Head-of-line Blocking in TCP and SCTP: Analysis and Measurements

Michael Scharf and Sebastian Kiesel
Institute of Communication Networks and Computer Engineering, University of Stuttgart, Germany
Email: {scharf,kiesel}@ikr.uni-stuttgart.de

***Abstract*— Many signaling protocols in IP networks need a protection against message loss, but they do not require a strict in-sequence data delivery. Since TCP provides reliable in-order transport, end-to-end delays may be unnecessarily increased due to head-of-line blocking. An alternative transport protocol is SCTP, which is optimized for signaling applications and provides mechanisms for reliable, partial-ordered or unordered message delivery. In this paper, we quantify the impact of head-of-line blocking on the response time of transaction-based signaling applications. In order to mitigate this problem, we compare different solutions based on TCP and SCTP. Both a new analytical model and measurements on state-of-the-art operating systems show to which extend SCTP can improve transport delays by leveraging transmission over multiple parallel streams or using unordered data delivery. Our analysis reveals that using one or multiple parallel TCP connections can result in much higher end-to-end delays, even for moderate packet loss probabilities. We also observe significant differences in the TCP performance of different operating systems.**

I. Introduction

"Next Generation Networks" (NGN), such as the "IP Multimedia Subsystem" (IMS) standardized by 3GPP, differ from traditional IP networks by deploying stateful application layer entities (e. g., softswitches) in the core network, which interact by different signaling protocols: The Session Initiation Protocol (SIP) is used for session control, and DIAMETER for authentication, authorization, and accounting. Furthermore, session border controllers (SBC) and middleboxes such as network address translators and firewalls must be controlled, e. g., by MEGACO/H.248, NSIS, or the Simple Middlebox Configuration Protocol (SIMCO). These signaling protocols have quite stringent delay requirements because transaction delay may contribute to call setup delays perceived by users.

The Transmission Control Protocol (TCP) is the default choice for reliable transport in the Internet. Since TCP ensures reliable in-order delivery, end-to-end delays are increased due to head-of-line blocking when IP packets get lost. This effect is particularly critical on links with high data rates, e. g., between large softswitches and other central NGN entities.

The Stream Control Transmission Protocol (SCTP), which has been developed as a transport layer protocol especially for signaling applications, addresses this problem by allowing unordered delivery of messages to the application. Furthermore, SCTP offers a partial-ordered service, i. e., preservation of sequence will be guaranteed only for subsets of all messages. In addition to mechanisms to mitigate the delaying effect of head-of-line blocking, SCTP also provides functions for environments with high reliability and security requirements. Therefore, SCTP is proposed as an alternative to TCP for many NGN signaling protocols, such as DIAMETER [1], H.248 [2], NSIS [3], or SIMCO [4]. SIP can benefit from SCTP, too [5].

While the reduction of head-of-line blocking is a well-known feature of SCTP, there are only few studies that quantify the impact of this effect on end-to-end delays. To the best of our knowledge, existing works either use simulations or measurements only, and a detailed analytical study has not been published so far. In this paper, we analyze the response time of transaction-based signaling protocols over TCP and SCTP. Potential alternatives to TCP for improving delays are multiple parallel TCP connections, multiple SCTP streams, and SCTP unordered mode. We compare these approaches and quantify the corresponding response times when facing IP packet loss, both by an analytical model and by measurements on different operating systems.

The remainder of the paper is organized as follows. In Section II, we discuss how signaling applications can leverage the protocol mechanisms provided by SCTP or parallel TCP connections. Section III presents an analytical model for reseqeuencing delays. In Section IV, we present measurement results for end-to-end delays and compare them to the analytical approximations. Finally, Section V concludes this paper.

II. Head-of-Line-Blocking in Transport Protocols

A. Message Delivery Modes

The classical Internet transport protocols either provide an ordered reliable service (TCP) or one that does not guarantee any ordered delivery and which is not reliable (UDP). However, reliability (i. e., protection against message loss) and ordered delivery (i. e., passing the messages to the receiving application in the same sequence as they were sent by the sender) are orthogonal issues [6]. Many signaling applications have high reliability requirements and thus need a reliable transport protocol, but they do not require an ordered delivery of signaling messages. With respect to ordering, the requirements of applications can be subdivided into three classes [6]: (1) *ordered*, (2) *partial-ordered*, or (3) *unordered* delivery. In the second case, the transport protocol must preserve only the ordering relation for subsets of all messages.

