RplTrc: A Tool for Emulating Real Network Dynamics for Performance Evaluation

Rainer Baumann, Ulrich Fiedler

Computer Engineering and Networks Laboratory

ETH Zurich, Switzerland

Abstract— **Evaluating the performance of distributed applications, such as mobile telephone or video conferencing devices, has become increasingly complex as the diversity of scenarios and data rates in networks increase. In this paper, we present the design and implementation of RplTrc, a network emulation tool. RplTrc uses traces to drive interceptions of the Linux protocol stack to delay, drop or duplicate packets. The length of traces is not limited. Thus, unlike other well-known emulators such as NIST Net or the emulator module in the Linux 2.6 kernel, RplTrc is capable to account for important performance characteristics inherent to real networks such as long-range dependence and self-similarity of cross-traffic. Tests show that RplTrc installed on a commodity PC is capable to emulate a 100MBit/s or a lightly-loaded 1GBit/s network and has a precision in reproducing packet delays in the order of 100 μs. RplTrc thus enables extensive performance evaluation of distributed applications in lab environments.**

I. Introduction

Evaluating the performance of distributed applications has become increasingly complex as the diversity of scenarios and data rates in networks increase. This includes performance evaluation of devices in contexts such as industry automation, distributed gaming and multimedia in both wired and wireless networks. Given this situation, employing network emulation for performance evaluation (see illustration in Figure 1) is an interesting alternative to test-bed experimentation as well as simulation. Moreover, in many test scenarios, commodity PCs offer sufficient capabilities for the implementation of network emulators.
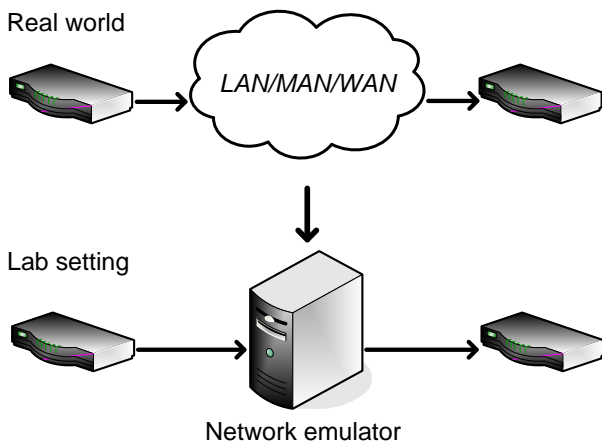

Fig. 1. Evaluating the performance of distributed applications with network emulation.

However, the implementation of such emulators is difficult for two reasons. First, emulating network dynamics is a task that includes time critical actions. Thus, kernel programming becomes necessary to protect time critical actions against unpredictable operating system scheduling delays. Second, the dynamics of real networks exhibit key invariant statistical characteristics such as long-range dependence/self-similarity[1] [2]. These characteristics have significant impact on performance evaluation [1] and are not accounted for in well-known emulators such as NIST Net [3] or the emulator module in the Linux 2.6 kernel [4] (see section II for details).

In general, there are two ways to reproduce the impact of long-range dependence in network cross traffic. The first is to employ packet traces that have been verified to reflect these characteristics to drive the network emulation. Traces that reflect long-range dependent characteristics are inherently very long [5]. The statistical characteristics are then inherited from the trace to the emulation. This is simple, flexible, and sufficient in many cases. A second way is to identify appropriate models, fit parameters to these models and implement the models in the emulator. Candidates for appropriate models are fractional Gaussian noise and fractional autoregressive integrated moving average (fractional ARIMA) [6]. These models have a large run time complexity and are difficult to compute in a real time environment. Moreover, it is a known difficult problem to find good fits for the parameters of these models to the dynamics observed in real networks [7].

Therefore, in this paper we focus on the approach to drive the emulation with packet traces and describe the design and implementation of RplTrc. Our contributions with RplTrc are as follows.

1) Rpl can reproduce the dynamics of wired and wireless networks inherent to large packet traces that were previously generated or captured;
2) Rpl employs kernel programming to protect the time critical task of network emulation against unpredictable process scheduling delays;
3) Rpl can be used to emulate a 100 MBit/s network or a lightly-loaded 1 GBits/s network on a commodity PC;
4) Rpl has a delay precision in the order of 100 μs.

The rest of this paper is structured as follows: Section II

---

[1]Note that in network traffic modeling the terms long-range dependence and self-similarity imply each other (see [1] for details).

reviews other network emulators. Section III describes the overall architecture of RplTrc. Section IV describes the implementation of RplTrc. Section V reports performance limitations and delay precision of RplTrc. Section VI reports a use case where RplTrc has successfully been employed to evaluate mobile telephone equipment before we conclude in Section VII.

