# Optimizing PR-SCTP Performance using NR-SACKs

Mohammad Rajiullah, Anna Brunstrom
Department of Computer Science, Karlstad University
SE-651 88 Karlstad, Sweden
Email: {mohammad.rajiullah | anna.brunstrom}@kau.se

***Abstract*—A partially reliable extension of SCTP, PR-SCTP, has been considered as a candidate for prioritizing content sensitive traffic at the transport layer. PR-SCTP offers a flexible QoS trade-off between timeliness and reliability. Several applications such as streaming multimedia, IPTV transmission, and SIP signaling have been shown to benefit from this. Our previous work, however, suggests that the performance gain can be very much reduced in a network with competing traffic. One of the most important factors in this case is the inefficiency in the *forward_tsn* mechanism in PR-SCTP. In this paper, we thoroughly examine the *forward_tsn* inefficiency and propose a solution to overcome it that takes advantage of the NR-SACKs mechanism available in FreeBSD. Moreover, we implement and evaluate the proposed solution. Our initial set of results show a significant performance gain for PR-SCTP with NR-SACKs. In some scenarios the average message transfer delay is reduced by more than 75%.**

***Index Terms*—SCTP; PR-SCTP; NR-SACKs; performance evaluation**

## I. Introduction

The transport layer sits above the network layer in the Internet layering architecture. TCP and UDP have been the most common transport protocols at this layer for more than two decades. A new transport protocol named Stream Control Transmission Protocol (SCTP) [1] was defined by the IETF Signaling Transport (SIGTRAN) working group in 2000. The original target application for SCTP was to transport SS7 signaling traffic over IP. SCTP can, however, also be used as a general purpose message based transport protocol. In addition to sharing common transport functionalities of TCP such as flow control, reliability, and congestion control, SCTP's support of some advanced features like multi-streaming, partial ordering, and multi-homing opens up possibilities for a whole new range of applications.

TCP and basic SCTP provide all the functionalities for reliable delivery of application data. On the contrary, UDP is unreliable for application data. Apart from using full reliability or no reliability, some applications such as real time multimedia may benefit from the more fine grained QoS trade-off offered by partial reliability. During congestion periods, very high delays or jitter from a loss recovery may result in poor performance for this type of applications. A limit on reliability for each application data unit is necessary. For example, if applications generate data with different priorities, it is possible to reduce high delays by considering only high priority data during a loss recovery.

Moreover, partial reliability might be advantageous for time sensitive data that are only useful for a limited time period. In this case, avoiding (re)transmission of stale data may not only improve application performance but also save network resources for fresh and crucial data. Pioneer work of Dempsey [2] and Papadopoulos and Parulkar [3] showed the feasibility of retransmission based partial reliability for multimedia traffic.

The message based abstraction in SCTP makes it easy to realize partial reliability for application data. RFC 3758 [4] introduces a Partial Reliability extension of SCTP, PR-SCTP. Since PR-SCTP provides partial reliability on a per message granularity, it is possible to define various reliability policies for different application data units. Several applications, such as streaming of real time video and audio, have been shown to benefit from PR-SCTP [5–8]. In our earlier work [9], we, however, showed that this gain from PR-SCTP can be very limited in a network with competing traffic. In such a network, several factors influence the PR-SCTP performance. One of the most important factors is an inefficiency in the Forward Cumulative TSN (*forward_tsn*) mechanism of PR-SCTP when application messages are lost in bursts.

In this paper, we analyze the *forward_tsn* inefficiency in PR-SCTP in greater detail for several traffic characteristics. Moreover, as further contributions, we propose and also implement a solution that utilizes Non-Renegable Selective Acknowledgements (NR-SACKs) [10, 11]. NR-SACKs can selectively acknowledge out-of-order but non-renegable data. The NR-SACKs mechanism is already available in FreeBSD; we modify the PR-SCTP sender implementation so that it uses the information from NR-SACKs to get a more efficient *forward_tsn* mechanism. The results after this optimization show significant improvements. Our results reveal that in some scenarios the average message transfer delay is reduced by more than 75%.

The remainder of the paper is organized as follows. Background and related work are described in Section II. In Section III, the *forward_tsn* inefficiency is examined in detail. In Section IV, our proposed NR-SACKs based optimization for solving the inefficiency in the *forward_tsn* mechanism is described. Further, the implementation and performance evaluation of our solution are also described in this section. Finally, we give some concluding remarks in Section V.

## II. Background

PR-SCTP accommodates all the transport functionalities in SCTP. Its message based abstraction allows preserving application message boundaries. Within a SCTP packet, specific building blocks called *chunks* are used to carry application messages and control information. Application messages are carried in data chunks whereas the control information for a SCTP association is carried in a number of different control chunks. If several of these chunks when combined are less than the maximum transfer unit (MTU) in length, these can be bundled into a single SCTP packet. This chunk based format especially facilitates extensibility in SCTP.