Head-of-line blocking can occur when transport protocols offer ordered or partial-ordered service: If IP packets get lost, subsequent messages have to wait for the successful retransmission in the receiver queue and are thus delayed.

Due to real-time requirements, resequencing delays are critical for signaling protocols. Of course, packet loss probabilities in signaling networks are usually small. However, on signaling links with a high amount of traffic, many messages may be affected even by rare packet loss. In the following, we discuss three different alternatives to the usage of one TCP connection, which is the default choice for many signaling protocols. We do not detail using UDP, since this requires an application-level error recovery, flow control, etc., which can hardly be realized as efficient as in the transport layer [5].

B. Multiple TCP Connections

A straightforward solution to reduce head-of-line blocking is to use several parallel TCP connections between the same two end systems. If one connection is subject to head-of-line blocking, other connections can still deliver messages. For partial-ordered delivery, all messages that are in a causal relationship have to be transmitted via the same connection.

This solution does not require a new transport protocol. However, it has several drawbacks compared to SCTP, which will be considered below: First, there is more overhead, as each TCP connection must be established, maintained, and closed separately. Second, due to the stream-oriented nature of TCP, the receiving application must be prepared to receive only a part of a message. The handling of the required buffers is more complex for parallel connections. And third, each TCP connections has to recover from packet loss independently, whereas SCTP applies error recovery and congestion control mechanisms to the aggregated message flow. We will show in the next sections that the latter strategy can improve delays.

C. SCTP Multistreaming

SCTP is a message-oriented, general purpose transport protocol optimized for signaling transport. It allows to split one association into up to 65536 logical subchannels per direction, so-called streams. Each user message is transmitted in one of these streams, and SCTP ensures in-order delivery only within the same stream. This is similar to using several parallel TCP connections, but avoids the drawbacks mentioned above.

An example for an application using partial-ordered transport with SCTP is the dynamic control of firewalls by SIMCO [7]. This protocol allows to establish “policy rules” in the firewall, e. g., for allowing the media streams of a VoIP call to pass the firewall. As specified in [4], signaling messages that create a new firewall policy rule are distributed evenly over several SCTP streams, e. g., by means of a round robin strategy. In contrast, messages that modify or delete an existing policy rule are sent via the same stream as the initial message that established this policy rule, in order to retain causality.

D. SCTP Unordered Transport

When sending messages in unordered mode, SCTP offers reliable transport, but delivers messages to the upper layer protocol as they arrive. This solution completely avoids head-of-line blocking. However, the upper layer protocol must have own mechanisms to deal with potentially reordered messages. This mode of operation is used, e. g., for SIP over SCTP [5].

E. Related Work

Resequencing delays are a well-known effect and sophisticated theoretical models for automatic repeat request (ARQ) protocols have been developed (see, e. g., [8]). However, to the best of our knowledge, existing analytical models do not consider the specific algorithms used by TCP and SCTP, such as the fast retransmit mechanism. Several simulation-based studies have compared end-to-end delays of TCP and SCTP, but their results are ambiguous: [9] compares the delay of SIP messages transported over UDP, TCP, or SCTP, respectively. In the latter case, only one stream with reliable unordered service is used. The authors conclude that for packet loss probabilities smaller than $0.3\,\%$ head-of-line blocking in TCP does not introduce a significant performance decrease compared to SCTP. A similar work [10] finds no significant differences between SCTP and TCP with selective acknowledgments. However, these simulation results contradict recent measurements based on FreeBSD [11]. A drawback of these papers is that they do not provide a detailed explanation of the observed effects, and they do not consider multiple SCTP streams. In [7], we propose an analytical model for the transport of SIMCO over SCTP that quantifies how transmission over multiple ordered SCTP streams can reduce resequencing delays. In the following, we extend our analytical model from [7] to signaling transport over TCP.