II. RELATED WORK

Existing emulators that run on a commodity PC/workstation pertain to two categories. The first category does not account for the fact that network emulation is a time critical task and implement the emulation and/or network interface reading/writing as a user space task. As a result packet delays can be distorted up to several milliseconds. The most well-known of these emulators are the ns emulation [8], the Ohio-Network-Emulator ONE [9], and the emulators described in [10] and [11].

The second category employs kernel programming to support real-time actions. The emulators in this category are

1) Dummynet [12],
2) NIST Net [3],
3) the network emulator module that is build in the Linux 2.6 kernel [4], and.
4) NetPath [13].

Dummynet is part of one of the IP firewalls in the FreeBSD kernel. However, its functionality is limited to modeling constant packet delay as well as packet delay that comes from a constant bandwidth limitation on a single link. There is no support to implement variable packet delay as induced by a network. Simple extensions of Dummynet, such as employing a script in user space to configure variable delays, would result in a violation of the real-time support.

NIST Net [3] is a Linux kernel module capable of modeling variable network delays. NIST Net implements data structures such as a radix sort delay list that are particularly optimized to store large amounts of delayed packets. Values for packet delay are generated from a table which is stored in the kernel. NIST Net randomly accesses this table, which codes the inverse CDF of the delay distribution, and corrects the table's value for the correlation with the previously generated value. However, this generation process cannot account for correlation structures other than short-range dependence as can be seen from the following argument. Formally, NIST Net's model to generate delay values is

$$d_i = \rho \cdot d_{i-1} + (1 - \rho) \cdot rnd_i \qquad \rho \in (-1, 1) \qquad (1)$$

where the d_i is the i-th value, ρ is the constant lag-one correlation and the rnd_i are independent random increments from a configurable distribution. With regard to correlation structure this simple model is comparable to the first-order autoregressive process

$$X_i = a \cdot X_{i-1} + \epsilon_i \qquad a \in (-1, 1) \qquad (2)$$

where the ϵ_i are independent random increments from a normal distribution. However, this autoregressive process is

known to lead to short-range dependent values which in turn has a major impact on performance evaluations. For details see [5]. As a consequence, care has to be taken when using NIST Net for performance evaluations.

The network emulator module built in the Linux kernel 2.6 as well as NetPath use the same table-based approach to generate delay values as NIST Net which likewise lead to short-range dependent values. Moreover, the emulator module built in the Linux kernel reuses existing data structures that are optimized for fast forwarding of packets. These structures need to be modified to store large amounts of delayed packets.

III. ARCHITECTURE

We start with giving an overview on RplTrc's architecture as depicted in Figure 2. The emulator kernel module intercepts the Linux TCP/IP protocol stack between layer 2 and 3. This module performs time critical actions on incoming packets (delay, drop, duplicate). The actions are controlled on a per flow[2] level with a packet action trace that is successively loaded by a preemptive low priority user space process. To facilitate the implementation, RplTrc reuses parts of NIST Net (version 2.0.12) code for the protocol stack interception and for the delaying of packets in the kernel emulator module. However, the trace reader module is completely new.

IV. IMPLEMENTATION

Implementation of the protocol stack interception, the kernel emulator module, and the trace reader module are as follows.

A. Protocol stack interception

The Linux protocol stack is intercepted between layer 2 and 3 to delay, drop or duplicate packets (see figure 3). The current version of RplTrc, is build on the GNU/Linux 2.4 kernel. Hence, the processing path from packet arrival until interception is as follows: When an Ethernet frame arrives at a receiving interface *RX dev*, it is temporarily stored in the device's memory before a triggered interrupt copies it into a socket buffer. The deposed frame is then unpacked and analyzed by *eth_type_trans* determining the appropriate layer 3 protocol. Then *netif_rx* transfers the packet to a queue and informs the responsible layer 3 protocol handler by triggering a software interrupt. At this point the protocol stack is intercepted by hooking in the emulator module as a packet handler. This handler replaces the handler for standard IP packets.

B. The emulator kernel module

RplTrc's kernel emulator module essentially consists of a packet handler, a multi-field classifier and action handlers for dropping, delaying or duplicating packets (see figure 4). The packet handler controls the emulation and a multi-field classification of the IP headers that enables us to identify the flow associated with a packet. Based on this classification, the

