Universität Stuttgart

INSTITUT FÜR KOMMUNIKATIONSNETZE UND RECHNERSYSTEME
Prof. Dr.-Ing. Dr. h. c. mult. P. J. Kühn

Interner Bericht / Internal Report

Nº 53

Titel / Title	**Performance Measurements of SIMCO over TCP and SCTP**
Verfasser / Author(s)	Sebastian Kiesel, Michael Scharf, Sebastian Beutel, Thomas Ruschival

Datum / Date	July 3, 2006
Umfang / Size	18 Seiten / Pages
Schlüsselworte / Keywords	Measurement, SIMCO, TCP, SCTP

Kurzfassung / Abstract

This technical report summarizes the results of the SIMCO performance measurements with different transport protocols.
SIMCO (SImple Middlebox COnfiguration) is a signaling protocol that can be used for controlling middleboxes such as firewalls and network address translators – small cable/DSL routers as well as large gateways between VoIP carrier networks, policing thousands of simultaneous calls. In the latter case, optimizing the SIMCO response time is important, as this contributes to the call setup delay perceived by the user in a negative way.
SIMCO assumes to be transmitted over a reliable transport layer protocol. However, it requires only partial in-order delivery of messages. Using a Linux and Solaris based prototype implementation and a wide area network emulator, the SIMCO transaction response time was measured both for TCP (Transmission Control Protocol) and SCTP (Stream Control Transmission Protocol) as transport layer protocols, with different parameter settings and for different workloads.
The measurement results reveal clear advantages of using SCTP, especially in scenarios with medium to high transaction rates.

1 Introduction

The SIMCO (SImple Middlebox COnfiguration) protocol [1] is a signaling protocol that implements the MIDCOM protocol semantics [2]. In the context of the MIDCOM architecture [3], it can be used for controlling middleboxes [4] such as firewalls and network address translators (NATs).

As outlined in [3], firewalls and NATs are potential obstacles to packet streams, for example if dynamically negotiated UDP or TCP port numbers are used, as in many peer-to-peer communication applications. SIMCO allows applications to communicate with middleboxes on the datagram path in order to request a dynamic configuration at the middlebox that enables datagram streams to pass the middlebox. Applications can request pinholes at firewalls and address bindings at NATs. A typical MIDCOM/SIMCO usage scenario is shown in figure 1.

When interactively establishing multimedia sessions, such as the SIP (Session Initiation Protocol [5]) based VoIP (Voice over IP) scenario shown in figure 1, optimizing the SIMCO response time is important, as this contributes to the call setup delay perceived by the user in a negative way.

SIMCO assumes to be transmitted over a reliable transport layer protocol. The SIMCO specification [1] mandates TCP (Transmission Control Protocol) [6] as the default transport for SIMCO. However, SIMCO requires only partial in-order delivery of messages: all SIMCO messages referring to firewall rules related to a specific VoIP call have to be delivered in sequence, whereas messages related to other calls can be transmitted independently. TCP ensures complete in-order delivery and therefore fulfills this requirement, but may cause unnecessary delays due to so-called head-of-line-blocking in case of packet losses and retransmissions in the network.

The Stream Control Transmission Protocol (SCTP) [7] has originally been designed as a part of the SIGTRAN architecture [8], for the transport of Signaling System No. 7 (SS7) messages over IP. However, this rather special purpose is achieved by adaptation layers on top of SCTP. SCTP itself has been designed as a generic transport protocol for IP networks, at the same layer in the protocol stack as TCP or UDP. SCTP offers several advantages compared to TCP for the transport of signaling protocols, especially in scenarios with high reliability requirements or high signaling traffic between two endpoints. In particular, the negative effect of head-of-line-blocking can be reduced by distributing the application layer messages over several so-called streams within one SCTP association, each having its own resequencing queue.

How to transport SIMCO over SCTP and in particular how to leverage SCTP's multiple streams feature has been specified in [9]. This technical report summarizes the results of the SIMCO performance measurements carried out at the IKR network laboratory. Using a Linux and Solaris based prototype implementation (derived and extended from [10]) and a wide area network emulator, the SIMCO transaction response time was measured both for TCP and SCTP transport layer protocols, with different parameter settings and for different workloads.

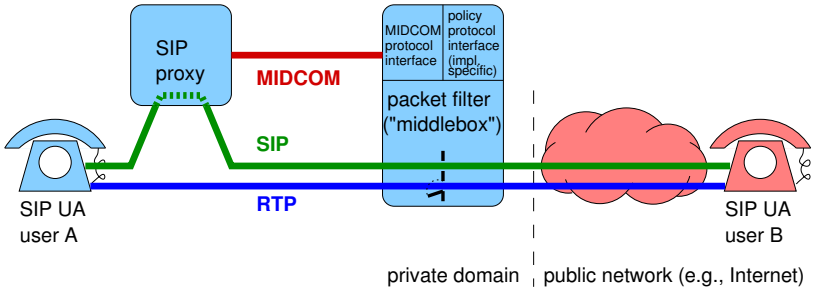

Figure 1: The MIDCOM architecture

2 Theoretical Explanation of Effects

Several effects contribute to the transaction response times presented in this report:

- Propagation delays of individual IP packets in the network between SIMCO agent and middlebox (in the following: RTT = round trip time, considered in the IP layer, assumed to be constant)
- Delays caused by retransmissions of lost packets
- Additional delays due to the behavior of the resequencing queue at the receiver in case of packet losses
- Local processing of requests, especially in the middlebox
- Delays at the sender in case of small congestion windows

In [11], we propose an analytical model that describes the additional resequencing delays when using multiple SCTP streams in ordered mode. The model is compared with measurements on the "SIMCO over SCTP" testbed. This technical report provides additional measurement results yielded from the same testbed, detailing a larger parameter space than presented in [11].

In [12], both the analytical and measurement based evaluation is extended to consider two additional configurations for signaling transport: distribution of messages over several parallel TCP connections, and SCTP using one stream in unordered mode. Adding support for parallel TCP connections to the SIMCO software would be rather complex. A completely unordered transport is not applicable for SIMCO. Therefore, all measurements presented in [12] are based on an even simpler testbed. It consists of a load generator, the WAN emulator and an echo server that simply returns the dummy messages sent by the load generator.

When using a single TCP connection or SCTP with multiple streams in ordered mode, the measurement results from the very simple testbed of [12] can be compared and are virtually identical to the SIMCO results presented in this report and in [11]. Therefore, one can conclude that local processing of SIMCO requests in the middlebox has only a low impact on the total response time for the considered parameter configuration.

3 Measurement Setup

Figure 2 shows the block diagram of the SIMCO agent (“client”) and middlebox (“server”) as they could be used in a scenario as depicted in figure 1.

3.1 SIMCO implementation

For the measurements documented in this technical report, a “SIMCO over SCTP” prototype was implemented and a testbed has been configured, as shown in figure 3. The SIP proxy has been replaced by a load generator.

