The results show that the proposed algorithm is able to provide performance of the two management algorithms in an emulated environment. The paper also compares the performance of the two management algorithms in an emulated signaling environment, using the lksctp implementation of SCTP. The results show that the proposed algorithm is able to provide significant reductions in loss recovery time.

I. INTRODUCTION

Emerging technologies like Voice over IP (VoIP) have lead to a revolution in the telecommunications industry. It is now possible for operators to replace expensive legacy environments with cost effective IP-based environments. The first step towards purely IP-based environments was to transmit media content (e.g. voice traffic) over IP-networks. However, signaling traffic needed for e.g. call initiation and billing, remained in separate signaling networks. The requirements that signaling traffic has was not met by IP-based environments at that time. Today, however, a lot of improvements have been done, and operators are now starting to use, or are already using, IP-based environments for signaling as well.

A key factor in this transition is the SIGTRAN architecture [1], an architecture for IP-based signaling. A major component of this architecture is the Stream Control Transmission Protocol (SCTP) [2], which is a transport protocol for signaling traffic. SCTP is based on Transmission Control Protocol (TCP) [3], [4], [5], but includes features to better support signaling transport. For instance, SCTP supports out-of-order delivery of data so that packet loss does not cause head-of-line blocking.

Signaling traffic is different from ordinary TCP bulk traffic as it often consists of small bursts of messages, where each message requires timely delivery. However, the loss recovery of SCTP, which was inherited from TCP, is still not fully optimized for such traffic. Two loss recovery mechanisms are used by SCTP: fast retransmit and retransmission timeout. Fast retransmit is preferred, as it detects packet loss quicker than retransmission timeout. However, fast retransmit does not work if the amount of outstanding data is too small, a situation more likely to occur for bursty signaling traffic than for TCP bulk traffic. In such situations loss recovery resorts to retransmission timeouts, triggered by the expiration of the retransmission timer. Unfortunately, the algorithm that manages the retransmission timer, unnecessarily extends the time needed for retransmission timeout by restarting the retransmission timer too late.

This problem has also been observed for TCP [6], [7], but is not regarded as a critical problem as TCP applications typically do not depend on the delay of individual TCP segments. For SCTP signaling traffic, however, an unnecessary increase in loss recovery time is problematic, as it can cause violations to the message delay requirements of the SIGTRAN architecture [1].

Therefore, we propose a new algorithm for managing the SCTP retransmission timer. The new algorithm does not unnecessarily extend the time needed for retransmission timeout, and is therefore better suited for applications that require timely delivery of data. In addition, we provide an experimental performance evaluation of the different algorithms using the Linux Kernel implementation of SCTP (lksctp) [8]. The experiments were conducted in an emulated network environment, allowing a real protocol implementation to be evaluated, using bursty traffic and network characteristics representative for signaling environments.

The results from the evaluation show that the new algorithm for managing the retransmission timer can reduce the loss recovery time significantly.

The rest of the paper is structured as follows. Section II gives a brief overview of SCTP loss recovery. Section III describes the algorithm that manages the SCTP retransmission timer, the performance problem associated with it, and our proposed solution to the problem. Section IV describes the experimental setup and design. Section V presents some of the results, and in Section VI a summary is provided.

II. SCTP LOSS RECOVERY

As mentioned in the introduction of this paper, SCTP has two ways of detecting and recovering from packet loss: retransmission timeout and fast retransmit. Fast retransmit is preferred, as it detects packet loss more quickly than retransmission timeout. To achieve the quick detection, fast retransmit uses the reception of duplicate acknowledgments for loss detection. Duplicate acknowledgments are sent from a
receiver when it receives packets that are out-of-order. Packets can arrive out-of-order at a receiver for two reasons: the network reorder packets, or a previously sent packet has been lost causing the following packets to arrive out-of-order. To disambiguate between these events, a SCTP sender waits for three consecutive duplicate acknowledgments before it infers packet loss and performs retransmissions [2].

However, in some situations it might be impossible for a sender to use fast retransmit for loss recovery. This usually happens when the amount of outstanding data is small. For example, if three packets are sent and the first is lost, the two following packets will not generate enough duplicate acknowledgments to trigger fast retransmit. In such situations, packet loss recovery relies on expiration of the retransmission timer. The expiration of this timer happens when a sent packet is not acknowledged by the receiver within a given period of time. This period of time is referred to as the retransmission timeout (RTO) and is based on measurements of the round-trip time between the sender and the receiver. To prevent sudden increases in the round-trip time from causing packets to be mistakenly regarded as lost, and unnecessarily retransmitted, a conservative lower bound on the timer is used.

