On Making SCTP Robust to Spurious Retransmissions

Sourabh Ladha[+] ladha@cis.udel.edu
Stephan Baucke[±] stephan.baucke@ericsson.com
Reiner Ludwig[±] reiner.ludwig@ericsson.com
Paul D. Amer[+] amer@cis.udel.edu

[+]Protocol Engineering Lab
University of Delaware

[±]Ericsson Research
Aachen, Germany

ABSTRACT

Network anomalies such as packet reordering and delay spikes can result in spurious retransmissions and degrade performance of reliable transport protocols such as TCP and SCTP. Past research showed that in certain networks or paths, such anomalies were fairly common. The result was a series of proposals (Eifel, DSACK-based, F-RTO) targeted towards making a transport robust to such events. Most of the proposals focused on TCP and assumed applicability to SCTP. This paper focuses on SCTP and makes three contributions. First, the effect of spurious retransmissions in SCTP is shown to be more aggravated than in TCP, leading to a problem of congestion window overgrowth. Second, it is shown that most but not all of the solution space is shared between TCP and SCTP. An extension to the Eifel algorithm is proposed. Third, a microscopic as well as macroscopic comparison of the Eifel algorithm and the DSACK-based algorithm is presented. Such a comparison will provide implementers and vendors a clearer view of the applicability and tradeoffs involved in different algorithms.

1. INTRODUCTION

Network anomalies that may degrade the performance of a reliable transport protocol are manyfold. This paper focuses on two such anomalies: packet reordering and delay spikes.

The statistics on packet reordering caused by a network element have been the cause of much debate during the past years. Some work shows that packet reordering is fairly common in certain paths on the Internet, with measurements showing that as many as 90% of the connections experience reordering [BPS99], while others show that packet reordering is a rare event with the degree of reordering below 2% [Pax97]. Nevertheless, all past work recognizes that packet reordering is a network property (one of the causes being a naïve load balancing scheme used by a router), and that there may be networks/paths in the Internet where an unacceptable amount of reordering is witnessed [BPS99, Pax97]. A granularity of packet reordering exceeding the duplicate ack threshold of the fast-retransmit algorithm may cause a sender to retransmit data and trigger congestion control [APS99, Jac88] unnecessarily.

A sudden increase in the end-to-end delay is referred to as a delay spike. The case for delay spikes is more pronounced and can be caused in the Internet by events such as route flipping [All00, AP99, Bol93]. If seen from a transport perspective, an aggressive retransmission timer calculation can result in more frequent delay spikes. It is well accepted that wireless links can exhibit delay spikes quite frequently due to link layer ARQ [LR01] or reactive resource allocations, besides others [Gur01, YK02]. Handovers or cell changes in wireless cellular networks are another cause for highly variable delays [GPA+02, KAG+01]. A delay spike can cause an expiration of the retransmission timer at the sender (a spurious timeout), leading to unnecessary retransmissions and triggering of congestion control [LK00].

The studies described above spun a series of research work targeted towards making a transport robust to such events. The Eifel algorithm [LM03] uses the timestamp option in TCP [JBB92] as an explicit signaling mechanism, and is able to detect spurious fast retransmits as well as spurious timeouts. The DSACK-based algorithm [BA04] uses the DSACK notification [FMM+00], a signal from the receiver indicating it has received a packet more than once, to detect both spurious fast retransmits and spurious timeouts. Forward RTO (F-RTO) [SKR04] is a heuristic algorithm which changes the retransmission pattern of a sender after a timeout event, and watches the incoming acks to detect spurious timeouts. The above schemes have been proposed within the IETF. The Eifel and DSACK-based algorithms are already experimental RFCs. The purpose of having multiple such algorithms is to give implementers and vendors a choice to use whichever algorithm is more appropriate and robust for their networks while keeping in mind the tradeoffs involved.

The Stream Control Transmission Protocol (SCTP) is an IETF standards track transport layer protocol [SXM+00] that is gaining acceptance both as a general purpose transport and as a next generation signaling transport [CIA+03, SM01]. Like TCP, SCTP provides an application with a full duplex, reliable, connection-oriented transmission service. The congestion control principles in SCTP are broadly derived from TCP. Unlike TCP, SCTP provides additional transport services such as multistreaming, multihoming and preservation of message boundaries. Sharing TCP's mechanisms in loss recovery and congestion control makes SCTP also vulnerable to the problem of spurious retransmissions. The Eifel algorithm has been proposed only for TCP. The DSACK-based algorithm has been proposed for both TCP and SCTP (SCTP has a counterpart to DSACK known as DupTSN and so in the SCTP context we refer to the algorithm as DupTSN-based algorithm). F-RTO is currently being revised to consider SCTP [SKR04]. However, two items missing in the past work are (a) research regarding interaction and applicability of these algorithms with SCTP, and (b) a comparison of the above algorithms for TCP or SCTP.

In this paper, we address these missing items. We study the problem space of spurious retransmissions with SCTP, and show that the effects of spurious timeouts are more severe in SCTP than in TCP. We discuss the features of SCTP that make it difficult to directly apply TCP based algorithms to SCTP. We propose an extension of the Eifel algorithm for SCTP, and finally we present comparisons of the Eifel algorithm and the DupTSN-based

* The majority of this work was performed while Sourabh Ladha interned at Ericsson Research during summer 2003. Follow-up work by Sourabh Ladha and Paul Amer was supported, in part, by the Communications and Networks Consortium sponsored by the U. S. Army Research Laboratory (DAAD19-01-2-0011), and by the CISCO University Partnership Program.

algorithm both from a microscopic and macroscopic perspective (this comparison is relevant to both SCTP and TCP). This paper does not consider the F-RTO algorithm for two reasons. First, F-RTO caters solely to the problem of spurious timeouts and not to spurious fast retransmits. Second, at the time of writing this paper F-RTO has not been modified completely to work in conjunction with SCTP.

The rest of this paper is organized as follows. Section 2 discusses the different proposals for detection of spurious retransmissions. Section 3 gives a background of SCTP features related to the work described in the paper. A reader familiar with SCTP may skip this section. Section 4 illustrates the problem space and discusses the more serious implications of spurious timeouts in SCTP. It also presents an extension to the Eifel algorithm for SCTP. Section 5 brings out a microscopic evaluation of the Eifel and DupTSN-based algorithms. Section 6 discusses the metrics used for macroscopic evaluation and the results obtained. Section 7 concludes the paper.

2. RELATED WORK

The Eifel algorithm was originally proposed in [LK00]. The algorithm as initially proposed could either use the existing TCP timestamp option [JBB92], or called for a new flag in the TCP header. Due to the limited number of available TCP header bits, the consensus in the IETF was to use the timestamp option. An Eifel-enabled sender [LM03] stores the timestamp value of the first retransmission sent in loss recovery in the RetransmitTS variable. On the receipt of the first acceptable ack, the sender compares the value of timestamp echoed in that ack with RetransmitTS. If the echoed timestamp value is found to be earlier, the sender concludes that the earlier retransmission (either due to a timeout or a fast retransmit) was spurious. The advantage of the Eifel algorithm, intuitively, is that it is able to detect a spurious retransmission soon enough to avoid further unnecessary retransmissions, and is a complete solution catering to the problem of packet reordering, delay spikes as well as packet duplication. The disadvantage is that the Eifel algorithm requires the timestamp option to be enabled for the connection, which is a 12 byte overhead per packet. The Eifel algorithm as defined in [LM03] does not consider SCTP; it only targets TCP.

The DSACK-based algorithm [BA04] uses DSACK information in TCP [FMM+00] to detect if a retransmission was spurious. A DSACK tells a sender that the receiver has received a packet more than once (if a sender had only sent the packet once, a DSACK implies that duplicates were created in the network). The algorithm calls for maintaining sender side state to store meta-information about the retransmitted segments in a particular loss recovery phase. On receiving DSACK notifications, the sender updates the data structure correspondingly. If the sender finds that all retransmitted segments have also been reported in DSACKs, it concludes that all retransmissions (fast-retransmits or timeout-based) were spurious. The DSACK-based algorithm does not add packet overhead. The disadvantage is that the algorithm is not robust to packet duplication within the network, and to loss of acks (as a DSACK is reported only once). Also, the DSACK information comes only after loss recovery has exited, so a sender cannot detect a spurious retransmission while in the loss recovery phase. Further, local state at the sender (for storing the meta-information of retransmissions) has to be maintained. SCTP's analogy to DSACK is called DupTSN (Duplicate TSN). The proposed DSACK-based algorithm applies to both TCP and SCTP.