PR-SCTP introduces functionalities for partial reliability in SCTP by allowing an association to choose (re)transmission policy on a per message basis [4]. This policy is decided at the application layer. *Timed reliability* is such a policy. In this policy, when forwarding a message to the transport layer, an application adds a content specific certain lifetime value to it. Upon expiration of this lifetime value, PR-SCTP simply abandons this message. No (re)transmission attempt is made. This is quite effective for time sensitive traffic which is valuable only for a certain time period. Delays from loss recovery are now only for fresh and important data. Network resources are saved from stale and meaningless data transportation. Overall, this enhances application performance. Moreover, using PR-SCTP, it is possible to multiplex both reliable and unreliable data over a single association which ultimately eliminates the necessity of using two separate protocols. In any case, a PR-SCTP receiver is completely unaware about any particular prioritization policy employed at the sender side.

PR-SCTP uses a special control chunk called *forward_tsn* to provide partial reliability. When a message is abandoned, the PR-SCTP sender informs this to the receiver by issuing a *forward_tsn* chunk. This chunk tells the receiver to forward its cumulative ACK point and not to expect that abandoned message any further. Although PR-SCTP may not retransmit a lost message, it adjusts all the congestion related variables to be fair with other competing flows in the network.

Several researchers have shown performance gain of PR-SCTP for applications such as streaming multimedia [5–8], IPTV transmission [12] and SIP signaling [13]. Nevertheless, our previous work [9] shows that in some situations this gain can be rather limited. This is especially the case when a PR-SCTP flow shares a bottleneck link with competing traffic. There are several contributing factors behind this limited performance. Firstly, if an application generates small messages, overhead per byte is increased. Secondly, several small messages may lead to less than full-sized packets when they are bundled before transmission. In case of packet based buffering in the network, loss rate per application bytes is increased compared to the flows using full-sized packets. Lastly, the inefficiency in the existing *forward_tsn* mechanism may limit the performance when network resources are shared. The existing *forward_tsn* mechanism in PR-SCTP becomes particularly inefficient for bursty losses of a mix of reliable and unreliable messages. This is particularly common when several small messages are bundled into a single SCTP packet and dropped in the network. Based on our findings in [9], we in this work further analyze the *forward_tsn* inefficiency in PR-SCTP. We, in addition, propose a NR-SACKs based optimization to mitigate this problem and also implement and evaluate our proposed solution.

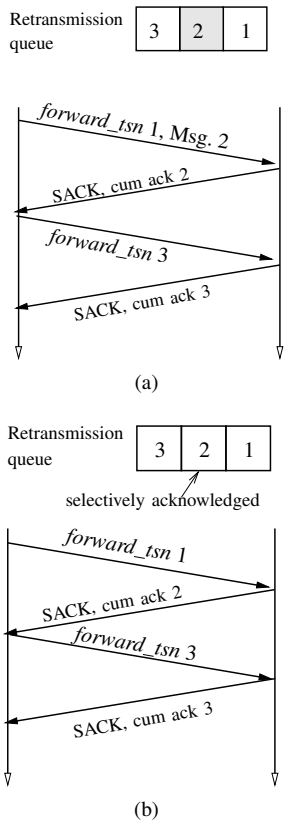



Fig. 1: *forward_tsn* inefficiency in the existing PR-SCTP. (a) Scenario 1: continuous message loss pattern. (b) Scenario 2: non-continuous message loss pattern.

## III. Evaluation of *forward_tsn* Inefficiency

In this section, we first describe the *forward_tsn* inefficiency. Then we discuss our experiment setup and the results from the evaluation of the *forward_tsn* inefficiency.

### A. *forward_tsn Inefficiency*

We will consider two loss scenarios to illustrate the inefficiency in the existing *forward_tsn* mechanism. In the first loss scenario, a large number of messages are lost in sequence. This could be either because loss in the network is bursty or because several small messages are bundled into a single packet that is lost. Consider a simple scenario where a PR-SCTP sender has three messages that have all been lost in its retransmission queue. If all these three messages are unreliable due to the expiration of their lifetime, then PR-SCTP is very efficient. It needs a single *forward_tsn* to update the receiver's cumulative acknowledgement point to the third message. However, if the

three messages are of mixed reliability, then the *forward_tsn* mechanism becomes slow. For instance, consider the retransmission queue in Fig. 1a. Message number 2 is the only reliable message and requires retransmission. In such a case, PR-SCTP first sends a *forward_tsn* for message number 1 and bundles message number 2 with it. Then, PR-SCTP must wait for the acknowledgement with cumulative acknowledgement up to message number 2 before it can send a *forward_tsn* for message number 3. At any point during transmission, a sender can only issue a *forward_tsn* for a particular message, if all the preceding reliable messages are cumulatively acknowledged. This makes PR-SCTP slow as compared to SCTP. Since SCTP is a fully reliable protocol, it can bundle all three messages together and retransmit. Loss recovery is at least one round trip time faster.