---

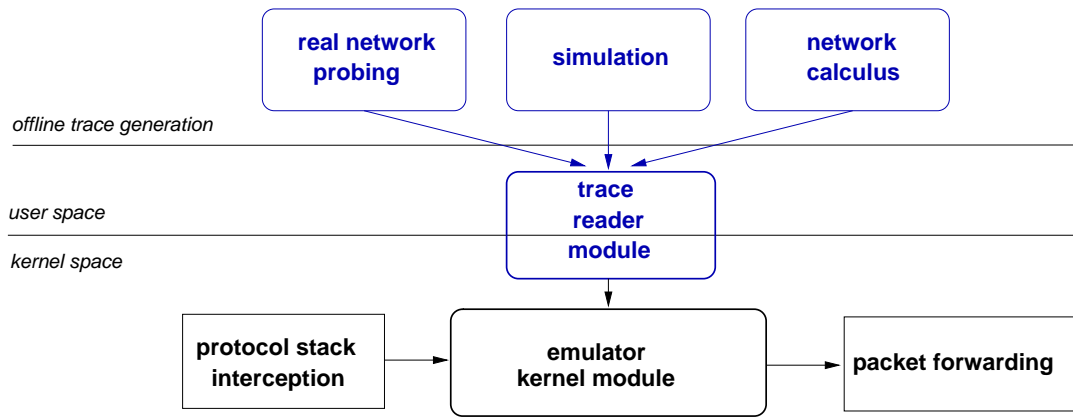[2]A flow is characterized by the port- and IP-addresses of the source and destination hosts.

Fig. 2. RplTrc's overall architecture: Previously generated packet action traces are employed to drive the network emulation.
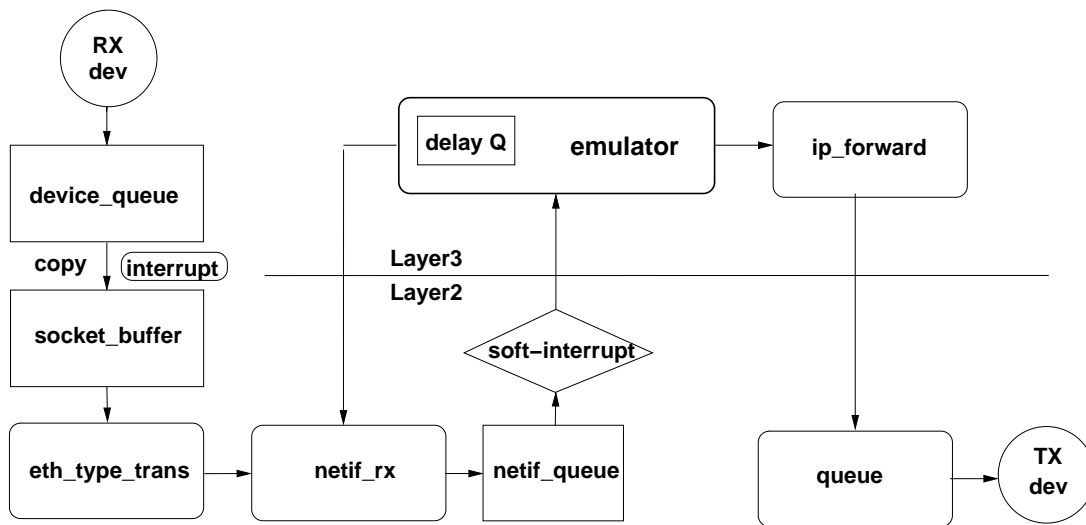

Fig. 3. The GNU/Linux protocol stack is intercepted between layer 2 and 3. The emulator is implemented as a packet handler that delays, drops, or duplicates packets.

action is determined from an instruction buffer that contains a chunk of the packet action trace.

Essentially, the emulator module consists of four functional units (see figure 4).

1) The control unit, implemented as a *packet_handler*.
2) The classification unit, implemented as the *multi_field_classification*. This unit determines which action is associated with the packet.
3) The action unit, containing the three action handlers for dropping, duplicating and delaying.
4) The packet play-out unit, containing the *interrupt_timer* and its handler (*do_timer*).

First of all, when the *packet_handler* receives a packet, it checks if the emulator is switched off or the packet has already been processed. In this case, the handler forwards the packet to the IP protocol handler (*ip_forward/ip_recv*) without further processing. In all other cases, the multi-field packet header classification (*multi_field_classification*) is invoked for

checking which action needs to be performed with this packet. For this purpose, we look up a two-level hash table containing per flow information. If no entry is found, the packet is directly forwarded to the IP protocol handler. Otherwise, the action is as specified in the packet action trace and the packet is forwarded to the corresponding handler.

This handler is either the *packet_drop* handler, the *packet_delay* handler, or the *packet_duplicate* handler. The *packet_drop* handler deletes the packet from the buffer and exits the module. The *packet_delay* handler forwards the packet into a delay list (*packet_delay_list*). This list is implemented with radix sort. Due packets are released from this list every 121 μs which is one tick in the interrupt frequency of the PC's MC146818 clock. Packets with the same due time are released in arbitrary order. Then the packets get marked and are re-injected into *netif_rx*. As a consequence of marking, the *packet_handler* of the emulator kernel module sends them directly to the standard *ip_forward* handler.