III. Approximation of Resequencing Delays

A. Model Assumptions

In this section, we analyze the effect of head-of-line blocking for different transport layer solutions. We consider partial-ordered data transmission over $N \geq 1$ SCTP streams, SCTP unordered mode and the usage of one or several TCP connections. We assume that the path between the two endpoints has a constant unidirectional delay of Δ and thus a minimum round-trip time $RTT = 2\,\Delta$. The path is supposed to suffer from symmetric random packet losses with loss probability p, which may be caused by congestion or transmission errors.

For simplicity, signaling messages are supposed to be sent with constant interarrival time d. Under the assumption that the total traffic $\lambda = 1/d$ is equally distributed over the N SCTP streams or TCP connections, the load on each of them is $\lambda_{\text{eff}} = \lambda/N$. Furthermore, we assume that the sender window does not restrict the amount of data that can be sent. As shown later, this is a reasonable assumption as long as p is small.

B. Error Detection in TCP and SCTP

TCP as well as SCTP can recover from packet loss by two mechanisms, which are used in a similar way in both protocols: The *fast retransmit* and the *retransmission timeout*. In the following, we explain their impact on end-to-end delays at the example of SCTP.

An SCTP endpoint can detect packet loss if transmission sequence numbers (TSNs) are missing in the selective acknowledgments (SACKs). A SACK, which is sent upon the reception of a data chunk on any stream, contains missing TSN reports for all streams. An SCTP endpoint retransmits

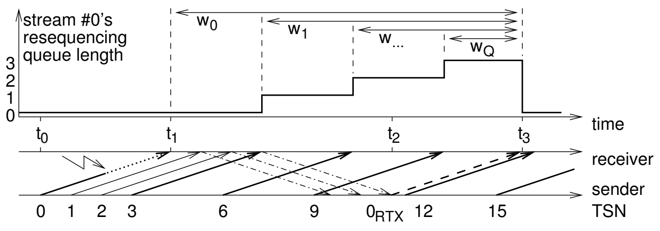

Fig. 1. Illustration of resequencing delays for 3 SCTP streams

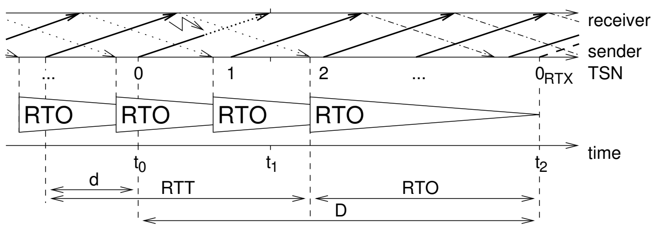

Fig. 2. Timeout-based error recovery (1 SCTP stream)

data when three subsequent SACKs include a missing report. This SCTP error recovery by a fast retransmit is illustrated in Fig. 1. For this figure, we assume that the SCTP association has $N = 3$ streams and a round robin scheduling strategy is applied, i. e., the data chunks with transmission sequence numbers 0, 3, 6, … are sent via stream #0. In this example, the data chunk with TSN 0 is lost, which is sent at t_0. At $t_2 = t_0 + RTT + 3\,d$ the sender has received 3 SACK chunks with missing reports and performs the retransmission. The time to detect the lost packet follows as $D_{\text{FXMT}} = t_2 - t_0 = RTT + 3\,d$.

The reliable data delivery is also ensured by a timeout mechanism: If the the oldest outstanding data chunk has not been acknowledged when the retransmission timeout expires, missing chunks are retransmitted. As depicted in Fig. 2, the timer is restarted whenever a new acknowledgment arrives. Thus, the error detection time is $D_{\text{RTO}} = RTO + \max(RTT - d, 0)$.

Considering both mechanisms, the error detection time is

$$D = \min(RTT + 3\,d, RTO + \max(RTT - d, 0)) \ . \quad (1)$$

However, this expression is an approximation only since more than one packet, the retransmission, or acknowledgments may get lost, too. This may trigger overlapping fast recovery periods or more complex retransmission scenarios, which are difficult to describe by a simple model. We neglect these effects since they hardly occur for small loss probabilities.

