Division for Information Technology
Department of Computer Science

Johan Eklund & Anna Brunstrom

Performance of Network Redundancy Mechanisms in SCTP

Karlstad University Studies
2005:48

Johan Eklund & Anna Brunstrom

Performance of Network Redundancy Mechanisms in SCTP

Karlstad University Studies
2005:48

Johan Eklund & Anna Brunstrom. *Performance of Network Redundancy Mechanisms in SCTP.*

Karlstad University Studies 2005:48
ISSN 1403-8099
ISBN 91-7063-019-4

Distribution:
Karlstad University
Division for Information Technology
Department of Computer Science
SE-651 88 KARLSTAD
SWEDEN
+46 54-700 10 00

www.kau.se

Printed at: Universitetstryckeriet, Karlstad 2005

Performance of Network Redundancy Mechanisms in SCTP

Johan Eklund and Anna Brunstrom

Department of Computer Science
Karlstad University
Sweden

{Johan.Eklund | Anna.Brunstrom}@kau.se

Abstract

One of the ambitions when designing the Stream Control Transmission Protocol was to offer a robust transfer of traffic between hosts. For this reason SCTP was designed to support multihoming, which presumes the possibility to set up several paths between the same hosts in the same session. If the primary path between a source machine and a destination machine breaks down, the traffic may still be sent to the destination, by utilizing one of the alternate paths. The failover that occurs when changing path is to be transparent to the application.

This paper describes the results from an experiment concerning SCTP failover performance, which means the time between occurrence of a break on the primary path until the traffic is run smoothly on the alternate path. The experiment is performed mainly to verify the Linux Kernel implementation of SCTP (LK-SCTP) and is run on the Emulab platform. The results will serve as a basis for further experiments.

The experiment is performed in a network without concurrent traffic and in conclusion the results from the experiment correspond well to the values found in other studies and they are close to the theoretical best values. As expected the parameter Path.Max.Retrans has a great impact on the failover time. One observation is that the failover time and the max transfer time for a message are dependent upon the status in the network when the break of the primary path occurs.

1 Introduction

The SIGTRAN working group of the IETF was formed in 1999 [3]. The task of SIGTRAN was to define an architecture for the transporting of real-time signaling traffic over IP networks. For this purpose a new transport protocol, Stream Control Transmission Protocol (SCTP), has been formed. SCTP is developed to offer a reliable, message oriented and multihomed transport service, and is defined in RFC 2960 [12]. The multihoming facility is designed primarily to provide robustness.

To set up a multihomed association between two hosts, each machine will need at least two network interfaces and the paths between these interfaces are in an ideal situation totally physically separated. This makes the paths totally redundant to each other. One of the paths is denoted

primary and as long as there are no problems all traffic will be run on that path. If the primary path breaks down, one of the alternate paths will be set as primary and the traffic between the hosts will continue. This failover is to be as transparent to the application as possible.

When to regard the primary path as broken presents a challenge. If, for example, there is congestion in the network, the concurrent traffic situation may be the reason for timeouts and retransmissions of messages on the path and not a failure. It is not possible for the sending machine to distinguish between retransmission timeouts depending on congestion and timeouts depending on path break. Due to this, reasons other than path failure may cause false failovers if the configuration is to tight. If, on the other hand, the configuration determining when the path is to be regarded broken is to liberal, the transport service will not be able to offer the service demanded by the application. Thus, there is a trade-off when to regard the path as broken and when to accept retransmissions due to congestion.

In the specification of SCTP there is a parameter called Path.Max.Retrans, which is tunable. This parameter indicates how many timeouts to accept before regarding the path as broken and commit a failover to the alternate path.

Another set of tunable parameters are the initial and the limiting values for the retransmission timeout (RTO). The RTO is calculated dynamically throughout the transfer of data based on estimates of the round-trip time, but an initial value as well as a minimum value and a maximum value for the parameter are set. If a timeout occurs the packet is resent and the RTO is doubled. For successive timeouts, this procedure is repeated until the maximum value is reached.

The signaling traffic has to be robust, but it also has demands concerning performance to be able to offer an acceptable service to the application. The demands from telecommunication signaling applications have been stated by the International Telecommunication Union Telecommunication Standardization sector (ITU-T). Their recommendation Q.706 [5] concerning failover time is that the failover procedure is not to take more than 800 ms. There are also some re-commendations concerning the maximum transfer time for a single message over a network. These recommendations are given by ISDN User Part (ISUP) [6] and by Transaction Capabilities Application Part (TCAP) [8]. Their analysis have shown that the maximum transfer time of a single packet for telephony signalling applications are in the range of 600 ms - 1000 ms. RFC 2960 has recommendations concerning both RTO and Path.Max.Retrans, but these recommendations have shown to be far to liberal to fulfill the demands concerning failover time [1] [10].

