Self-synchronizing One-way Delay Measurement in the Internet

Frank Eyermann
Institut für Technische Informatik
Universität der Bundeswehr
Munich, Germany
Frank.Eyermann@unibw.de

***Abstract*—End-to-end packet delay is the network parameter with maximum impact on performance of distributed applications. This is especially true for soft real-time applications, which are delay-sensitive by definition, but also for applications relying on the TCP protocol whose sliding window mechanism performs badly in case of high packet delays. Therefore, measuring packet delay is an important task for both network operators and application developers. This paper presents a tool set for measuring and evaluating one-way end-to-end delay and packet loss that can be operated on standard PCs without additional external timing sources. We chose a script-based approach that can even be executed on virtualized platforms. The self-synchronization mechanism embodied in the trace evaluation is a distinctive feature that omits the need for expensive external clocks (as e.g., GPS receivers). We also show a wide-ranging set of measured traces and their most prominent statistical properties.**

Keywords-One-way delay, packet loss, measurement, tools

I. INTRODUCTION

Even though first mechanisms for ensuring quality of service (QoS) in IP networks have been proposed almost 15 years ago [3], they are still rarely used in the Internet. In general, only best-effort services that treat all packets equally and do not respect special requirements of single packets are available.

However, applications requiring an elevated level of QoS, as e.g., Voice-over-IP (VoIP), IP Television (IPTV), or Video-on-Demand (VoD), become more and more important for private users but also for Business-to-Customer (B2C) and Business-to-Business (B2B) communication. These applications only work satisfactorily if one-way end-to-end delay, packet loss, delay variation (also called jitter), and/or throughput are above or below a certain threshold. As techniques for ensuring these QoS parameters are still not embodied in today's networks, users can only "hope" that the network has sufficient performance.

Therefore, it is important to test regularly the actual performance of the Internet with respect to the above mentioned performance parameters. This work presents a tool set for measuring one-way delay, delay variations, and loss. The tools have already been used to capture a wide-ranging set of traces in EmanicsLab [4], [5].

The paper is structured as follows: First, in Section 2 we present a summary of the requirement analysis for the tools, followed, in Section 3, by related work on this topic. Later, in Section 4 we describe the tools we have developed for measuring traces and evaluating them. Section 5 discusses measured traces and their evaluation. Finally, Section 6 concludes the paper and summarizes the main results.

II. REQUIREMENTS

Analyzing suitability of networks for multimedia or real-time services requires testing the network over a long period of time in order to cancel different load situation as regularly observed at different times of day or week. As continuous measurement would generate an enormous amount of measurement data, it may be preferable measuring short intervals of a few minutes scattered over a period of days or weeks. This requires the capability to flexibly schedule measurement runs. The number of packets sent during such a measurement run, the size of the packets as well as the frequency of packet generation has to be easily configurable. Furthermore, for documentation purposes and for easier repetition of experiments a script-based approach would be beneficial.

In addition, there are a number of non-functional requirements. First, the tools should be executable on different operating systems, including at least Linux and Windows. As more and more servers—especially in testbeds—are virtualized, the tools have to be tested on such platforms, too.

Precise one-way delay measurement is typically performed using additional hardware for synchronizing the sender and receiver host (e.g., GPS receivers). This increases efforts and costs drastically especially if cables have to be installed.

Synchronization, however, is not necessary if only relative delay or inter-packet delay variation is of interest: Packet delay consists of static and dynamic delay components: Static components are propagation, serialization, and processing delay [7]; Queuing delays are dynamic components. Even though the static components cannot be neglected, these constant values are only troublesome in case of satellite communication and do typically not exceed some tenth of milliseconds. More problematic is the dynamic part of the delay as, first, its share might be bigger than the static one and, second, the changes in delay lead to unpredictable arrival times at the receiver. Therefore, the tools should also be able to work with unsynchronized hosts and be able to measure the dynamic delay components.