The design and implementation of the testbed components (SIMCO server and load generator), as well as some first measurements are documented in [10]. The SIMCO implementation is compliant to Version 7 of the SIMCO Internet draft [13]. However, some features such as support for PDR and some policy rule sanity checks were omitted. Adding support for these features might slow down local processing of PER requests in the SIMCO server. SCTP support is compliant to [9], the outbound stream number is chosen as TID modulo number of streams [9, section 4.4.1.2.]. The Nagle algorithm for TCP and its equivalent for SCTP have been disabled (socket options TCP_NODELAY and SCTP_NODELAY). For the Linux SCTP implementation this does not make a difference; it seems that this algorithm is not yet implemented in the version used for the measurements. For the performance measurements, the interface between the SIMCO server and the kernel packet filter has been disabled, i.e., the firewall rules were not added to the kernel, in order to evaluate the transport layer performance without the impact of the packet filter API.

The SIMCO implementation used for the measurements was slightly modified compared to the final outcome of [10]. The source code was made more portable in order to compile and run under Sun Solaris ([10] supported only Linux). Furthermore, several minor bug fixes and error handlers were added to avoid software crashes under rare error conditions.

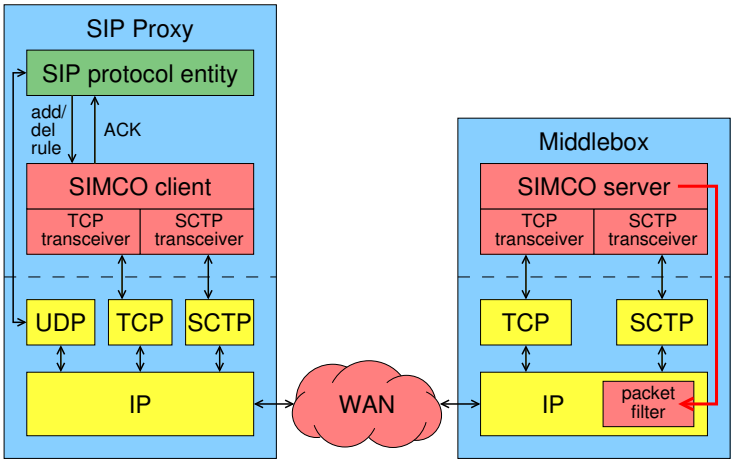

Figure 2: SIP proxy as SIMCO agent and middlebox block diagram

3.2 Operating System Software & Hardware Configuration

Both the SIMCO client (load generator) and SIMCO server (middlebox) were compiled from their C++ source code and installed on two different hardware and operating system platforms.

Linux/x86

Operating System	linux-2.6.16 / Fedora Core 3
CONFIG_HZ in Kernel	250
GCC version	3.4.3 20050227 (Red Hat 3.4.3-22.fc3)
glibc version	glibc-2.3.5-0.fc3.1
libsctp version	1.0.2
CPU	Intel Pentium 4 2.80GHz, stepping 04, hyperthreading disabled
RAM	512 MByte
Ethernet NIC	D-Link DFE-500TX (Digital DS21143 “Tulip” revision 65)

Solaris/UltraSPARC

Operating System	Sun Solaris 10 (SunOS 5.10 Generic_118833-02 sun4u sparc SUNW)
GCC version	3.4.3 (csl-sol210-3_4-branch+sol_rpath)
Server Model	Sun Netra AX 1105-500
CPU	UltraSPARC IIe, 500MHz
RAM	768 MByte

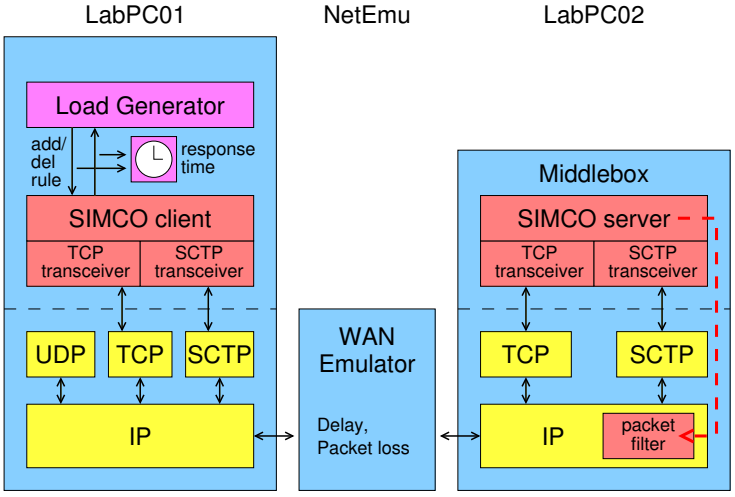

Figure 3: Testbed with load generator used for measurements

3.3 WAN Emulator & Computer Interconnection

All results presented in this report were measured using the NistNet WAN emulator. For some parameter settings, measurements were repeated using the Shunra\Cloud WAN emulator, without noticing any significant differences.

The computers were interconnected by means of Fast Ethernet hubs (100 Mbps half-duplex). No other traffic was present on these network segments during the measurements.

NistNet

WAN emulator	NistNet 3.0.alpha
Operating System	Linux 2.6.12-b5 / Ubuntu Linux 5.10
CPU	Intel Pentium 4 2.80GHz, stepping 04, hyperthreading disabled
RAM	512 MByte
Ethernet NIC	2 * D-Link DFE-500TX (Digital DS21143 “Tulip” rev. 65)

Shunra\Cloud

WAN emulator	Shunra\Cloud V4.0 Build 10149
Operating System	Microsoft Windows2000 Build 5.00.2195
CPU	2 * Intel Pentium 3 500MHz (symmetric multiprocessing)
RAM	1024 MByte
Ethernet NIC	2 * D-Link DFE-500TX (Digital DS21143 “Tulip” rev. 65)

4 Experiments 1 – 3 : Delay over Loss

4.1 Parameters for Experiments 1 – 3

Protocols:	TCP, SCTP (1, 2, 4, 8, 16, 1024 streams, ordered)
SIMCO transaction rate:	100 $\frac{1}{s}$
IP packet loss in WAN emulator:	0.0 % . . . 10.0 % per direction, applied in both directions, random distribution with indicated mean value
One-way latency in WAN emulator:	Experiment 1: 10 ms, Experiment 2: 20 ms, Experiment 3: 40 ms, constant, applied in both directions
Number of measured transactions:	Experiment 1: 80,000 per parameter combination Experiments 2 & 3: 40,000 per parameter combination

At higher packet loss probabilities, the Solaris SCTP measurements are distorted by sporadic longer stalls of the SCTP association, which seem to occur without any obvious reason. In this case, the measurements were aborted.

Error bars indicate 95% T-test confidence intervals.