In this paper we focus on the impact of different values for Path.Max.Retrans on SCTP failover time and max transfer time for an individual message. The implementation of SCTP used is the Linux kernel implementation (LK-SCTP) [2]. The traffic is being sent at a constant rate and no concurrent traffic is sent over the network. The results indicate that the tuning of Path.Max.Retrans is crucial for the performance of SCTP failover. It is also shown that the status of the transfer at the time of the primary path break impacts the failover time as well as the maximum transfer time for a message. The results are compared to the results from a similar study by Grinnemo and Brunstrom [1] and the comparison displays big similarities between the results. The results produced in this experiment will later also serve as reference for further studies of SCTP in more complex traffic environments.

The failover mechanism of SCTP is of interest to many applications which rely on a ro-

bust transport service. The same scenario as the one used for the experiment in this study has also been investigated by Grinnemo and Brunstrom [1]. They performed their study using the TietoEnator-Ericsson implementation of SCTP and used the MTP-L3 User Adaption Layer (M3UA) on top of SCTP. As mentioned above, the results from their study will serve as reference for the results from this study. Jungmaier et al. have also performed a study with many similarities to the study described in this paper. Their study aimed to find parameter settings for SCTP to meet the requirements from telephony signaling. Their study considered an adaptation layer (M2PA) on top of SCTP [10]. Furthermore, Caro Jr. et al. have made simulation studies related to SCTP failover performance. They have, among other, suggested an alternative mechanism to reduce the number of unnecessary retransmissions during failover as an improvement to the existing SCTP failover mechanism [9].

The rest of this paper is structured as follows. In chapter 2 we describe the experimental setup, the network scenario to emulate, the experimental parameters and the data gathering. Chapter 3 describes the experimental results. The chapter starts with a general overview, followed by a detailed analysis of the results where a few specific effects are pointed out. In the final chapter we draw a few conclusions from the results and look forward to future work.

2 Experimental Setup

2.1 Scope

As described previously, the focus of the experiment described in this paper is the impact of different values for Path.Max.Retrans on SCTP failover performance. The scope of the experiment includes four different configurations, where the only varying parameter is Path.Max.Retrans. RFC 2960 recommends 5 as the value for Path.Max.Retrans, but Grinnemo and Brunstrom showed in their study [1] that 5 retransmissions before considering the primary path unusable was too liberal. This recommendation does not fulfill the demands in the ITU-T recommendation which says that the failover time must be less than or equal to 800 ms [7]. Another parameter which could possibly have impact on the failover performance is the path propagation delay, but since the study by Grinnemo and Brunstrom showed this parameter to have marginal impact on the result in an experiment with the configuration used in this experiment, we only use one path propagation delay (20 ms).

The network used in this study consists of two dual homed hosts where one serves as source of the traffic and the other one as sink for the same traffic.

2.2 Network Scenario

The network evaluated in this study is shown in Figure 1.

In this figure source and destination are the two traffic endpoints. Each machine is running an SCTP application on top of the transport protocol SCTP, which is run on top of the IP-protocol. Source serves as traffic generator and is the sending point, while destination is the traffic destination and serves as sink for the SCTP traffic.

The traffic is sent over a network and the session between the endpoints is dual homed. The two paths between the endpoints are fully separated. The traffic is sent over the primary path as

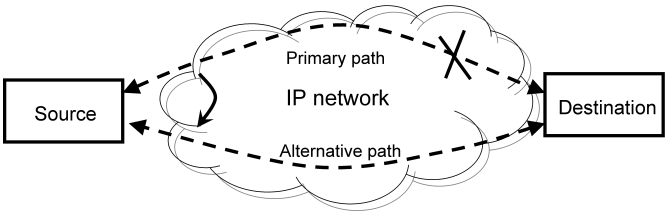

Figure 1: *Evaluated network scenario*

long as there are no problems. When the sending application notices problems on the primary path in the form of timeouts the traffic is resent, but now on the alternate path. The new traffic is still sent on the primary path. This is repeated until a certain number of timeouts, indicated by the parameter Path.Max.Retrans, is reached. At this moment the sending machine regards the primary path unusable and abandons it. From this moment on all traffic is sent on the alternate path, which is now regarded as primary. The failover has occurred.

The evaluation of the failover performance is performed on a controlled experiment setup, see Figure 2, on the Emulab platform [13]. This is an emulated network, which implements the network in Figure 1.