F-RTO is a heuristic algorithm that changes a sender's transmission pattern after a timeout. On a timeout a sender retransmits the oldest unacknowledged segment. As the first ack arrives, the sender transmits new data segments as opposed to the normal timeout recovery of sending retransmissions. If the next ack covers a segment that was not retransmitted, the sender concludes that the timeout was spurious. The advantage of F-RTO is that it adds no packet overhead and is a sender side only algorithm. The disadvantage is that the algorithm works on heuristics leading to corner cases and becomes complicated if the application has no new data to send when called upon by F-RTO. F-RTO caters only to the problem of spurious timeouts and not to spurious fast retransmits. F-RTO is in the process of being revised for use with SCTP.

The above mentioned schemes can "detect" spurious retransmissions. Also required is a response algorithm [LG03, BDA03] for correcting or undoing the previous congestion control changes. Explicitly out of scope of this paper is the evaluation of the different response algorithms. The interested reader is pointed to [BA02, GL03] for such evaluations.

Note that packet reordering in SCTP can also be caused due to application initiated changeovers [ICA+03] or transport layer Concurrent Multipath Transfer (CMT) [ISA+03]. Solutions to transport reordering attempt to avoid spurious retransmissions in the first place. On the other hand, spurious retransmissions caused by network reordering are often unavoidable. This paper discusses the latter, i.e. the problem of packet reordering caused by the network.

3. SCTP BACKGROUND

We briefly describe two SCTP features that the reader should be familiar with to understand the discussion that follows in this paper.

- *Chunk Based Transmission:* SCTP [SXM+00] is a message-oriented protocol, in contrast to the byte-streaming nature of TCP. An application write is treated as a message and is translated into one or more SCTP DATA chunks, depending on whether the message fits in one Path Maximum Transmission Unit (PMTU). Each DATA chunk is assigned a Transmission Sequence Number (TSN). While TCP assigns sequence numbers to bytes, SCTP assigns TSNs to chunks. If the application is writing small messages, more than one DATA chunk can be "bundled" into the same SCTP packet. This concatenation mechanism is known as chunk bundling. Further, SCTP incorporates control chunks such as SACK, HB (Heart Beat), ECNE (ECN Echo), etc., which can also be bundled with DATA chunks.
- *Multihoming:* A host is said to be multihomed if it can be addressed by multiple IP addresses. TCP as a transport does not support multihoming. At any point in time during a TCP connection, if an endpoint's IP address becomes inaccessible (say, due to interface/link failure), TCP's connection will timeout multiple times and eventually abort, thus forcing the application or user to recover (assuming the application is providing data to TCP). On the other hand, SCTP has a built in failure detection and recovery system, known as failover [SXM+00], which allows associations to dynamically send traffic to an alternate peer IP address when needed without losing the end-to-end association or requiring the application to intervene. This failover occurs after a threshold number of consecutive timeouts to the "primary"

destination have occurred. SCTP also exploits this path/interface redundancy in its retransmission policy.

The following definitions are applicable throughout this paper. (a) A *primary destination* is an interface at an SCTP endpoint, which the peer SCTP endpoint uses to send new data transmissions. The primary destination can be set either by the peer application at any time in the association lifetime, or chosen by the peer's SCTP stack during the establishment of an association. (b) A *primary path* is a path followed to reach the primary destination. (c) An *alternate destination* is an interface at an SCTP endpoint that is used for redundancy purposes. (d) An *alternate path* is a path followed to reach an alternate destination.

The current retransmission policy in SCTP states that all retransmissions, resulting from fast retransmit or a timeout, should go to an alternate destination. The expectation is that if the paths are diverse, then sending retransmissions via an alternate path will avoid the congestion on the primary path. However, the question of how to determine if a chosen alternate destination results in an alternate and diverse path is an open research issue. Some research provides evidence that sending retransmissions on the primary path is better even if the paths are diverse [CAI+03]. Further note that SCTP (a) maintains congestion control state, timers, etc., per destination, and (b) continually monitors the reachability of idle alternate destinations using heartbeats; however, the recommended rate of sending heartbeats once every thirty seconds provides minimal feedback.

4. THE PROBLEM, EFFECTS AND SOLUTION

Simulation Setup

The graphs presented in this paper are a result of simulations done using the Network Simulator (ns2.1b8) [NS]. The SCTP module used in ns, currently part of the latest ns2 distribution, was developed at [PEL]. For the purpose of studying spurious retransmissions, the "hiccup" module was used [LK00, Sch00], which artificially introduces packet reordering and delay spikes with script-specified parameters.

Unless mentioned otherwise, the following simulation setup was used. All links (corresponding to end-to-end paths) are 1Mbps duplex with a propagation delay of 100ms in each direction. The delayed ack timer is set to a value of 200ms. The bottleneck buffer for each path can hold up to 100 packets, and the PMTU for each path is 1500 bytes.

A singlehomed simulation corresponds to an SCTP source and an SCTP sink connected via a 1Mbps duplex link simulating the end-to-end path. A multihomed simulation corresponds to the setup shown in Figure 1.

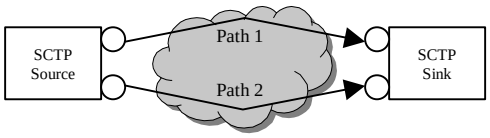

Figure 1: Multihoming setup used for simulations

In Figure 1, both endpoints have two interfaces. Traffic for path1 is routed between the first (upper, in Figure 1) set of interfaces, and traffic for path2 is routed between the second (lower, in Figure 1) set of interfaces (separate paths can be assumed for reasons such as path diversity, policy based routing, load balancing). We have assumed the paths are diverse (i.e., do not share any bottleneck queue), and all retransmissions go on the alternate path.

Spurious Fast Retransmits

Packet reordering (not due to loss) with a granularity exceeding the duplicate ack threshold causes an SCTP sender to infer that a packet was lost. An SCTP sender then triggers loss recovery by retransmitting the packet, and triggers congestion control by cutting back its congestion window and slow start threshold [APS99, SXM+00]. This behavior is inappropriate as there was no loss due to congestion, just a packet reordering. The problem of spurious fast retransmits is shown in Figure 2(a). Sequence number 21 is reordered such that enough missing reports are generated for the sender to retransmit 21. The sender cuts back its congestion window, unnecessarily lowering its offered load.

Spurious Timeouts

We show the problem of spurious timeouts in both

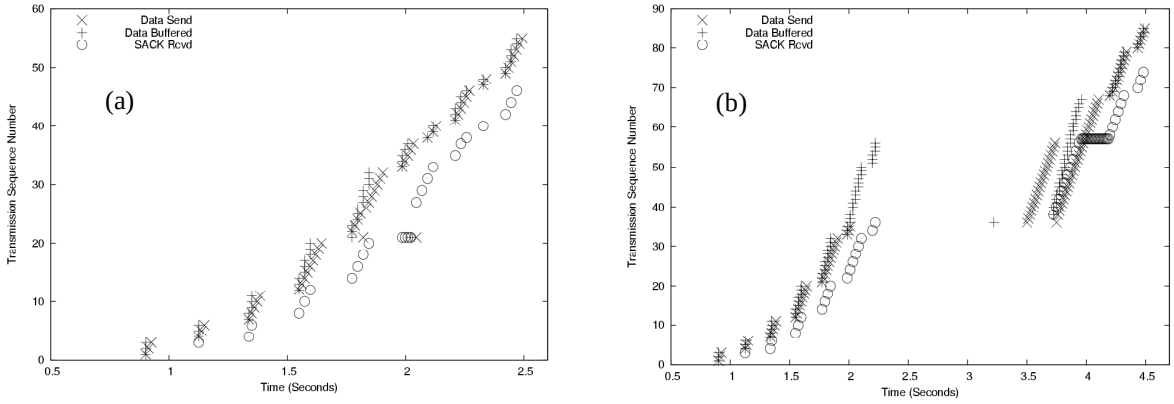

Figure 2: Singlehomed configuration: (a) a spurious fast retransmit (b) a spurious timeout