III. RELATED WORK

This section presents related work on delay, delay variation, and loss measurement in the IP networks.

A. IPPM

The goal of the *IP Performance Metrics working group* (IPPM WG) of the Internet Engineering Task Force (IETF) is to define metrics that can be applied to the quality, performance and reliability of Internet data delivery services [9]. In addition, the working group defined a general framework for accurately measuring and documenting the metrics [13]. The IPPM WG does not define or suggest how the performance parameters are measured. They emphasize on definitions and the unambiguous understanding what a parameter expresses, so that measurement results can be compared, shared and validated by different entities.

B. Measurement tools

Quite a number of tools for delay and loss measurement are available, including, e.g., *ping*, *cing* [1], *king* [8], *netperf* [11], or *scriptroute* [16]. All tools use probing techniques, i.e., they inject artificial packet (so-called probes) into the network and observe their behavior. A sub-group of these tools uses so called inference techniques: While delay measurement in general requires two programs, i.e., a sender that generates the probes, and a receiver that evaluates the probes, tools embodying inference techniques use standard behavior of protocol stack implementations on nodes in the network to receive feedback. Thus, these tools can combine sender and receiver functionality in one program but can only measure round-trip delay and not one-way delay. Examples for such programs include *ping* or *traceroute*.

None of the programs mentioned above is able to perform flexible script-based long-term one-way delay and delay variation measurements. Either the tools use inference techniques that by definition cannot measure one-way performance or their design does not include the possibility to schedule measurement runs. Furthermore, none of these tools have been tested on virtualized platforms or with unsynchronized hosts.

IV. MEASUREMENT TOOL

A. Tool design

One-way delay, delay variation, and packet loss measurement requires a pair of programs: a sender, generating the probing packets as well as a receiver, collecting the probes and writing a log file.

All time intervals and timestamps are stored and transferred in units of 100µs. This value is a compromise between timer resolution and storage space. On the one hand, sub-milliseconds resolution is approximately one magnitude smaller as typical measurement values, and therefore, the effect of the rounding error is negligible. On the other hand, a signed 32-bit integer counting steps of 100 µs overflows only every 60 hours – long enough to detect any overflow of counters.

Measuring such small time differences is not possible using the built-in real-time clock (RTC) of PCs. Furthermore, accessing the RTC is quite slow thereby reducing program performance. Intel invented a quickly accessible, high-resolution timing source for their Pentium processor. The TSC (time-stamp counter) is a 64-bit processor register counting its clock cycles. The time resolution of this register is more than sufficient (on a 1 GHz processor, the register counts microseconds) and access to this processor register takes only a couple of processor cycles. All other x86 processor manufacturers later adopted the TSC for their processors [12].

The drawback of using this counter is the varying processor speed from computer to computer, and accordingly, the necessity to calibrate the TSC in order to produce comparable results. Furthermore, state-of-the-art processors may reduce their speed for preserving power in times with low load. This also influences the TSC counter. Therefore, during calibration and during send cycles busy waiting is necessary in order to prohibit power-saving features.

B. Scripting language definition

Deterministic probe sending schedules can be flexibly described using the following four script commands implemented by the tool:

- `at` *absoluteTime*
  waits until *absoluteTime*. The parameter time can be a real point in time (e.g., 13:05:23), or the syntax `*/`*`n`* can be used for hour, minute, or second. In this case `at` will wait until the current hour, minute, or second, respectively, can be divided through *n* without remainder (e.g., `at */2:00:00` will wait until the next full even hour).
- `send` *destAddress repeat size delayMillis*
  sends a burst of probes. Probes of *size* bytes are sent every *delayMillis* milliseconds to IP address *destAddress* for *repeat* times.
- `Loop` *iter*
      body
  `bend`
  processes the block *body* for *iter* times. If *iter* is 0, the command loops infinitely. *body* can be a series of commands including other loop statements.
- `wait` *millis*
  waits for *millis* milliseconds.

