# Transmission Scheduling Optimizations for Concurrent Multipath Transfer

Thomas Dreibholz, Erwin P. Rathgeb
University of Duisburg-Essen
Institute for Experimental Mathematics
Ellernstrasse 29, 45326 Essen, Germany
{dreibh,rathgeb}@iem.uni-due.de

Robin Seggelmann, Michael Tüxen
Münster University of Applied Sciences
Department of Electrical Engineering and Computer Science
Stegerwaldstrasse 39, 48565 Steinfurt, Germany
{seggelmann,tuexen}@fh-muenster.de

## ABSTRACT

SCTP is a general-purpose Transport Layer protocol with out-of-the-box support for multi-streaming as well as multi-homing. A protocol extension, which is denoted as CMT-SCTP, extends SCTP by supporting Concurrent Multipath Transfer (CMT). That is, multiple network paths are utilized simultaneously in order to improve the payload data throughput. However, dissimilar paths – i.e. paths having different delays or bandwidths – are challenging and also very likely in internet setups.

In this paper, we show how CMT-SCTP data transport performance can be improved by combining multi-streaming with an advanced stream scheduling policy and SCTP API enhancements. The performance benefit of our approach in dissimilar path setups is proven by simulations.[1][2]

**Keywords:** Concurrent Multipath Transfer, Scheduling, Optimizations, Analysis

## 1. INTRODUCTION

The Stream Control Transmission Protocol (SCTP), which is described in [14], has originally been developed for telephone signaling, but evolved into a general-purpose transport layer protocol. It is reliable, connection- and message-oriented. Multiple small messages can be bundled into a single packet to avoid overhead. Additional features are multi-homing, to use multiple network paths for failover, and multi-streaming to reduce the impact of head-of-line blocking. The packet format is extensible, so features can be added without changing the base protocol.

With the increasing popularity of multi-homed networks – being deployed to achieve link redundancy – there is also a rising demand for load sharing by utilizing *all* available network paths (for which providers are paid for) to enhance application data throughput. The Concurrent Multipath Transfer (CMT) extension, CMT-SCTP [3,4,8], allows such a load sharing.

[1]Parts of this work have been funded by the German Research Foundation (Deutsche Forschungsgemeinschaft).
[2]The authors would like to thank the anonymous reviewers for their helpful comments and Sangtae Ha for his friendly support.

To realize reliable transport, SCTP acknowledges the latest message of received continuous data with a so-called cumulative acknowledgement, similar to the Transmission Control Protocol (TCP). However, it also supports the preliminary acknowledgement of messages received out of order, hence the acknowledgements of SCTP are called selective (SACK). Since the acknowledgement of out-of-order data is just preliminary, the sender still has to keep the messages in its retransmission queue until they are cumulatively acknowledged. With the extension Non-Renegable Selective Acknowledgements (NR-SACK) [10], the receiver will also acknowledge out-of-order messages definitively, which then can be removed from the sender's retransmission queue immediately.

The multi-streaming feature of SCTP requires a scheduler on the sender and also receiver side to decide on the order in which messages of different streams are sent and delivered to the application, respectively. Using a specific scheduling algorithm depending on the scenario has proven to be beneficial [12]. Currently used schedulers, like Round-Robin which simply cycles through the streams, do not consider multiple paths. Therefore, in this paper we will analyze if there is also potential for optimizations with a CMT-aware scheduling, when using multiple streams and multiple paths with CMT enabled.

The structure of this paper is as follows: in Section 2, multi-path transfer is introduced. Details on multi-streaming and the respective scheduling are given in sections 3 and 4. Section 5 describes the simulation model and setup, followed by the performance analysis in Section 6.

## 2. MULTI-PATH TRANSFER

Figure 1 illustrates two dual-homed endpoints $A$ and $B$, each equipped with two network interface cards and therefore two IP addresses. This results in two paths (path #1 and path #2) for each direction ($A \rightarrow B, B \rightarrow A$). Clearly, a straightforward approach is to use a Transport Layer protocol which is capable of this so-called Concurrent Multipath Transfer (CMT). Two protocol approaches are actively discussed in the IETF Transport Area Working Group:

- CMT-SCTP [1,8] is simply a CMT extension for the already multi-homed Stream Control Transmission Pro-