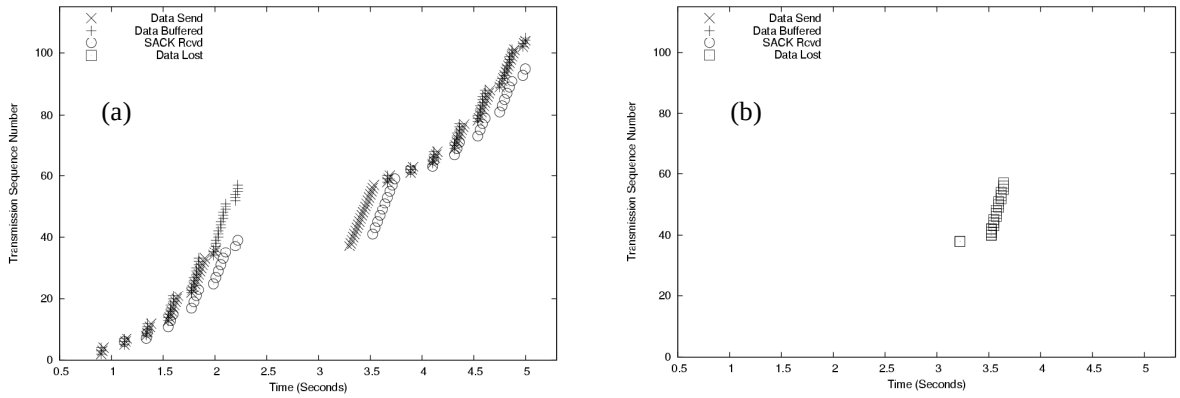

Figure 3: Spurious timeout for multihomed SCTP endpoints (a) primary interface (b) secondary interface (the problem of congestion window overgrowth)

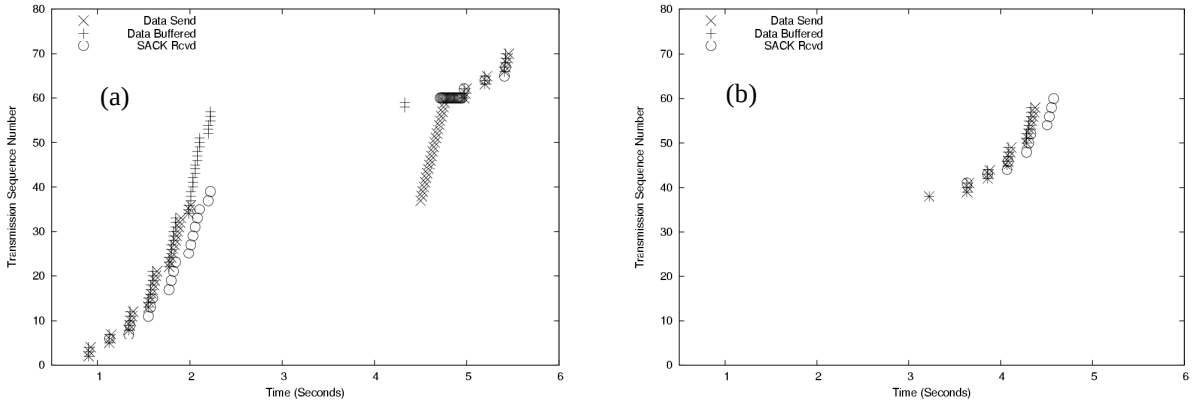

Figure 4: Spurious timeout for multihomed SCTP endpoints (a) primary interface (b) secondary interface (the case of a long delay spike)

singlehomed and multihomed scenarios. The illustration of a delay spike can be seen in Figure 2(b) and Figure 3 as the difference in time between 'Data Buffered' and 'Data Send'. Singlehomed SCTP is fairly similar to TCP in the context of spurious timeouts. Figure 2(b) demonstrates the effect of a single spurious timeout. As acks for the original transmissions arrive at the sender, more retransmits are triggered until the Go-Back-N recovery is complete. Thus the sender transmits N unnecessary retransmissions over the network and severely cuts back its congestion window, unnecessarily reducing its load.

Figure 3 shows a similar problem in a multihomed environment. SCTP's current retransmission policy states that all retransmissions should go to alternate destinations, and new data should continue to be transmitted to the primary destination. In our simulation setup the retransmissions go on an alternate path as well (as we assumed diverse paths, see Figure 1). As the acks for original transmissions are received from the primary path (Figure 3(a)), the sender sends more retransmits on the alternate path (Figure 3(b)). In the same simulation (Figure 3), to aggravate the problem, we simulated a link outage on the alternate path (causing all packets on the alternate path to be dropped) that the sender was unable to detect before the spurious timeout occurred. Because heartbeats are infrequent, it may take a significant time (several consecutive timeouts) for an SCTP sender to detect that the alternate path has failed. (Note: Transient congestion on the alternate path is another scenario where a similar situation will be experienced.)

In such a situation, a sender is doing what is called congestion window overgrowth, i.e. the sender is increasing the congestion window of the alternate path on receiving acks for original transmissions. All the retransmissions sent on the alternate path are lost (see Figure 3(b)), but the sender is unable to detect their loss. If the alternate path was witnessing congestion, this congestion window overgrowth could add severely to the problem. The root cause is that the sender is unable to distinguish between acks corresponding to original transmissions and those corresponding to retransmissions; this is called the *retransmission ambiguity*, first defined in [KP87]. Congestion window overgrowth as described above can be seen as an extended effect of this ambiguity.

Another situation of a spurious timeout in a multihomed SCTP association is demonstrated in Figure 4. Due to either significantly different round trip times on the primary and alternate paths or due to an extended delay spike on the primary path, it is possible that the SCTP sender does the entire Go-Back-N retransmissions on the alternate path before even the original transmissions reach the SCTP receiver. In Figure 4 a delay spike of sufficient length is introduced to cause this scenario.

A Solution: Extending the Eifel Algorithm

The Eifel algorithm [LM03] uses TCP's timestamp option [JBB92] to resolve the retransmission ambiguity and determine if a retransmission was spurious. The original proposal of the Eifel algorithm [LK00] also had the option of using a new bit called the "retransmit bit" to resolve the retransmission ambiguity. SCTP on the other hand lacks both of these signaling mechanisms.

We propose either adding a per-packet timestamp chunk extension to SCTP (first described in [CAI+03]), or a new "retransmit bit" in the SCTP DATA and SACK chunks. Timestamps cost 12 bytes overhead for every packet, but have the advantage of being useful for other mechanisms apart from the Eifel algorithm, such as Round Trip Time Measurement (RTTM) defined in [JBB92]. We have developed extensions to the Eifel algorithm for SCTP considering both of these signaling mechanisms. The algorithm using either a timestamp chunk or a retransmit bit is outlined in Figure 6. Also specified in Figure 6 are the rules for transmitting and echoing the retransmit bit.

The first extension to the Eifel algorithm introduces "cover acks" (see Figure 6). A "cover ack" is an ack that acknowledges (via the cum-ack field) all data up to the highest TSN retransmitted. A "cover ack" is required to avoid incorrectly interpreting congestion-based losses as spurious timeouts, as chunk bundling in SCTP allows retransmitting more than one TSN in the same packet (restricted by the PMTU). Figure 5 depicts a scenario where an SCTP sender using the Eifel algorithm as described in [LM03] would incorrectly interpret congestion loss as a spurious fast retransmission. In Figure 5, when SACK 11(2-5) is received (an acceptable ACK), the Eifel-enabled sender would conclude that the retransmission containing DATA 11 and DATA 12 was spurious. However, this would suppress the real congestion based loss of DATA 12 (which had been bundled and retransmitted with DATA 11). A "cover ack" ensures that false positives as described in this example are not generated.

The second extension to the Eifel algorithm is to cater to the retransmission policy in multihomed hosts in SCTP. As shown in Figure 4, it is possible to do a complete Go-Back-N recovery on an alternate path even before the series of original transmissions reaches an SCTP receiver. An SCTP sender will be unable to detect the timeout spurious until duplicate notifications arrive from the receiver. The solution is an optional extension that calls for integrating the DupTSN-based algorithm as defined in [BA04] with the Eifel algorithm to yield a more robust spurious retransmission detection algorithm.

Apart from the above, we have further added a new state "AckRcvd" to the Eifel Algorithm (the timestamp variant) that avoids any known corner cases, and also makes the algorithm robust for the case where all acks for the original transmissions are lost (see section 3.3 in [LM03]).