Respective scripts are interpreted by `Lattes`, a C++ application described in Section D.

C. Structure of Probes

The probes consist of three unsigned 32-bit integers as well as random padding data filling the rest of the packet. The first integer is the *Flow ID*. Each burst gets its own *Flow ID*. During start-up of `Lattes` a *Flow ID counter* is initialized with a value based on the current time. Every time `Send::doit()` is called the *Flow ID* is incremented by 1. The second integer is the *Serial number*, starting from 1 for the first probe of a burst and incremented by 1 for each probe sent. And finally, the third integer stores the time on the sender in units of 100 µs when the probe was generated. The probes are sent as UDP packets; therefore, the actual size of the IP packet is 28 bytes larger than stated in the *size* parameter.

D. *Lattes*

`Lattes` is an interpreter of the scripting language defined above. This section presents the design, implementation, and interfaces of `Lattes`. The receiver of the probes, `Latreceiver`, will be described in the following section.

Each command is implemented as a class inheriting from the class `Command` (see Fig. 1). The class `Command` has a virtual function `doit()`, which must be implemented by each not abstract child class and contain the functionality of the command. Based on this approach `Lattes` can be easily extended by simply implementing a new class inheriting from `Command`.

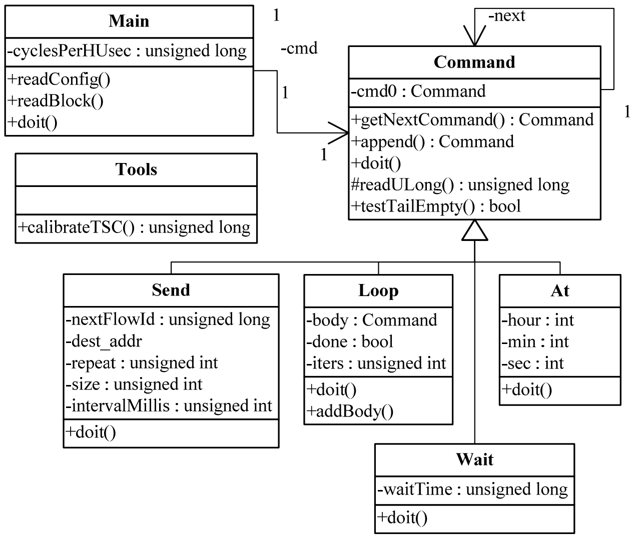

Figure 1. Class diagram for Lattes

During start-up of the program the `Main` class reads in the script. The method `Main::readBlock` reads line by line from the script and compares the start of a line with the command literals. If one command literal is recognized, the respective object is created and the rest of the line is handed over to its constructor, which evaluates the command's parameters. The objects are stored as a linked list.

Furthermore, the `Main` class calls `Tools::calibrateTSC()`, a static method that measures the increase of the TSC register per time unit.

After reading in the entire script and constructing all `Command` objects, the main program calls the `doit()` method of the first command. Each `doit()` method calls `doit()` of the next command in the chain after performing its functionality.

The `loop` command is the only command manipulating the control flow of the script. The class `Loop` has an additional attribute `body` that points to the first command of the body. The `next` attribute of the `Loop` class points to the first command following the corresponding `bend` statement. The `doit()` method of the class `Loop` calls `body->doit()` for *iter* times before continuing with the next command after `bend` (*i.e.*, `next->doit()`).

E. *Latreceiver*

`Latreceiver` was developed as receiver for the probes sent by `Lattes`. Common functions (*e.g.*, calibration of the TSC values or probe format definition) are shared between both programs. The program writes all events to a log file.

`Latreceiver` maintains a list of all currently running flows. If a probe is received, this list is searched for the Flow ID stored in the probe. If the ID cannot be found, the start of a new flow is detected and a new record containing information on the received packet is added to the list as well as a respective remark is written to the log file.