Depending on the number of messages that can be bundled into a single packet and the distribution of reliable and unreliable messages in the packet, the delay from the inefficient *forward_tsn* mechanism grows as the number of lost messages in sequence grows. In this case, if the network resources are dynamic, application performance is considerably punished.

In the second loss scenario, packets are lost non-continuously. In this case, several packets that are close to each other, but not in sequence, are lost due to high loss rate. Consider Fig. 1b where there are again three messages in the retransmission queue because none of them have been cumulatively acknowledged. However, message number 2 has been selectively acknowledged. Message number 1 and 3 have been lost in the network. In this case, even if all three messages are unreliable, the sender will not send a *forward_tsn* for message number 3. This happens as message number 2 has been selectively acknowledged and is therefore no longer considered for retransmission or abandonment through the *forward_tsn* mechanism. Message number 2 must be cumulatively acknowledged before sending a *forward_tsn* for message number 3. So, again the loss recovery is one round trip time longer than in the case when all messages are reliable and can be retransmitted all together. Furthermore, this delay from the inefficient *forward_tsn* mechanism grows as this non-continuous loss sequence grows.

### B. Experiment Setup

We adopt a single bottleneck, emulation based experiment setup to investigate the *forward_tsn* inefficiency. All the experiments are carried out in a local LAN test bed. Fig. 2 shows the setup we use. All of the nodes have the same hardware configuration of 4 GB RAM and an Intel Core 2 duo processor (2.6 GHz). Both end machines are configured with FreeBSD 8.2. As shown in Fig. 2, the client and server machines are connected to each other through a network emulator. This emulator is configured with the Dummynet traffic shaper [14], which is used to set the buffer size, the delay, and the bandwidth in the network. Bandwidth (up and down) is limited to 10 Mbps in the experiment network. Besides, one way delay is kept at 40 ms and the buffer size at the network is set to 73 KB which corresponds to Dummynet's default buffer setting

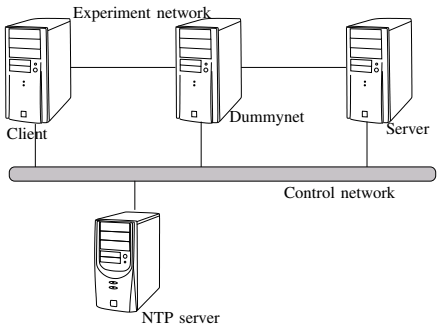



Fig. 2: Client-server based network setup used in the experiments.

of 50 packets. Byte based buffering is used in the experiments. This alleviates the sharing inefficiencies that may occur due to small packets [9] when packet based buffering is used.

In the experiments, we vary the message sizes. The number of messages that are bundled in a packet varies as the message size varies. This means that the number of messages that are lost in sequence as a result of a single packet loss also varies with different message sizes. Moreover, in some experiments, we introduce variability in the message sizes by using normally distributed message sizes with a standard deviation of 20% of the mean message size. Additionally, the fixed message sizes are selected in such a way that full-sized packets can be formed when messages are bundled. Moreover, the largest fixed message size is limited to 716 bytes, since messages of larger sizes cannot be bundled for the MTU size of 1500 bytes considered in the experiments. Although bursty loss is possible in the network scenario under investigation, the *forward_tsn* inefficiency will likely be most visible when several messages are bundled into a single packet that is lost in the network.

During each run of the experiment, there are four background greedy TCP flows in the network. These flows are network limited. Therefore, when we start a PR-SCTP flow, the fair share for every flow is 2 Mbps. We vary the application send rate as a fraction of this share to generate several loads. However, we limit the send rate to values that are sustainable for the network over time as the timed reliability service in PR-SCTP is not intended for bulk traffic. A queue may still build up temporarily in the send buffer due to congestion control. A queue in the send buffer causes the sender to bundle several messages. Moreover, in each experiment run, the server sends about 14 Mbytes of data.

We create a PR-SCTP based application that generates messages according to a Poisson arrival process. In the experiments, the application generates messages of different priorities: important messages and normal messages. We use several fractions of important messages starting from 1% to 100%. In these experiments, we use a timed reliability based PR-SCTP policy. We use a time to live (TTL) of 5000 ms for important messages and a TTL of 100 ms for normal messages. For practical delays, a TTL of 5000 ms means that

important messages in effect are reliable. Besides, PR-SCTP uses unordered delivery as many applications such as syslog [15] generate semantically independent messages. In addition, we perform 30 repetitions for each experiment to allow for 95% confidence intervals.