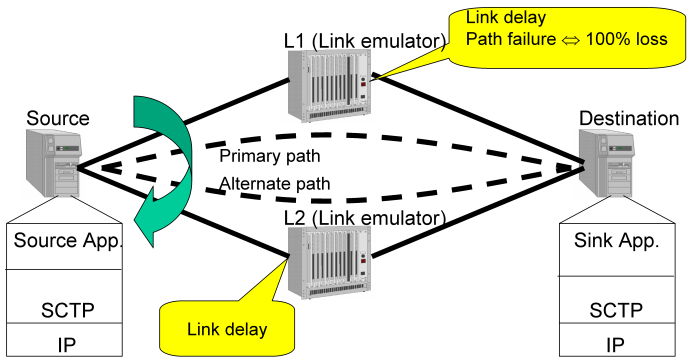

Figure 2: *Experiment setup*

The bandwidth, the loss rate and the propagation delay of both paths between the source and the destination machines are emulated by a machine running Dummynet [11]. The bandwidth

is set to 1 Mbit/second[1], the one way propagation delay to 20 ms, the Dummynet router queue is set to 6 packets and initially the loss rate is 0% on both paths. The path MTU is set to 1500 Bytes. After 8 seconds of traffic the primary path is emulated to break down. This is done by setting the loss rate to 100%. The traffic of each test is ended after 20 seconds, which is more than enough to perform the failover and reach stable traffic on the alternate path.

2.3 Experimental Parameters and Data Gathering

The traffic generating application on source, see Figure 2, paces out messages of size 250 Bytes at a constant rate of 100 messages per second. Tests are run with the parameter Path.Max.Retrans set to 2, 3, 4 and 5 and each test is repeated 40 times, which gives a total of 160 tests. The RTO parameter is initially set to 80 ms, which is also the minimum value for the parameter, while the maximum RTO value is set to 150 ms for all the tests. All tests are automated and are managed by a separate machine running several scripts. These scripts control all parameters and repetitions as well as logging of the traffic.

The session is initiated by the application on the destination machine connecting to the application on source. After setup of the association the application on the source machine starts generating traffic. In all tests, logging takes place both at the application level and at the transport level. For each repetition of the test, both the source and the destination applications are restarted.

The results are calculated and analyzed from the data collected on the application level. The logging on the transport level is used to confirm the behavior seen on the application level.

Every message sent from the source application is timestamped and gets a specific sequence number. When the message arrives at the destination application, the message gets another timestamp. This gives the possibility to easily calculate the maximum transfer time for a message.

The failover time is the time between the failure of the primary path until the path is regarded useless by the sending machine, which then starts sending all data on the alternate path. This time is not possible to calculate exactly in a traffic situation. To estimate the failover time in this test we will use an approximation where the time for the failover is counted between two points of time. The first point is the original send time for the first message that is later resent. This message is sent in close connection to the primary path failure time. The first retransmission will happen as soon as the RTO times out due to the primary path failure. The second point is the time when the message with the longest transfer time is resent on the alternate path. This is the last packet that was originally sent on the primary path during the failover procedure[2]. All the following messages will be sent directly on the alternate path, and will therefor have shorter transfer times. The alternate path is from this point on regarded as primary.

[1] This bandwidth is narrow, but still it is not a restricting factor in this experiment

[2] An alternate way to extract the time for the failover is by listening to notifications from the kernel.

3 Experimental Results

3.1 Overview

Figure 3 and Figure 4 summarize the results from the experiment. The results connected by the solid lines[3] show the sample means of the 40 repetitions and the error bars indicate a confidence interval of 95%.