The traffic pattern of signaling traffic is different from that of ordinary TCP traffic, presumably bursty in the case of SS7 signaling [9], which increases the risk of having a small number of packets outstanding when packet loss occurs. Thus, the risk of recovering packet loss with the means of lengthy retransmission timeouts, instead of fast retransmits, is larger in signaling scenarios. This is unfortunate, as signaling traffic has more strict requirements on timely delivery than ordinary TCP traffic. What makes this problem worse is that SCTP’s retransmission timer management algorithm unnecessarily extends the time needed for retransmission timeout even further. In the next section we describe this problem, and propose a solution to it.

III. MANAGEMENT OF THE RETRANSMISSION TIMER

The algorithm that manages the SCTP retransmission timer is given in [2], and is identical to TCP’s algorithm [10]. In

![Fig. 1. Two scenarios involving the retransmission timer management](image)

In the next section we describe this problem, and propose a simple change to the retransmission timer management algorithm. By modifying the retransmission timer management in [2] to remove the offset each time the timer is restarted, the new algorithm will keep the timer in a correct state at all times. Thus, the timer will always expire after exactly RTO ms if no acknowledgment is received. The new algorithm requires that SCTP stores the time at which each packet was transmitted. However, such bookkeeping is often conducted by SCTP implementations, as the information is needed by other parts of the protocol.

IV. EXPERIMENTAL SETUP & DESIGN

To evaluate the SCTP retransmission timer management and the proposed algorithm in a signaling context, we modified the standard parameterization of SCTP. The parameterization is shown in Table I, and is based on recommendations from telecommunication vendors. Compared to the standard parameterization [2], this parameterization is more aggressive. This is required and also supported by most signaling environments, as signaling traffic typically have higher requirements on availability and timely message delivery [1]. In the experimental

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTO_wait (ms)</td>
<td>200</td>
</tr>
<tr>
<td>RTO_min (ms)</td>
<td>100</td>
</tr>
<tr>
<td>SACK_delay (ms)</td>
<td>20</td>
</tr>
<tr>
<td>Bottleneck bandwidth (Kbit/s)</td>
<td>500, 1000, 2000</td>
</tr>
<tr>
<td>One way end-to-end delay (ms)</td>
<td>5, 10, 15, 20, 25, 30</td>
</tr>
</tbody>
</table>

**TABLE I**

**EXPERIMENTAL PARAMETERS**

1For example, in the implementation we used for our evaluation, this information is already available as it is used for round-trip time estimation purposes.
evaluation we also used end-to-end delays and bandwidth characteristics relevant to such environments. These are listed in Table 1 as well.

To show the impact of the management algorithm, we used a traffic pattern that consisted of bursts of four messages with intermediary idle periods of 200 ms. This pattern was used as it forced SCTP to recover most packet losses with retransmission timeouts, revealing the impact of the problem.

The environment used for the evaluation consisted of three ordinary hosts that acted as sender, receiver, and network emulator. The sender and receiver, which both used the Linux Kernel SCTP implementation (lksctp) [8], communicated via the network emulator, which delayed and dropped incoming packets as instructed.

To achieve precise control over packet loss the network emulator was equipped with KauNet [11], an extended version of the Dummynet network emulation software [12]. KauNet allows precise control over packet loss with the possibility to specify these on a per packet basis, using precomputed patterns.

The experiments were conducted as follows. The receiver initiated an association with the sender, which responded by transmitting a fixed number of bursts according to the previously mentioned traffic pattern. When this traffic passed through the network emulator, a single packet from the flow was dropped by the emulator. The time needed for the lost packet to be received by the receiver was then measured.

Using this method, the whole procedure was repeated until packets in all the different positions of the flow had been subjected to loss. The metric used to evaluate the retransmission timer management algorithm was the Message Transfer Time (MTT). The MTT is defined as the time required for a message to travel from the sender to the receiver application. When a message was lost within the emulated network, the time needed for loss detection and retransmission was, of course, included in the MTT.

For each combination of bottleneck bandwidth, end-to-end delay, and packet loss position, three replications were made and the median MTT was used as the result. The reason for executing three replications of each scenario was to eliminate possible random outliers.

V. RESULTS

In Figure 2 the results of a representative experiment are shown. The y-axis of the graph shows the MTT of individual SCTP messages. The MTT in this scenario is the time it took for a message to be sent, lost, retransmitted, and received by the other end. The x-axis shows the Transaction Sequence Number (TSN) of the corresponding message. For the experiments shown in this graph twelve bursts, each containing four messages, were measured. Furthermore, the bottleneck bandwidth was 1000 Kbit/s, and the end-to-end delay 15 ms.