If a lost packet is detected, i.e., the serial numbers of two probes of the same flow are not consecutive, a remark is written to the log file. For consecutive packets the packet-to-packet delay variation Δt_i^{ptp} , i.e., the difference of the delay of the previous packet and the current packet, as well as the (not-normalized) relative delay Δt_i^{rel} is calculated and written to the log file (see following section). Reordered packets that have been passed by a successive packet are dropped by the receiver.

EmanicsLab uses the *MyPLC platform* [14] consisting of virtualized Linux systems. The virtualization produces a long latency between receiving a frame on the physical interface and the processing of the packet in user-space. This leads to linear dependency between consecutive probes as depicted in Fig. 2. Only virtualized systems show this effect.

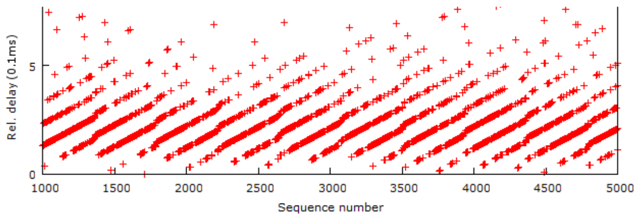

Figure 2. Trace collected on virtualized host

As discussed in [16] and [10], these distortions can be minimized or even eliminated if time stamping is already performed in the interrupt service routine, which copies the frame from the physical network device to memory. Consequently, `Latreceiver` uses the *pcap library* [18] for time stamping the packet already in the kernel allowing the usage of the programs also on virtualized systems.

`Latreceiver`'s main focus is on probe reception. All further processing, *e.g.*, splitting the log file in files containing only information about one flow, is task of post-processing tools described in subsequent sections.

F. *Measurement value evaluation*

`Latreceiver` as well as some post-processing tools are used to calculate different parameters from the measured values. `Latreceiver` directly calculates the packet-to-packet delay variation Δt_i^{ptp} and the (not-normalized) relative delay Δt_i^{rel} (see Equations 1 and 2)

$$\begin{aligned}\Delta t_i^{Sender} &= t_i^{Sender} - t_{i-1}^{Sender}\\ \Delta t_i^{Receiver} &= t_i^{Receiver} - t_{i-1}^{Receiver}\\ \Delta t_i^{ptp} &\equiv \Delta t_i^{Sender} - \Delta t_i^{Receiver}\\ &\equiv t_i^{Sender} - t_{i-1}^{Sender} - \left(t_i^{Receiver} - t_{i-1}^{Receiver}\right)\end{aligned} \tag{1}$$

$$\Delta t_i^{rel} \equiv \sum_{j=0}^{i} \Delta t_j^{ptp} \tag{2}$$

with t_i^{Sender} being the sending time of probe i as stored in the probe and $t_i^{Recevier}$ the point of time probe i was received. The variable Δt_i^{rel} denotes the accumulated delay variation, and thus, represents the dynamic component of the packet delay.

The term Δt_i^{ptp} can be positive or negative. If queues in routers grow, the packet-to-packet delay variation is positive as successive packets spend more and more time in router queues. In case the queues are shrinking, packet-to-packet delay variation is negative as each packet spends less time in router queues.

Assuming that the network is only temporarily overloaded and router queues are empty at some time the relative delay can be normalized in a way that the smallest value for the relative delay is 0 (see Equation 3):

$$\overline{\Delta t_i^{rel}} \equiv \Delta t_i^{rel} - \min_j(\Delta t_j^{rel}) \tag{3}$$

The relationship of the variables Δt_i^{ptp} and $\overline{\Delta t_i^{rel}}$ is visualized in Fig. 3. In this example the sender generates probes every 20 ms. These probes arrive at the receiver with different delays. In the example the network was overload in the time from t_1^{Sender} to t_3^{Sender}, and thus router queues grew; afterwards, until t_5^{Sender} the queues empty again.