4.2 Experiment 1: One-Way Latency 10 ms

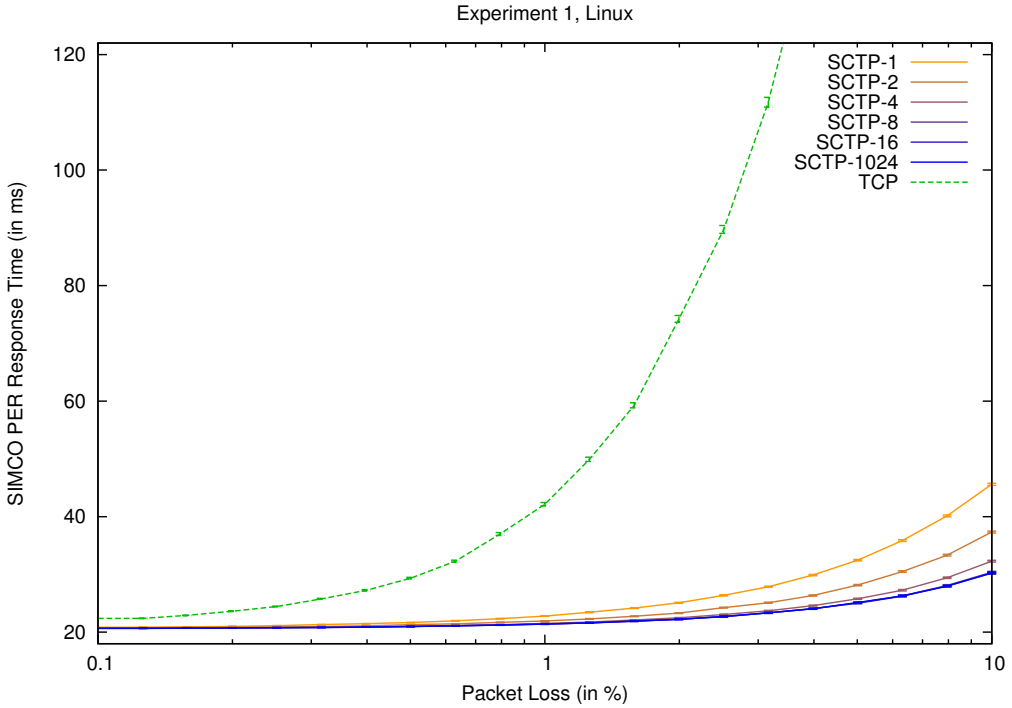

Figure 4: Experiment 1: Delay over Loss (Linux)

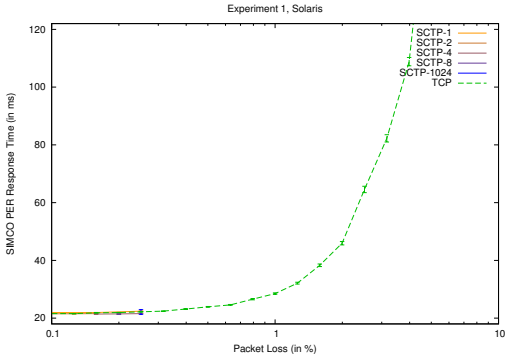

Figure 5: Experiment 1: Delay over Loss (Solaris), SCTP measured only for low loss probabilities due to problems – see text

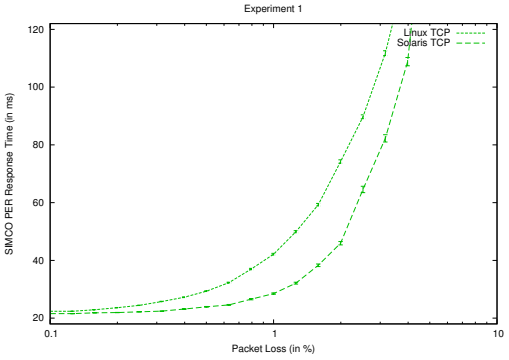

Figure 6: Experiment 1: Delay over Loss (Linux TCP vs. Solaris TCP)

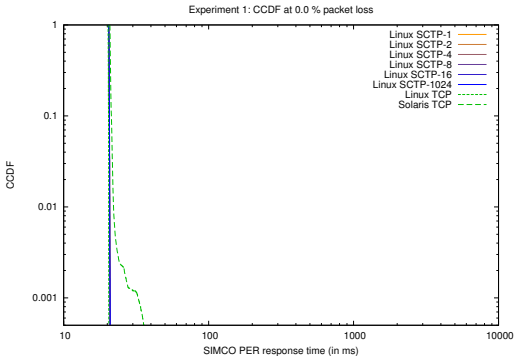

Figure 7: Experiment 1: Delay CCDF at 0.0 % packet loss

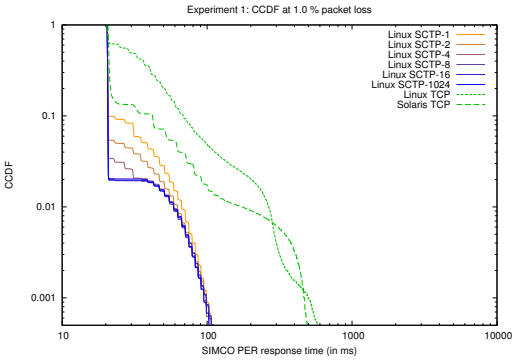

Figure 8: Experiment 1: Delay CCDF at 1.0 % packet loss

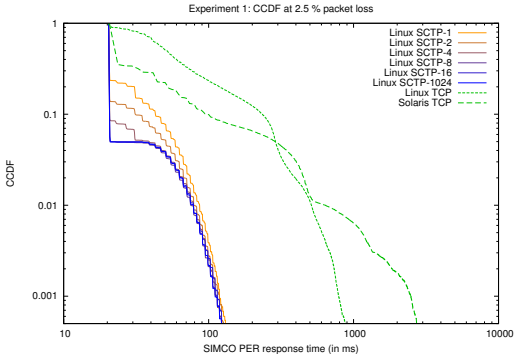

Figure 9: Experiment 1: Delay CCDF at 2.5 % packet loss

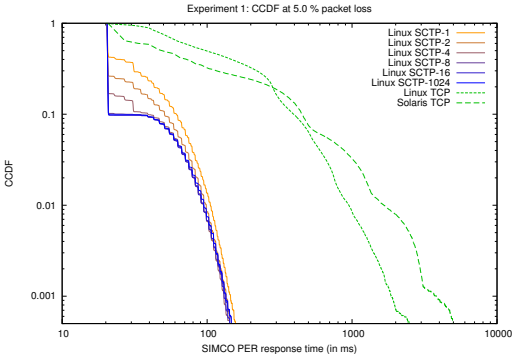

Figure 10: Experiment 1: Delay CCDF at 5.0 % packet loss

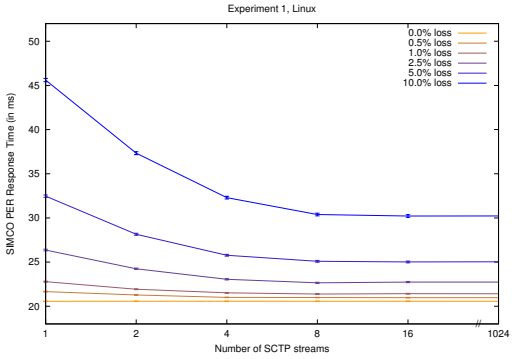

Figure 11: Experiment 1: Delay over Number of SCTP Streams (Linux)