C. SCTP Multistreaming

As shown in Fig. 1, data chunks have to wait in stream #0's resequencing queue until the retransmission arrives at the receiver at $t_3 = t_2 + RTT/2$, and the waiting times w_n depend on the time $D = w_0$ to detect the packet loss. The number of data chunks that have to be queued until the retransmission arrives is $Q = \lfloor \frac{D}{d\,N} \rfloor$. The resequencing delay of the first data chunk after the lost one is $w_1 = D - N\,d$. The subsequent waiting times are $w_2 = D - 2\,N\,d, \ldots, w_Q = D - Q\,N\,d$. The mean waiting time is the sum of all w_i divided by the mean number of data chunks between two losses, which is $1/p$. The mean increase of the unidirectional end-to-end delay is thus

$$W = p \sum_{i=0}^{Q} w_i = p\left((Q+1) \cdot D - \tfrac{Q\,(Q+1)}{2}\,N\,d\right) \ . \quad (2)$$

D. SCTP Unordered Transport

Messages with the unordered flag set can be delivered to the upper layer protocol as they arrive. If all messages are sent in this mode, i. e., $w_0 = D$, $w_1 = w_2 = \cdots = 0$, eq. (2) yields

$$W = p \cdot D \ . \quad (3)$$

The same result applies for partial-ordered transport over a sufficiently large number of streams, i. e., if the stream to be used for the next message transmission has already recovered from a potential previous loss ($Q = 0$). This is approximately fulfilled for $N \geq M$ with $M = \lceil \frac{D}{d} \rceil$. In both cases head-of-line bocking can be avoided completely.

E. Multiple TCP Connections

From a theoretical point of view, the resequencing delay of one SCTP stream and one TCP connection should be identical, since both use similar error recovery algorithms. If several parallel TCP connections are used, each connection has an independent error recovery, i. e., acknowledgments only refer to data transmitted over the same connection. As a result, the time to trigger a fast retransmit is $D = RTT + 3\,d\,N$ and increases with the number of connections N. The mean waiting time can obtained by substituting D in (2) with

$$D = \min(RTT + 3\,d\,N, RTO + \max(RTT - N\,d, 0)) \ . \quad (4)$$

F. End-to-end Delays

In the considered scenario, the mean unidirectional end-to-end delay is $RTT/2 + W$. Many signaling protocols are transaction-based and require a reply for each signaling message. As a consequence, the important metric is the bidirectional response time (see, e. g., [7]). Furthermore, response times are easy to measure. Since head-of-line blocking may occur in both directions, the mean response time follows as

$$R = RTT + 2\,W + \delta \ , \quad (5)$$

where δ represents the processing time in the end systems.

IV. Analytical and Measurement Results

A. Measurement Setup

In order to measure transport layer delays, we set up the testbed depicted in Fig. 3. The software emulates a transaction-based signaling protocol and consists of a load generator and an echo server. Both were designed to be as simple as possible to isolate delays caused by the transport protocol from local processing effects. The load generator generates messages according to a given interarrival time (IAT) distribution, each having a constant size (64 octets). Using a round robin strategy,

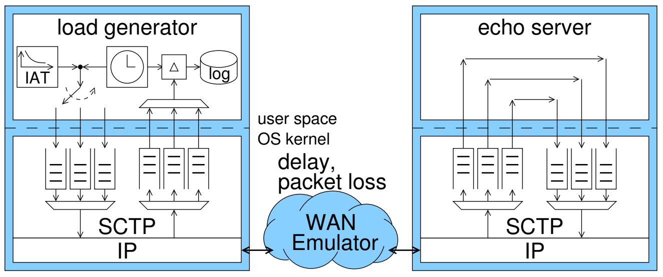

Fig. 3. Measurement setup: Load generator, WAN emulator and echo server

messages are distributed over a configurable number of parallel TCP connections or streams within one SCTP association. The echo server waits until it has received a complete message and returns it on the same connection or stream pair. Upon reception of the echoed message the load generator calculates the response time and writes it to a log file.