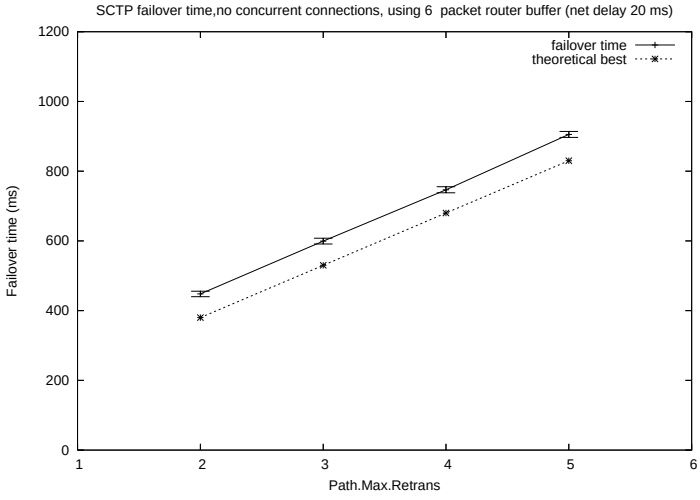

Figure 3: *Failover time vs. Path.Max.Retrans*

In figure 3 the failover times are shown vs. Path.Max.Retrans. It is seen that the failover time is highly dependant on the maximum number of retransmissions and that the increase seems to be linear. Considering the exponential back-off of the retransmission timeout (RTO) the expected increase of the failover time would be exponential. The reason why this experiment shows a linear increase is that the RTO has reached its maximum value (150 ms) already for the second retransmission. In figure 3 there is also a dotted line indicating the theoretical best failover times for the settings. It is possible to calculate this theoretical best failover time from the parameters set in the experiment. The calculation is based on the number of retransmissions that are needed for failover to occur and the minimum possible RTO for each one of these retransmissions. When Path.Max.Retrans is set to 2 this indicates the failover time to be 80 ms +150 ms +150 ms = 380 ms. Likewise when Path.Max.Retrans is 3 the theoretical failover time is 530 ms, for 4 retransmissions it is 680 ms and for 5 retransmissions it is 830 ms. From figure 3 it is easy

[3]There are no values between the discrete values in the pictures, the line between these values are inserted only for visibility purpose.

to see that the mean failover times correspond well to the theoretical best times for all different settings of Path.Max.Retrans.

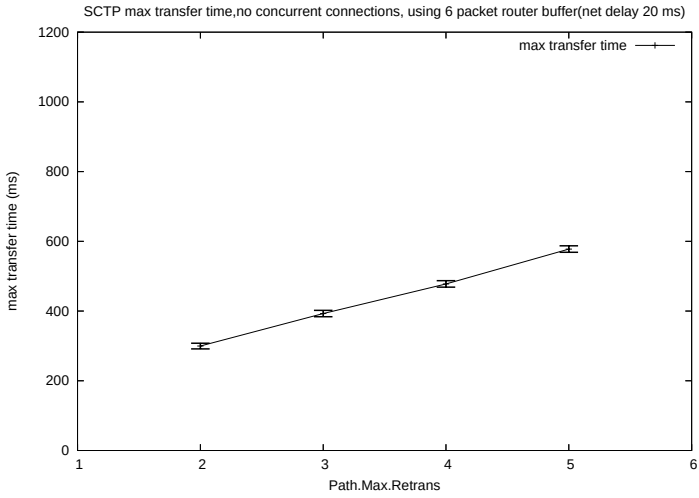

Figure 4: *Max message transfer time vs. Path.Max.Retrans*

In Figure 4 the max transfer times vs. Path.Max.Retrans are shown. One observation from these results is that the increase of the max transfer time, just as the increase in failover time, appears linear and tightly related to Path.Max.Retrans. However, the increase in max transfer time is not as steep as the increase in failover time. The comparison between these values give at hand that the failover times raise faster than the max transfer time for a message. For the max transfer time of a message it is not as easy to talk of a theoretical best value, since such a value is dependant upon the relation between the RTO, the number of retransmissions, the send rate and also the path propagation delay. In this experiment a lower bound for Max Transfer Time is 150 ms (max RTO reached) plus 20 ms (path propagation delay) for all settings.

3.2 Detailed Analysis

A detailed look upon the individual results for failover times indicates that they are not normally distributed around the mean value. They are instead divided into three distinct groups, whereof one result just occurs a few times. The deviation of the times in each group is a maximum of 1 ms. These three different failover times are the only ones noticed among all the 40 repetitions. This pattern is seen for all different values of Path.Max.Retrans. When looking at the individual values for max transfer time for a message there are only 2 distinct groups of times found and the pattern is the same for all settings. These results indicate that there are two or perhaps three

different scenarios behind the different results. When comparing the individual failover times to the individual maximum transfer times there is also a correlation indicating that two of the different groups of times found for failover times are subsets of the same scenario.

Failover						
Path Max Retrans	Scenario A		Scenario B		Scenario C	
	Time (ms)	Number of occurences	Time (ms)	Number of occurences	Time (ms)	Number of occurences
2	422	19	464	14	484	7
3	573	20	622	17	642	3
4	723	23	772	13	792	4
5	874	17	923	17	943	6

Table 1: *Different failover times*

The different groups of failover times are shown in Table 1 together with the number of occurrences for each failover time. In the same way the different groups for maximum transfer times are shown together with the number of occurrences in Table 2. In this table the times for scenarios B and C are merged, since these two scenarios have the same max transfer times.

Noticeable is that the two different max transfer times have almost the same prevalence, which indicates that the underlying scenarios are equally likely to occur.

It is noticeable that the magnitude of the difference between the results for failover times as well as for max transfer times is fairly big. When comparing failover times when Path.Max.Retrans is set to 2, for example, the difference between the failover time in scenario A and the failover time in scenario C is almost 15%. The difference in the max transfer time between the different scenarios is of the same magnitude.

