Beneficial Transparent Deployment of SCTP: the Missing Pieces

Michael Welzl
Department of Informatics
University of Oslo, Norway

Florian Niederbacher
Institute of Informatics
University of Innsbruck, Austria

Stein Gjessing
Department of Informatics
University of Oslo, Norway

***Abstract*—The Internet-wide deployment of new transport protocols such as the Stream Control Transmission Protocol (SCTP) is a difficult matter. While SCTP could be beneficial in many cases, it is still a major challenge to enable applications to use the new protocol. We believe that its deployment could be significantly accelerated by introducing an intermediate step: transparent usage below TCP, such that TCP-based applications automatically obtain a benefit. We have implemented a Linux based TCP-SCTP mapping tool that exploits SCTP's multi-streaming feature, and carried out measurements which show that such an approach can significantly improve the performance of TCP applications. However, and perhaps more importantly, we also encountered difficulties, which lead us to make some concrete recommendations regarding future research and the implementation and specification of SCTP itself.**

I. Introduction

For decades, the Internet's transport layer has been defined by two protocols: TCP and UDP. This narrow choice does not match the heterogeneity of services that is needed by applications and could be provided by today's infrastructure. New transport protocols such as SCTP, DCCP and UDP-Lite have been designed to address these shortcomings; however, their deployment in the Internet is sluggish, to say the least. This is partly due to a problem which has been identified in [1]:

> There is a vicious circle – application developers will not use a new protocol (even if it is technically superior) if it will not work end-to-end; OS vendors will not implement a new protocol if application developers do not express a need for it; NAT and firewall vendors will not add support if the protocol is not in common operating systems; the new protocol will not work end-to-end because of lack of support in NATs and firewalls.

We need to find a way to break this vicious circle and make a transition from today's two tier architecture (with TCP and UDP as the only transport protocols) to an Internet transport layer where multiple transport protocols are available. This is called "accommodating the tussle" in [2].

One possibility is to let the Operating System use a new transport protocol in such a way that a benefit is automatically attained if it is available, without having to change existing applications. If such a mechanism falls back to standard TCP or UDP when the new transport protocol is unavailable, no harm is done, but a benefit is achieved. This could give NAT and firewall vendors an incentive to support the new protocol, and, as the protocol becomes more available in this fashion, application developers could eventually become interested in fully exploiting its features. In other words, with this intermediate step, it could be possible to gradually deploy a new transport protocol.

To investigate the possibility of deploying SCTP in this fashion, we developed a prototypical Linux-based TCP-SCTP mapping tool which exploits SCTP's multi-streaming feature by mapping TCP connections between the same pair of hosts onto streams of a single SCTP association. In this way, it should, for instance, be possible to automatically let a newly starting transfer exploit the larger congestion window (cwnd) of an ongoing or immediately preceding transfer.

While some tests with this tool showed extremely encouraging results, we also encountered difficulties. These lead us to make some recommendations regarding future research and the implementation and specification of SCTP.

After a brief overview of related work in the next section, we will explain how we implemented the tool in section III. Some results of our tests are presented in section IV, leading to the aforementioned recommendations which we conclude with.

II. Related work

Neither mapping TCP to SCTP nor reusing the cwnd of a previous or ongoing connection are new ideas. Examples of the former are the "withsctp" application which is included in the "Linux Kernel Stream Control Transmission Protocol" (LKSCTP)[1] project, and the "shim layer" [3], but neither of these two approaches seems to attain a significant performance benefit except for conditions of very high packet loss. Accessing a common cwnd with multiple TCP connections has been proposed in [4], [5] and, in a somewhat more generic way for parallel data transfers, in [6], [7]. To the best of our knowledge, none of these methods are fully (just [4] partially, e.g. in Linux) implemented in common operating systems; this may be due to the complexity of such an implementation, combined with the fact that the performance benefit is not always clear from a (selfish) single user's point of view (we will get back to this problem in our conclusion).

[1] http://lksctp.sourceforge.net/

Structured Streams (SST) [8] is a transport protocol that was designed to better support multiple application streams by providing hierarchies and dynamic creation and deletion of streams as well as stream priorities. While it might seem to be even better suited for our purpose than SCTP, the many similarities between these two protocols and the fact that SCTP is standardized makes SCTP a more attractive choice than SST.