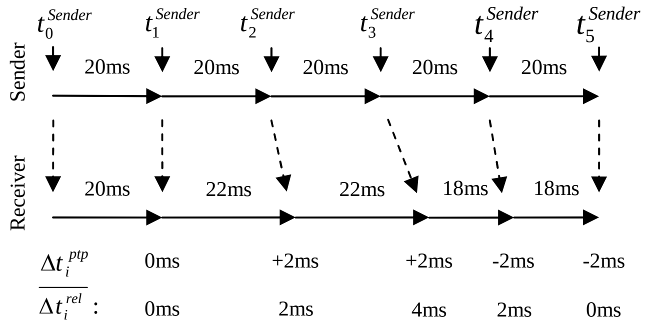

Figure 3. Impact of queueing on measurement variables

G. *Post-processing*

Fig. 4 shows the packet-to-packet delay variation Δt_i^{ptp} and the normalized relative delay $\overline{\Delta t_i^{rel}}$ of a flow consisting of 7000 probes (approx. 5 min) sent between two nodes.

In contrast to the theoretical concepts discussed above, the relative delay is increasing. All traces captured look similar; however, the slope of the relative delay changes from trace to trace including also negative slopes.

The reason identified is synchronization errors resulting from two factors: The first-order error of real-time clocks is not zero, *i.e.*, the time difference between two clocks is not constant but gets greater or smaller with time. This effect is called skew and can easily be observed on most clocks. This error, however, is typically much smaller than the error observed in the traces. In Fig. 4 the error is 110 ms per 300 s or approx. 30 seconds per day.

Additional measurement errors might be introduced because time at sender and receiver is measured in processor cycles and is then converted to real time units. Each time one of the programs is started the conversion factor is calculated with the help of a calibration routine (see Section IV.D). This calibration might add an additional error component.

Other reasons for changing delay, like path changes or traffic shaping by providers, cannot be held liable for this effect, as they do not cause continuously increasing or decreasing relative delay over minutes.

A more accurate calibration of clocks that would not only eliminate the clock offset but also the skew is only possible with additional hardware. But as the error is linear in time it also can be corrected *ex post*. Therefore, for each flow the slope of the dynamic delay has to be determined. This can be done by fitting a line onto the lowest delay values. This line represents a dynamic delay of zero.

The line and all observations are then projected to the x axis. This converts Fig. 4 into Fig. 5 (packet-to-packet delay variation is not shown as it does not change; scale of y axis is adapted).

H. *Self-synchronization*

Fitting the line to the observation values is not a trivial task. Quite a lot of algorithms exist for fitting straight lines into a cloud of observations (*e.g.*, ordinary least square estimation, OLS) but all these assume positive and negative