Max Transfer Time				
Path Max Retrans	Scenario A		Scenario B and C	
	Time (ms)	Number of occurences	Time (ms)	Number of occurences
2	272	19	324	21
3	363	20	422	20
4	452	23	512	17
5	543	17	603	23

Table 2: *Different max message transfer times*

ther major scenario, scenario B, is a scenario where the last SACK sent from the destination

machine does not reach the source machine. This scenario is illustrated in Figure 6. In this case, 2 messages (number 54 and 55) that have already reached the destination will be unnecessarily retransmitted after the first timeout, since no SACK has arrived for these messages. In this case the destination machine waits for two packets to arrive before sending a SACK, since the last event before the primary path failure, from the perspective of the destination, was that a SACK was sent. This implies that also after the retransmission on the alternate path, some data will not be SACK'ed by the destination until the next timeout. This situation will, as a chain reaction, lead to further unnecessary retransmissions. This situation will prolong the failover time in scenario B.

When analyzing the traffic on the network level it is found that the different results depend on the state of the transmission at the moment the primary path is broken. What happens when the primary path is broken is that along this path no more data reach its destination and no returning SACK's reach the source machine to verify that a message has reached its destination. In this experiment we use the default behavior of the implementation, where a SACK is sent from the destination machine to the source after 2 messages have arrived or 200 ms has elapsed, whichever occurs first. The different scenarios shown in the tables above are results from the different status in the network at the time of failure.

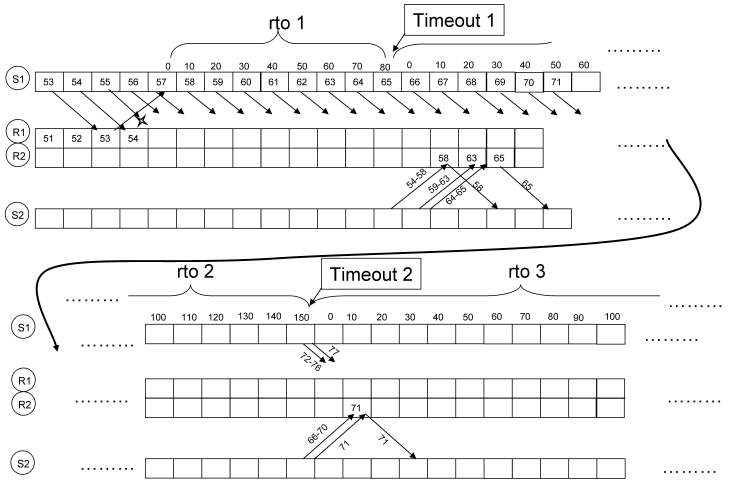

Figure 5: *Visualization of traffic scenario A*

Figures 5 through 7 are attempts to visualize the different traffic scenarios found in this experiment. The figures illustrate the traffic sequence between the source and destination machines. (To make it possible to show these figures in this paper the figures are split into two parts connected by the long curved arrow).

In the figures the closely connected vectors represent the destination machine interface 1 (R1) and interface 2 (R2), the topmost vector represents the source machine interface 1 (S1), whereas the bottom vector represents source machine interface 2 (S2).

The numbers in the vectors represent transport sequence numbers for the messages. The digits above the topmost vector, surrounded by the curly bracket indicate the timer time (in ms) elapsing. The arrows indicate packets sent over the network, the ones sent towards the middle vectors are the data packets, and the ones sent in the other direction are the SACK's. The star in the transfer is a symbol for the path break, which implies that no packets can reach its destination on this path after that moment.

The case in Figure 5 shows scenario A, where the destination machine (R1) is waiting for another message before sending the next SACK when the path failure occurs. This scenario has the shortest failover times.

In scenario A, it is seen that the first retransmission, after 80 ms, includes one message that has already reached the destination (number 54), but for which no SACK has been sent. This first packet contains 5 bundled messages (since the path MTU is 1500 bytes) and it leads to a SACK, since it is the second unacknowledged packet that reach the destination (the first packet arrived on interface D1). Since the retransmission consists of a total of 12 messages, which will be sent in 3 separate packets, the last packet with retransmitted data leads to another SACK. This implies an empty congestion window at the source machine at the time of the next retransmission, after another 150 ms.