Two different data sets are shown in the figure: Standard RTO Management and Modified RTO Management. For the standard set, we can see the MTT of packets retransmitted using the RTO management rules as defined in [2]. Similarly, the modified set shows the MTT of retransmitted packets using our proposed RTO management algorithm. We can clearly see that there exist large differences between the two data sets. The largest difference can be found when the message with TSN 8 was lost. For a loss of this message, the modified RTO management was able to reduce the MTT with 38%. On average, for this scenario, the MTT was reduced with 24%.

Aside from the first burst, which will be covered later, a general pattern in the MTT for the different bursts can be observed. The first message (M1) in each burst had the same MTT for both versions of the management algorithm. The reason for this is that three messages (M2, M3, M4) succeeded M1. These messages generated three duplicate acknowledgments at the receiver, which in turn triggered a fast retransmit of M1. Thus, the RTO management algorithm did not have any effect on the loss detection time of M1.

The two following messages (M2, M3), in each burst, had a MTT that was reduced with approximately 25% using the modified version. When M2 was lost, and the standard algorithm was used, the duplicate acknowledgment triggered by M3 restarted the timer, as it also acknowledged M1. This made the retransmission of M2 occur roughly $\text{RTT} + \text{RTT}_M$ ms after it was originally sent. Thus, one extra round-trip time was implicitly added to the retransmission timer. For a loss of M3 the situation was similar. In this case the extra round-trip time, added to the loss detection, came from the restart of the retransmission timer triggered by the acknowledgment of M2.

For a loss of the fourth message (M4) in each burst, the MTT was reduced with approximately 35% for the modified version. The reason why a loss of this message further increased the benefit of our algorithm is related to the 20 ms SACK delay that was used in the experiments. Let us once again consider
the actions of the standard algorithm. Similar to the previous case, when the acknowledgment to $M_2$ restarted the timer, one extra round-trip time was added to the loss detection time. However, as $M_4$ was lost, and therefore unable to cause an acknowledgment to be generated, the receiver had to wait for its delayed acknowledgment timer, associated with $M_3$, to expire before sending an acknowledgment. Thus, $M_4$ was retransmitted approximately $RTO + RTT_{M3} + SACK_{delay}$ ms after it was originally transmitted.

As our proposed algorithm keeps the retransmission timer associated with the earliest outstanding packet at all times, neither the round-trip time nor the SACK delay affects the expiration of the retransmission timer. Thus, by using this algorithm the time needed for loss recovery was roughly constant throughout the entire scenario.

As previously mentioned, the first bursts in the experiments deviate from the MTT pattern of the following bursts. Firstly, the MTT of the first message was much higher than for all other messages. The reason for this is a bug in the lksctp implementation, which makes it impossible to trigger fast retransmit when the first data packet in an association is lost. Furthermore, as no RTO calculation had been conducted at that time the timer was set to its initial value of 200 ms. Secondly, we had no peak in the MTT for a loss of the fourth message in this burst. This occurred as SCTP uses an acknowledgment policy that requires the first data packet in an association to be acknowledged immediately, and not delayed. Therefore, a displacement in the acknowledgment generation occurs and a loss of the fourth message in this burst becomes similar to a loss of the third message in the other bursts. Lastly, the MTT is generally higher for all messages in this burst, than for succeeding bursts. This is an effect of the slow RTO convergence from initial to minimum value.

In Figure 3 we can see an aggregation of all experiments with a bottleneck bandwidth of 1000 Kbit/s. The $y$-axis of this graph shows the average MTT reduction, provided by the modified timer management. The different lines in the graph represent the different end-to-end delays used, and the $x$-axis shows the different message positions within the bursts ($M_i$, $1 \leq i \leq 4$). For instance, when $x = 2$ then $y$ shows the average MTT reduction for the messages that were the second messages in their corresponding bursts.

Consistent with the more detailed view of the results, shown previously, we can see that the largest benefit of using the modification was achieved when the last message in a burst was lost. Furthermore, we can also verify that the benefit of using our proposed algorithm becomes larger as the end-to-end delay increases. This was also verified for the other combinations of bandwidth and end-to-end delay, shown in Table I. More details about these results can be found in [13].

VI. CONCLUSIONS

The algorithm that manages SCTP’s retransmission timer often restarts the timer too late, which causes loss detection to happen slowly. Our experiments, conducted in an emulated network environment using the Linux Kernel implementation of SCTP, show that it is possible to achieve faster loss detection using a new algorithm that keeps the timer is a correct state at all times. The new algorithm is able to keep the timer in a correct state by reusing round-trip time measurements to calculate the correct retransmission timeout value. The results from the experiments show that the new algorithm is able to provide much faster loss detection without making SCTP more agressive.

REFERENCES