4.3 Experiment 2: One-Way Latency 20 ms

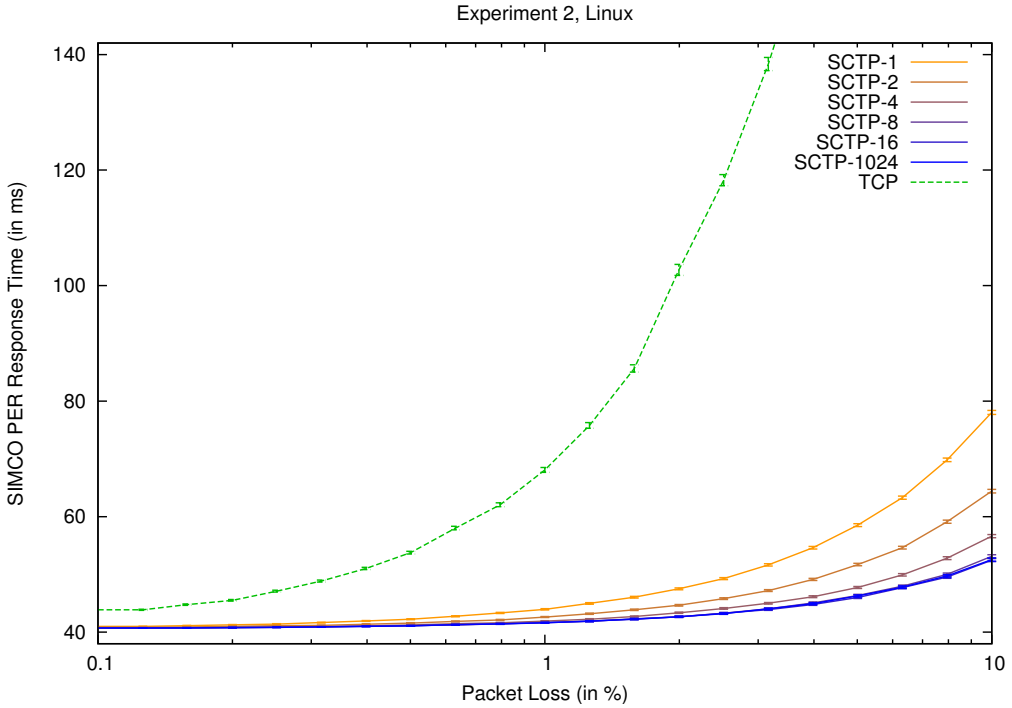

Figure 12: Experiment 2: Delay over Loss (Linux)

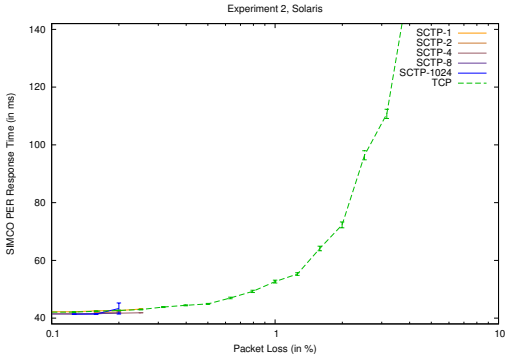

Figure 13: Experiment 2: Delay over Loss (Solaris), SCTP measured only for low loss probabilities due to problems – see text

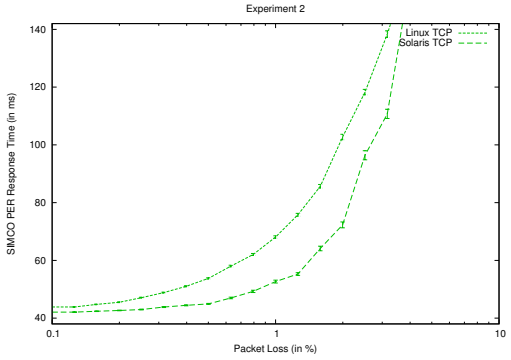

Figure 14: Experiment 2: Delay over Loss (Linux TCP vs. Solaris TCP)

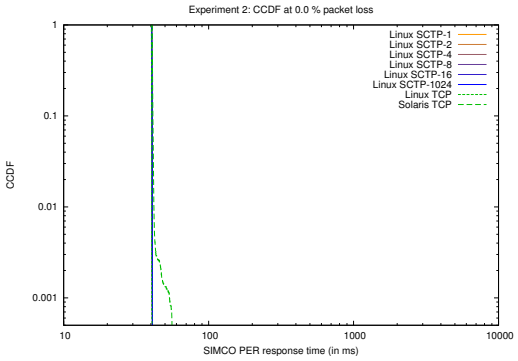

Figure 15: Experiment 2: Delay CCDF at 0.0 % packet loss

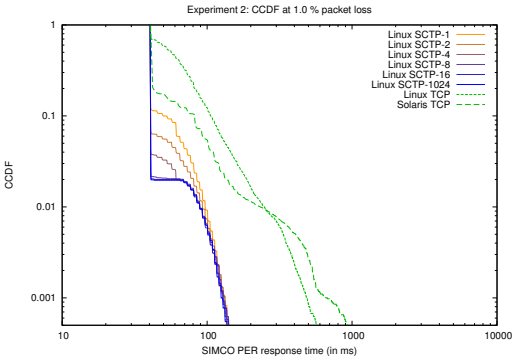

Figure 16: Experiment 2: Delay CCDF at 1.0 % packet loss

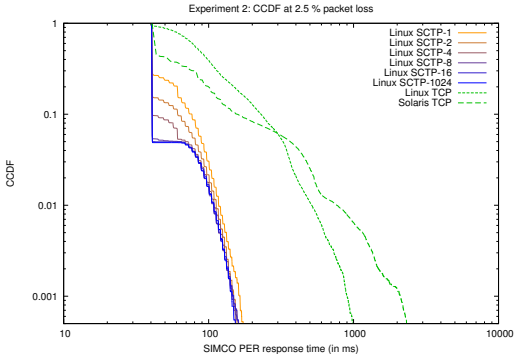

Figure 17: Experiment 2: Delay CCDF at 2.5 % packet loss

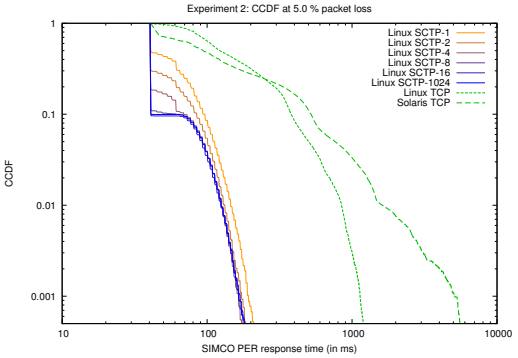

Figure 18: Experiment 2: Delay CCDF at 5.0 % packet loss

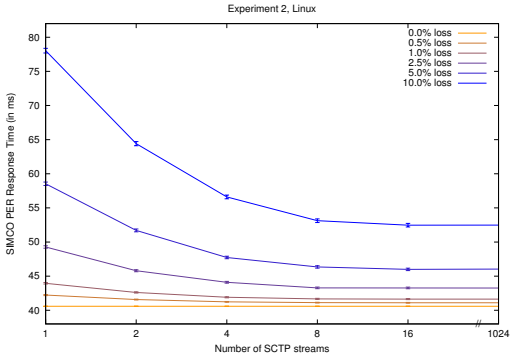

Figure 19: Experiment 2: Delay over Number of SCTP Streams (Linux)