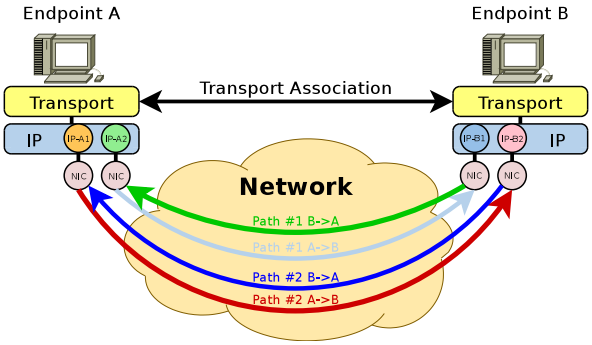


**Figure 1: The Principle of Multi-Homing**

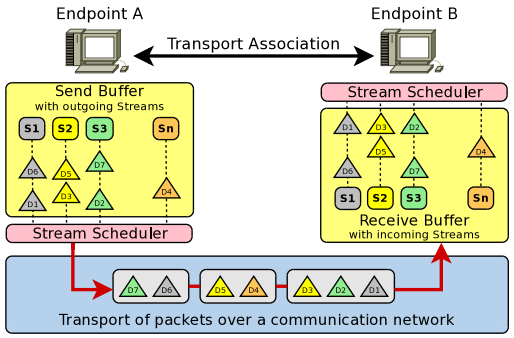


**Figure 2: The Principle of Multi-Streaming**

tocol (SCTP) [14]. The variant CMT/RP-SCTP [1, 2] applies Resource Pooling (RP) [17] to achieve fair bandwidth usage in the presence of shared bottlenecks.

- Multipath TCP (MPTCP) [7] is an extensive CMT modification of the well-known TCP [11] protocol, with strong emphasis on interoperability with and backward compatibility to existing middleware (like NAT boxes or transparent proxies) which are not aware of the CMT transport.

For its applications, MPTCP provides – just like TCP – the reliable and ordered transport of a single byte stream over a connection. Clearly, the challenge of this kind of service is to timely reorder the data transmitted over paths with different characteristics (i.e. delay, bandwidth) at the receiver side, at a small memory footprint and management overhead. SCTP, on the other hand – although being able to provide a reliable and ordered transport similar to TCP – has more flexibility with its concept of multi-streaming. The introduction of multi-streaming can simplify the transport task. In the following, we will show how to utilize this feature to improve the performance of SCTP multi-path data transmission.

## 3. MULTI-STREAMING

Multiple streams within a connection allow the separation of logically independent data. The application assigns each message to a stream, messages belonging together to the same one. In case of SCTP this is done with an identifier for each message, indicating the stream. With this identifier the protocol only needs to restore the sequence of messages belonging together, i.e. those of the same stream, while messages of different streams can arrive unordered. Hence, after a packet loss only messages of the affected streams have to be delayed to restore the sequence, while on other streams the transmission can be continued. This results in a reduced average delay compared to other reliable protocols without multi-streaming, like TCP, where all following messages are delayed after a loss, resulting in a so-called head-of-line blocking.

Figure 2 provides an illustration of transferring data of multiple streams over a single connection. Every message passed by the sending application is inserted into the corresponding stream buffer. The messages are then bundled into packets for sending. The order of the messages of a single stream is given, however, the order of messages of different streams has to be determined by a scheduler. Upon reception, the messages are sorted into stream buffers again, to restore their order. When passing them to the receiving application, another scheduler is necessary to decide on the order in which messages of different streams are delivered.

## 4. TRANSMISSION SCHEDULING

For multi-streaming, sender and receiver schedulers are necessary. The sender scheduler has to decide on the order in which messages are sent, while the receiver scheduler on the order in which the data is passed to the application. In the SCTP specification there is no statement about how these schedulers have to be realized and this is therefore open to the implementation. Today's implementations use for sender scheduling the generic algorithms Round-Robin (FreeBSD) and First-Come, First-Served (Linux, Solaris), respectively. However, in some scenarios it can be beneficial to use a specific scheduler. The receiver scheduler only decides on the order in which messages are delivered, so there is little benefit in varying the algorithm. The sender scheduler on the contrary, can affect the behavior on the wire and therefore different algorithms can be used as an optimization. The benefits of using several common scheduling approaches are described in [12].