The evaluation section of [8] focuses on the performance benefit attained by removing the connection setup and teardown overhead for extremely short connections – a scenario that we do not consider in this paper, but for which it seems likely that our approach can attain a similar benefit. Accordingly, the authors of [9] have shown that, for HTTP, SCTP with multi-streaming can perform even better than multiple parallel TCPs. However, as a major difference to our work, they had to change the application to achieve this.

Generally, what we present in this paper could be done in many ways, e.g. by adding SCTP-like features to TCP. It seems logical that this direction should not be pursued further at this stage, where we have protocols like SCTP available but not used much on the Internet – and when automatically obtaining a benefit for end users with SCTP could foster deployment of this protocol, eventually leading to a more comprehensive set of available transport layer services. In fact, we believe that our approach is an indispensable element for gradually shifting towards an entirely new transport architecture such as the one described in [10].

III. Implementation

The goal of our tool is to map multiple TCP connections between the same pair of hosts onto different independent streams of a single SCTP association. This way, they would all conjointly make use of a single congestion control instance, and the connection setup/teardown delay for the individual connections could theoretically be avoided too. Because sequential delivery is enforced per stream (if so desired) but not *between* streams by SCTP, message loss of one stream does not cause head-of-line blocking delay for messages of another stream.

In a first step, we considered redirecting the Linux TCP socket calls onto SCTP sockets. One inherent problem with this approach is that the SCTP API [11] does not allow a logical separation of streams at the user level. That is, if thread 1 (expecting data on stream 1) reads from the SCTP assocation, it might just as well get data from stream 2 (intended for thread 2), along with the number "2", so that it can accordingly hand over the data. Since a TCP application would only expect to receive data that was intended for it, it is clear that streams must first be separated somehow, e.g. by a common receiver thread or process that continuously reads from the socket buffer and then copies the data into per-receiver buffers, as shown in Figure 1.

The size of dedicated per-receiver stream buffers may be large, but will always be limited – thus, one of the buffers can get full and subsequently block all other receivers. In the scenario shown in Figure 1, this could, for example, happen

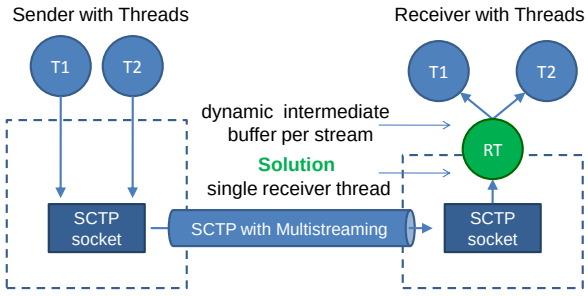

Fig. 1. Parallel transmissions with SCTP's multi-streaming

when the buffer of thread T1 is full, but the single receiver thread RT must give the next data block to T1. Simply dropping messages is not an option because this would break the semantics of reliable transfer: at this stage, the messages have already been acknowledged, as they have correctly arrived at the SCTP socket buffer. As a possible solution, per-stream flow control could be applied, but this is not available as a native protocol function.

Because of the complexity of implementing the above functionality in kernel space, we investigated the possibility of redirecting socket calls in user space, but found no efficient way to do so. A detailed description of the problems that we encountered is provided in [12].

We then decided to implement our mapping tool in the form of a proxy (which we called "Connection Manager Gateway" (CMG)), as shown in Figure 2. Our idea was not to transparently modify the application code in order to use SCTP, but to influence the connection itself. This method has the obvious disadvantage of additional TCP overhead (connection management and congestion control) between the hosts and the CMGs, but this side effect can be minimized by bringing the CMGs as close to the hosts as possible (in our setup, they were applications running directly on hosts A and B, respectively). On the positive side, this implementation method makes it very easy to fall back to standard TCP behavior, and allows for deliberate placement of the CMG if needed (e.g. to avoid problems with NATs).

TCP connections were redirected by accordingly modifying only the connect() function of the socket API, using a technique called "preloading". The socket address parameters forwarded by this function call were exchanged with the local host address as well as the used CMG TCP port. Additionally, the original address parameters were sent to the CMG after the successful connection setup. Between the CMGs, a simple protocol (operating on stream 0 of the SCTP assocation) took care of communicating a TCP connection attempt (or teardown) from one side to the other, such that TCP connections were available for forwarding at both CMGs as long as they were needed.