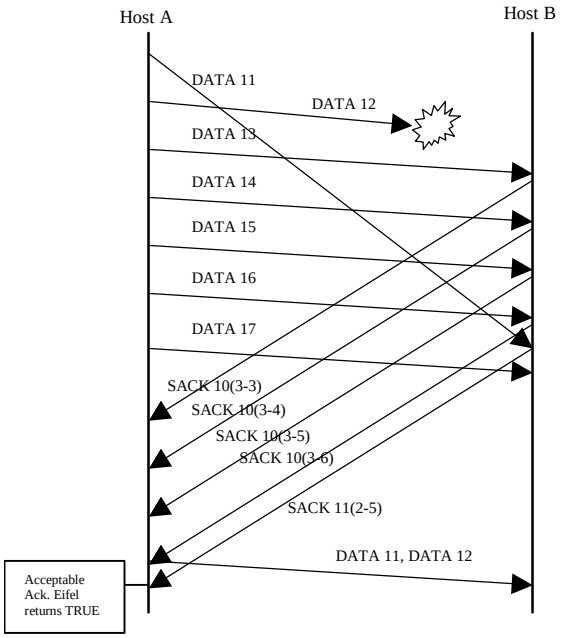

Figure 5: Chunk bundling may result in congestion based losses being incorrectly interpreted as spurious retransmits.

5. A MICROSCOPIC EVALUATION

We now verify the functionality of the Eifel algorithm for SCTP on a micro basis, and compare it with the DupTSN-based algorithm. We have implemented both algorithms in ns2.1b8 [NS] with the SCTP module [PEL].

The simulations described follow the topology and link characteristics described in section 4. For the Eifel algorithm, we implemented and simulated the timestamp variant. For plain SCTP and SCTP with DupTSN-based detection, a data chunk size of 1448 bytes was used. For SCTP with the Eifel algorithm, a data chunk size of 1436 bytes was used to allow room for the 12 byte timestamp chunk.

Responding to Spurious Retransmissions

The IETF came to the consensus to separate the detection and response algorithms for spurious retransmissions, and to let the response be independent of the detection. Thus, any of the response algorithms [BDA03, LG03] can be used with a detection algorithm [LM03, BA04, SKR04], and vice versa. To be fair in our evaluation, we kept the same response algorithm for both Eifel and the DupTSN-based algorithms. The chosen response algorithm is described in [LG04], which is in the last phase of standardization. Briefly, the response algorithm comprises two elements:

(i) Restore the congestion control state (note that a decay factor keeps a check on how much state is safe to restore once a spurious retransmission is detected). The restoration of congestion

Data Structure: The sender maintains a simple list structure to hold the TSNs of the retransmitted chunks, and corresponding to each, a Boolean variable indicating if a DupTSN report has been received for that chunk. This information can also be maintained by extending the SACK scoreboard.

I. Eifel Algorithm using per packet TIMESTAMP chunk (Timestamp echo rules following [JBB92]):

```
(1) Upon Initiation of loss recovery:
        (a) Set "Spurious Recovery" = FALSE;
        (b) Set "RetransmitTS" = TSVAL of the first retransmitted chunk sent;
        (c) Set "AckRcvd" = FALSE;
(2) Set "MaxRetransmit" = Highest TSN retransmitted;
(3) On the arrival of the ACK:
        (a) Update the duplist data structure;
        (b) If the Ack is a "cover Ack"      // A "cover Ack" is an Ack that acknowledges via cum-ack point all data up to MaxRetransmit
              goto Step 4;
            else
              { set "AckRcvd" = TRUE ;
                goto Step 2; }
(4) If the value of TSECHO field in the "cover ACK" TIMESTAMP chunk is smaller than the value of RetransmitTS
        goto step 5;
    else
        goto DONE;
(5) If {loss recovery was by timeout && AckRcvd == FALSE && DupTSN received for up to MaxRetransmit}
        goto DONE;
   else
        goto step 6;
(6) If {loss recovery was by Fast Retransmit (FR)}
        Set "Spurious Recovery" = dupacks + 1;
    else
        Set "Spurious Recovery" = SPUR_TO;
    goto RESP;
(RESP) Placeholder for Response Algorithm;
(DONE) See Multihoming Considerations;
```

II. Eifel Algorithm using per DATA/SACK chunk retransmit bit (Retransmit Bit Rules specified below):

```
(1) Upon Initiation of loss recovery:
        (a) Set "Spurious Recovery" = FALSE;
(2) Set "MaxRetransmit" = Highest TSN retransmitted;
(3) On the arrival of the ACK:
        (a)  If the Ack is a "cover ACK"    // A "cover Ack" is an Ack that acknowledges via cum-ack point all data upto MaxRetransmit
               goto Step 4;
             else
               { if the ACK has the retransmit bit set
                   goto DONE;
                 else
                   goto Step 2; }
(4) If the "cover ACK" has the retransmit bit set
        goto DONE;
(5) If {loss recovery was by FR}
          Set "Spurious Recovery" = dupacks + 1;
    else
          Set "Spurious Recovery" = SPUR_TO;
    goto RESP;
(RESP) Placeholder for Response Algorithm;
(DONE) See Multihoming Considerations;
```

Multihoming Considerations: It may be possible for spurious timeouts caused by delay spikes of more than RTO(primary) + RTT(alternate) to not be detected by the above algorithm due to the retransmission policy specified in RFC2960 (i.e., retransmissions SHOULD go on alternate path). In such scenarios the retransmitted chunk can reach the receiver before the original chunk. For such scenarios, the DupTSN-based detection [BA04] can be used.

Retransmit Bit Rules:

(a) All DATA chunk retransmissions should have the retransmit bit set in the chunk header.
(b) If a SACK chunk is being sent out corresponding to one or more DATA chunks which had the retransmit bit set, then the SACK chunk should also have the retransmit bit set in the chunk header.

Figure 6: **Two variants of the Eifel algorithm for SCTP and the rules associated with the retransmit bit approach (see [LM03] for detailed definitions and explanation of the algorithm functionality)**

state is done as per the following:

cwnd= FlightSize + min(Bytes Acked, Initial Window) ;
ssthresh = pipe_prev;
where pipe_prev was set to max(FlightSize, ssthresh) before cwnd and ssthresh were updated upon loss recovery initiation.

(ii) For spurious timeouts, resume transmission of new data and adapt the retransmission timer.

We have not considered adapting the duplicate ack threshold in the response algorithm as it has been removed from recent versions of the response document [LG04], and evaluating response algorithms is beyond the scope of this paper. However, we strongly suggest that future work try to address the issue of adapting the duplicate ack threshold [BA02].

Spurious Fast Retransmissions

Figure 2(a) shows the effect of a spurious fast retransmit caused by packet reordering on SCTP. Figure 7(a) shows the effects of the Eifel algorithm on spurious retransmissions. On receipt of the ack that covers the retransmitted segment, the Eifel-enabled sender is able to detect the spurious retransmission and undo the resulting changes to the congestion control state.

With the Eifel algorithm and a single spurious retransmission, goodput increases significantly as compared to plain SCTP. Figure 7(b) shows the effect of the DupTSN-based algorithm to detect spurious retransmissions. The sender detects and responds to the spurious fast retransmit; however, the decision can only be taken after having received the DupTSN report which arrives after the loss recovery phase has ended. Thus, the amount of congestion state (cwnd) that can be restored is reduced due to a smaller flightsize. However, at a macro level, the Eifel and DupTSN-based algorithms should perform close to each other for spurious fast retransmissions (described later in section 6). Note also that the DupTSN-based algorithm is dependent on the DupTSN report in the SACK corresponding to the retransmission. A DupTSN report is generated only once by an SCTP receiver. This makes the DupTSN-based algorithm vulnerable to ack loss.