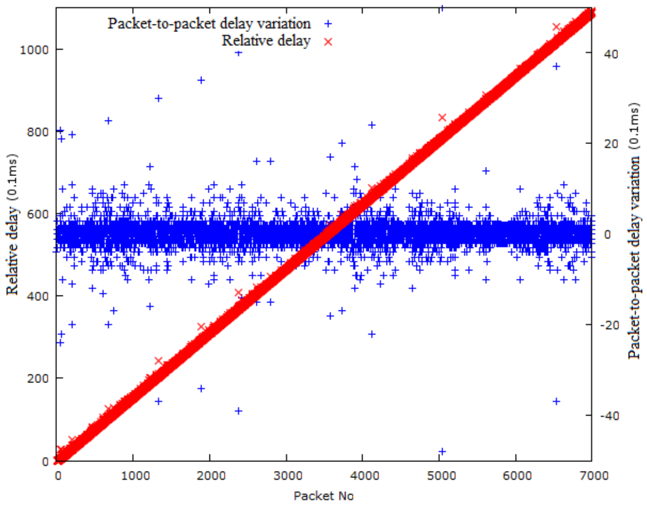

Figure 4. Sample trace showing the skew in relative delay

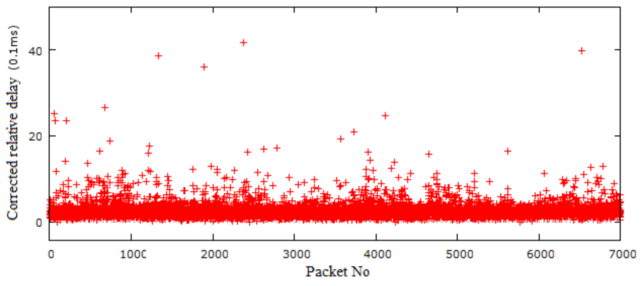

Figure 5. Corrected relative delay

variations from the mean (and not only positive as in the present case). For this work several approaches were developed and tested. The one that shows the best results iteratively searches from the right and left side for a pair of points (p_1, p_2), which best represent the line. The algorithm works as follows:

The smallest observations in a 1% interval from the right and from the left, respectively, are chosen as start values for p_1 and p_2. Based on the line through those both points, the observations are transformed for the first time. This typically produces also negative observations, as the guessed p_1 and p_2 have not been optimal.

The solution is improved by choosing the smallest point p' of the transformed observation set and replacing either p_1 or p_2, depending if p' is in the right or left half of the set. Afterwards, the set is transformed according to the newly constructed line again. This step is repeated as long as there are negative observations. Regularly only a few iterations are necessary.

This procedure describes a very robust and fast algorithm. Several thousand measurement traces (see next chapter) have been processed and afterwards their plausibility was checked statistically. The algorithm is robust to outliers as in the present scenario outliers can only be positive, the algorithm, however, considers only the lowest measurement values. Furthermore, the algorithm is universal as it does not rely on the structure of the data or any input parameters.

V. TRACES

With the help of the tools developed a wide ranging set of delay measurements is performed. All measurement runs are collected in EmanicsLab.

A. *EmanicsLab*

EmanicsLab is a European research network consisting of 20 nodes at 10 sites across Europe (see Fig. 6). EmanicsLab partners use the network for research activities in the area of network and service management, including distributed flow collection and analysis, distributed intrusion detection systems, as well as distributed monitoring and accounting systems. It is funded by the *European Network of Excellence for the Management of Internet Technologies and Complex Services (EMANICS)*.

EmanicsLab is based on MyPLC, the backend management infrastructure of PlanetLab [14]. Project partners can run their services and applications in own *slices*. A slice is a fraction of resources on a set of nodes implemented as virtual machines [2]. The nodes run a customized Linux operating system. Access control and establishment of slices is controlled remotely by a central management system. Project partners typically get root access to their slice.

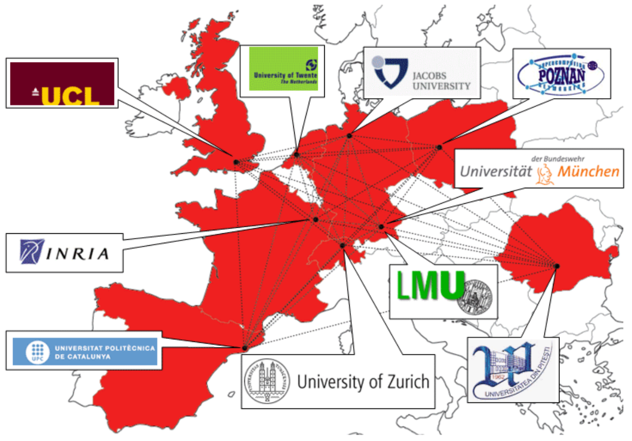

Figure 6. EmanicsLab nodes in Europe ([4])

B. *Measurement runs*

Seven out of ten sites of EmanicsLab have been participating in trace collection. The other three institutes joined EmanicsLab only after the measurements have already begun. In total, five different test runs have been scheduled. All tests are structured similarly in order to produce comparable results. Each run measures 36 times for five minutes the 42 unidirectional vertexes in a fully-meshed graph of all participants in a time frame of 72 hours. This adds up to 1512 measurement runs per test. The schedule of runs within a test ensures that one station is either sender or receiver of probes at one instance of time. In total about 18 Mio single measurements have been performed.