Multi-path transfer is also a scenario where a special scheduler can be used for optimization. The most severe issue with multiple paths is the reordering of the messages, caused by differing delays on the paths. The more messages have been reordered, the more complex is the restoration for the receiver and the more buffer space is necessary. The available buffer space has to be at least equal to the bandwidth-delay product, which will be shown in detail in section 6.1, otherwise the data transfer has to be slowed down. The simplest approach to mitigate reordering is to assign streams to paths and send messages of the same stream on the same path. This avoids that some of the messages sent in-order pass the ones on the slower path. However, just assigning streams to paths may not be the optimal solution in case the delays of the paths or the amount of data of the streams differ largely from each other. In such cases the slower paths can be overloaded while the faster paths remain unchallenged. Hence, depending on the scenario, multiple specific streams may have to be assigned to one path or a stream with a large amount of data split up on multiple paths. This can require a very complex and resource-intensive scheduler. To examine the potential of a scheduler which considers multiple and also diverse paths, we will analyze the possible performance benefits with an optimized scheduler and compare it to the standard implementation. This optimized scheduler will always choose the optimal combination of streams and paths to achieve the maximum optimization, depending on the scenario.

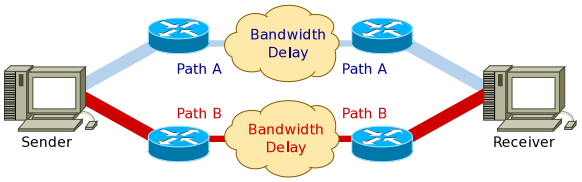


**Figure 3: The Simulation Setup**

# 5. SIMULATION MODEL AND SETUP

For our performance evaluation, we have used the OMNeT++-based INET framework with our CMT-SCTP simulation model [3]. The SimProcTC [6] tool-chain has been used for parametrization and result processing. The result plots in this paper show the average values of 24 runs and their 95% confidence intervals. Figure 3 illustrates the simulation setup: sender and receiver have been connected via two paths. The following configuration parameters have been used, unless otherwise specified:

- The QoS parameters (bandwidth, delay, bit error rate) of the links between the routers of each path have been configurable (default: bandwidth of $\rho_P$=10 Mbit/s, delay of $\delta_P$=10ms and no bit errors on each path $P$). The bottleneck network interfaces use FIFO queues having a capacity of 100 packets.
- After association establishment and transmission start, the actual throughput measurement has been started after a settling time of 19s. The measurement duration has been 300s, after which the results do not vary anymore.
- The sender has been saturated (i.e. it has tried to transmit as much data as possible); the message size has been 1,452 bytes at an MTU of 1,500 bytes (i.e. MTU-sized packets [14]; overhead is about 3.2%). It has equally distributed its messages on two streams. All messages use ordered transmission (i.e. the packet sequence is preserved within each stream).
- Send and receive buffers have the same length $\sigma$. Buffer Splitting [4] and NR-SACK [9,10] have been turned on; the effect of NR-SACKs will be examined in Subsection 6.3.

# 6. PERFORMANCE ANALYSIS

For our performance analysis, we first examine the basic scenarios of path dissimilarity with respect to delay (Subsection 6.1) and bandwidth (Subsection 6.2) with two streams. After that, we analyze a more advanced scenario (Subsection 6.3).

## 6.1 Delay Dissimilarity

Figure 4 shows the resulting CMT-SCTP payload throughput for varying the send/receive buffer sizes $\sigma$ of a transmission on paths $A$ and $B$; $\delta_B$ denotes the delay on path $B$ in ms. The bandwidths on both paths – $\rho_A$ and $\rho_B$ – are constant at 10 Mbit/s. In order to fully utilize a path, it is necessary to have at least the number of bytes given by the RTT-bandwidth product in flight. Therefore, the send and receive buffers need at least a size of:

$$\sigma \geq \underbrace{2 * \delta_A * \rho_A}_{\text{Path A}} + \underbrace{2 * \delta_B * \rho_B}_{\text{Path B}} + \epsilon. \qquad (1)$$

Additional delays are introduced by queues within the network (their average size depends on the amount of congestion), which further increases the needed buffer size by a term $\epsilon$. This is reflected by the results of the 10ms/10ms symmetric scenario (curves 1 and 4): at $\sigma$=64 KiB (the pure RTT-bandwidth-product according to Equation 1 is 50,000 bytes), the expected CMT-SCTP payload throughput of around 19.2 Mbit/s is reached – regardless of the scheduling policy (Σ=Round-Robin for round-robin scheduling or Σ=Fixed for mapping stream #0 to path $A$ and stream #1 to path $B$).

