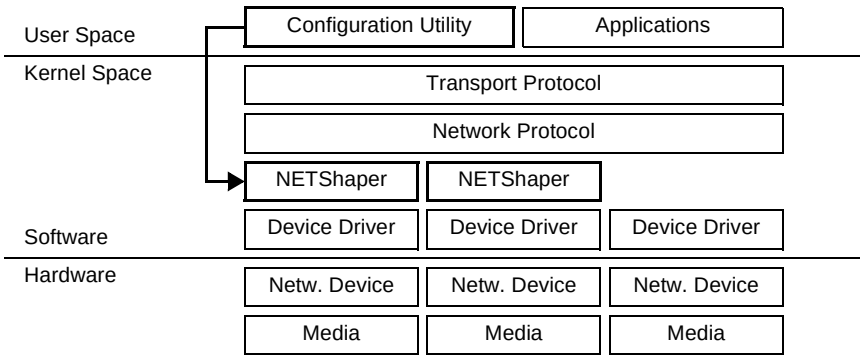

Figure 3: Placement of NETShaper within the Protocol Stack.

1. Our emulator emerged from a tool to "shape" (smooth out) network traffic.

- Because NETShaper uses the same interface to network layer protocols as an actual network device, we can use the same bandwidth throttling mechanisms as network device drivers. If a network device cannot process any more packets, it sets a “busy” flag to prevent network layer protocols from sending packets to it. Since NETShaper also uses this flag to throttle bandwidth to the specified rate, the network layer protocol above NETShaper will perceive no difference to a real network device. This means in particular that queueing algorithms on network layer will work with NETShaper, which makes our bandwidth throttling more realistic than the common drop-tail queue based approaches (see above).

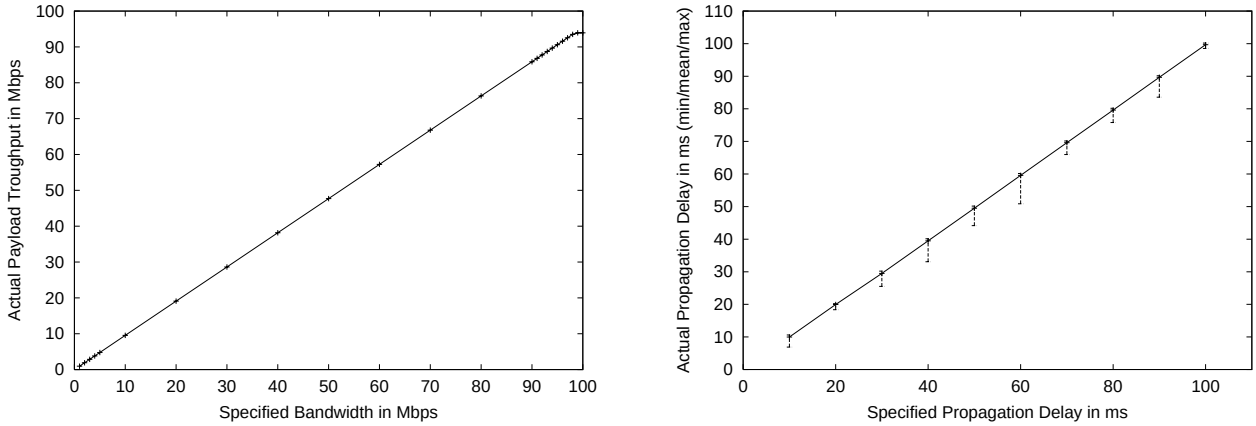

Figure 4: Measurements of Bandwidth Limitation and Delay.

To proof our emulation tool works in an efficient way, we conducted some measurements[1] (see Fig. 4). The overhead introduced by the additional layer in the protocol stack turned out to be marginal: Traversing the NETShaper module delays a packet by at most 3 microseconds, if no delay is specified. As the left chart shows, bandwidth limitation scales up to the physical limit (100 Mbps in this case). The difference between specified bandwidth and payload throughput emerges from the actual protocol overhead introduced by network and transport layer, and is therefore realistic. The right chart shows that the emulated propagation delay can vary from the specified value to some extent. Although the mean delay of 1000 packets sent through NETShaper (depicted by the continuous line) matched the specified delay very well, single packets were sent out up to 9 ms too early. This behavior results from the limited timer accuracy of the operating system we used (Linux for x86 architecture, timer granularity 10 ms). Measurements with other emulation tools for UNIX systems show the same problem [14, 15]. If a finer delay emulation is needed, NETShaper can be used on systems with reduced timer granularity. Timer granularity can easily be modified by recompiling the operating system kernel with different settings. Finer timers will come at the cost of an increased scheduling overhead [1]. Of course, these settings affect all timers within the system. As a result, the behaviour of other code may also be affected (e.g., TCP retransmit timers will be more accurate).