Spurious Timeouts

The positive effects of the Eifel algorithm on spurious timeouts are more significant as an Eifel sender avoids the Go-Back-N retransmissions that were shown in Figure 2(b). Figure 8(a) shows the effect of the Eifel algorithm on a single

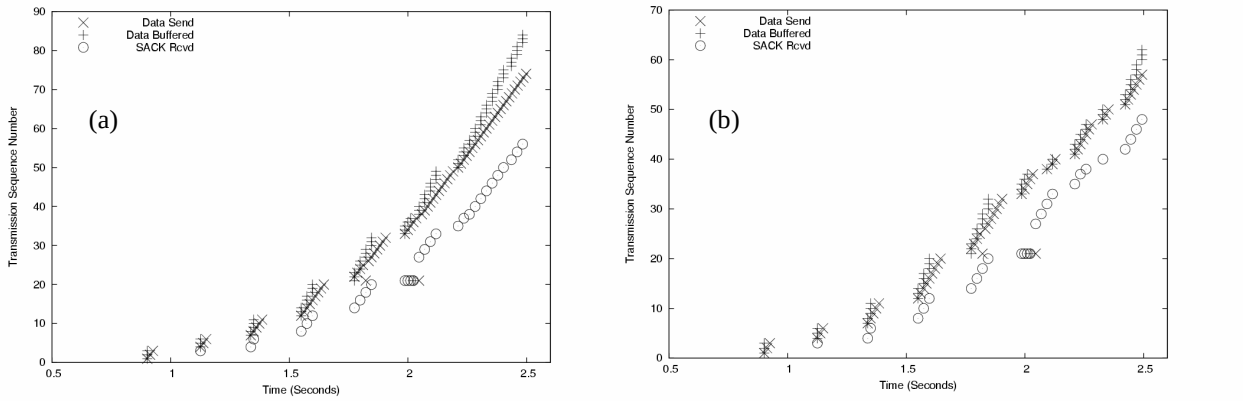

Figure 7: Effect of (a) Eifel and (b) DupTSN-based algorithm on a spurious fast retransmission (singlehomed SCTP)

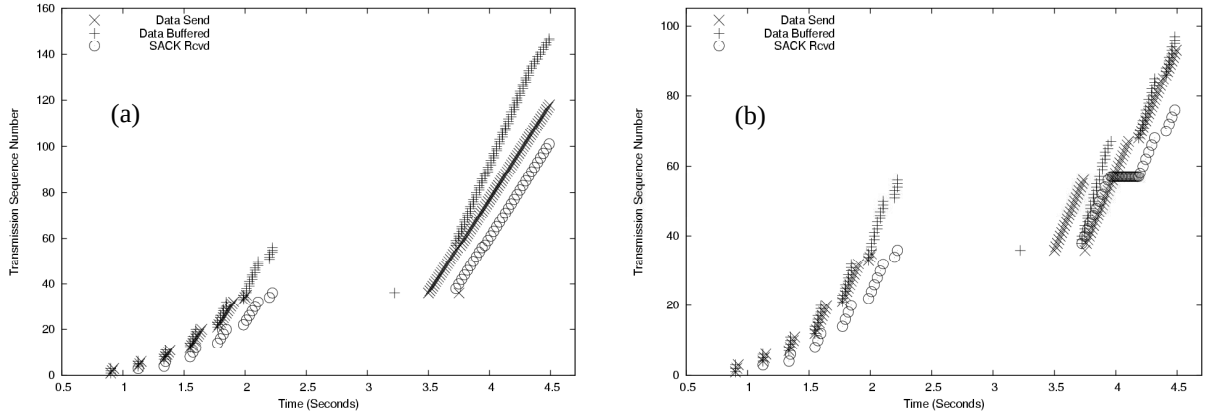

Figure 8: Effect of (a) Eifel and (b) the DupTSN-based algorithm on a spurious timeout (singlehomed SCTP)

spurious timeout (corresponding to Figure 2(b)), for a singlehomed environment. As the first ack corresponding to the retransmitted segment arrives, the Eifel-enabled sender detects the spurious timeout, and the response algorithm is invoked which restores the congestion control state and then resumes transmission from the top, suspending the unnecessary Go-Back-N retransmissions. The effective increase in goodput is significant as compared to plain SCTP. Figure 8(b) shows the DupTSN-based algorithm with a single spurious timeout. As noted earlier, the DupTSN-based sender waits for the arrival of all N acks for DupTSN reports after the loss recovery. Only then can the sender detect a spurious timeout. Moreover, the loss of a single ack out of the N acks carrying the DupTSN reports will cause the DupTSN-based algorithm to fail as it is unable to distinguish between a genuine loss and a needless retransmission.

As seen in Figure 8(b), the DupTSN-based algorithm is unable to avoid the Go-Back-N retransmissions invoked unnecessarily by the SCTP sender. The congestion window that can be restored after the detection is also smaller as compared to the Eifel algorithm. Figure 9 shows the effect of a long delay spike causing two spurious timeouts at an SCTP sender in a singlehomed scenario. The Eifel and DupTSN-based algorithms detect this spurious timeout and respond accordingly. The analysis is similar to that of a single spurious timeout in Figure 8, with the difference that the amount of congestion control state that can be restored is decayed [LG04].

While a spurious fast retransmit in both the single- and multihomed scenarios can be conveniently detected by the Eifel algorithm without relying on DupTSN reports, the detection of a long spurious timeout in a multihomed scenario has to rely on the DupTSN reports. Figure 10 shows the multihomed scenario of spurious timeouts (the multihomed topology corresponds to Figure 1). The graphs in Figure 10 (a)-(f) correspond to the combined trace for both primary and alternate interfaces.

Figure 10(a) depicts a single spurious timeout with plain SCTP. Figure 10(c) shows that the Eifel algorithm is able to detect this spurious timeout as the ack for the original transmit reaches the SCTP sender before the ack for the retransmit. The response is similar to that described earlier for a singlehomed scenario, and the effective increase in goodput is more than a factor of two. The Eifel algorithm also avoids the congestion window overgrowth problem described in section 4.

Figure 10(e) shows the effect of the DupTSN-based algorithm. Figure 10(b) shows a similar spurious timeout with plain SCTP in a multihomed scenario; however the length of the delay spike is increased. In this case, the retransmitted packet reaches the SCTP receiver via an alternate path before the original transmission reaches the SCTP receiver. Since the ack corresponding to the retransmission reaches the sender first, a decision whether or not the timeout was spurious cannot be taken. A sender can only take such a decision after receiving the DupTSN report for the retransmitted segment. Thus, the Eifel algorithm for SCTP as described in Figure 6 relies on DupTSN reports in such an event. This is the worst case behavior followed by the Eifel algorithm for SCTP. Figure 10(d) shows this behavior of the Eifel algorithm where DupTSN reports are used to detect a spurious timeout. Thus, Figure 10(d) is identical to Figure 10(f). The Go-Back-N retransmissions cannot be avoided. The only gain is restoring the congestion control state and accounting the spurious retransmissions. Note that all of the simulated links had the same propagation delay. In a scenario where the alternate path has a higher propagation delay, the Eifel algorithm may be able to detect the spurious timeout without relying on DupTSN reports. In general, the Eifel algorithm will have to rely on DupTSN reports to detect spurious timeouts if the following condition is true:

Delay-Spike[primary] > RTO[primary] + RTT[alternate]

6. A MACROSCOPIC EVALUATION

In this section, we evaluate the Eifel algorithm for SCTP on a macro basis and compare it with the DupTSN-based detection. The first metric used for evaluation is the achieved goodput, measured as the highest TSN an SCTP sender is able to send within a fixed interval of 200s. The second metric is the number of unnecessary retransmissions. This metric is useful especially for wireless scenarios and networks where battery power is directly proportional to the transmission power consumed in data (re)transmissions. Finally we also measure the download time taken for a 4M bulk data transfer. Unless specified, the network topology is the same as described in section 4. Sixty runs were performed for each scenario; the averages are shown in the figures.

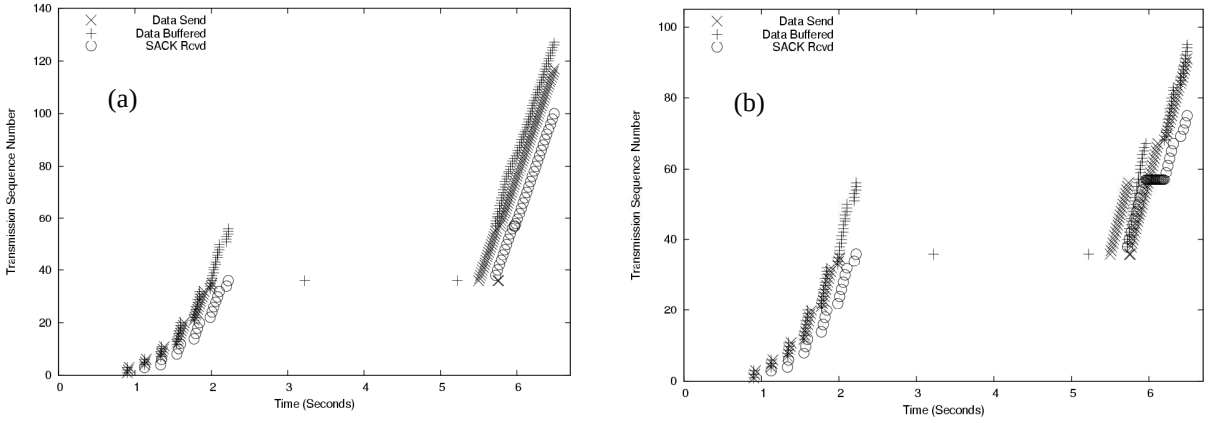

Figure 9: Effect of (a) Eifel and (b) DupTSN-based algorithm on a delay spike spanning two spurious timeouts (singlehomed SCTP)