Functional Units

control unit   *classification unit*   *action unit*   *packet playout unit*
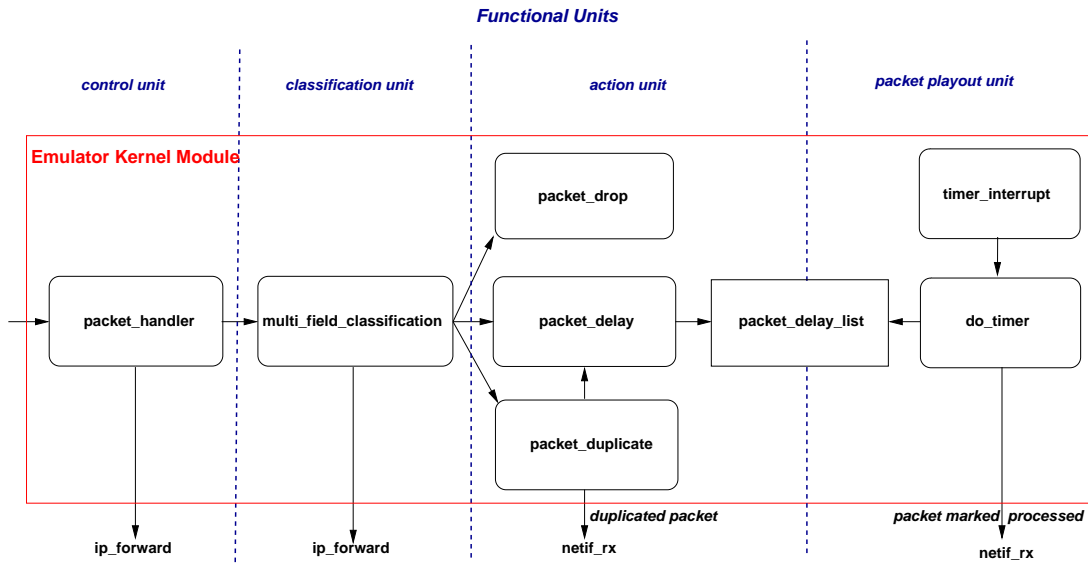
Fig. 4.   The kernel emulator module consists of a packet handler, a multi-field classifier and action handler for dropping, delaying or duplicating packets.

The *packet_duplicate* handler creates a new instance of a packet. This duplicated packet is then re-injected into *netif_rx* and handled separately. The original instance of the packet is transfered to the *packet_delay* handler. This design choice for duplication opens the possibility to create cascaded instances of the same packet.

C. The trace reader module

The trace reader module transfers packet action traces from the file system (e.g. from a disk) into packet action buffers that are located in the kernel. The emulator kernel module then reads from these buffers (see figure 5). Conceptually, trace reading from the file system into the kernel is a classical producer-consumer problem with the trace reader module being the producer and the emulator kernel module the consumer. However, this producer-consumer problem has a few constraints. These are the timely delivery of chunks of packet action traces required to drive the emulator and the fact that any distortions of the emulation due to the CPU requirement of the transfer must be avoided. From the point of implementation the trace reader is a user space process that transfers packet action traces associated with a flow. Transfers are coordinated with system calls and signals.

Alternatives to transfer traces

Before implementing the trace reader, we have evaluated the following mechanisms to transfer packet action traces between user and kernel space:

1) Transfer over **shared memory** by mapping of pages of physical memory into both kernel and user space;
2) transfer over the **process file system** by employing call back functions which invoke the process file system's *copy_from_user* function.

3) transfer over **device files** by employing their limited interface and the device files *copy_from_user* function.

Results of the evaluation including performance tests are reported in [14]. From these tests we conclude that all three mechanisms can reach transfer rates faster than 200 Mbit/s. Since a maximum transfer rate of 8.7 Mbit/s is required to support the emulation of a 100 MBit/s network[3], any of the three transfer mechanisms can be employed to implement the tracer reader module.

Implementation

We decided to employ the process file system to implement the transfer of packet action traces into kernel space since this option has the lowest implementation complexity. Loading packet action traces can be triggered by the command line interface (CLI) and is performed in three steps. First, a unique flow ID is requested from the emulator kernel module using a specific process file, and a pair of buffers associated with this flow is initialized. Second, a new producer process is initialized with the obtained flow ID and the filename of the trace. Third, the producer process gets registered in the flow table of the emulator's kernel module and starts filling the two buffers.

The trace reader module uses two buffers instead of just one to ensure continuous operation. While one empty buffer is reloaded, the emulator reads from the other. The reloading process of an empty buffer is initiated by sending a signal to the associated producer process.

We denote that a dedicated packet action trace needs to be loaded for each flow.