TABLE I. TRACE OVERVIEW

Name	Burst size (# of packets)	Probe size (byte)	Interval (ms)
Test01	30000	60	10
Test02	7000	60	50
Test03	3500	60	100
Test04	150000	60	3
Test05	250000	60	2

C. *Measurement results*

The measured data contains a huge amount of information. The data has already been used for the assessment of a multi-domain auditing system for end-to-end SLAs [6]. In the following some statistical properties as a proof of concept for the measurement tools are shown. These data, however, could also be helpful for application developers and network operators.

1) *Moments*

Fig. 7 and Fig. 8 show the mean, the 95%-, the 97.5%- and the 99%-percentiles of $\overline{\Delta t_i^{rel}}$. Percentiles are used instead of standard deviation or variance as the distribution of the packets typically belongs to the class of so-called heavy-tailed distributions, which do not have a finite standard deviation. Percentiles do not suffer from this effect.

As can be seen from the figures, for approx. two thirds of the measured paths the relative delay is negligible (the relative delay of 99% of all packets is between 1 and 2 ms). For some of the measured paths these percentiles are 100 times bigger. The reasons are heavily loaded Internet connection at two sites. Tests scheduled after these sites have upgraded their uplink show a more homogenous result.

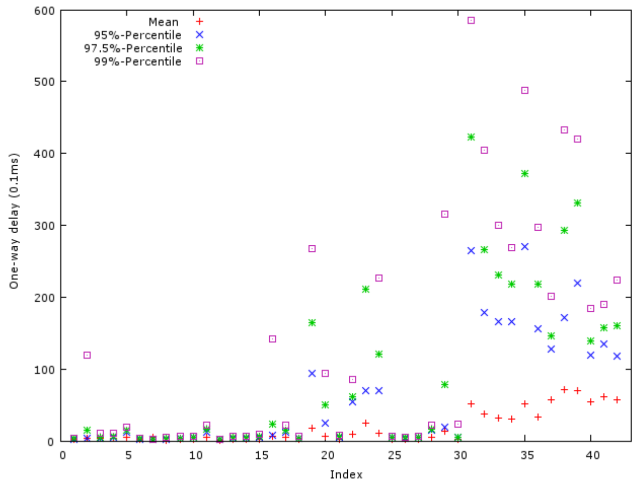

Figure 7. Mean, 97.5%- and 99%-Percentile of Test01

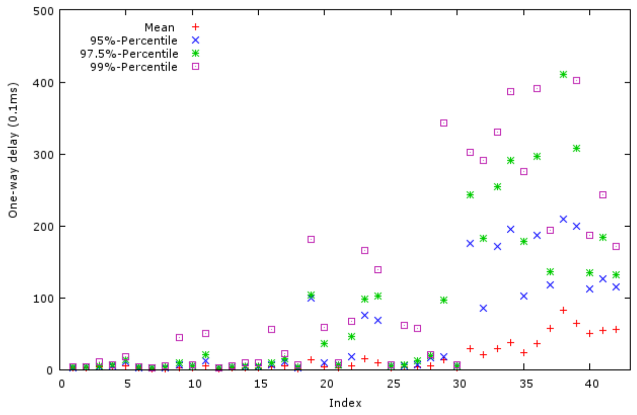

Figure 8. Mean, 97.5%- and 99%-Percentile of Test02

VI. CONCLUSION

This paper presented a tool set for measuring one-way delay and delay variations. Additional hardware for clock synchronization might be used but if not available as a fall-back the tools may remove first-order error of the clock, i.e., skew, in a post-processing step. Furthermore, the tool set is script-based allowing automated measurements over a longer period of time. Furthermore, the tools have been tested on virtualized platforms as frequently used in testbeds (e.g., PlanetLab).