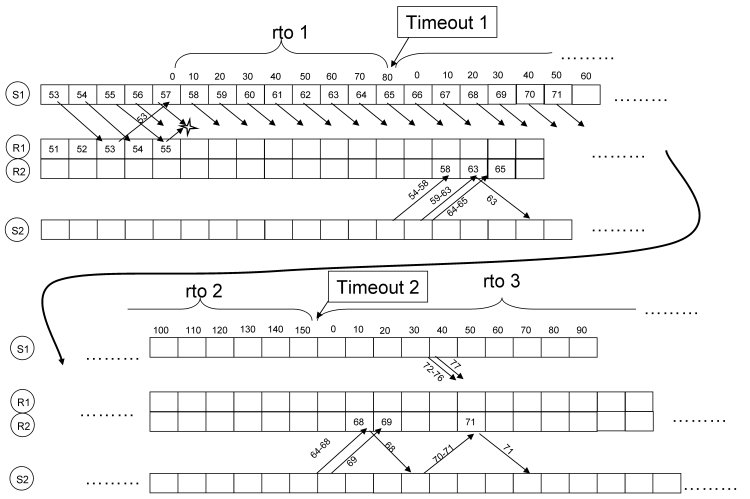

Figure 6: *Visualization of traffic scenario B*

The other major scenario, scenario B, is a scenario where the last SACK sent from the

destination machine does not reach the source machine. This scenario is illustrated in Figure 6. In this case, 2 messages (number 54 and 55) that have already reached the destination will be unnecessarily retransmitted after the first timeout, since no SACK has arrived for these messages. In this case the destination machine waits for two packets to arrive before sending a SACK, since the last event before the primary path failure, from the perspective of the destination, was that a SACK was sent. This implies that also after the retransmission on the alternate path, some data will not be SACK'ed by the destination until the next timeout. This situation will, as a chain reaction, lead to further unnecessary retransmissions. This situation will prolong the failover time in scenario B.

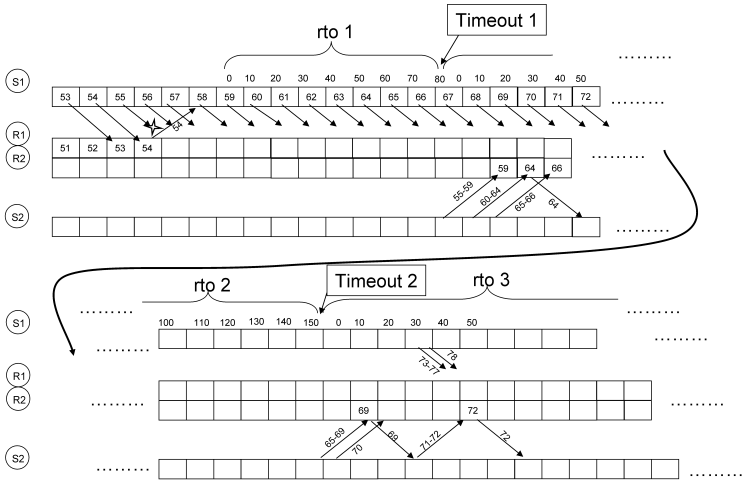

Figure 7: *Visualization of traffic scenario C*

Scenario C, shown in Figure 7 occurs just a few times. This scenario is a variant of scenario B above. The difference here is that in scenario B the SACK does not reach the source machine before the path failure, but in scenario C the SACK proceeds all the way to the source machine. The traffic situation in scenario B and C will later in the failover procedure cause more unnecessary retransmissions compared to scenario A.

The different failover times between the scenarios come up due to the delayed SACK, which causes the destination to wait for two packets before sending a SACK. This mechanism also causes the different maximum transfer time for a message between scenario A and the other scenarios. The delayed SACK causes the same effect on scenario B and C in aspect of maximum transfer time for a message.

The different scenarios affect the failover times and among these scenarios the optimal situation is that one message has reached the destination after the SACK has left. This will trigger the

coming retransmitted traffic to be compatible with the SCTP implementation and the path MTU for sending SACKs from the destination. This compatibility will cause the congestion window to be free to retransmit all the messages that have not been SACK'ed at once. In the other scenarios the retransmission is delayed since the congestion window is not empty. Scenario B and scenario C suffer from this delay and due to this they both show a longer failover time as well as a longer max transfer time for a message compared to scenario A.

The scenarios seen in the experiment are all results of the SACK delay which causes the destination to wait until two packets have arrived before sending a SACK. The differences between the scenarios depend upon the configurations for these specific tests. There are dependencies upon the path MTU (1500 Bytes), the size of the messages and on the traffic intensity. Furthermore, the implementation of the SCTP protocol in the LK-SCTP-implementation may also have an impact on the results. The reason for scenario A to have the shortest failover times is that the number of messages and the size of the messages matches the path MTU in such a way that all resent messages are SACK'ed in the same retransmission cycle. In scenario B and scenario C not all resent messages are SACK'ed in the same cycle which causes messages to timeout and as a chain reaction even these messages are resent. If every single packet reaching the destination would have been SACK'ed at once these three scenarios would not have occured in the same way.

In a real situation there could of course be different variants of the scenarios found in this experiment. Nevertheless, the three scenarios found here indicate the importance of the status in the network at the time of the failure for the failover performance as well as for max transfer time for a message.