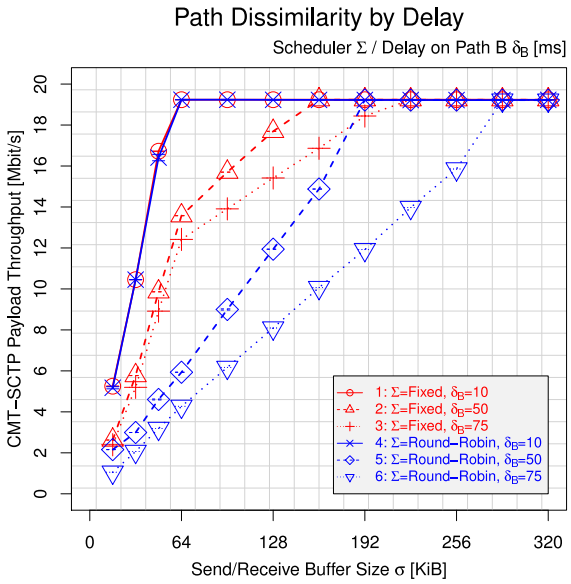


**Figure 4: Throughput on Dissimilar Delay Paths**

With dissimilar delay paths (here: $\delta_B$=50ms or $\delta_B$=75ms; curves 2 and 5 or curves 3 and 6, respectively), the choice of the stream scheduler makes a difference: for $\delta_B$=75ms, the fixed stream mapping (i.e. Σ=Fixed) already achieves the full bandwidth for send and receive buffer sizes of $\sigma$=224 KiB – while $\sigma$=288 KiB is required when using the round-robin scheduling (i.e. Σ=Round-Robin). When the packets traverse dissimilar delay paths, the necessary packet reordering at the receiver side requires some space, i.e. out-of-sequence chunks cannot leave the receive buffer until all preceding chunks of the same stream have been received. Clearly, this negative effect is smaller when the chunks of a stream have a similar delay – which is achieved by the fixed stream mapping.

The corresponding average message delay (i.e. the time from message generation until reception at the application layers) is depicted in figure 5. As expected, there is no significant message delay difference between stream #0 (left-hand plot) and stream #1 (right-hand plot) in the symmetric 10ms/10ms case for round-robin scheduling (i.e. Σ=Round-Robin; see curves 1 and 4). Note, that the message delay linearly increases with the send and receive buffer size $\sigma$, once $\sigma$ has exceeded the minimum given by Equation 1: a higher setting of $\sigma$ cannot make the transmission faster – it just increases the time the chunks of a message have to wait in the send buffer until they actually can be transmitted over the network.

As expected, round-robin scheduling (i.e. Σ=Round-Robin) results in both streams having similar message delays, re-