The software is implemented in C and runs under Linux (kernel version 2.6.15) or Solaris 10. The Linux variant either uses the "lksctp"-kernel module or standard Linux TCP. For both protocols the "nodelay" socket options have been enabled. It should be noted that the SCTP-specific parts of the source code are much simpler because of SCTP's message oriented API. For Solaris, we only present the TCP results. Our Solaris SCTP measurements are distorted by sporadic longer stalls of the SCTP association, which seem to occur without any obvious reason. Measurements were made using two 2.4 GHz Pentium 4 or 500 MHz UltraSPARC IIe computers connected by 100 Mbps Ethernet to a "NIST Net" network emulator, which adds a delay of $\Delta = 10\,\text{ms}$ in each direction and randomly drops IP packets with a given probability p.

B. Impact of Packet Loss on SCTP

Fig. 4 shows the mean response time as a function of the packet loss probability p, when SCTP is used as transport protocol. The load generator is configured to send requests with neg.-exp. IAT $d = 10\,\text{ms}$, corresponding to $\lambda = 100\,1/\text{s}$. As in all other diagrams, the values have been obtained by averaging over the transaction response time during a measurement period of at least 1000 s. Fig. 4 reveals that using more than one stream can significantly improve the response time R even for moderate loss probabilities such as $p = 2\,\%$. The difference gets larger for higher p, but such situations will hardly occur in well-dimensioned signaling networks.

The measurement results match very well the response time predicted by the analytical model in eq. (5), with a processing delay assumed to be $\delta = 0.5\,\text{ms}$. The slight underestimation for $p > 1\,\%$ is probably due to the impact of multiple fast retransmits and timeouts that cannot be neglected for large p. The cumulative complementary distribution function (CCDF) for $p = 1\,\%$ is depicted in Fig. 6. As the number of streams N gets larger, the CCDF asymptotically approaches the case of unordered delivery. When N is larger than a certain value (here: $M = 6$), unordered delivery and multiple steams have the same performance. Furthermore, small steps at $R = 30\,\text{ms}$ can be observed. Looking at traces reveals that this additional delay of 10 ms is caused by sporadic bundling of data chunks: When two subsequent chunks are assembled to one packet, the first one is delayed by one IAT, i. e., in the mean by $d = 10\,\text{ms}$.

C. Impact of Packet Loss on TCP

Fig. 5 presents measurements both for Linux and Solaris TCP, with the same testbed setup as for SCTP. Unlike one might expect, the mean response time for TCP is larger than for a single SCTP stream and differs from our analytical model. For $p < 1\,\%$, the difference is small, but in particular the Linux response time is larger than predicted. Both TCP implementations are not able to transport the offered load of 100 messages/s for $p > 7\,\%$, which is manifested by socket buffer overflows. This can be explained by the TCP congestion control that limits the throughput when losses occur.

The CCDF in Fig. 6 reveals a non-negligible probability for high delays (for $p = 1\,\%$). The 99 % quantile of the response time is approximately 200 ms. For Solaris TCP, there is a significant probability for delays of about 500 ms. They seem to be caused by retransmission timeouts with a minimum duration of $RTO \approx 500\,\text{ms}$. Linux TCP uses a smaller minimum value of $RTO = RTT + 200\,\text{ms}$ [12] and thus does not suffer that much from long delays. However, there is a significant probability of small delays. An analysis of traces shows that Linux sometimes does not send data immediately to the network, but aggregates them to larger packets, even though the "nodelay" option is set.

D. Variable Load

In the following, we vary the offered load λ, and we study multiple TCP connections, too. Figures 7, 8, and 9 present the mean response time for Linux SCTP, Linux TCP and Solaris TCP, respectively. The TCP graphs are plotted as a function of the data rate $\lambda_{\text{eff}} = \frac{\lambda}{N}$ per TCP connection. As to be expected, multiplexing the total data rate of λ over N TCP connections results in the same delay as sending $\frac{\lambda}{N}$ over one connection, since different TCP connections operate independently. All three diagrams can be subdivided into three regions:

(1) For small data rates, the response time is increased by the RTO, which expires before a fast retransmit is triggered. Here, SCTP performs worst due to its high minimum RTO of 1 s. As λ increases, R increases due to head-of-line blocking, except for a sufficiently large number of SCTP streams.