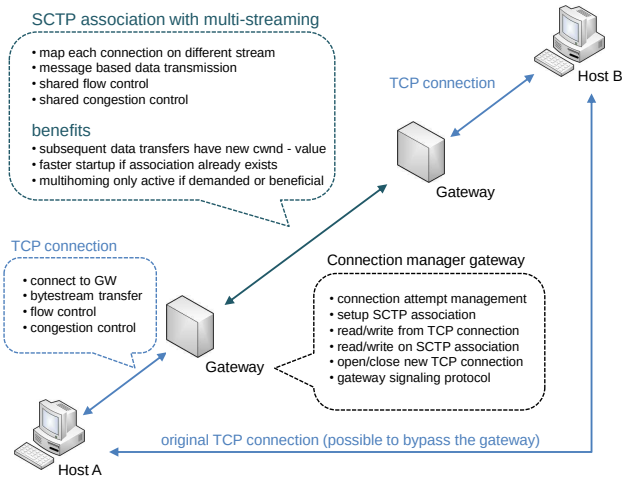

Fig. 2. The Connection Manager Gateway (CMG)

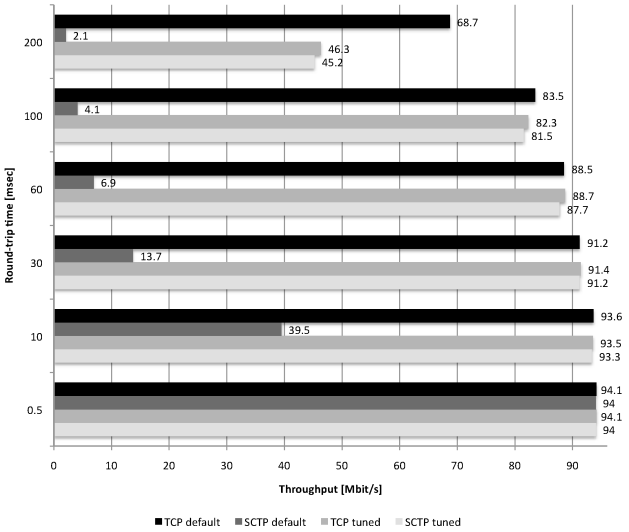

Fig. 4. Iperf measurements with default and tuned parameters

IV. Evaluation

We tested our implementation in a typical SOHO (small office/home office) network, shown in Figure 3, using standard PCs connected via Ethernet with 100 Mbit/s and a user message size of 1452 bytes. A network emulator (netem)[2] was applied on the intermediate router, where it introduced a constant delay of 30 ms (unless otherwise noted) in each direction. With our prototypical user-level implementation, we sometimes saw significant fluctuations from OS scheduling. In order to get more trustworthy numbers, all results presented in this section were obtained by taking the average of 30 experiments. For the sake of clarity, we have not plotted the standard deviation, as we believe it would not add any information that is of importance for the main conclusion of this paper.

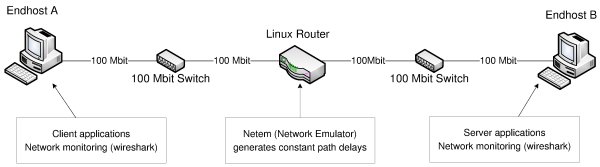

Fig. 3. Network testbed used for the practical measurements

A. Parameter tuning

In order to fairly compare TCP and SCTP, we performed measurements with a ”tuned” TCP version, using Reno congestion control, since this is the only congestion control mechanism available in SCTP. Also, since SCTP does not auto-tune the socket receive buffers, they were manually tuned to fit the RTT (for example to 732 kbytes when the RTT was 60 ms).

[2]netem is an integral part of the Linux operating system installed on all hosts (Ubuntu 9.04 – Kernel 2.6.31); it allows emulating variable delay, loss, duplication and re-ordering.

Figure 4 shows that, in initial 10-second-long experiments with various different round-trip time (RTT) values – as a means to change the bandwidth×delay product (BDP) – SCTP (from lksctp-tools version 1.0.11) was less and less efficient in comparison to TCP as the RTT grew. For values up to 100 ms, we found that this is largely due to TCP buffer auto-tuning. Additionally, with 200 ms, the BDP is large enough for the default TCP congestion control mechanism (CUBIC) to play out its advantage.