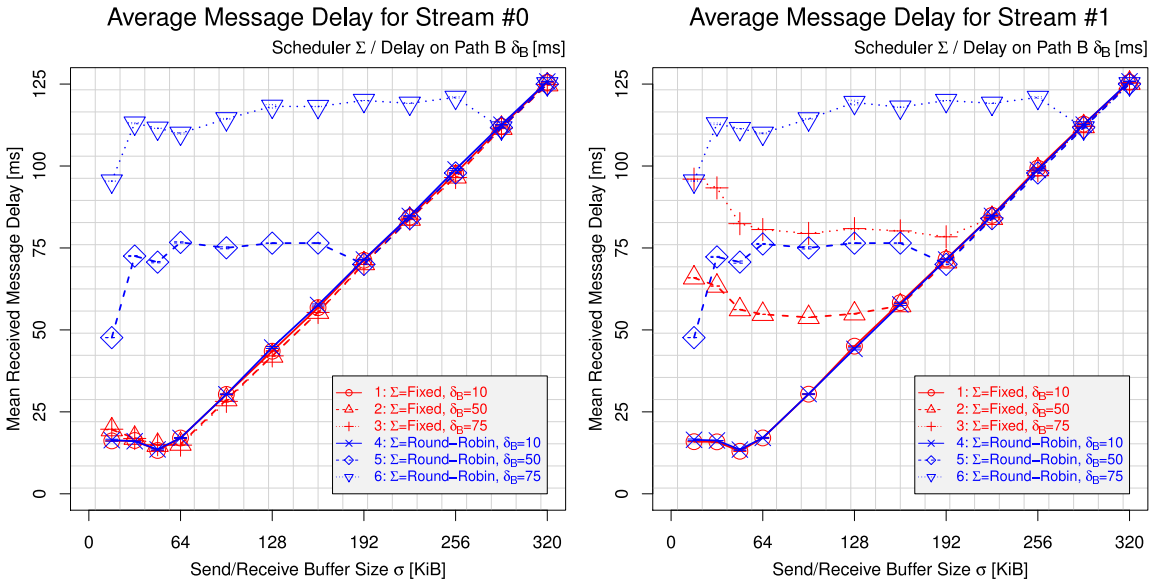


**Figure 5: Per-Stream Average Message Delay on Dissimilar Delay Paths**

gardless of the path delay dissimilarity. Clearly, the fixed stream mapping (i.e. Σ=Fixed) shows significant differences: stream #0, using path $A$ with a latency of $\delta_A$=10ms, has a significantly smaller message delay than stream #1, using path $B$ with a higher latency of $\delta_B \geq$50ms (compare curves 2 and 3). But the bandwidths of both streams remain equal (not plotted here due to space limitations), as long as Equation 1 is satisfied. We will examine bandwidth dissimilarity in the following Subsection 6.2.

In summary, the fixed stream mapping can – in comparison to round-robin scheduling – achieve the maximum throughput and reduced delay at smaller send and receive buffer sizes.

## 6.2 Bandwidth Dissimilarity

In order to show the effects of bandwidth dissimilarity, figure 6 presents the CMT-SCTP payload throughput results for varying the send/receive buffer sizes $\sigma$ at path $B$ whose bandwidth $\rho_B$ is set to 25 Mbit/s and 50 Mbit/s, respectively. Using round-robin scheduling (i.e. Σ=Round-Robin), the behavior is as anticipated (curves 5 and 6): for a sufficiently large setting of $\sigma$ (according to Equation 1), the expected payload throughput of nearly $\frac{\rho_A+\rho_B}{2}$ per stream is reached, i.e. the bandwidth is equally splited between the two streams.

For the fixed stream mapping (i.e. Σ=Fixed; stream #0 on path $A$, stream #1 on path $B$), the result is different: while the throughput of stream #0 remains at 10 Mbit/s (since path $A$'s bandwidth is set to $\rho_A$=10 Mbit/s) as expected, the stream #1 is *not* able to utilize the higher bandwidths $\rho_B$ of path $B$ (curves 1 and 2). Even worse, the achieved payload throughput remains at around 10 Mbit/s, i.e. the same throughput as for stream #0. The reason for this problem is the application layer, which – using the SCTP socket API [16] – has no information on the current per-stream contents in the send buffer. That is, the application layer is notified to refill the buffer when its occupancy has dropped under a certain threshold – but it has no information which stream is currently low on data. Providing data in a round-robin manner down to the SCTP layer leads to filling up the buffer with data for stream #0 on the slow path $A$ – while stream #1 on the fast path $B$ cannot get enough data to utilize its path's capacity.

Our solution is a small API extension [5] which allows the sender to query the per-stream send buffer utilization. By making the application layer aware of the stream dissimilarity introduced by the fixed stream mapping, the sender application can generate an appropriate amount of data for each stream. This is particularly useful for all kinds of SCTP-based tunneling applications, e.g. Secure Shell (SSH) channels [18] or signaling applications like SS7 [13]. With our API extension (curves 3 and 4), the throughput behavior of stream #1 is as desired: it can utilize the bandwidth $\rho_B$ of path $B$, and the overall payload throughput of the association is nearly $\rho_A + \rho_B$.

In summary, the fixed stream mapping requires an API extension to make applications aware of the stream dissimilarity. This is necessary for utilizing streams with different bandwidths. This API extension has been contributed as Internet Draft [5] to the IETF SCTP standardization process.