B. An Application: Emulating a Network Scenario

In this paragraph, we show a possible way how several NETShaper instances in combination with suitable testbed hardware can make up a comprehensive network emulation facility.

The Network Emulation Testbed (NET), which is currently being set up at the University of Stuttgart, mainly consists of a 40+ node PC cluster connected by both a high-performance switch and a separate control network (Fig. 5). The connection between nodes can be set up by virtual LANs (VLANs) defined in the switch. A VLAN with two participating nodes would be equivalent to a separate connection between these two nodes. If a node is part of several VLANs, the operating system can address each VLAN by a separate virtual network device, transparent to protocols and applications. Through this concept, VLANs can be used to set up any virtual topology. Each of the virtual network devices has a

1. measurements were conducted on 800 MHz PIII PCs with 256 MB RAM and Fast Ethernet NICs

separate NETShaper instance attached, affecting the respective outgoing traffic on the link. The combination of connection topology emulation with VLANs and link parameter emulation with NETShaper creates a comprehensive network scenario emulation, with the possibility of attaching SUT on every node within the network topology.

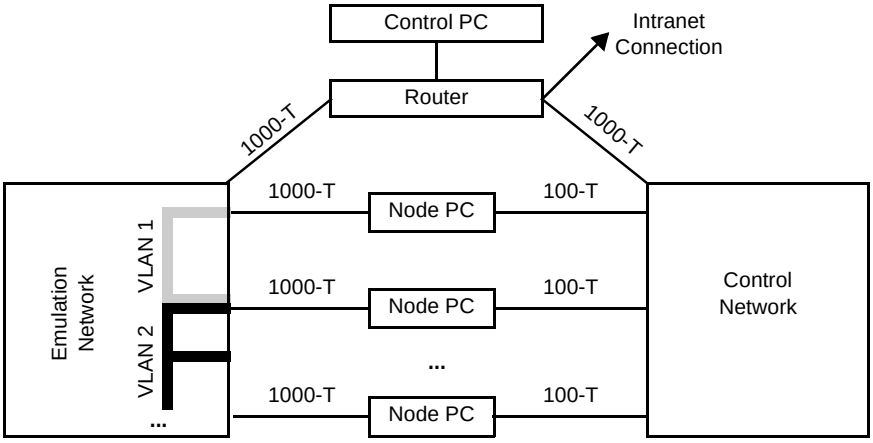

Figure 5: The Network Emulation Testbed: Facilitating the Emulation of Comprehensive Scenarios.

C. Dynamic Parameter Changes

The emulation of a dynamic network model requires an additional functionality: According to the changes in the network model, the settings of the respective emulation tools have to be updated in real-time. Therefore, we have extended the configuration utility running on each node with NETShaper instances to act as a configuration daemon (see Fig. 6). The daemon waits for parameter update messages in UDP packets and propagates them down to the NETShaper instances. The update messages come from a central update controller, which distributes changes in a dynamic network model to the respective configuration daemons. There are many possibilities how to trigger changes in the network model. We specify dynamic network scenarios in an XML-based language, including tables and state diagrams to describe parameter changes [8]. With this approach, a variety of interesting scenarios can be set up with low effort (e.g. failing or fading links, burst losses, etc.).

We conducted tests with up to 1000 update messages per second, revealing that the accuracy of emulation is not affected in any way by the updating process. The processor usage of a node processing one update message every ms increased by only 2-3%.

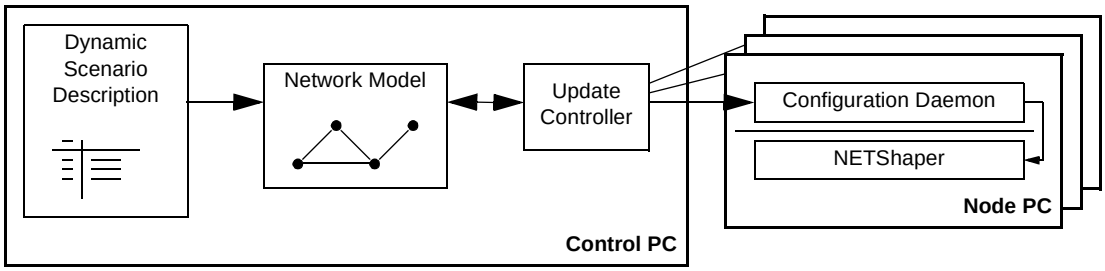

Figure 6: Emulation of a Dynamic Network Scenario.