As Figure 4 shows, the performance of tuned TCP and SCTP is quite similar. Note that this figure depicts the “raw” SCTP performance, i.e. from SCTP without our CMG, with which we would get additional delay from connection / association establishment, and from manipulating the intermediate buffers. While the impact of the former overhead declines with growing transfer duration, the impact of the latter overhead grows, causing SCTP to be slower than TCP by approximately 50 ms for an 8 Mbyte file and 400 ms for a 64 Mbyte file. We do not present a deeper investigation of this problem at this point because it is merely a side effect of our prototypical implementation. The differences between TCP and SCTP shown in Figure 4, however, are general, and apply no matter how efficiently a TCP-SCTP mapping tool is implemented.

We will now proceed to discuss the results obtained in experiments with sequential and parallel transfers, highlighting the pro’s and con’s of using our tool to exploiting the cwnd of an immediately preceding or currently ongoing transfer, respectively.

B. Sequential transfers

We used the “wget” tool (v1.11.4) to download an HTML document (113 bytes) with five images of equal size from a web server. The experiment was repeated for different web pages (different image files) and with different SCTP situations: “start-up”, i.e. the wget transfer is the first transfer

seen by the CMG, causing it to establish the SCTP association to its peer, and "open cwnd", where we ran iperf for 5 seconds before the wget transfers. iperf would then establish the connection and build up a cwnd that wget reused. However, in our experiments we had a problem running wget back-to-back with iperf (the execution gap was well above the RTT). Hence, to facilitate these tests, we applied a kernel patch that causes SCTP's cwnd to only decrease upon the reception of a "heartbeat" message instead of every retransmission time-out (RTO) if the destination address is in idle state.

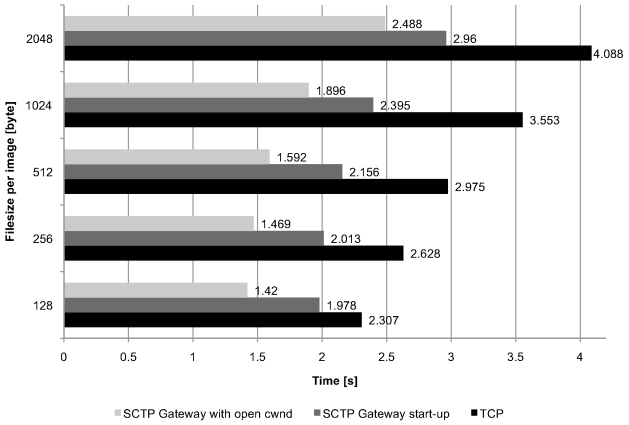

Fig. 5. Time required for a serial download of an HTML page with five images of the same size

Figure 5 shows the results from a serial download with a new connection for each single file (one by one): because each transfer was able to reuse the cwnd of the previous one, the download time was consistently reduced by more than a second, even in the start-up case. Clearly, such an improvement is noticeable by an end user. Since wget initiated transfers one after another, and TCP connection setup and teardown occurred between logical entities on the same host only (sender to CMG), TCP's connection handling delay was below the SCTP RTT of roughly 60 ms. Hence, the cwnd could be reused from one wget transfer to the next (also without of patch). Direct consecutive TCP connections, however, did not reuse the same cwnd value.

The results shown in Figure 6, for which the same experiment was repeated with a persistent connection (a single connection which is reused by HTTP 1.1 for the subsequent transmissions), indicate that we should not always apply our tool. In this particular case, the performance was generally better than in the case without persistent connections – which is not surprising because of the reduced overhead (e.g., no connection setup and teardown between the downloads) – but the TCP-SCTP mapping only yielded a benefit when the cwnd was already opened by a preceding iperf transfer, and was otherwise even detrimental. This is because, by using only one TCP connection for the entire communication, persistent connections effectively let each transfer reuse the cwnd of the previous transfer without the overhead that our tool adds to the communication.