## 6.3 Advanced Scenario

To further examine the performance of the fixed stream mapping, we have used a more advanced setup consisting of the paths $A$ and $B$ with constant bandwidths settings of $\rho_A$=$\rho_B$=10 Mbit/s and delays $\delta_A$=$\delta_B$=10ms. To this symmetric setup, we add a third path $C$ with a bandwidth setting of $\rho_C$=1 Mbit/s and delay $\delta_C$=50ms. In the real world, such a scenario is not unlikely when dynamically setting up a DSL-based backup link. Using SCTP Dynamic Address Reconfiguration (Add-IP) [15], the new path would be dynamically added to the already existing paths of the association. Figure 7 presents the resulting CMT-SCTP throughput (left-hand plot) and average message delay over all streams (right-hand plot) for varying the send/receive buffer sizes $\sigma$. seven streams are multiplexed over the three paths; the fixed stream mapping assigns streams #1, #3, #5

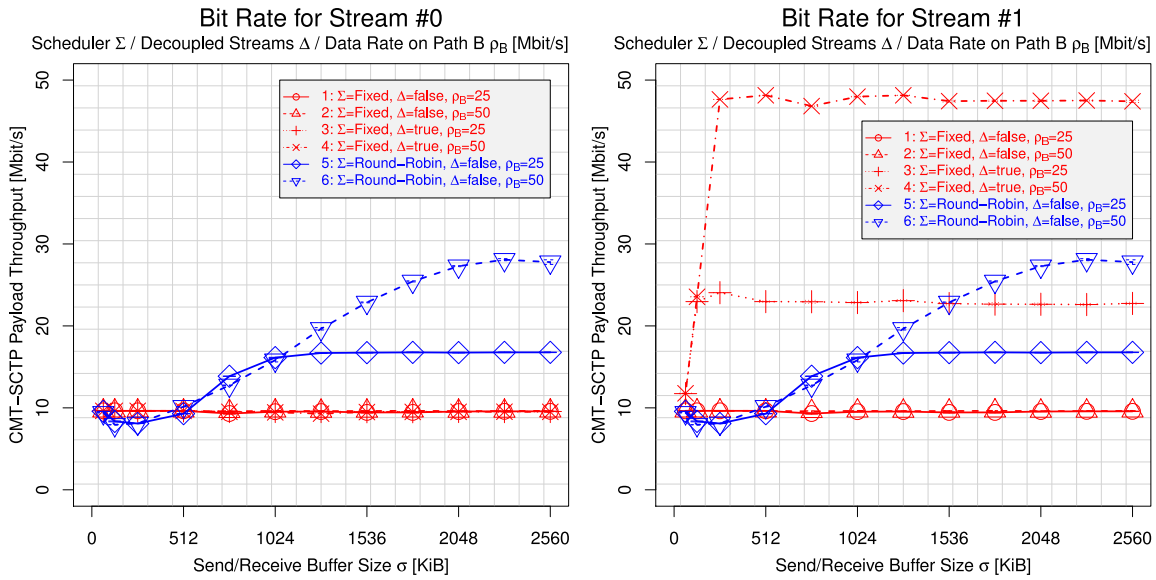


**Figure 6: Per-Stream Bit Rate on Bandwidth Dissimilarity**

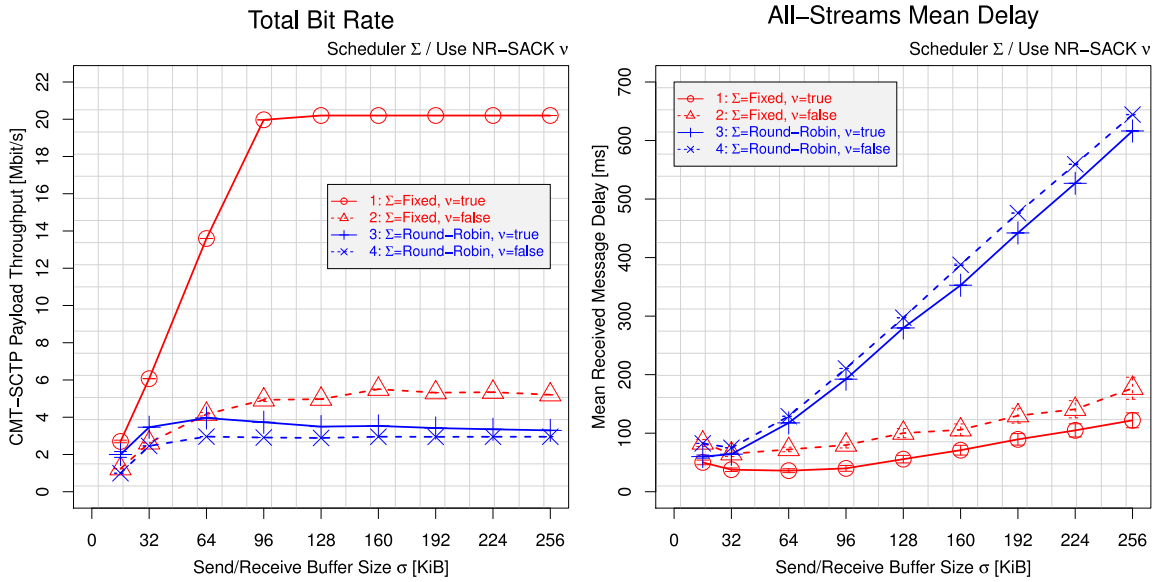


**Figure 7: Advanced Setup with Seven Streams over Three Paths**

to path $A$, streams #2, #4, #6 to path $B$ and stream #0 to path $C$. A real-world use case for such a seven-stream setup is e.g. an SSH-based tunneling scenario [18].

Clearly, round-robin scheduling (i.e. Σ=Round-Robin) does not perform well in this setup (see curve 4). Even for a send/receive buffer size setting of $\sigma$=256 KiB, the throughput remains below 3 Mbit/s – at a message delay of more than 600ms! In case of an SSH tunneling setup – which may transport interactive session data – this performance is unacceptable. Using the fixed stream mapping (i.e. Σ=Fixed), the payload throughput performance improves to slightly above 5 Mbit/s and a message delay of around 150ms (see curve 2). While the delay improvement is significant, the throughput performance is still below the expectations.

The reason for this low performance is packet reordering – due to path delay dissimilarity – and the implied queuing effects: while the receive buffer can provide its per-stream messages to the application layer and remove the corresponding chunk data (i.e. gaining space for new chunks), once the data of a stream is in sequence, the sender may only drop them on reception of a cumulative acknowledgement covering these chunks. That is, a chunk can only be removed from the send buffer when *all* preceding chunks – regardless of their stream numbers – have been acknowledged. This is necessary, since a receiver may revoke gap acknowledgements above the cumulative acknowledgement at any time. In order to overcome this problem, the NR-SACK extension [9, 10] becomes highly useful: the receiver may non-revokably acknowledge chunks above the cumulative ac-