3.3 Comparison

A similar experiment as the one in this report has been performed by Grinnemo and Brunstrom [1] using another implementation. It is possible to compare the results from these experiments even if the situations are not exactly the same. The comparison concerning failover times is shown in figure 8.

In this figure the horizontal dotted line indicates the maximum acceptable failover time, interpreted from the specification of signaling system No.7, by ITU-T [7]. When looking at the figure one should keep a few differences in mind. The first is that the one way path propagation delay is 20 ms in this experiment and 10 ms in the experiment by Grinnemo and Brunstrom. Nevertheless, the results should be comparable since Grinnemo and Brunstrom noticed the propagation delay to have minor impact on the result under the configuration used in this experiment. The reason for not comparing the same propagation delays is that Grinnemo and Brunstrom's experiment was not made with all values of Path.Max.Retrans for a 20 ms propagation delay. The second difference to remember is that the measurements are probably performed in different ways. A third component to have in mind is that the implementation used by Grinnemo is a user space implementation while the implementation in this experiment is a kernel implementation. Even with these differences in mind it is seen that the failover times are similar and that the results diverge a little as the Path.Max.Retrans increases. It is seen that in this experiment a Path.Max.Retrans of 4 is acceptable with respect to the ITU-T limit, while 3 seems to be the maximum in Grinnemo and Brunstrom's case.

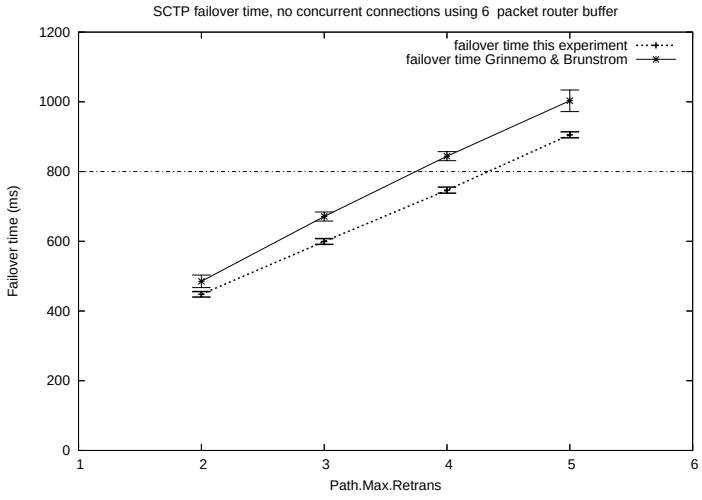

Figure 8: *Comparison failover time vs. Path.Max.Retrans*

In figure 9 the comparison for max transfer times is shown. For the max transfer time it is hard to find an exact upper bound for what could be accepted. The two dotted horizontal lines in the figure indicate an interval in which the maximum acceptable transfer time for a single message would fall, according to the recommendations by ISUP [6] and TCAP [8]. It is shown that the results converge as Path.Max.Retrans increases, in contrast to the failover times, and that in both experiments at least a value of 4 for Path.Max.Retrans is below the indicated interval.

4 Conclusions and Future Work

4.1 Conclusions

In the experiment presented in this paper the focus has been on SCTP failover performance. The experiment has been performed in an isolated network, run on the LK-SCTP implementation, without concurrent traffic. The traffic has been sent at a constant rate in a single threaded, multihomed SCTP association.

The conclusions to be drawn from this experiment are that under these optimal conditions the LK-SCTP implementation of SCTP performs well. It is seen that the setting of the parameter Path.Max.Retrans has a major impact both on the failover time and the max transfer time for a single message. These results correspond well to the results found in other studies. Another factor that was found to have impact on the failover performance is the state in the network at the time of the primary path failure.

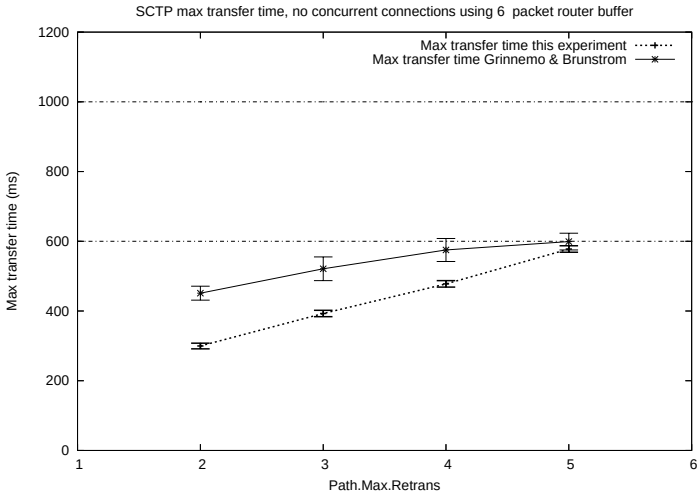

Figure 9: *Comparison max message transfer time vs. Path.Max.Retrans*

The results from the experiment have been compared to similar a study by Grinnemo and Brunstrom [1], performed on another SCTP-implementation. This comparison indicates that the LK-SCTP kernel implementation performs slightly better than the implementation used by Grinnemo and Brunstrom. However, the results do not differ that much between the implementations, and since the situations are not exactly the same no evident conclusions can be drawn.