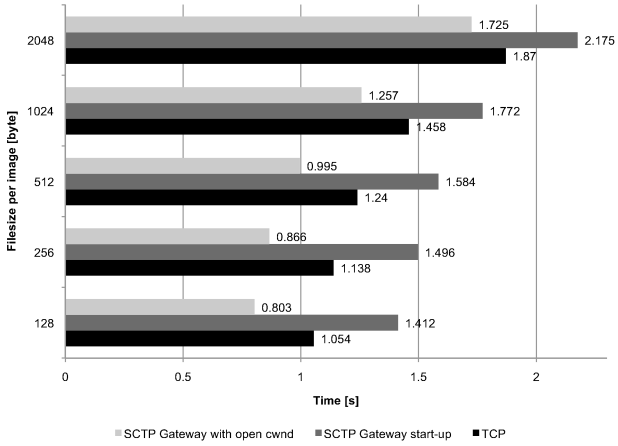

Fig. 6. Time required for a serial download of an HTML page with five images of the same size using persistent connections

C. Parallel transfers

Consider a user browsing a website that contains movies and pictures, e.g. for sharing them with friends and relatives; here, starting a movie download, and clicking on a picture while this download proceeds seems like a very obvious use case. In this scenario, mapping both the early long transfer (the movie) and the later short one (the picture) onto one SCTP association should be beneficial because the later transfer could make use of the large cwnd that the earlier transfer has already attained.

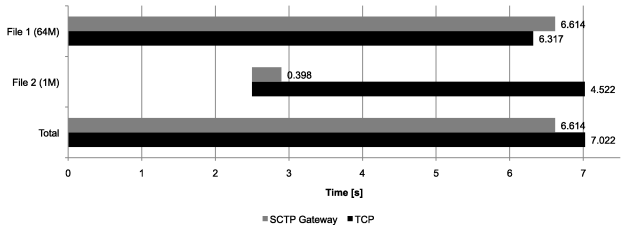

Fig. 7. Timeline of transferring two files when the longer transfer (file 1) is joined by a shorter one (file 2) after 2.5 seconds

We emulated this situation by a transfer of 64 Mbytes (file 1) which is joined by a transfer of 1 Mbytes (file 2) after 2.5 seconds; the results are shown in Figure 7. The duration of file 2 with TCP was surprisingly long (in a separate test, it took only 0.689 seconds to transfer this file in isolation). This extremely unfair TCP rate allocation was caused by the very different cwnd values at the time of starting the second transfer, combined with the exceedingly long default queue used in Linux for the outgoing network interface (the `txqueuelen` parameter of the eth device) – an additional experiment where we added the "ping" command showed that the RTT grew from about 60.2 ms to 711 ms when the second transfer began. File 1 took slightly longer with SCTP, which is due to its length – as mentioned before, the buffer management overhead of our prototypical implementation grows with the file size – and because, other than in the TCP case, the

bandwidth is fairly shared among the two files with SCTP.

Despite the overhead of our solution, mapping the two transfers onto streams of an SCTP association greatly improved the overall situation. Since the two transfers were no longer competing for bandwidth, file 2 was transferred much faster than with TCP. Because file 2 could benefit from the larger cwnd of the already ongoing transfer, it was also transmitted faster than in isolation. Note that, in this experiment, the SCTP association was already established before the test began (but without artificially opening the cwnd with a preceding iperf transfer, as in the previous experiments). The inclusion of connection / assocation establishment would have added another 60-100 msec to the communication, and thus would not have significantly affected the general outcome.

V. Conclusion

Our goal was to see if a significant performance benefit can be achieved by transparently mapping TCP connections between the same pair of hosts onto streams of a single SCTP association. As we have shown in the previous section, the answer to this question is "yes". However, we have also seen that doing this is not always beneficial, and sometimes even detrimental. We refrained from carrying out more tests to evaluate under which conditions our tool would be worth applying because we considered our implementation too prototypical to further proceed with it, and addressing the problems would take us far beyond merely updating the tool. At this point, we consider it more valuable to take a step back and examine the lessons learned from our exercise.

Ideally, we would carry out our TCP-SCTP mapping in a way that is only beneficial and never harmful; this would mean to automatically enable it only when an advantageous situation is detected. The decision would have to depend on several factors, including the starting times and sizes of transfers, the number of flows, and the attained SCTP cwnd of an ongoing transfer at the time of mapping a new TCP flow. Since this would also include knowing the maximum transfer time, it might have to be done in an application-dependent way, and in fact it may often only be possible to make a correct decision in hindsight.

How to inform a receiver of a sender's intention to carry out a mapping is another, entirely different, open issue. The major challenge here is to do this without requiring additional RTTs before starting the communication [13]. In our tests, our tool always carried out the mapping and both sides were statically configured to assume that SCTP is used for nothing else.