4.4 Experiment 3: One-Way Latency 40 ms

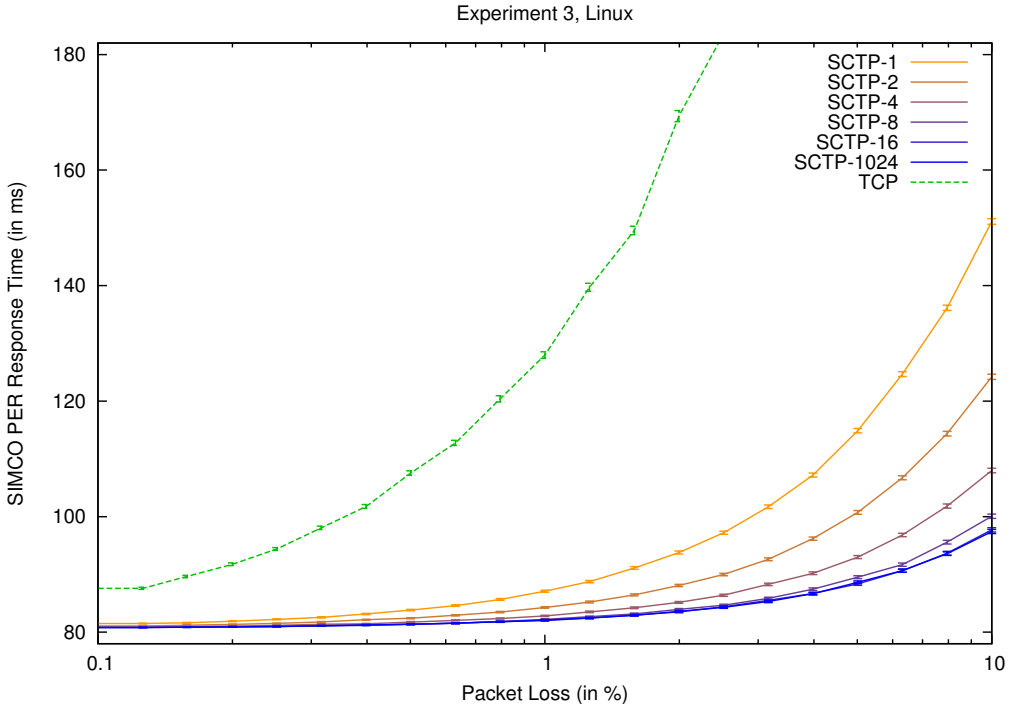

Figure 20: Experiment 3: Delay over Loss (Linux)

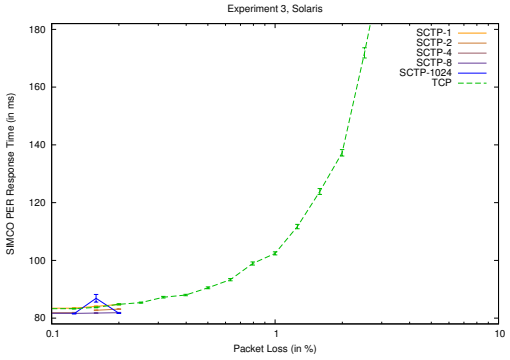

Figure 21: Experiment 3: Delay over Loss (Solaris), SCTP measured only for low loss probabilities due to problems – see text

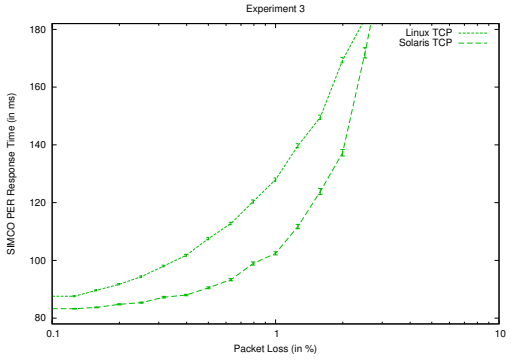

Figure 22: Experiment 3: Delay over Loss (Linux TCP vs. Solaris TCP)