(2) For $\lambda > \alpha = \frac{3}{RTO}$ (in case of SCTP) or $\lambda_{\text{eff}} > \alpha = \frac{3}{RTO}$ (for TCP), the fast retransmit allows a faster error recovery and reduces the end-to-end delay significantly. For Linux SCTP and Solaris TCP, the measured response time is rather close to the value predicted by our analytical model, up to a message rate of about $100\,\frac{1}{\text{s}}$. For Linux TCP, again an additional delay of about 10 ms can be observed. The TCP delays are approximately minimal at $\lambda_{\text{eff}} \approx \mu = \frac{2\sqrt{3}}{RTT}$.

(3) The behavior at high data rates differs significantly: The Linux SCTP delay is close to the RTT when a sufficient number of streams or unordered mode is used. The maximum data rate that we could transport with the "experimental" Linux

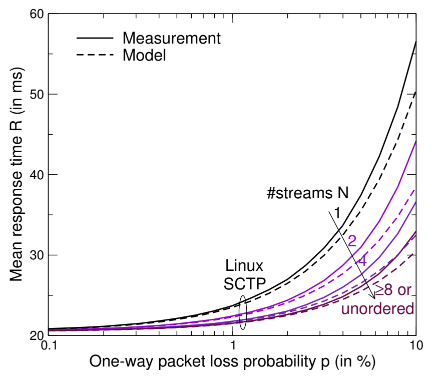

Fig. 4. Response time using (Linux) SCTP

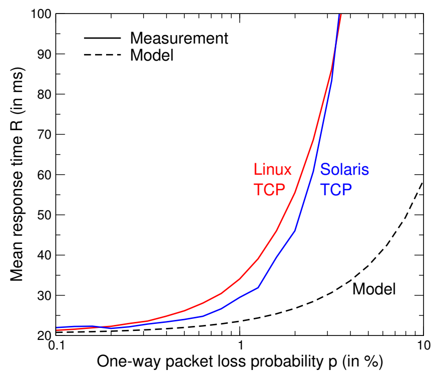

Fig. 5. Response time using (Linux/Solaris) TCP

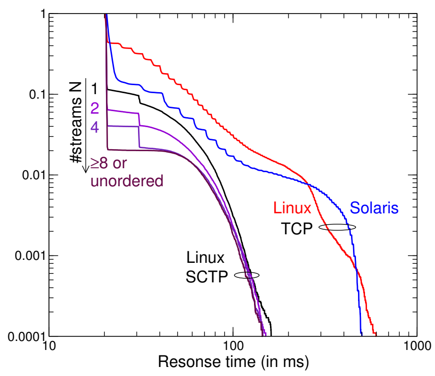

Fig. 6. Response time CCDF ($p = 1\,\%$)

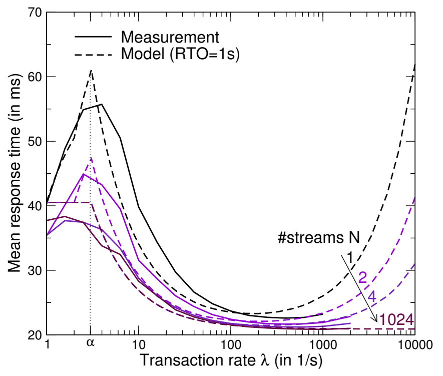

Fig. 7. Linux SCTP for variable load ($p = 1\,\%$)

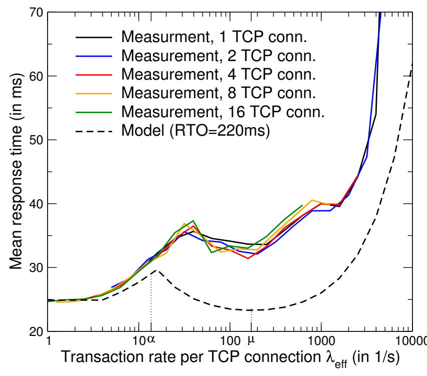

Fig. 8. Linux TCP for variable load ($p = 1\,\%$)