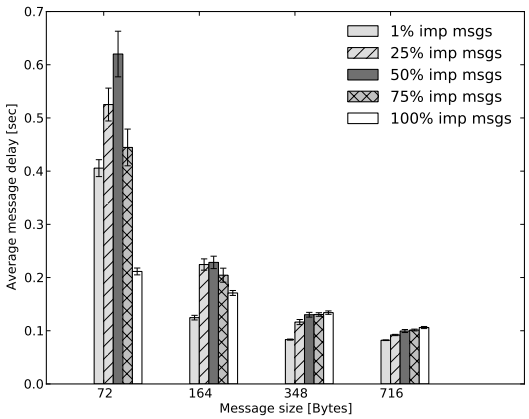
### C. Results

In all the experiments, we consider the message transfer delay as the performance metric, since as seen in the background section, most applications that have considered PR-SCTP are data limited. Message transfer delay is a typical choice in this case. This delay is calculated as the time difference between when a message is generated at the sender application and then received at the receiver application. We only present the results of the average message delay for important messages, since the results for normal messages are very similar.

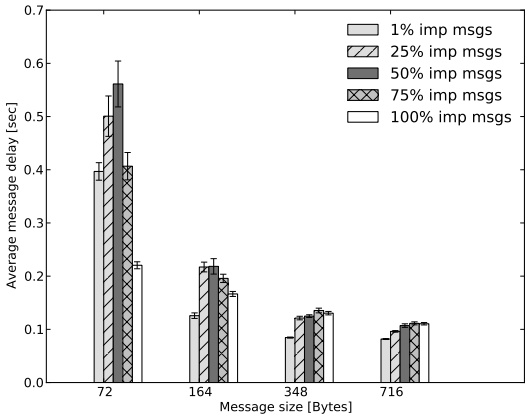
Fig. 3a shows the average message delay comparison of several PR-SCTP flows with different important message fractions and different fixed message sizes. The send rate for each PR-SCTP flow is kept to a value such that on an average it uses only 30% of its fare share capacity. A PR-SCTP flow with 100% important messages can be generally regarded as a standard SCTP flow. In this figure, we see that as the message size increases, average delay decreases. When the message sizes are small, many messages can be bundled into a single packet. In consequence, when such a packet is lost we get a long burst of lost messages. Since only a fraction of all messages are important, this results in a mix of reliable and unreliable messages being lost. Due to this mix, several separate *forward_tsn* chunks need to be sent to the receiver for a single packet loss which increases the delay as described in Section III-A.

Fig. 3a shows that when the message sizes are as small as 72 bytes and the fraction of important messages is 50%, then the average message delay becomes the highest among all the results. In this case, we have the highest possibility of both the largest sequence of lost messages and an even distribution of reliable and unreliable messages. This effect gradually decreases as the message size increases or the fraction of important messages changes. This clearly illustrate the inefficiency in the existing *forward_tsn* mechanism.

Fig. 3b also shows the average message delay comparison for different message sizes and fractions of important messages. In this case, the message sizes represent mean message sizes, since the size of each message is drawn from a normal distribution. The resulting variability in message size may cause less than full-sized packets. However, as packet sizes are taken into account in byte based buffering, we do not see any significant change in the results. In both figures, we see that the inefficiency in the *forward_tsn* mechanism causes notable differences in delay as the fraction of important messages changes. This is especially clear in the case of a message size of 72 bytes. Besides, small messages increase the overhead due to the use of more chunk headers. This effect is clearly shown in both Figs. 3a and 3b for 100% important messages.




(a) Fixed message sizes



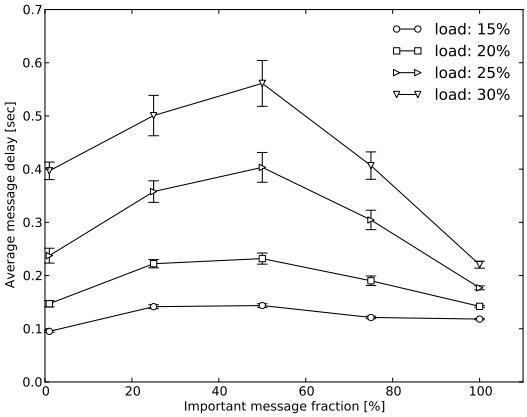

(b) Normally distributed message sizes

Fig. 3: Average delay comparison at 30% load.