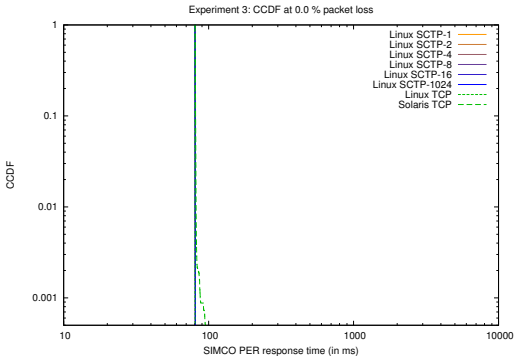

Figure 23: Experiment 3: Delay CCDF at 0.0 % packet loss

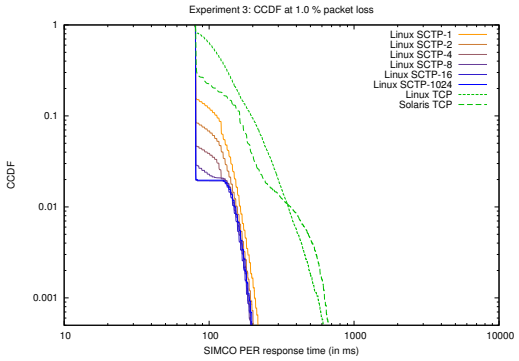

Figure 24: Experiment 3: Delay CCDF at 1.0 % packet loss

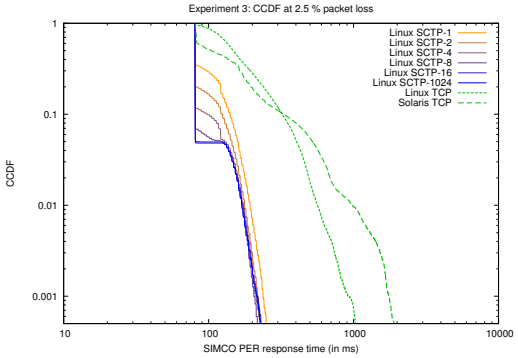

Figure 25: Experiment 3: Delay CCDF at 2.5 % packet loss

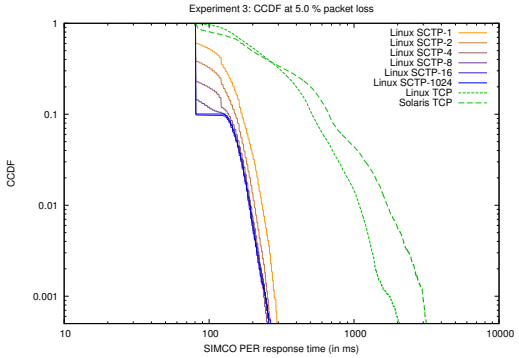

Figure 26: Experiment 3: Delay CCDF at 5.0 % packet loss

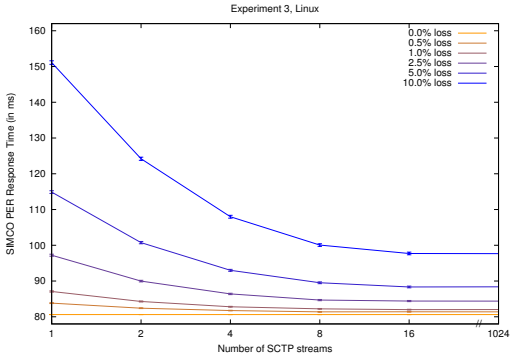

Figure 27: Experiment 3: Delay over Number of SCTP Streams (Linux)

5 Experiments 4 – 6 : Delay over Load

5.1 Parameters for Experiments 4 – 6

Protocols:	TCP, SCTP (1, 2, 4, 1024 streams, ordered)
SIMCO transaction rate:	$0.1\ \frac{1}{s} \ldots 10{,}000\ \frac{1}{s}$
IP packet loss in WAN emulator:	Experiment 4: 0.0 % Experiment 5: 1.0 % Experiment 6: 2.0 % per direction, applied in both directions, random distribution with indicated mean value
One-way latency in WAN emulator:	10 ms constant, applied in both directions

Number of measured transactions per parameter combination:

Transaction rate	Experiment-4	Experiment-5	Experiment-6
$\geq 0.1\ \frac{1}{s}$	≥ 100	≥ 1000	-
$\geq 1\ \frac{1}{s}$	≥ 400	$\geq 10{,}000$	-
$\geq 10\ \frac{1}{s}$	$\geq 4{,}000$	$\geq 100{,}000$	$\geq 4{,}000$
$\geq 100\ \frac{1}{s}$	$\geq 40{,}000$	$\geq 100{,}000$	$\geq 40{,}000$

As very long measurement durations are required to produce statistically significant results for low transaction rates, only Linux TCP, Solaris TCP and Linux SCTP with 1024 streams were measured only in measurements 4 and 5 for transaction rates lower than $10\ \frac{1}{s}$. In each measurement the sending rate was increased until the load generator could not generate the desired rate during the whole measurement duration. That is, the measurement was aborted for one protocol if the load generator could not pass messages to TCP or SCTP within a time threshold of 1 s after the scheduled sending time, e. g., because even repeated calls of the *send()* system call failed due to full socket buffers.

Error bars indicate 95% T-test confidence intervals.