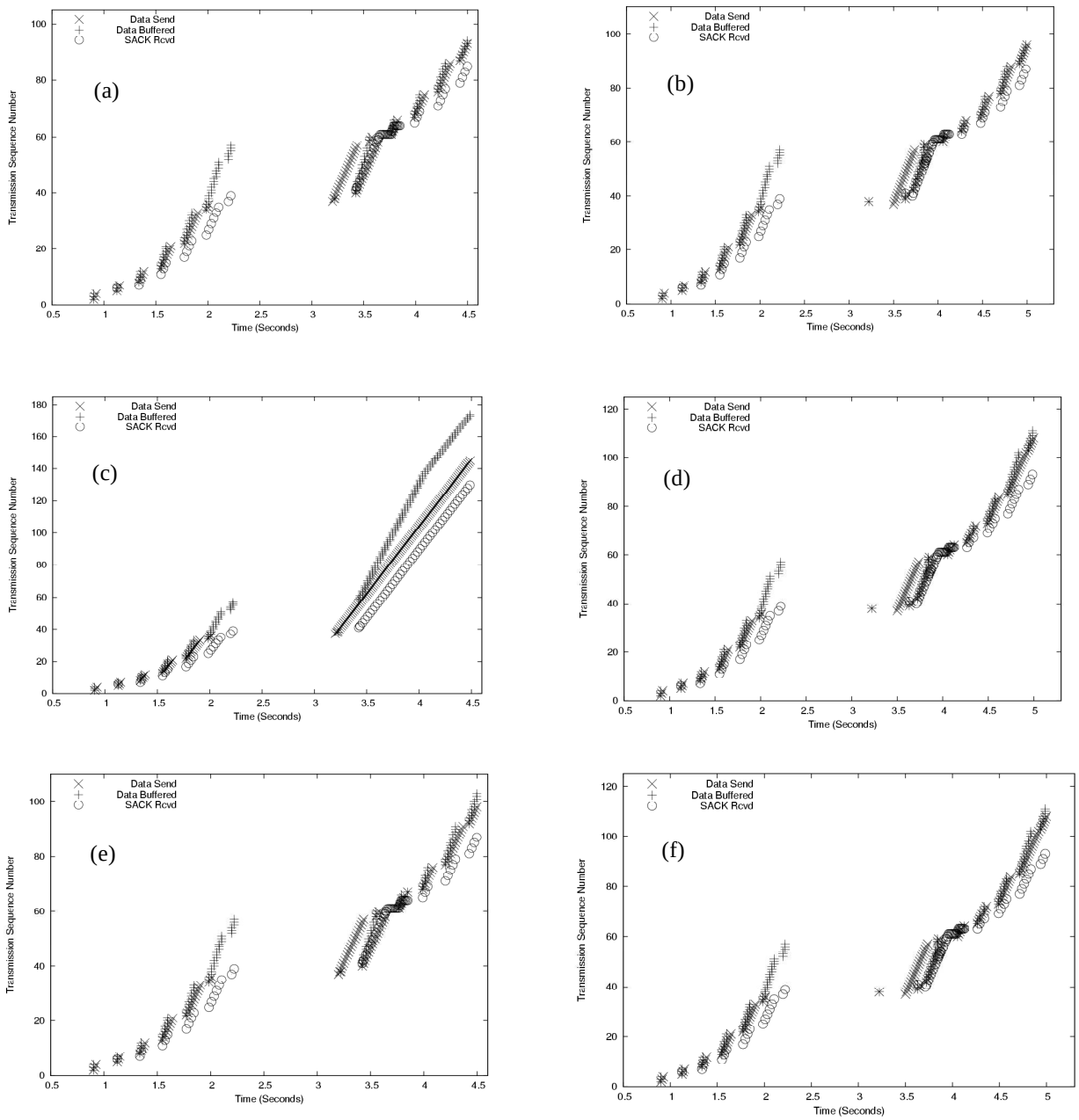

Figure 10: Effect of delay spikes in multihomed SCTP (combined interfaces):
(a), (c), (e) – A spurious timeout with (a) plain SCTP; (c) Eifel; (e) DupTSN-based algorithm
(b), (d), (f) – Long delay spike and a spurious timeout with (b) plain SCTP; (d) Eifel; (f) DupTSN-based algorithm
(note: (d) shows the failure of unmodified Eifel algorithm and the fallback to the DupTSN mechanism, thus (d) and (f) are similar)

Spurious Fast Retransmits

We evaluated the effect of packet reordering (simulated only on the forward path) using the hiccup module. Note that if the number of packets by which a particular packet is displaced is less than the duplicate ack threshold, the effect of the reordering is negligible (i.e. the reordering will not result in a spurious fast retransmit). However, our purpose was to cause packet reordering such that spurious retransmissions occur. Thus, the granularity of packet reordering was set such that *every reordering event causes a spurious fast retransmission.*

Figure 11(a) shows the goodput achieved by plain SCTP, SCTP with the Eifel algorithm, and SCTP with the DupTSN-based detection, in a bulk data transfer lasting 200 seconds. The reordering rate specifies the percentage of packets of the total transfer that experience reordering. A wide range of reordering rates from 0.5% to 15% has been evaluated, corresponding to measurements in [Pax97, BPS99]. Figure 11(a) shows that at a macro level the Eifel algorithm and the DupTSN-based detection perform nearly same and achieve higher goodput than plain SCTP at moderate reordering rates (0.5-5%) (even a 1% reordering rate significantly degrades plain SCTP's performance). At reordering rates above 5%, Eifel achieves higher goodput than both plain SCTP and SCTP with DupTSN-based detection. This is because at high reordering rates, reordering occurs for each window of data, and as seen in the micro simulations described earlier, the amount of cwnd that can be restored immediately by the DupTSN-based detection is smaller as compared to the Eifel algorithm.

Note that we used the timestamp variant of the Eifel algorithm in the simulations, hence every packet sent carries 12 bytes less payload. However, from measuring the download time taken for a 4M transfer, we noticed that this overhead is offset by the added robustness of the Eifel algorithm.

Figure 12 shows the transfer time taken for a 4M download with a reordering rate of 1%. Even with timestamp chunks enabled, the Eifel-enabled sender achieves faster download times than plain SCTP, and nearly the same as the DupTSN-based detection. The retransmit bit variant of the Eifel algorithm does not add the 12 byte overhead, and may thus result in yet faster download times.

Algorithm / File Size	Plain SCTP	SCTP-Eifel	SCTP-DupTSN Detection
4M	53.3s	34.7s	35.0s

Figure 12: Download times for a 4M transfer in a persistent reordering scenario with 1% reordering rate

Spurious Timeouts

To evaluate the effect of spurious timeouts, we created random delay spikes during a bulk transfer of 200 seconds and measured the goodput and the number of unnecessary retransmissions. The delay spikes were set to occur in time intervals that were chosen randomly from various intervals, specified by their minimum and maximum values: 10-20s, 30-40s, 50-60s, and 70-80s. The length of the delay spike was picked randomly between 3 and 10 seconds, a range that corresponds to realistic scenarios in some wireless systems [GPA+02].

Figure 11(b) shows the goodput achieved for the various delay spike time-intervals in a singlehomed SCTP scenario. SCTP with the Eifel algorithm outperforms both plain SCTP and SCTP with the DupTSN-based detection significantly by approximately 1000 packets (~9%) for the worst case (delay spikes occurring every 10-20s). In the case of longer intervals of 70-80s between delay spikes, Eifel achieves a gain in goodput of approximately 400 packets (~3%). SCTP with the DupTSN-based detection performs nearly same as plain SCTP. The reason has been identified in the microscopic evaluation in section 5, where it was shown that the DupTSN-based detection is not able to avoid the Go-Back-N loss recovery, whereas the Eifel algorithm detects a spurious timeout upon the arrival of first "cover ack", thus avoiding Go-Back-N retransmissions and restoring the congestion window to continue with regular transmissions.