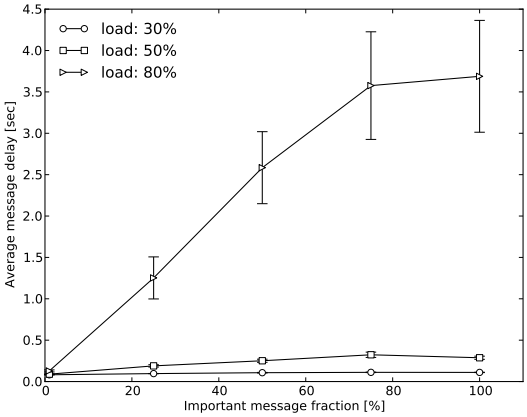
Fig. 4 illustrates the impact of load on the results. Since our application send rate is always a fraction of its 2 Mbps fare share capacity, load is expressed as a percentage. Recall that the PR-SCTP flow and the four background flows share a bottleneck link of 10 Mbps capacity. Further, as the results for fixed and normally distributed message sizes are similar, we only display the results for the more realistic case of normally distributed message sizes.

Fig. 4a shows the effect of load on the performance for a mean message size of 72 bytes. This is the case in which the largest number of messages can be bundled. The results show that until the load becomes too small to generate significant packet loss, as seen for 15% load, the effects from the inefficient *forward_tsn* mechanism are clear. In every case except the 15% load case, average delay reaches considerably higher values when on an average half of all messages are important.

Fig. 4b shows the effect of load on the performance for a mean message size of 716 bytes. This size is chosen to see the effect on performance for larger message sizes. In




(a) Mean message size: 72 bytes




(b) Mean message size: 716 bytes

Fig. 4: Average delay comparison for different loads.

this case, any two messages of size larger than 716 bytes, which can be produced due to variability, cannot be bundled. The results show that the average message delay increases as the fraction of important messages increases and PR-SCTP needs to retransmit more and more messages. It is particularly noticeable at higher loads, since higher load generates more loss. In the 80% load case, loss becomes very high. In this case, PR-SCTP can hardly keep up with the application send rate and the average message delay increases very rapidly as the fraction of important messages increases. In the extreme case where the network cannot keep up with the application send rate, the average message delay depends on the amount of data transmitted which gives the very large delays seen in the figure. However, the effect of the inefficient *forward_tsn* mechanism is very limited due to the absence of significant burst loss. Each PR-SCTP packet now typically contains only one or two application messages.

## IV. Proposed NR-SACKs based Optimization for PR-SCTP

In our previous work [9], we mentioned a possible mitigation that requires an extension of the existing *forward_tsn* chunk to solve the inefficiency in the *forward_tsn* mechanism. Implementing this solution, however, not only requires a modification of the existing *forward_tsn* chunk but also requires both sender and receiver side adaptation to use the modified *forward_tsn* chunk. We therefore propose an alternative solution that is a NR-SACKs based mitigation to enhance the inefficiency of the *forward_tsn* mechanism in PR-SCTP. NR-SACKs are already available in the current FreeBSD kernel and are in line with ongoing IETF standardization work. In this section, we describe NR-SACKs and our NR-SACKs based optimization for PR-SCTP. Furthermore, we describe the implementation and performance evaluation of this optimization.

### A. Non-Renegable Selective Acknowledgements (NR-SACKs)

The Selective Acknowledgement (SACK) mechanism allows efficient loss recovery for both TCP and SCTP. It not only cumulatively acknowledges the most recent data that arrive in order, but also selectively acknowledges those data that arrive out of order at the transport receiver. These out-of-order data are, however, implicitly renegable. A receiver can later discard any selectively acknowledged data. This means that selectively acknowledged information can only be used as advisory for intelligent retransmission and thus a sender must keep every data it sends unless cumulatively acknowledged.

TCP is only allowed to send in-order data to the application. On the contrary, SCTP can provide the out-of-order data delivery service. In such a case, data that has been delivered to the application layer is no longer renegable by a transport receiver. The original SACK mechanism, however, does not provide any distinction of selectively acknowledged data that have been delivered to the application and are non-renegable as compared to selectively acknowledged data that have not yet been delivered to the application and are renegable. NR-SACKs allow a transport receiver to explicitly inform a transport sender that all or some of the selectively acknowledged data are non-renegable [10, 11]. If PR-SCTP is modified to use information from NR-SACKs, the inefficiency in the *forward_tsn* mechanism can be partially solved.