---

[3]46 bytes minimal packet size for IP packets without payload, 271739 packets per second, 4 bytes of action trace data per packet
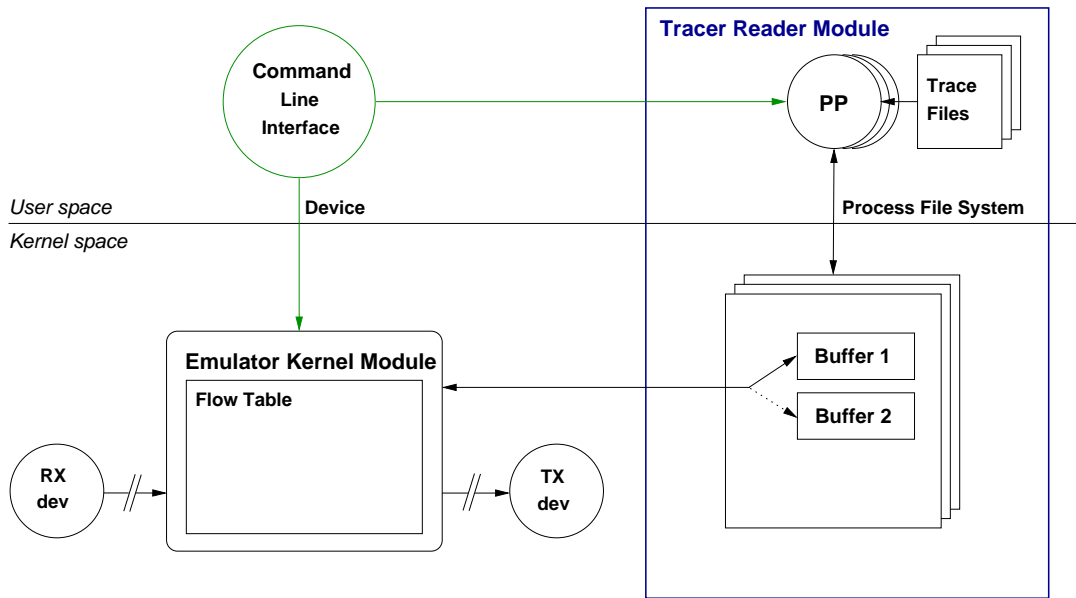
Fig. 5. The trace reader module transfers packet action traces from the file system into packet action action buffers that are located in the kernel. The emulator kernel module then reads from these buffers.
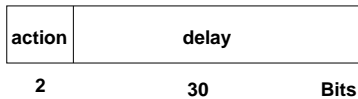
Packet trace format

Fig. 6. The specific trace format developed to encode per packet actions.

A specific trace format has been developed to code instructions for packet actions (drop, duplicate, delay) in a way that minimizes the amount of trace data transfered from the file system to the kernel emulator module. The instruction is encoded in 4 bytes (32 bits) as depicted in figure 6. The first two bits encode the type of action as listed in table I.

Action type	Code
Drop packet	00
Normal packet	01
Duplicate packet	10
unused	11

TABLE I

LIST OF CODES FOR VARIOUS PACKET ACTION TYPES

The remaining thirty bits specify the packet delay in microseconds. This encoding enables us to specify delays at a very fine granularity while the maximum of more than 17 seconds delay will likely never be reached. Action type 10 duplicates a packet. The remaining thirty bits of the code specify the delay of the original packet. A subsequent 01 action type specifies the delay of the duplicated packet. Cascaded duplications, i.e. triplications etc., are also possible.

Section Summary

RplTrc is based on

- intercepting the Linux TCP/IP stack between layer 2 and 3,
- an emulator kernel module that is registered as a layer 3 packet handler and performs the time critical task of delaying packets, and
- a trace-reader module that employs the process file system to perform the low priority task of loading chunks of packet action traces into the kernel where they are used to drive the emulation.

V. PERFORMANCE LIMITATION AND DELAY PRECISION

In this section, we report on a test of RplTrc, installed on a commodity PC[4], to find out the emulation's performance limitation and delay precision. The test setup is as depicted in figure 7.

A. Performance limitation

First of all, we test up to what rate the emulator can process offered load. We employ the sender to offer an increasing load of UDP/IP packets that contain no payload. At the same time we monitored system parameters such as CPU usage at the emulator and check the receiver to see whether the emulator timely forwards the offered load. We found that the emulator can process offered load up to 105000 packets/s, which corresponds to a minimum rate of 32 MBit/s. At higher

---

[4]Results reported here were achieved on a Dell Precision 340 Workstation with Intel Pentium P4 2.0GHz, 512MB RAM, 40GB Hitachi Deskstar 120GXP, two 3Com Tornado 3c905C NICs running Debian Sarge 2.4.27.
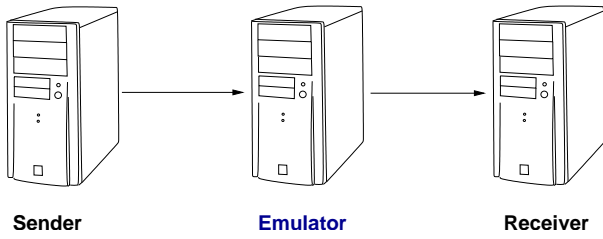
Fig. 7. Test Setup to measure performance limitation and delay precision of the emulator.