In EmanicsLab, a testbed arisen out of the Network of Excellence EMANICS, a wide ranging set of measurement traces with over 18 Mio singleton measurements have been performed.

ACKNOWLEDGMENT

The authors wish to thank the members of the Chair for Communication Systems and Internet Services at the Universität der Bundeswehr Munich, headed by Prof. Dr. Gabi Dreo Rodosek, for helpful discussions and valuable comments on previous versions of this paper. The Chair is part of the Munich Network Management Team.

REFERENCES

[1] Anagnostakis, K., Greenwald, M., and Ryger, R.; “cing: Measuring Network-Internal Delays using only Existing Infrastructure”, 22nd Annual Joint Conference of the IEEE Computer and Communications (INFOCOM), pp. 2112-2121, 2003.

[2] Bavier, A., et al; “Operating system support for planetary-scale network services”, Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation 2004 (NSDI'04), pp. 19-19, San Francisco, USA.

[3] Braden, R., Zhang, L., Berson, S., Herzog, S., and Jamin, S.; “Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification”, RFC 2205, Sept 1997

[4] EmanicsLab, subproject of the European Network of Excellence for the Management of Internet Technologies and Complex Services (EMANICS), www.emanicslab.org

[5] European Network of Excellence for the Management of Internet Technologies and Complex Services (EMANICS), Project Number: FP6-IST #026854, www.emanics.org.

[6] Eyermann, F.: “An Auditing System for Multi-for Mulit-domain IP Carrying Service Level Agreements”, Doctoral Dissertation, unpublished.

[7] Filsfils, C. and Evans, J.; “Engineering a multiservice IP backbone to support tight SLAs”, Computer Networks, Volume 40, Issue 1, pp. 131-148, September 2002.

[8] Gummadi, K., Saroiu, S., and Gribble, S; “King: Estimating Latency between Arbitrary Internet End Hosts”, SIGCOMM Internet Measurement Workshop, pp. 5-18, 2002.

[9] IETF IP Performance Metrics working group, http://www.ietf.org/html.charters/ippm-charter.html

[10] Jain, M.; “Probing for Bandwidth Measurements”, article of the Planetlab-users mailing list from May 11th, 2005.

[11] Netperf homepage, http://www.netperf.org/netperf/NetperfPage.html, visited Jan. 2011.

[12] Pásztor, A. and Veitch, D., “PC Based Precision Timing without GPS”, Proceedings of the ACM SIGMETRICS international Conference on Measurement and Modeling of Computer Systems, Marina Del Rey, pp. 1-10, June, 2002.

[13] Paxson, V., Almes, G., Mahdavi, J., and Mathis, M.; “Framework for IP Performance Metrics”, RFC 2330, May 1998

[14] PlanetLab, An open platform for developing, deploying, and accessing planetary-scale services, www.planet-lab.org

[15] Scharf, M.; “The Impact of Delay Variations on TCP Performance”, Proceedings of the 2nd Workshop on Modeling and Optimization in Mobile, Ad hoc and Wireless Networks (WiOpt '04), pp. 419-420, Cambridge, 2004.

[16] Spring, N., Wetherall, D., and Anderson, T.; "Scriptroute: A Public Internet Measurement Facility", USENIX Symposium on Internet Technologies and Systems (USITS), pp. 17-17, 2003.

[17] Spring, N., Peterson, L., Bavier, A., and Pai, V.; Using PlanetLab for network research: myths, realities, and best practices, SIGOPS Oper. Syst. Rev. 40(1), pp 17-24, 2006.

[18] TCPDUMP.org http://www.tcpdump.org/, visited Jan 2011.