4.2 Future Work

The analysis of the results from the experiment in this study showed the occurrence of three different scenarios depending on the status of the traffic in the network at the time of failure. These three scenarios occur due to the implementation of the delayed SACK. To further investigate the impact of this parameter additional experiments with different values for the SACK timer will be performed and analyzed.

The experiment presented in this paper has been performed under ideal conditions. The traffic has been sent at a constant rate, the router buffers in Dummynet have been big enough and no concurrent traffic has been present in the network. Due to this no traffic has been lost due to reasons other than path failure. In a real network there is almost always concurrent traffic competing for bandwidth and the traffic load causes congestion. In some situations the router buffers will be overloaded and some packets are lost. The signalling traffic, which is of interest for SCTP, is typically sent as bursts of messages rather than at a constant bit rate.

Therefore a natural continuation of this experiment is to introduce bursty traffic and concur-

rent traffic in the network. For such experiments the results from the experiment presented in this paper will serve as a valuable reference.

All the above mentioned experiments are first to be emulated to get a controlled environment and to be able to repeat the individual experiments. In a more distant future an experiment in a real network could also be possible, perhaps by using PlanetLab [4].

References

[1] K-J. Grinnemo and A. Brunstrom. Performance of SCTP-controlled Failovers in M3UA-based SIGTRAN Networks. In *Proceedings of Advanced Simulation Technologies Conference 2004 (ASTC'04)*, Hyatt Regency Crystal City, Arlington, Virginia, USA, April 2004.

[2] http://lksctp.sourceforge.net/. SCTP for the Linux Kernel, 2005.

[3] http://www.ietf.org/. IETF Home Page, 2005.

[4] http://www.planet lab.org/. Planet Lab Home Page, 2005.

[5] ITU-T. Q.706: Signalling System No.7 - Message Transfer Part Signalling Performance. *ITU-T*, March 1993.

[6] ITU-T. Q.771: Signalling System No. 7 - Functional Description of Transaction Capabilities. *ITU-T*, June 1997.

[7] ITU-T. Q.707: Signalling System No. 7 -Message Transfer Part Signaling Performance. *ITU-T*, June 1999.

[8] ITU-T. Q.761: Signalling System No. 7 - ISDN user part functional description. *ITU-T*, December 1999.

[9] A. Caro Jr., J.R. Iyengar, P. Amer, G. Heinz, and R. Stewart. Using SCTP multihoming for fault tolerance and load balancing. *SIGCOMM Computer Communnication Review*, 32(3):23–23, 2002.

[10] A. Jungmaier, E. Rathgeb, and M. Tuexen. On the Use of SCTP in Failover Scenarios. In *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics*, pages 363–368, Orlando, US, July 2002.

[11] L. Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, January 1997.

[12] R. Stewart, Q. Xie, K.Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rythina, M. Kalla, L. Zhang, and V. Paxson. RFC 2960: Stream Control Transmission Protocol. Internet Engineering Task Force, October 2000.

[13] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation, pages 255-270*. Boston, MA, USA, December 2002.

Performance of Network Redundancy Mechanisms in SCTP

One of the ambitions when designing the Stream Control Transmission Protocol was to offer a robust transfer of traffic between hosts. For this reason SCTP was designed to support multihoming, which presumes the possibility to set up several paths between the same hosts in the same session. If the primary path between a source machine and a destination machine breaks down, the traffic may still be sent to the destination, by utilizing one of the alternate paths. The failover that occurs when changing path is to be transparent to the application.

This paper describes the results from an experiment concerning SCTP failover performance, which means the time between occurrence of a break on the primary path until the traffic is run smoothly on the alternate path. The experiment is performed mainly to verify the Linux Kernel implementation of SCTP (LK-SCTP) and is run on the Emulab platform. The results will serve as a basis for further experiments.

The experiment is performed in a network without concurrent traffic and in conclusion the results from the experiment correspond well to the values found in other studies and they are close to the theoretical best values. As expected the parameter Path.Max.Retrans has a great impact on the failover time. One observation is that the failover time and the max transfer time for a message are dependent upon the status in the network when the break of the primary path occurs.

Karlstad University Studies
ISSN 1403-8099
ISBN 91-7063-019-4