offered load the trace reader process cannot refill the trace buffers in the kernel emulator module. As a consequence these buffers under-run. The underlying reason for this under-run is the large amount of interrupts caused by incoming packets that have higher priority than the user level trace reader process. Thus, the CPU usage is the limiting factor. Table II lists the CPU usage for various offered amounts of load. Table III lists reference values of CPU usage with the emulator been switched off. From this test we infer that RplTrc, installed on a commodity PC, can be employed to emulate a well-loaded 100 MBit/s network or a lightly-loaded 1 GBits/s network.

Packets [x1000/s]	CPU usage [%]	Buffer under-run
50	48	no
100	89	no
105	92	no
110	96	yes

TABLE II

CPU USAGE OF THE EMULATOR AT VARIOUS OFFERED LOADS.

Packets [x1000/s]	CPU usage [%]	Buffer under-run
100	70	no
110	75	no

TABLE III

CPU USAGE OF THE EMULATOR PC AT VARIOUS OFFERED LOADS (EMULATOR SWITCHED OFF).

B. Delay precision

Besides performance limitation, delay precision is a key performance characteristic of an emulator. We thus determine the delay precision of RplTrc on a commodity PC. We run various traces and inject/log traffic at the sender/receiver with a real-time measurement infrastructure that has a precision of 3μs. We investigate the emulator's delay precision under various synthetic traffic patterns and traces which we have gathered by probing ETH's network. For details see [14]. From our results, we infer that the emulator has a precision of two clock ticks which is approximately 242μs on the commodity PC used for the tests. We denote that this precision is more than five times more accurate than the

precision of the network emulator build in the Linux 2.6 kernel [4].

Section Summary

Performance limitation and delay precision of RplTrc installed on a commodity PC can be summarized as follows: Tests show that RplTrc can process at least 32 MBit/s of offered load. Thus RplTrc can be used to emulate a 100 MBit/s network or a lightly-loaded 1 GBits/s network. The delay precision with which RplTrc reproduces delays is in the order of 100μs.

VI. USAGE EXAMPLE

We have employed RplTrc to evelute the performance of synchronization algorithms in newly developed circuit emulation adapters. The adapters are employed to connect mobile telephone base stations over packet-switched networks such as Metropolitan Gigabit Ethernets (see figure 8 for an illustration of the scenario and [15] for further details). The task of the sending adapter is to encapsulate the plesiochronous TDM signals from base stations and send the resulting frames over the Metropolitan Gigabit Etherent. The task of the receiving adapter is to decapsulate and play-out the TDM signal at a rate synchronized to the sending base station. Plesiochronous TDM signals of a base station have a nominal frequency of 8000Hz with a variation of $\pm$ 50 ppm. However, despite this rather large tolerance of 50 ppm, each base station's specific frequency within this tolerance needs to be preserved across the Ethernet. This perservation is necessary to prevent long-term buffer over- or underrun and has to be such that the maximum time interval error (MTIE) requirement as specified by ITU-T can be met [16]. This task is usually achieved by time stamping encapsulated signal at the sending and receiving adapter, estimating the minimal time stamp difference and subsequently correcting for any change in this difference. The rationale behind taking the minimal difference for synchronization is that this difference does not depend on the traffic characteristics of the Metropolitan Gigabit Etherent. At perfect synchronization, this difference is equal to the minimal network delay, which is in the order of 100 microseconds. Thus, a network emulator that is employed to evaluate synchronization algorithms in circuit emulation adapters needs to have a delay precision which is at least in this order to get meaningful results. This delay precision requirement is met with RplTrc.

Figure 9 depicts the lab setting we have used to evelute the performance of various synchronization algorithms in circuit emulation adapters with RplTrc by measuring the MTIE. Figure 10 depicts a sample result. The initial skew between the clocks in sending and receiving adapter has been set to 50 ppm. The depicted sample result has been measured after an initial phase of 10 seconds. The packet trace replayed for this evaluation is from a an active probing measurement in the ETH Zurich campus network.