Figure 13(a) shows the goodput achieved for the same delay spike scenarios as described above, but with multihomed SCTP endpoints. With the current retransmission policy of SCTP, it may

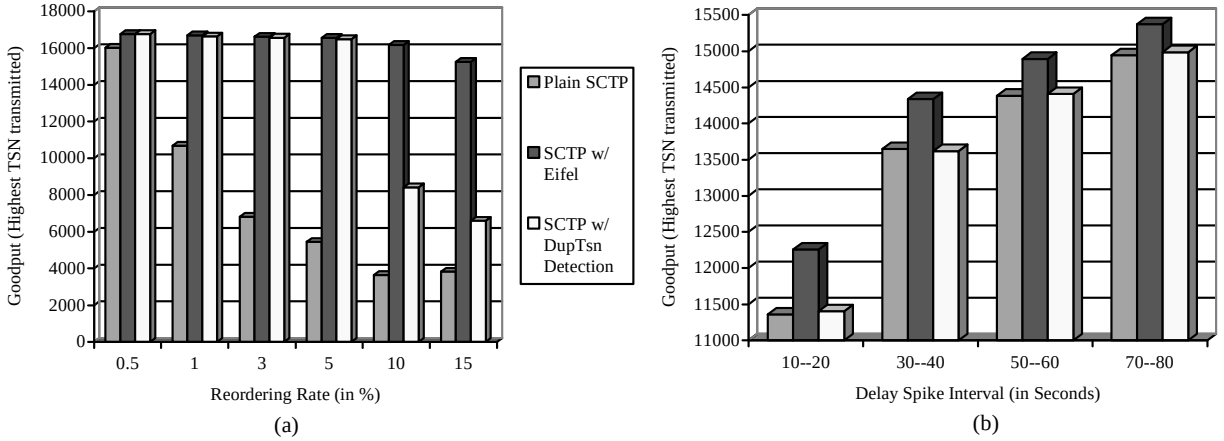

Figure 11: Goodput achieved in a 200s transfer in the case of (a) packet reordering (b) spurious timeouts due to delay spikes in singlehomed SCTP

be possible to perform the entire Go-Back-N recovery on the alternate path even before the series of original transmissions reaches the SCTP receiver. For a scenario where the retransmission overtakes the original transmission, we extended the Eifel algorithm to rely on DupTSN reports to detect spurious retransmissions. Thus, as seen from Figure 13(a), the Eifel and DupTSN-based algorithms exhibit similar performance and do not present any noticeable benefits in performance over plain SCTP.

Figure 13(b) shows the number of unnecessary retransmissions with plain SCTP, SCTP with the Eifel algorithm, and SCTP with DupTSN-based detection in a singlehomed SCTP configuration. As expected, the Eifel algorithm reduces the number of unnecessary retransmissions significantly. This is because an Eifel-enabled sender detects spurious timeouts on the receipt of the first "cover ack" and prevents further unnecessary retransmissions. However, the DupTSN-based detection does not detect a spurious retransmission until having received all DupTSN reports, and is therefore unable to avoid unnecessary retransmissions.

Figure 14 shows the time taken for a 4M transfer with delay spikes of length 3-10s, occurring in intervals of 10-20s. In this simulation we used a link bandwidth corresponding to a satellite link with a bandwidth of 250Kbps, and a propagation delay of 100ms (the previous simulations used 1Mbps and 100ms, respectively). We lowered the link bandwidth to witness the effect of delay spikes and the gain achieved by the Eifel algorithm and the DupTSN-based algorithm. Figure 14(a) shows the download time in a singlehomed SCTP scenario. The Eifel algorithm reduces the download times by approximately 35s (~18%) in comparison with plain SCTP, and by 15s (~8%) in comparison with DupTSN-based detection. Figure 14(b) shows the same transfer for a multihomed SCTP scenario. The Eifel algorithm reduces the download times by approximately 17s (~9%) in comparison with plain SCTP, and by 10s (~5%) in comparison with DupTSN-based detection. Note that we again used the timestamp variant of Eifel, thereby introducing an additional 12 byte overhead. The retransmit bit approach would eliminate the need for this overhead and thus further reduce the download time.

Algorithm/ File Size	Plain SCTP	SCTP-Eifel	SCTP-DupTSN Detection
4M	197.2s	163.4s	178.3s

(a)

Algorithm/ File Size	Plain SCTP	SCTP-Eifel	SCTP-DupTSN Detection
4M	199.8s	182.1s	191.9s

(b)

Figure 14: Download times for a 4M transfer in a scenario of delay spikes of length 3-10s, occurring every 10-20s with (a) singlehomed SCTP, and (b) multihomed SCTP

7. CONCLUSION

This paper discussed the problem space of network anomalies such as packet reordering and delay spikes in the context of SCTP. It was shown that spurious timeouts may result in congestion window overgrowth on the alternate destination due to the retransmission ambiguity in SCTP. An extended version of the Eifel algorithm was proposed for SCTP, including an option to integrate the DupTSN-based algorithm with the Eifel algorithm. Another extension was the introduction of new local state at the sender ("cover ack") to ensure that congestion-based losses and retransmissions are not masked by spurious retransmits. Since standard SCTP has no mechanism equivalent to TCP timestamps, it was proposed that the Eifel algorithm could use either a new SCTP timestamp chunk type, or a "retransmit bit" in the headers of DATA/SACK chunks. Finally, this paper evaluated the Eifel and DupTSN-based algorithms on a microscopic as well as a macroscopic basis using simulations, and showed that SCTP with the Eifel algorithm in many cases achieves better performance than both plain SCTP and the DupTSN-based algorithm.

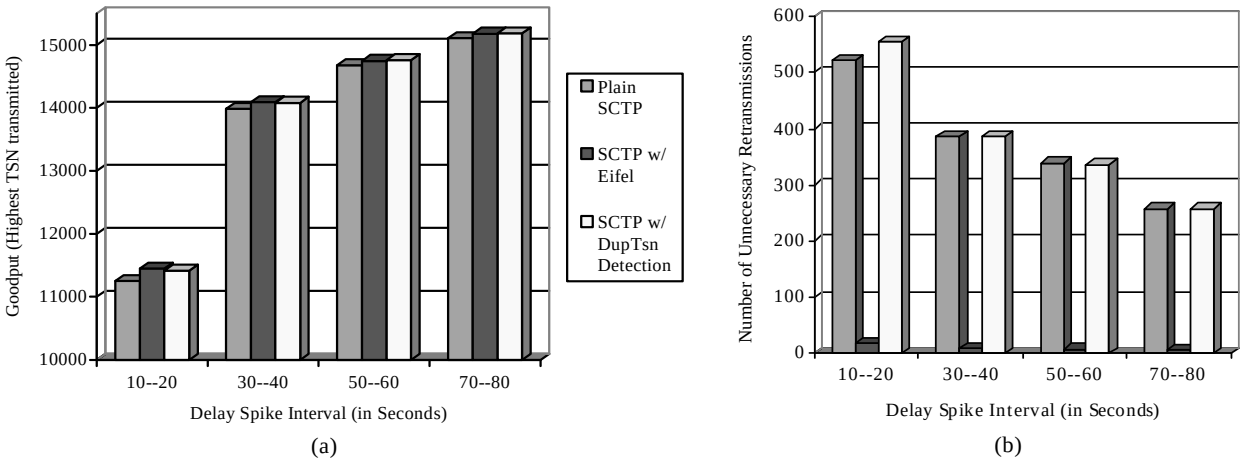

Figure 13: (a) Goodput achieved in a 200s transfer in the case of spurious timeouts in multihomed SCTP (b) Unnecessary retransmissions caused by spurious timeouts in singlehomed SCTP

The first set of results presented in this paper evaluated the Eifel and DupTSN-based algorithms on a microscopic basis. It was shown that the Eifel algorithm is able to detect and respond more quickly to spurious fast retransmits as compared to the DupTSN-based algorithm, thus being able to revert congestion control state faster. For delay spikes leading to spurious timeouts, we discussed both singlehomed and multihomed scenarios. A singlehomed SCTP scenario behaves much like TCP, and the Eifel algorithm detects spurious timeouts soon enough to avoid unnecessary Go-Back-N retransmissions. The DupTSN-based algorithm is unable to avoid Go-Back-N retransmissions. It was discussed that the loss of a single ack carrying DupTSN information causes the DupTSN-based algorithm to fail. For a multihomed scenario, the Eifel algorithm was able to avoid the congestion window overgrowth problem discussed earlier. For delay spikes on the primary path large enough to cause the retransmission on the alternate path to arrive before the original transmission, the Eifel algorithm may fall back to the DupTSN-based detection as there is no clear indication that the original transmission was delayed or lost.