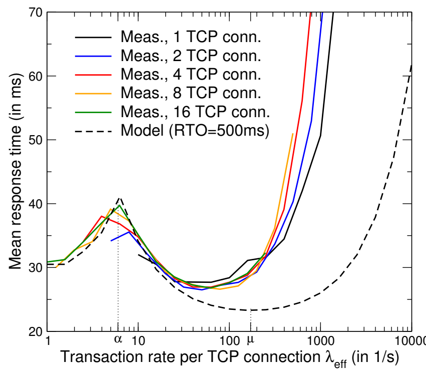

Fig. 9. Solaris TCP for variable load ($p = 1\,\%$)

SCTP is about $\lambda = 2,000\,\frac{1}{s}$. Under Solaris TCP, delays increase for $\lambda_{\text{eff}} > 500\,\frac{1}{s}$. Here the TCP congestion control seems to limit the sending rate. Our analytical model does not consider this effect and thus underestimates the response time. Interestingly, Linux TCP can transport at least four times this load before delays start to increase. Apparently, the Linux TCP algorithms trade off delay and throughput.

Summing up, the transport delay of multiple SCTP streams, or of SCTP in unordered mode, is close to the theoretical minimum given by our analytical model, whereas using one or multiple TCP connections can result in significantly larger delays, in particular for data rates of the order of $100\,\frac{1}{s}$. For a given data rate λ, the response time for transport over TCP can be minimized by using $N_{\text{opt}} \approx \frac{\lambda \cdot RTT}{2\sqrt{3}}$ parallel connections.

V. Conclusions

In this paper, we study the performance of the TCP and SCTP as transport protocols for transaction-based signaling protocols. We analyze how the impact of head-of-line blocking can be mitigated by using several parallel TCP connections, SCTP multistreaming, or SCTP unordered mode, respectively. We propose an analytical model for the end-to-end delay of signaling messages, and we verify it with measurements both under Linux and Solaris. Our results reveal that signaling transport over one TCP connection may suffer from significant delays even for moderate packet loss probabilities, whereas a small number of SCTP streams or SCTP unordered mode can avoid this head-of-line blocking. The alternative solution of multiple TCP connections performs worse in most cases.

References

[1] B. Aboba and J. Wood, "Authentication, authorization and accounting (AAA) transport profile," RFC 3539, June 2003.

[2] "Gateway control protocol: Transport over stream control transmission protocol (SCTP)," ITU-T Recommendation H.248.4, Nov. 2000.

[3] X. Fu, C. Dickmann, and J. Crowcroft, "General internet signaling transport (GIST) over SCTP," IETF Draft, work in progress, Oct. 2005.

[4] S. Kiesel, "SIMCO over SCTP," IETF draft, work in progress, Oct. 2005.

[5] J. Rosenberg, H. Schulzrinne, and G. Camarillo, "The stream control transmission protocol (SCTP) as a transport for the session initiation protocol (SIP)," RFC 4168, Oct. 2005.

[6] P. D. Amer, C. Chassot, T. J. Connolly, M. Diaz, and P. Conrad, "Partial-order transport service for multimedia and other applications," *IEEE/ACM Trans. Netw.*, vol. 2, no. 5, 1994.

[7] 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.

[8] Y. Xia and D. Tse, "Analysis on packet resequencing for reliable network protocols," *Perf. Eval.*, vol. 61, 2006.

[9] G. Camarillo, R. Kantola, and H. Schulzrinne, "Evaluation of transport protocols for the session initiation protocol," *IEEE Network*, vol. 17, no. 5, 2003.

[10] M. Lulling and J. Vaughan, "A simulation-based comparative evaluation of tranport protocols for SIP," *Comp. Commun.*, vol. 29, no. 4, 2006.

[11] K.-J. Grinnemo, T. Andersson, and A. Brunstrom, "Performance benefits of avoiding head-of-line blocking in SCTP," in *Proc. ICAS/ICNS 2005*, Papeete, Tahiti, Oct. 2005.

[12] P. Sarolahti and A. Kuznetsov, "Congestion control in Linux TCP," in *Proc. USENIX Annual Technical Conference*, Monterey, California, USA, June 2002.