knowledgement, allowing the sender to drop these chunks from its buffer – and gaining space to accept new data from its application layer. Using NR-SACKs (i.e. $\nu$=true) significantly improves the performance when using the fixed stream mapping: at $\sigma$=128 KiB, it reaches the expected payload throughput of around 20 Mbit/s, while further reducing the message delay to around 60ms (curve 1). Note, that NR-SACK only has a small effect when using round-robin scheduling (i.e. $\Sigma$=Round-Robin; curve 3), since messages of all streams are transported over all paths – and waiting for acknowledgements on other paths is necessary anyway.

In summary, it is highly desirable to combine a fixed stream mapping with the NR-SACK extension in order to achieve maximum throughput at a small delay in dissimilar setups, especially because there are no disadvantages with this optimization.

# 7. CONCLUSIONS

Using CMT with a single byte stream is a challenging task, particularly when the memory footprint and management complexity have to be kept reasonably small. But unlike MPTCP, the SCTP protocol supports multi-streaming. Using CMT-SCTP for SCTP-based CMT transport, we have introduced an advanced stream scheduling policy to achieve an improved service performance: by mapping each stream to a certain path, the buffer sizes can be kept small. In comparison to round-robin scheduling, our approach utilizes the network with smaller buffer sizes while also reducing the per-stream message delay.

As part of future work, we are going to design and develop a dynamic scheduling algorithm which automatically maps streams to paths, according to given QoS properties of the streams. Our proposed enhancements to SCTP will also be realized in the FreeBSD kernel implementation, in order to allow for tests in real-world setups. Finally, we are also going to contribute our results into the ongoing IETF standardization process of SCTP.

# 8. REFERENCES


[1] Becke, M., Dreibholz, T., Iyengar, J., Natarajan, P., and Tüxen, M. Load Sharing for the Stream Control Transmission Protocol (SCTP). Internet-Draft Version 00, IETF, Network Working Group, July 2010. draft-tuexen-tsvwg-sctp-multipath-00, work in progress.