A technical report [17] documents the complete evaluation with various classes of synthetic and measured packet traces.
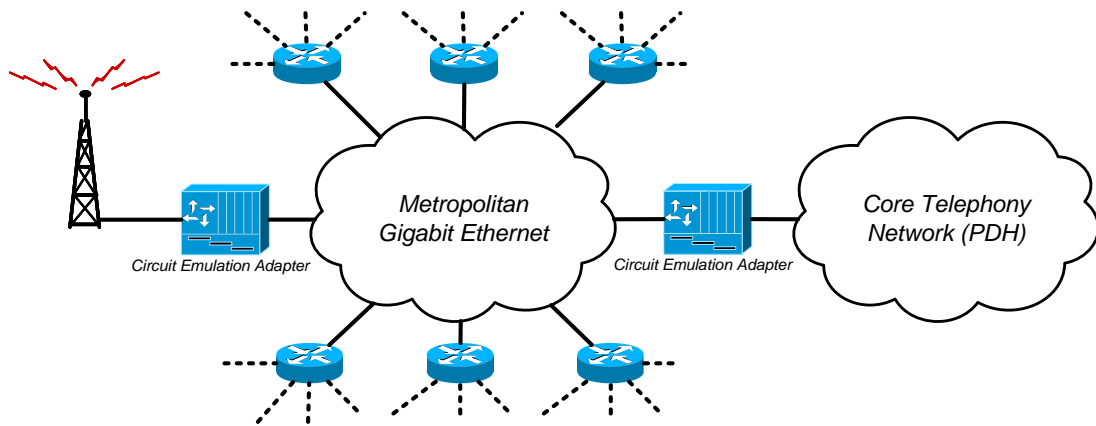
Fig. 8. Usage example: The emulator has been used to evaluate the performance of synchronization algorithms in circuit emulation adapters that connect mobile telephony base stations over Metropolitan Gigabit Ethernets.
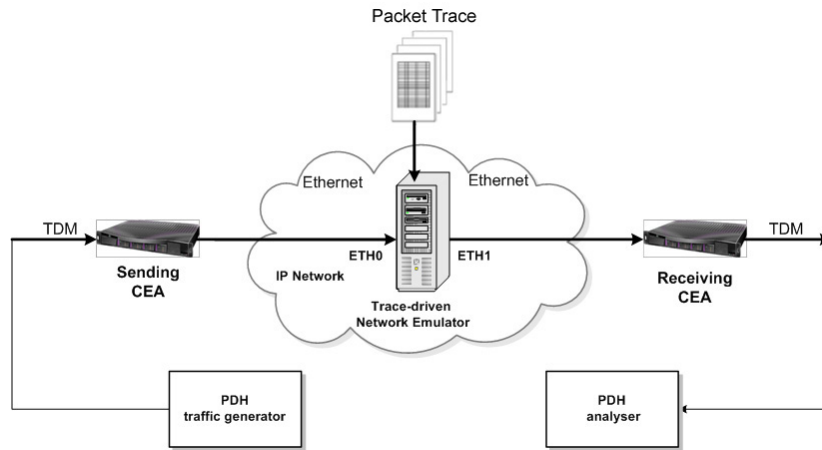

Fig. 9. Evaluation setup to measure the perfomance of synchronization algorithms in circuit emulation adapters with RplTrc in terms of MTIE (maximum time interval error).
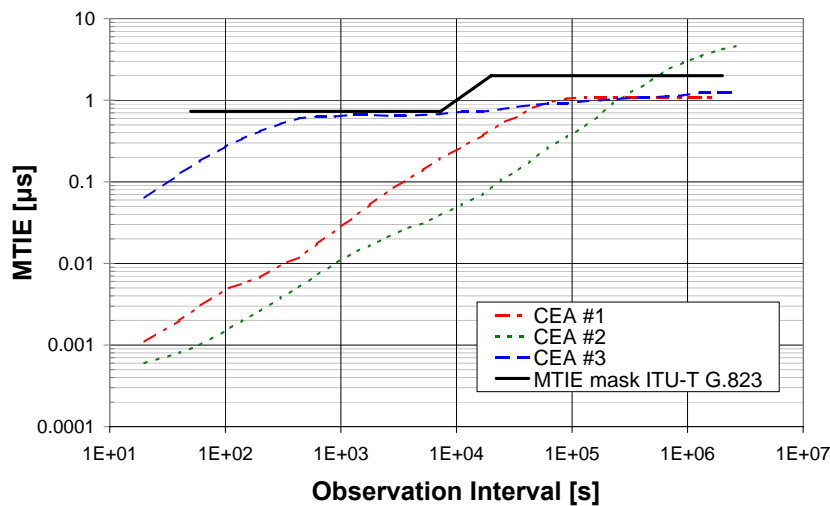

Fig. 10. MTIE results for three synchronization algorithms. The synchronization algorithms work by estimating the minimal network delay, which does not depend on network dynamics. This delay is in the order of 100 microseconds for Metropolitan Gigabit Ethernets. Thus, the emulator needs to reproduce packet delays at a precision which is at least of the same order.