V. Conclusion and Future Work

Comparative performance measurements of distributed applications and protocols in a changing environment require a network emulation facility. We proposed a simple, efficient tool for the emulation of network links that facilitates the evaluation of applications and protocols down to the network layer. In combination with suitable testbed hardware, several instances of our tool can make up a comprehensive network scenario. We also introduced a concept to include dynamic changes in the network sce-

nario. We believe that the realistic emulation of dynamic network scenarios will fill the gap between simulation tools and live measurements.

The current implementation of our emulation tool is suitable for the emulation of point-to-point links based on a dynamic network model. Future work includes the emulation of shared media communication and the integration of mobility models to influence the network parameters automatically.

REFERENCES

[1] J. S. Ahn, P. B. Danzig, Z. Lui, and L. Yan, "Evaluation of TCP Vegas: Emulation and Experiment," in *ACM Computer Communication Review*, vol. 25, no. 4, pp. 185-195, 1995.

[2] W. Almesberger, "Linux Traffic Control – Implementation Overview," *Technical Report SSC/1998/037*, Ecole polytechnique fédérale de Lausanne (EPFL), Nov. 1998.

[3] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, "Advances in Network Simulation," *IEEE Computer*, vol. 33, no. 5, pp. 59-67, 2000.

[4] M. Carson, "NISTNet Network Emulator," available from *http://www.antd.nist.gov/itg/nistnet/*

[5] N. Davies, G.S. Blair, K. Cheverst, and A. Friday, "A Network Emulator to Support the Development of Adaptive Applications," in *Proceedings of the 2nd USENIX Symposium on Mobile and Location Independent Computing*, pp. 47-55, Ann Arbor, 1995.

[6] D. Eckhardt and P. Steenkiste, "Measurement and analysis of the error characteristics of an in-building wireless network," in *Proceedings of the ACM SIGCOMM '96*, pp. 243-254, Stanford, 1996.

[7] K. Fall, "Network emulation in the VINT/ns simulator," in *Proceedings of the Sixth International Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems*, pp. 244-250, Red Sea, Egypt, 1999.

[8] D. Herrscher, A. Leonhardi, and K. Rothermel, "Modeling Computer Networks for Emulation," in *Proceedings of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*, Las Vegas, June 2002 (to appear).

[9] D. Ingham and G. Parrington, "Delayline: A Wide-Area Network Emulation Tool," in *Computing Systems*, Vol. 7, No. 3, pp. 313-332, 1994.

[10] Q. Ke, D. Maltz, and D. Johnson, "Emulation of Multi-Hop Wireless Ad Hoc Networks," in *Proceedings of the 7th International Workshop on Mobile Multimedia Communications*, Tokyo, Japan, October 2000.

[11] M. Kojo, A. Gurtov, J. Manner, P. Sarolahti, T. Alanko, K. Raatikainen, "Seawind: A Wireless Network Emulator," in *Proceedings of 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems*, Aachen, Germany, September 2001.

[12] J. Lepreau, C. Alfeld, D. Andersen, K. van Maren, "A Large-Scale Network Testbed," *SIGCOMM '99 New Research Session*, Cambridge, 1999.

[13] J. Nagle, "On Packet Switches With Infinite Storage," *IEEE Transactions on Communications*, vol. 35, no. 4, pp. 435-438, April 1987.

[14] B. Noble, M. Satyanarayanan, G. Nguyen, and R. Katz, "Trace-Based Mobile Network Emulation," in *Proceedings of the ACM SIGCOMM '97*, pp. 51-61, Cannes, France, 1997.

[15] L. Rizzo, "Dummynet: A simple approach to the evaluation of network protocols," in *ACM Computer Communication Review*, vol. 27, no. 1, pp. 31-41, 1997.

[16] R. Simmonds and B. Unger, "Towards Scalable Network Emulation," in *Proceedings of SPIE Vol. 4526 (2001): Scalability and Traffic Control in IP Networks*, Denver, August 2001.

[17] X. Zeng, R. Bagrodia, and M. Gerla, "GloMoSim: a Library for Parallel Simulation of Large-scale Wireless Networks," in *Proceedings of the 12th Workshop on Parallel and Distributed Simulations*, pp. 154-161, Banff, Canada, May 1998.

[18] Y. Zhang and B. Bhargava, "A Facility For Experimenting Distributed Software in the Internet," in *Proceedings of the IEEE Workshop in Advances in Parallel and Distributed Systems*, pp. 40-45, Princeton, 1993.