[2] Dreibholz, T., Becke, M., Pulinthanath, J., and Rathgeb, E. P. Applying TCP-Friendly Congestion Control to Concurrent Multipath Transfer. In *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)* (Perth/Australia, Apr. 2010), pp. 312–319. ISSN 1550-445X.

[3] Dreibholz, T., Becke, M., Pulinthanath, J., and Rathgeb, E. P. Implementation and Evaluation of Concurrent Multipath Transfer for SCTP in the INET Framework. In *Proceedings of the 3rd ACM/ICST International Workshop on OMNeT++* (Málaga/Spain, Mar. 2010). ISBN 978-963-9799-87-5.

[4] Dreibholz, T., Becke, M., Rathgeb, E. P., and Tüxen, M. On the Use of Concurrent Multipath Transfer over Asymmetric Paths. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM)* (Miami, Florida/U.S.A., Dec. 2010).

[5] Dreibholz, T., Seggelmann, R., and Becke, M. Sender Queue Info Option for the SCTP Socket API. Internet-Draft Version 00, IETF, Network Working Group, Nov. 2010. draft-dreibholz-tsvwg-sctpsocket-sqinfo-00, work in progress.

[6] Dreibholz, T., Zhou, X., and Rathgeb, E. P. SimProcTC – The Design and Realization of a Powerful Tool-Chain for OMNeT++ Simulations. In *Proceedings of the 2nd ACM/ICST International Workshop on OMNeT++* (Rome/Italy, Mar. 2009), pp. 1–8. ISBN 978-963-9799-45-5.

[7] Ford, A., Raiciu, C., Barré, S., and Iyengar, J. Architectural Guidelines for Multipath TCP Development. Internet-Draft Version 02, IETF, Network Working Group, Oct. 2010. draft-ietf-mptcp-architecture-02, work in progress.

[8] Iyengar, J. R., Amer, P. D., and Stewart, R. Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths. *IEEE/ACM Transactions on Networking 14*, 5 (Oct. 2006), 951–964. ISSN 1063-6692.

[9] Natarajan, P., Amer, P., Yilmaz, E., Stewart, R., and Iyengar, J. Non-Renegable Selective Acknowledgements (NR-SACKs) for SCTP. Internet-Draft Version 06, IETF, Network Working Group, Aug. 2010. draft-natarajan-tsvwg-sctp-nrsack-06, work in progress.

[10] Natarajan, P., Ekiz, N., Yilmaz, E., Amer, P. D., and Iyengar, J. Non-Renegable Selective Acknowledgments (NR-SACKs) for SCTP. In *Proceedings of the 16th IEEE International Conference on Network Protocols (ICNP)* (Orlando, Florida/U.S.A., Oct. 2008), pp. 187–196. ISBN 978-1-4244-2506-8.

[11] Postel, J. Internet Protocol. Standards Track RFC 791, IETF, Sept. 1981.

[12] Seggelmann, R., Tüxen, M., and Rathgeb, E. Stream Scheduling Considerations for SCTP. In *Proceedings of the 18th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)* (Split/Croatia, Sept. 2010).

[13] Sidebottom, G., Morneault, K., and Pastor-Balbas, J. Signaling System 7 (SS7) Message Transfer Part 3 (MTP3) – User Adaptation Layer (M3UA). Standards Track RFC 3332, IETF, Sept. 2002.

[14] Stewart, R. Stream Control Transmission Protocol. Standards Track RFC 4960, IETF, Sept. 2007.

[15] Stewart, R., Xie, Q., Tüxen, M., Maruyama, S., and Kozuka, M. Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration. Standards Track RFC 5061, IETF, Sept. 2007.

[16] Stewart, R., Xie, Q., Yarroll, Y., Wood, J., Poon, K., and Tüxen, M. Sockets API Extensions for Stream Control Transmission Protocol (SCTP). Internet-Draft Version 24, IETF, Transport Area Working Group, Oct. 2010. draft-ietf-tsvwg-sctpsocket-24.txt, work in progress.

[17] Wischik, D., Handley, M., and Braun, M. B. The Resource Pooling Principle. *ACM SIGCOMM Computer Communication Review 38*, 5 (Oct. 2008), 47–52. ISSN 0146-4833.

[18] Ylonen, T., and Lonvick, C. The Secure Shell (SSH) Connection Protocol. Standards Track RFC 4254, IETF, Jan. 2006.