5.2 Experiment 4: Packet Loss 0.0 %

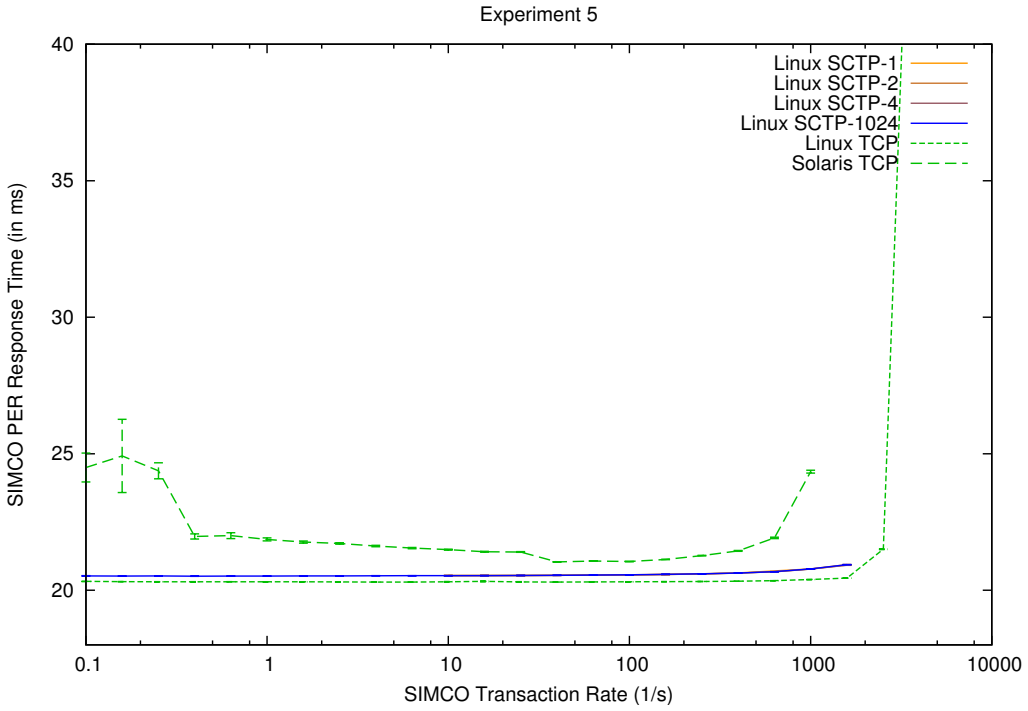

Figure 28: Experiment 4: Delay over Load

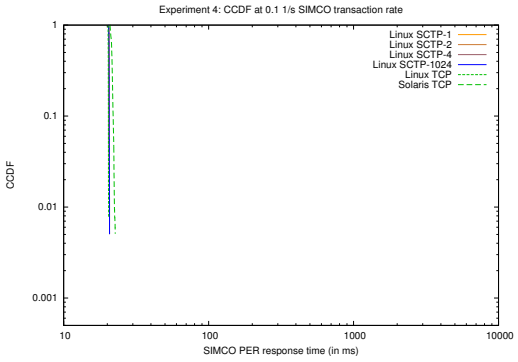

Figure 29: Experiment 4: Delay CCDF at 10 $\frac{1}{s}$ SIMCO transaction rate

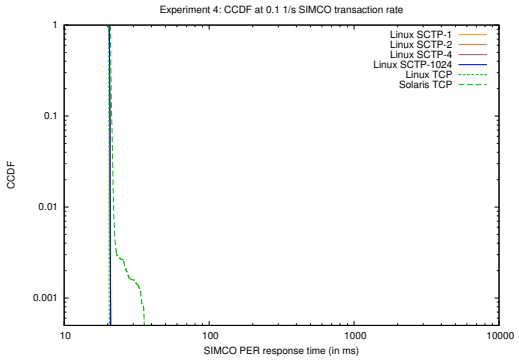

Figure 30: Experiment 4: Delay CCDF at 100 $\frac{1}{s}$ SIMCO transaction rate

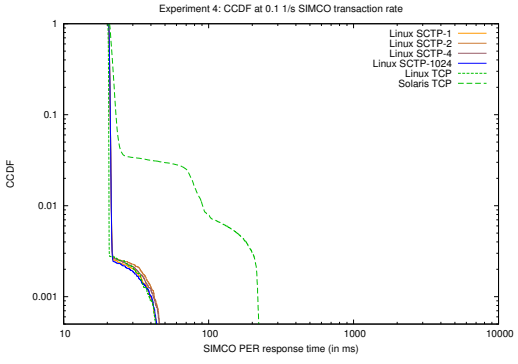

Figure 31: Experiment 4: Delay CCDF at 1000 $\frac{1}{s}$ SIMCO transaction rate

5.3 Experiment 5: Packet Loss 1.0 %

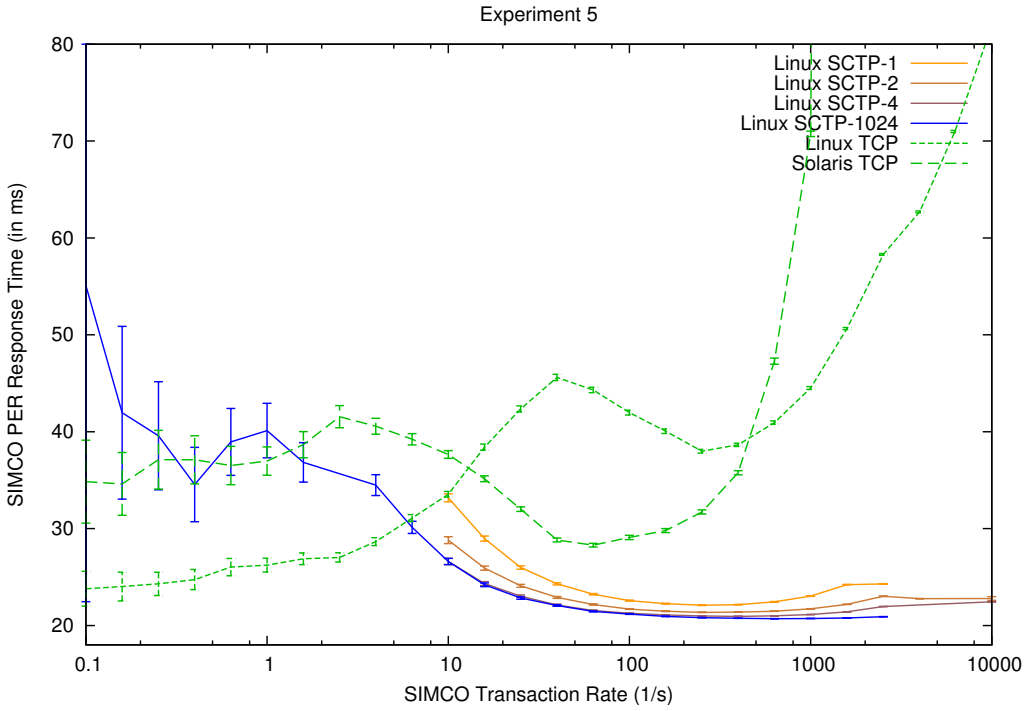

Figure 32: Experiment 5: Delay over Load

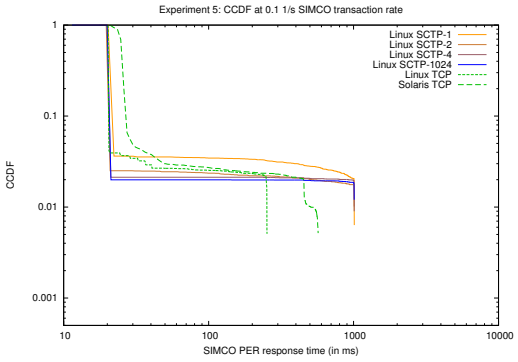

Figure 33: Experiment 5: Delay CCDF at 1.0 $\frac{1}{s}$ SIMCO transaction rate

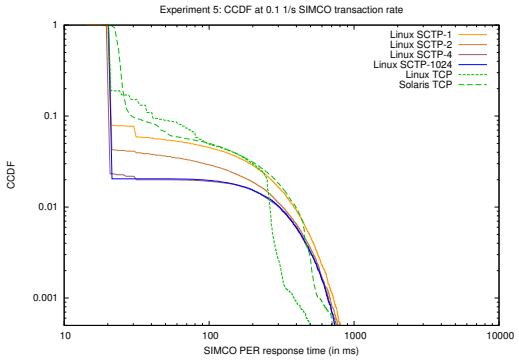

Figure 34: Experiment 5: Delay CCDF at 10 $\frac{1}{s}$ SIMCO transaction rate

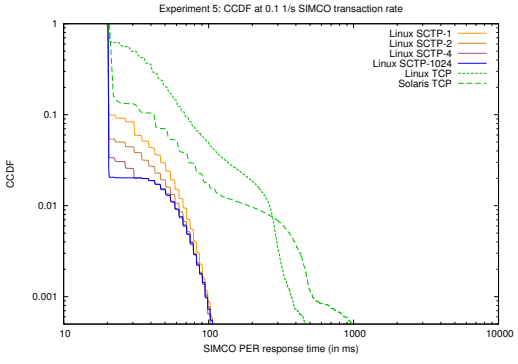

Figure 35: Experiment 5: Delay CCDF at 100 $\frac{1}{s}$ SIMCO transaction rate

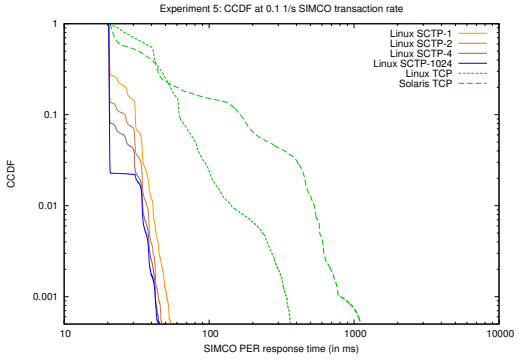

Figure 36: Experiment 5: Delay CCDF at 1000 $\frac{1}{s}$ SIMCO transaction rate

5.4 Experiment 6: Packet Loss 2.0 %

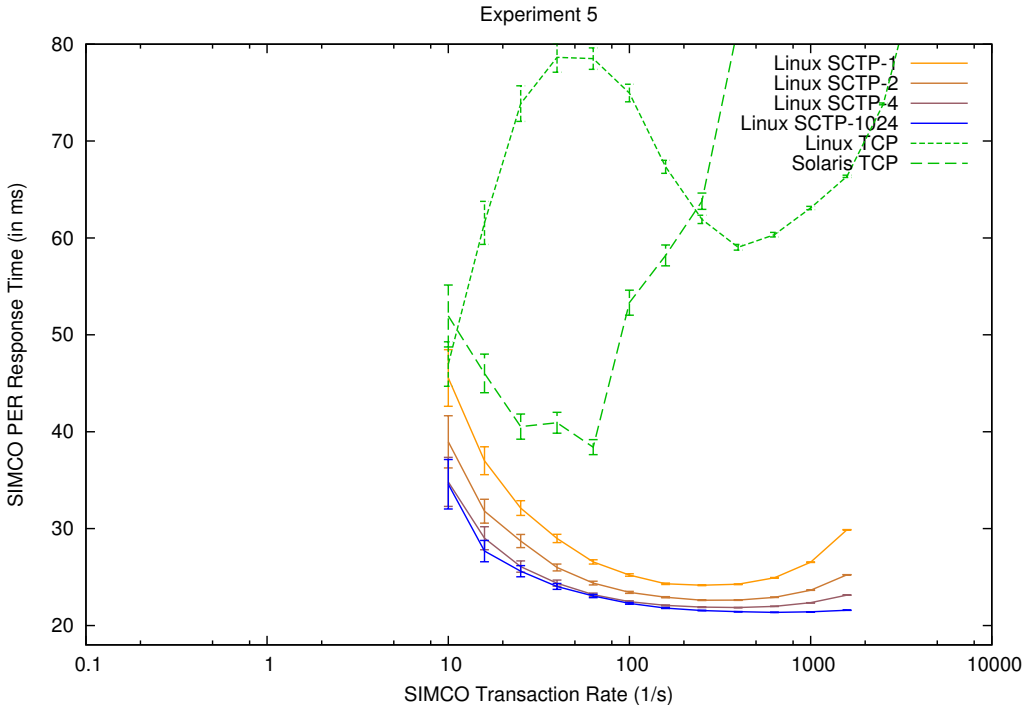

Figure 37: Experiment 6: Delay over Load

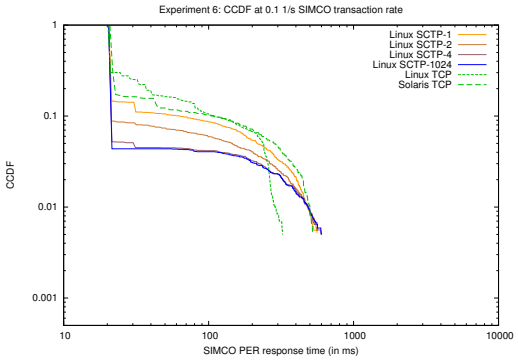

Figure 38: Experiment 6: Delay CCDF at 10 $\frac{1}{s}$ SIMCO transaction rate

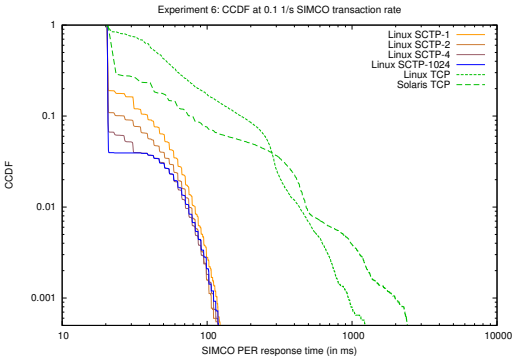

Figure 39: Experiment 6: Delay CCDF at 100 $\frac{1}{s}$ SIMCO transaction rate

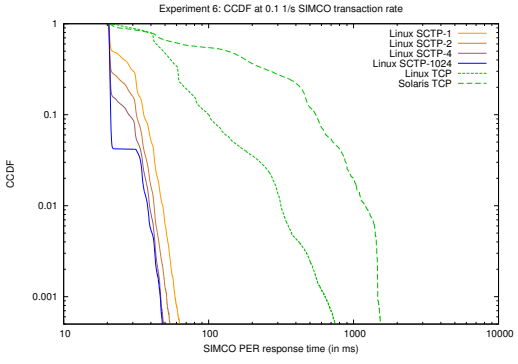

Figure 40: Experiment 6: Delay CCDF at 1000 $\frac{1}{s}$ SIMCO transaction rate

6 Observations and Conclusions

- Except for very low transaction rates, SIMCO transaction delays were significantly lower when using SCTP instead of TCP transport, for all considered delay and packet loss combinations and numbers of SCTP streams.
- Even for moderate packet loss probabilities TCP suffers from rather large end-to-end delays and a significant quantile of the transactions exceed the average delay by far.
- In general, increasing the number of streams within an SCTP association reduces transmission delays by alleviating the effect of head-of-line blocking. For the considered parameter space a rather small number of streams is sufficient – further increasing the number of streams does not yield even lower delays.