### B. NR-SACKs based Optimization

In all of our experiments, PR-SCTP uses an unordered message delivery service. Intuitively, PR-SCTP messages with various priority levels are often semantically independent and thus can be treated independently at the application layer. Unordered messages, therefore, can be immediately delivered to the application layer. In this case, NR-SACKs can be used to tell a PR-SCTP sender about the non-renegability of out-of-order messages at the receiver. A PR-SCTP sender can therefore use the non-renegability information from NR-SACKs to use the *forward_tsn* mechanism more effectively. For instance, Fig. 5a shows a representation of information

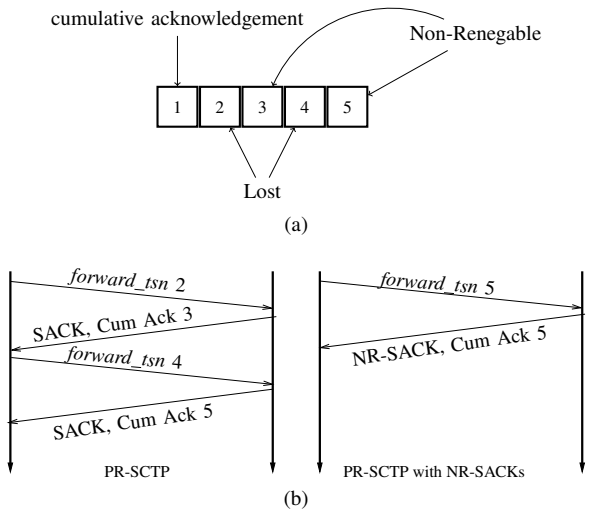



Fig. 5: (a) Case 1: Information extracted from a NR-SACK chunk. (b) Improvement of PR-SCTP using NR-SACKs.

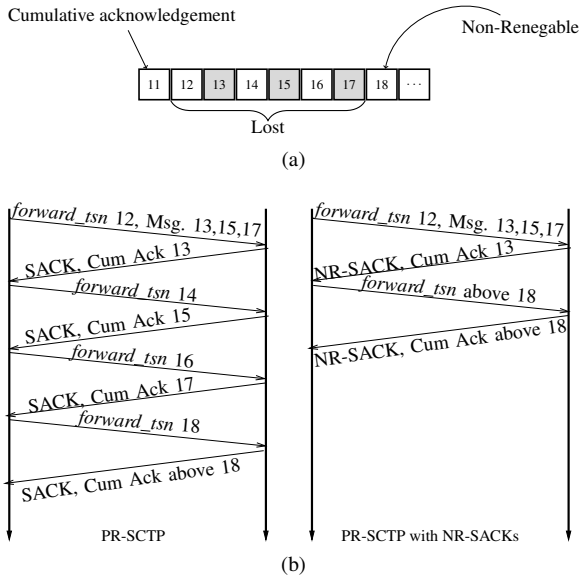



Fig. 6: (a) Case 2: Information extracted from a NR-SACK chunk. (b) Improvement of PR-SCTP using NR-SACKs.

derived from a NR-SACK chunk which tells that message number 1 reached the receiver in order. Additionally, it tells that messages number 3 and 5 reached out of order, since messages number 2 and 4 have not yet reached the receiver and perhaps are lost in the network. Further assume that messages 2 and 4 are both unreliable or that their lifetimes have expired.

A PR-SCTP sender can use the NR-SACK information shown in Fig. 5a very effectively. As both messages number 2 and 4 can be discarded, the sender can send a *forward_tsn* chunk to tell the receiver to forward its cumulative ACK point up to message number 5. Unnecessary delays can be saved as illustrated by the time sequence diagrams shown in Fig. 5b . In this figure, the left part shows the sequence of events in loss recovery when existing PR-SCTP is used. On the contrary, the right part shows the quicker loss recovery possible when PR-SCTP uses NR-SACKs.

One other possible scenario is when some of the lost messages are reliable and require retransmission. This is also applicable for burst loss scenarios when for instance a packet consisting of multiple bundled small messages is lost. To illustrate the use of NR-SACKs in this case, we can use fig. 6a as a representation of information derived from a NR-SACK chunk. In this case, messages number 12 to 17 are bundled into a single SCTP packet. As this packet is lost, there is a large transmission sequence number (TSN) gap at the receiver as shown in the fig. 6a. Moreover, a fraction of these lost messages are reliable as indicated by shaded TSNs in the figure.