It seems to us that addressing these two open questions is the biggest challenge for beneficial transparent deployment of SCTP. Additionally, we make the following recommendations for future work:

- Results in Section IV have shown that our connection manager implementation is not ideal. It would be better to directly replace the TCP function calls with SCTP in the OS, which, it seems, can only be done in the kernel because macros cannot be preloaded in Linux. To map independent processes to streams of a single SCTP association, such an implementation would have to support per-stream independent read and write functions as well as per-stream flow control, which is also missing in the SCTP implementation and its API specification.
- Dynamic creation and deletion of streams, as in SST [8], would be desirable for our work – with current SCTP, we have to decide how many streams should be used beforehand, which either leads to unnecessary memory overhead or to an upper limit on the number of TCP connections that can be mapped. This change to SCTP is already partially addressed in [14], which defines how streams could be dynamically added (but not deleted).
- To ensure that SCTP always performs at least as good as TCP (see Figure 4), buffer auto-tuning and pluggable congestion control should be added to its implementation.

The Internet-wide deployment of SCTP, which we believe to be possible in the transparent fashion that we have described, would be a significant change to the transport layer that our applications are working with today. Should it happen, this change could also create an impetus for trying to deploy other transport protocols in a similar fashion. This way, it might finally be possible to break the vicious circle, and change the Internet's transport layer for good – and it all begins with following the above recommendations.

References

[1] M. Handley, "Why the Internet only just works," *BT Technology Journal*, vol. 24, no. 3, pp. 119–129, 2006.

[2] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden, "Tussle in Cyberspace: defining tomorrow's Internet," *IEEE/ACM Trans. Netw.*, vol. 13, no. 3, pp. 462–475, 2005.

[3] R. W. Bickhart, "Transparent TCP-to-SCTP translation shim layer," Master's thesis, University of Delaware, 2005. [Online]. Available: http://www.cis.udel.edu/∼amer/PEL/poc/pdf/BickhartMSthesis.pdf

[4] J. Touch, "TCP Control Block Interdependence," RFC 2140, Internet Engineering Task Force, Apr. 1997.

[5] L. Eggert, J. Heidemann, and J. Touch, "Effects of Ensemble-TCP," *SIGCOMM CCR*, vol. 30, no. 1, pp. 15–29, '2000. [Online]. Available: http://www.isi.edu/ johnh/PAPERS/Eggert00a.html

[6] H. Balakrishnan and S. Seshan, "The Congestion Manager," RFC 3124, Internet Engineering Task Force, Jun. 2001.

[7] H. Balakrishnan, H. S. Rahul, and S. Seshan, "An integrated congestion management architecture for internet hosts," in *SIGCOMM*. New York, NY, USA: ACM, 1999, pp. 175–187.

[8] B. Ford, "Structured streams: a new transport abstraction," in *SIGCOMM*. New York, NY, USA: ACM, 2007, pp. 361–372.

[9] P. Natarajan, F. Baker, and P. Amer, "Multiple TCP Connections Improve HTTP Throughput – Myth or Fact?" in *IEEE IPCCC*, 2009.

[10] B. Ford and J. Iyengar, "Breaking up the transport logjam," in *HotNets-VII*, Calgary, Alberta, Canada, October 2008.

[11] R. Stewart, K. Poon, M. Tuexen, V. Yasevich, and P. Lei, "Sockets API Extensions for Stream Control Transmission Protocol (SCTP)," Internet-draft (work in progress) draft-ietf-tsvwg-sctpsocket-23.txt, Internet Engineering Task Force, Jul. 2010. [Online]. Available: http://tools.ietf.org/html/draft-ietf-tsvwg-sctpsocket-23.txt

[12] F. Niederbacher, "Beneficial gradual deployment of SCTP," Master's thesis, University of Innsbruck, 2010. [Online]. Available: http://heim.ifi.uio.no/michawe/teaching/dipls/florian_niederbacher.pdf

[13] B. Ford and J. Iyengar, "Efficient cross-layer negotiation," in *HotNets-VIII*, New York City, NY, October 2008.

[14] R. Stewart, P. Lei, and M. Tuexen, "Stream Control Transmission Protocol (SCTP) Stream Reconfiguration," Internet-draft (work in progress) draft-ietf-tsvwg-sctp-strrst-05.txt, Internet Engineering Task Force, Aug. 2010. [Online]. Available: http://tools.ietf.org/html/draft-ietf-tsvwg-sctp-strrst-05