- There are significant differences when using an SCTP association with only one stream per direction and ordered transport instead of a TCP connection. Except for very low transaction rates delays are lower when using SCTP.
- There are clearly noticeable differences in the behavior of the TCP implementations of Linux and Sun Solaris.
- From an implementation point of view it is worth noticing that the SCTP-specific parts of the SIMCO client and server source code are shorter and simpler than the TCP-specific parts, because of the message based SCTP API as opposed to TCP's byte stream based API.

References

[1] M. Stiemerling, J. Quittek, and C. Cadar, "NEC's Simple Middlebox Configuration (SIMCO) Protocol Version 3.0," IETF, RFC 4540, May 2006.

[2] M. Stiemerling, J. Quittek, and T. Taylor, "Middlebox Communications (MIDCOM) Protocol Semantics," IETF, RFC 3989, Feb. 2005.

[3] P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, and A. Rayhan, "Middlebox communication architecture and framework," IETF, RFC 3303, Aug. 2002.

[4] B. Carpenter and S. Brim, "Middleboxes: Taxonomy and Issues," IETF, RFC 3234, Feb. 2002.

[5] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," IETF, RFC 3261, June 2002.

[6] J. Postel, "Transmission Control Protocol," IETF, RFC 793, Sept. 1981.

[7] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, "Stream Control Transmission Protocol," IETF, RFC 2960, Oct. 2000.

[8] L. Ong, I. Rytina, M. Garcia, H. Schwarzbauer, L. Coene, H. Lin, I. Juhasz, M. Holdrege, and C. Sharp, "Framework Architecture for Signaling Transport," IETF, RFC 2719, Oct. 1999.

[9] S. Kiesel, "SIMCO over SCTP," IETF, draft-kiesel-midcom-simco-sctp-01, IETF draft - work in progress, Apr. 2006.

[10] C. Blankenhorn, "Evaluation of SCTP as transport protocol for transaction-based applications at the example of a protocol for firewall control," Student project (in German), IKR, University of Stuttgart, 2005.

[11] S. Kiesel and M. Scharf, "Modeling and performance evaluation of SCTP as transport protocol for firewall control," in *Proc. IFIP-TC6 Networking Conference, Springer LNCS*, Coimbra, Portugal, May 2006.

[12] M. Scharf and S. Kiesel, "Head-of-line blocking in TCP and SCTP: Analysis and measurements," in *Proc. IEEE Globecom 2006*, San Francisco, USA, Nov. 2006.

[13] M. Stiemerling, "Simple Middlebox Configuration (SIMCO) Protocol Version 3.0," IETF, draft-stiemerling-midcom-simco-07, IETF draft - work in progress, May 2005.