The packet traces represent traffic patterns that can be expected in metropolitan Gigabit Ethernets. This includes patterns that correspond to a change in the physical delay after a spanning tree re-computation and traffic bursts as measured in the ETH campus network.

In addition to use RplTrc to evelute the performance of synchronization algorithms in circuit emulation adapters, we intend to use RplTrc to evelute synchronization algorithms in other contexts. These contexts include industry automation, distributed gaming and multimedia over wireless networks.

VII. CONCLUSION

We have presented the design and implementation of RplTrc, a network emulation tool. RplTrc's architecture is based on a combination of a kernel module that performs the time critical task of network emulation and a user space module that successively transfers packet traces into the kernel to drive the emulation. Since the length of traces is not limited, RplTrc is capable to account for important performance characteristics inherent to real networks such as long-range dependence and self-similarity of cross-traffic. This feature differentiates RplTrc from other emulators such as NIST Net and the emulator module in the Linux 2.6 kernel. Moreover, we have shown that RplTrc, installed on a commodity PC, can be used to emulate a 100 MBit/s network or a lightly-loaded 1 GBits/s network at a delay precision in the order of 100 μs.

Our experience and the feedback from our industry partner indicate that RplTrc can be employed for a wide variety of testing purposes in lab environments. This includes performance evaluation of devices and applications in contexts such as industry automation, distributed gaming and multimedia in both wired and wireless networks. Moreover, we are currently porting RplTrc into the 2.6 Linux kernel. [Comment: A release can be expected together with the camera ready version of this paper.] Further work is to port RplTrc to a real-time Linux (RTLinux [18] or RTAI [19]) to get the delay precision below the order of 10 μs.

The code of RplTrc is available via *www.tik.ee.ethz.ch/rpltrc*.

VIII. ACKNOWLEDGMENT

REFERENCES

[1] K. Park and W. Willinger, *Self-Similar Network Traffic and Performance Evaluation*. New York, NY, USA: John Wiley & Sons, Inc., 2000.

[2] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson, "Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level," *IEEE/ACM Transactions on Networking*, vol. 5, no. 1, pp. 71–86, 1997.

[3] M. Carson and D. Santay, "Nist net: A linux-based network emulation tool," in *ACM SIGCOMM Computer Communications Review*, vol. 33, 2003, pp. 111–126.

[4] "The Linux Kernel Archives," www.kernel.org.

[5] J. Beran, *Statistics for long-memory processes*. Chapman and Hall, 1994.

[6] J.-Y. L. Boudec, "Performance evaluation lecture notes (methods, practice and theory for the performance evaluation of computer and communication systems)," Lecture Notes EPFL, 2005, http://ica1www.epfl.ch/perfeval/.

[7] M. M. Thomas Karagiannis and M. Faloutsos, "Long-range dependence, ten years of internet traffic modeling," in *IEEE Internet Computing*. IEEE Computer Society, September 2004, pp. 1089–7801.

[8] "Network emulation with the ns simulator," http://www.isi.edu/nsnam/ns/ns-emulation.html.

[9] Ohio University's Internetworking Research Group, "ONE - the Ohio Network Emulator," http://masaka.cs.ohiou.edu/one/.

[10] B. N. G. Nguyen, R. Katz and M. Satyanarayanan, "Trace-based mobile network emulation," in *Proc. of the ACM SIGCOMM'97*, Cannes, France, sept 1997, pp. 51–61. [Online]. Available: citeseer.ist.psu.edu/noble97tracebased.html

[11] L. S. Brakmo and L. L. Peterson, "Experiences with network simulation," in *Measurement and Modeling of Computer Systems*, 1996, pp. 80–90. [Online]. Available: citeseer.ist.psu.edu/brakmo96experiences.html

[12] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *ACM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, 1997. [Online]. Available: citeseer.ist.psu.edu/rizzo97dummynet.html

[13] S. Agarwal, J. Sommers, and P. Barford, "Scalable network path emulation." in *MASCOTS*, 2005, pp. 219–228.

[14] R. Baumann and U. Fiedler, "A Tool for Emulating Real Network Dynamics. TIK Report 218," ETH Zurich, Tech. Rep., Mai 2005.

[15] ——, "Towards Connecting Base Stations over Metro Gigabit Ethernets," in *Proceedings of Third International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks (QSHINE'06)*, Waterloo, Ontario, Canada, 2006.

[16] ITU-T Recommandation G.823, "The control of jitter and wander within digital networks which are based on the 2048 kbit/s hierarchy," March 2000.

[17] Rainer Baumann and Ulrich Fiedler, "Evaluation of Circuit Emulation Adapters. TIK Report 254," ETH Zurich, Tech. Rep., June 2006.

[18] FSMLabs, "RTLinux (Real-Time Linux)," www.fsmlabs.com.

[19] "RTAI (Real-Time Application Interface)," www.rtai.org.