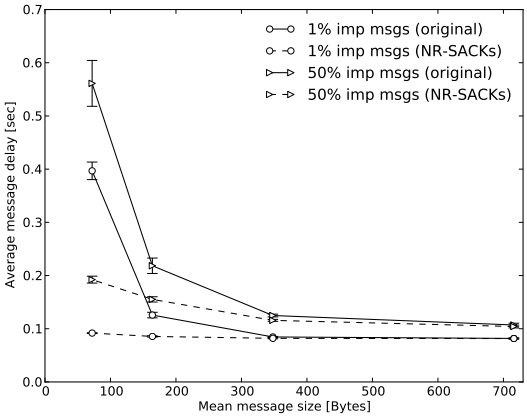
Fig. 6b shows the loss recovery steps of PR-SCTP with or without NR-SACKs using time sequence diagrams. The NR-SACKs case is shown in the right part of the figure. Since only the messages corresponding to shaded TSNs are eligible for retransmission, the sender will send an original *forward_tsn* chunk for message number 12 and along with this chunk, messages number 13, 15 and 17 can be retransmitted if space is not limited. Upon reception of both the *forward_tsn* chunk and all the retransmitted messages, the receiver can issue a NR-SACK chunk that cumulatively acknowledges message number 13; furthermore, the retransmitted messages along with the previously received out-of-order messages are selectively acknowledged and all are now non-renegable. When the sender will receive this NR-SACK, it can issue a *forward_tsn* chunk to tell the receiver to forward the cumulative ACK point up to a message number above 18.

NR-SACKs clearly provide faster recovery as shown in Fig. 6b. PR-SCTP using NR-SACKs requires two round trip times less to recover than the existing PR-SCTP. In addition, this improvement using NR-SACKs will increase as the length of the sequence of lost messages increases, provided that there is a mix of reliable and unreliable messages.

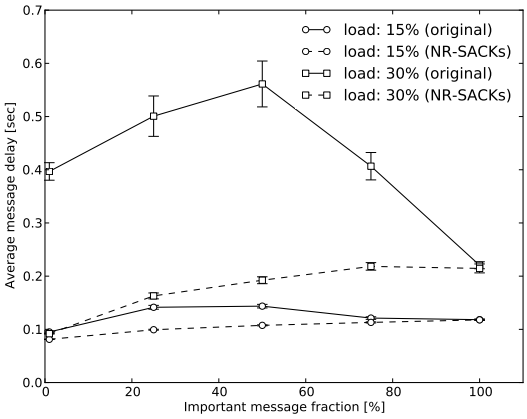
### C. Evaluation of NR-SACKs Based Optimization

Since NR-SACKs are already available in the FreeBSD kernel, we only tweak the PR-SCTP sender implementation so that it can use information from NR-SACKs for sending *forward_tsn* chunks more effectively. When a NR-SACK chunk is received, selectively acknowledged and non-renegable messages are marked. During PR-SCTP processing, these marked messages are treated as if they are cumulatively acknowledged. Next, we describe the performance evaluation of PR-SCTP using NR-SACKs.

We use the same experiment setup discussed in Section III-B to evaluate the performance of PR-SCTP when using NR-SACKs. We only present here the results for normally distributed message sizes. Fig. 7a shows the average message delay comparison for different mean message sizes. The send rates for all the PR-SCTP flows are on an average kept to a value so that only 30% of their fair share is used. At this rate,




(a) 30% load




(b) Mean message size: 72 bytes

Fig. 7: Average delay comparison of PR-SCTP with or without NR-SACKs.

PR-SCTP can keep up with the application send rate over time. There may be momentary queue build up in the send buffer due to congestion control that may cause message bundling. Moreover, only a few fractions of important messages are displayed to make the figure readable.

The performance improvement of PR-SCTP with NR-SACKs in terms of message transfer delay for smaller messages is clearly shown in the figure. When message sizes become smaller than 348 bytes, the *forward_tsn* inefficiency starts to hamper performance and PR-SCTP with NR-SACKs starts to show gain. Our results suggest that for a message size of 72 bytes, the average message delay is reduced by over 75% by using NR-SACKs when the fraction of important messages is 1% and by over 65% when the fraction of important messages is 50%.

As discussed in section IV, loss recovery is faster with NR-SACKs when a large number of messages in sequence are lost. For the smaller message sizes, many small messages are bundled into a single packet due to buffering in the send queue before transmission. In addition, since there are other flows competing for the network resources, the delay caused by the inefficient *forward_tsn* mechanism may restrict the flow in bandwidth sharing compared to other flows.

Fig. 7b shows the results for additional important message fractions and loads. Since the *forward_tsn* inefficiency becomes critical for small messages, we focus on the results for a mean message size of 72 bytes in the figure. The results suggest that even when the load is varied, PR-SCTP with NR-SACKs has significant gain over the existing one. Additionally, the shape of the graphs using PR-SCTP with NR-SACKs becomes as expected; average message delays grow as the fraction of important messages is increased. This is in sharp contrast to what we see for the existing PR-SCTP where the *forward_tsn* inefficiency creates a very different shape. Furthermore, since the inefficiency in the *forward_tsn* mechanism is absent when all messages are reliable, the graphs for both PR-SCTP versions coincide when all messages are important.