The second set of results presented in this paper evaluated the Eifel algorithm and the DupTSN-based algorithm on a macroscopic basis. The metrics used for evaluation were goodput, download time, and number of unnecessary retransmissions. For spurious fast retransmits, both the Eifel and DupTSN-based algorithms achieved higher goodput than plain SCTP, and performed close to each other. Eifel outperformed the DupTSN-based algorithm by reducing the download time further, even when using a 12 byte timestamp chunk. For spurious timeouts with singlehomed SCTP endpoints, the DupTSN-based algorithm did not offer a substantial increase in goodput as compared to the Eifel algorithm, which increased goodput significantly. The Eifel algorithm reduced the number of unnecessary retransmissions significantly as compared to plain SCTP or the DupTSN-based algorithm. For spurious timeouts with multihomed SCTP endpoints, both the Eifel and DupTSN-based algorithms performed close to each other.

The results show that adding the Eifel algorithm to SCTP improves performance for networks experiencing packet reordering and/or large delay variations. Even with the added overhead of using 12 byte timestamp chunks, the Eifel algorithm performs better. Using a per-chunk retransmit bit instead of the timestamp chunk would eliminate the need for this 12 byte overhead and may thus result in further performance improvement.

Future work or an extension to the work presented in this paper includes the following:

- This paper uses a simplistic network topology and model for reordering and delay spikes. A more realistic simulation setup would improve the accuracy of the results presented in this paper.
- While simulations capture the essence of mechanisms and provide candidate solutions, a real world implementation would further validate our conclusions.
- This paper does not evaluate response algorithms and does not present an SCTP specific response algorithm. Future work could include both.
- F-RTO, a heuristic scheme for detecting spurious timeouts, could be evaluated and compared with the Eifel and DupTSN-based algorithms.

8. ACKNOWLEDGEMENTS

Hannes Ekström provided much valuable help with the simulation traces. Armando Caro implemented the timestamp chunk for SCTP in NS. Randall Stewart, Kacheong Poon, Wolfgang Schrüfer, Brian Bidulock and several other members of the Transport Area discussion list provided early feedback on the proposal. Our sincere thanks to all.

The authors acknowledge the contribution of Armando Caro, Janardhan Iyengar and Keyur Shah for their comments on an earlier draft of this paper. Our sincere thanks go to John Wroclawski and the anonymous reviewers of ACM CCR for their valuable feedback.

9. REFERENCES

[All00] M. Allman. *A Web Server's View of the Transport Layer*. ACM Computer Communication Review, vol. 30(5), October 2000.

[AP99] M. Allman, V. Paxson. *On Estimating End-to-End Network Path Properties*. ACM SIGCOMM, September 1999.

[APS99] M. Allman, V. Paxson, W. Stevens. *TCP Congestion Control*. RFC 2581, April 1999.

[BA02] E. Blanton, M. Allman. *On Making TCP More Robust to Packet Reordering*. ACM Computer Communication Review, 32(1), January 2002.

[BA04] E. Blanton, M. Allman. *Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions*. RFC 3708, February 2004.

[BAF+03] E. Blanton, M. Allman, K. Fall, L. Wang. *A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP*. RFC 3517, April 2003.

[BDA03] E. Blanton, R. Dimond, M. Allman. *Practices for TCP Senders in the Face of Segment Reordering.* draft-blanton-tcp-reordering-00.txt, February 2003. (Work in Progress)

[Bol93] J. Bolot. *Characterizing End-to-End Packet Delay and Loss in the Internet.* Journal of High Speed Networks, vol. 2(3), September 1993.

[BPS99] J. Bennet, C. Partridge, N. Shectman. *Packet Reordering is not a Pathological Network Behavior*. IEEE/ACM Transactions on Networking, December 1999.

[Bra89] R. Braden. *Requirements for Internet Hosts–Communication Layers*. RFC1122, October 1989.

[CAI+03] A. Caro, P. Amer, J. Iyengar, R. Stewart. *Retransmission Policies with Transport Layer Multihoming*. ICON, September 2003.

[CIA+03] A. Caro, J. Iyengar, P. Amer, S. Ladha, G. Heinz, K. Shah. *SCTP: A Proposed Standard for Robust Internet Data Transport*. IEEE Computer, 36(11):56-63, November 2003.

[FMM+00] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky. *An Extension to the Selective Acknowledgement (SACK) Option for TCP*. RFC 2883. July 2000.

[Gur01] A. Gurtov. *Effect of Delays on TCP Performance*, Proceedings of IFIP Personal Wireless Communications, 2001.

[GL02] A. Gurtov, R. Ludwig. *Evaluating the Eifel Algorithm for TCP in a GPRS Network.* Proceedings of European Wireless, February 2002.

[GL03] A. Gurtov, R. Ludwig. *Responding to Spurious Timeouts in TCP*. IEEE INFOCOM, April 2003.

[GPA+02] A. Gurtov, M. Passoja, O. Aalto, M. Raitola. *Multilayer Protocol Tracing in a GPRS Network*. Proceedings of IEEE Vehicular Technology Conference, September 2002.

[ICA+03] J. Iyengar, A. Caro, P. Amer, G. Heinz, R. Stewart. *Making SCTP Robust to Changeover*. SPECTS, July 2003.

[ISA+03] J. Iyengar, K. Shah, P. Amer, R. Stewart. *Concurrent Multipath Transfer Using SCTP Multihoming.* Technical Report TR2004-02, CIS Dept, U of Delaware, September 2003.

[Jac88] V. Jacobson. *Congestion Avoidance and Control*. ACM SIGCOMM 1988.

[JBB92] V. Jacobson, R. Braden, D. Borman. *TCP Extensions for High Performance*. RFC 1323. May 1992.

[KAG+01] J. Korhonen, O. Aalto, A. Gurtov, H. Laamanen, *Measured Performance of GSM HSCSD and GPRS*, IEEE Conference on Communications, June 2001.

[KP87] P. Karn, C. Partridge. *Improving Round Trip Time Estimates in Reliable Transport Protocols.* ACM SIGCOMM, August 1987.

[LG04] R. Ludwig, A. Gurtov. *The Eifel Response Algorithm for TCP*. draft-ietf-tsvwg-tcp-eifel-response-05.txt, March 2004. (Work in Progress)

[LK00] R. Ludwig, R. H. Katz. *The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions*. ACM Computer Communication Review, 30(1), January 2000.

[LM03] R. Ludwig, M. Meyer. *The Eifel Detection Algorithm for TCP*. RFC3522, April 2003.

[LR01] D. Loguinov and H. Radha. *Measurement Study of Low-bitrate Internet Video Streaming*. ACM SIGCOMM Internet Measurement Workshop, November 2001.

[MMF+96] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow. *TCP Selective Acknowledgement Options*. RFC 2018, October 1996.

[NS] UC Berkeley, LBL, USC/ISI, Xerox Parc. *Ns-2 software and documentation, version 2.1b8*. http://www.isi.edu

[PA00] V. Paxson, M. Allman. *Computing TCP's Retransmission Timer*. RFC2988, November 2000.

[Pax97] V. Paxson. *End-to-End Internet Packet Dynamics*. ACM SIGCOMM, September 1997.

[PEL] Protocol Engineering Laboratory. http://pel.cis.udel.edu

[Pos81] J. Postel. *Transmission Control Protocol*. RFC 793, September 1981.

[Sch00] M. Schläger, *NS TCP Eifel Page*. http://www-tkn.ee.tu-berlin.de/~morten/eifel/ns-eifel.html

[SKR04] P. Sarolahti, M. Kojo, K. Raatikainen. *F-RTO: A TCP RTO Recovery Algorithm for Avoiding Unnecessary Retransmission*. draft-ietf-tsvwg-tcp-frto-01.txt, February 2004. (Work in Progress)

[SM01] R. Stewart, C. Metz. *SCTP: A New Transport Protocol for TCP/IP*. IEEE Internet Computing, December 2001.

[SOA+03] R. Stewart, L. Ong, I. Arias-Rodriguez, K. Poon, P. Conrad, A. Caro, M. Tuexen. *Stream Control Transmission Protocol (SCTP) Implementers Guide*. draft-ietf-tsvwg-sctpimpguide-10.txt, November 2003. (Work in progress)

[SXM+00] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson. *Stream Control Transmission Protocol*. RFC 2960, October 2000.

[YK02] M. Yavuz, F. Khafizov. *TCP over Wireless Links with Variable Bandwidth*. IEEE Vehicular Technology Conference, September 2002.