It can be noted that it is also possible to improve the PR-SCTP performance in FreeBSD if expired messages are discarded from the send buffer using late binding of TSN/SSN[1]. In late binding, TSN/SSN assignment actually happens when a message reaches the head of the send queue. According to RFC 3758 [4], a message can be skipped from the send queue before it is even transmitted and no corresponding *forward_tsn* chunk needs to be sent. This may happen when an application uses a timed reliability based policy. In this case, the lifetimes of messages may expire when they are queued in the send buffer due to congestion control. Although the possibility of skipping messages from the send buffer is mentioned in the FreeBSD documentation [16], this has not been implemented in FreeBSD 8.2. The lifetime property is not evaluated at the initial transmission. However, while dropping expired messages from the send buffer will improve PR-SCTP performance in general, our NR-SACKs based optimization will not be affected.

## V. Conclusions

In this paper we have examined an inefficiency in the existing *forward_tsn* mechanism of PR-SCTP in detail. We have proposed a solution to mitigate this problem. Furthermore, we have implemented the solution which uses the NR-SACKs functionality existing in the current FreeBSD kernel. Our initial experiment results expose a notable performance gain in terms of average message transfer delay in PR-SCTP by using NR-SACKs. If this modification is widely deployed, several applications that require prioritized transport will be greatly benefited. We are currently doing a larger number of experiments considering several network scenarios for a complete investigation of the performance of PR-SCTP using our proposed solution.

[1]SSN-Stream Sequence Number

## REFERENCES


[1] R. Stewart, Q. Xie, and K. Morneault, "RFC 4960: Stream control transmission protocol," September, 2007.

[2] B. Dempsey, T. Strayer, and A. Weaver, "Adaptive error control for multimedia data transfers," *In International Workshop on Advanced Communications and Applications for High-Speed Networks (IWACA)*, pages 279-289, Munich, Germany, March 1992.

[3] C. Papadopoulos and G. Parulkar, "Retransmission-based error control for continuous media applications," *In 6th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 512, Zushi, Japan, April 1996.

[4] R. Stewart, M. Ramalho, Q. Xie, and M. Tuexen, "RFC 3758: Stream Control Transmission Protocol (SCTP) Partial Reliability Extension," may, 2004.

[5] T. Maeda, M. Kozuka, and Y. Okabe, "Reliable Streaming Transmission Using PR-SCTP," in *Ninth Annual International Symposium on Applications and the Internet, SAINT'09*, pp. 278–279, IEEE, 2009.

[6] H. Sanson, A. Neira, L. Loyola, and M. Matsumoto, "PR-SCTP for real time H. 264/AVC video streaming," in *The 12th International Conference on Advanced Communication Technology*, vol. 1, pp. 59–63, IEEE, 2010.

[7] H. Wang, Y. Jin, W. Wang, J. Ma, and D. Zhang, "The performance comparison of PRSCTP, TCP and UDP for MPEG-4 multimedia traffic in mobile network," in *International Conference on Communication Technology Proceedings, ICCT*, vol. 1, pp. 403–406, IEEE, 2003.

[8] M. Molteni and M. Villari, "Using SCTP with partial reliability for MPEG-4 multimedia streaming," in *European BSD Conference*, 2002.

[9] M. Rajiullah and A. Brunstrom, "On the Effectiveness of PR-SCTP in Networks with Competing Traffic," in *The IEEE symposium on Computers and Communications (ISCC 2011)*, pp. 898-905, Corfu, Greece, 2011.

[10] E. Yilmaz, N. Ekiz, P. Natarajan, P. Amer, J. T. Leighton, F. Baker, and R. Stewart, "Throughput analysis of non-renegable selective acknowledgments (NR-SACKs) for SCTP," in *Computer Communications*, volume-33, pp. 1982-1991, October, 2010.

[11] N. Ekiz, P. Amer, P. Natarajan, R. Stewart, J. Iyengar, "Non-Renegable Selective Acknowledgements (NR-SACKs) for SCTP ,"*IETF Internet Draft: draft-natarajan-tsvwg-sctp-nrsack-08* (work in progress)

[12] S. Kim, S. Koh, and Y. Kim, "Performance of SCTP for IPTV Applications," in *The 9th International Conference on Advanced Communication Technology*, vol. 3, pp. 2176–2180, IEEE, 2007.

[13] X. Wang and V. Leung, "Applying PR-SCTP to transport SIP traffic," in *Global Telecommunications Conference, GLOBECOM'05,* vol. 2, pp. 5–780, IEEE, 2006.

[14] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, 1997.

[15] J. Postel, "RFC 3164: The BSD Syslog protocol," August, 2001.

[16] FreeBSD Manual, http://www.freebsd.org/cgi/man.cgi