TCP Maintenance and Minor Extensions (tcpm)
Internet-Draft
Intended status: Experimental
Expires: October 10, 2015

P. Hurtig
A. Brunstrom
Karlstad University
A. Petlund
Simula Research Laboratory AS
M. Welzl
University of Oslo

April 8, 2015

TCP and SCTP RTO Restart
draft-ietf-tcpm-rtorestart-06

Abstract

This document describes a modified algorithm for managing the TCP and SCTP retransmission timers that provides faster loss recovery when there is a small amount of outstanding data for a connection. The modification, RTO Restart (RTOR), allows the transport to restart its retransmission timer more aggressively in situations where fast retransmit cannot be used. This enables faster loss detection and recovery for connections that are short-lived or application-limited.

Status of This Memo

Copyright Notice

1. Introduction

TCP uses two mechanisms to detect segment loss. First, if a segment is not acknowledged within a certain amount of time, a retransmission timeout (RTO) occurs, and the segment is retransmitted [RFC6298]. While the RTO is based on measured round-trip times (RTTs) between the sender and receiver, it also has a conservative lower bound of 1 second to ensure that delayed segments are not mistaken as lost. Second, when a sender receives dupACKs, the fast retransmit algorithm infers segment loss and triggers a retransmission [RFC5681]. Duplicate acknowledgments are generated by a receiver when out-of-order segments arrive. As both segment loss and segment reordering cause out-of-order arrival, fast retransmit waits for three dupACKs before considering the segment as lost. In some situations, however, the number of outstanding segments is not enough to trigger three dupACKs, and the sender must rely on lengthy RTOs for loss recovery.

The number of outstanding segments can be small for several reasons:

(1) The connection is limited by the congestion control when the path has a low total capacity (bandwidth-delay product) or the connection's share of the capacity is small. It is also limited by the congestion control in the first few RTTs of a connection or after an RTO when the available capacity is probed using slow-start.

(2) The connection is limited by the receiver's available buffer space.

(3) The connection is limited by the application if the available capacity of the path is not fully utilized (e.g. interactive applications), or at the end of a transfer.

While the reasons listed above are valid for any flow, the third reason is most common for applications that transmit short flows, or use a bursty transmission pattern. A typical example of applications that produce short flows are web-based applications. [RJ10] shows that 70% of all web objects, found at the top 500 sites, are too small for fast retransmit to work. [FDT13] shows that about 77% of all retransmissions sent by a major web service are sent after RTO expiry. Applications with bursty transmission patterns often send

data in response to actions, or as a reaction to real life events. Typical examples of such applications are stock trading systems, remote computer operations, online games, and web-based applications using persistent connections. What is special about this class of applications is that they often are time-dependant, and extra latency can reduce the application service level [P09].

The RTO Restart (RTOR) mechanism described in this document makes the RTO slightly more aggressive when the number of outstanding segments is too small for fast retransmit to work, in an attempt to enable faster loss recovery for all segments while being robust to reordering. While RTOR still conforms to the requirement in [RFC6298] that segments must not be retransmitted earlier than RTO seconds after their original transmission, it could increase the risk of spurious timeout. Spurious timeouts can degrade the performance of flows with multiple bursts of data, as a burst following a spurious timeout might not fit within the reduced congestion window (cwnd). There are, however, several techniques to mitigate the effects of such unnecessary retransmissions (cf. [RFC4015]).

While this document focuses on TCP, the described changes are also valid for the Stream Control Transmission Protocol (SCTP) [RFC4960] which has similar loss recovery and congestion control algorithms.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document introduces the following variables:

The number of previously unsent segments (prevunsnt): The number of segments that a sender has queued for transmission, but has not yet sent.

RTO Restart threshold (rrthresh): RTOR is enabled whenever the sum of the number of outstanding and previously unsent segments (prevunsnt) is below this threshold.

3. RTO Restart Overview

The RTO management algorithm described in [RFC6298] recommends that the retransmission timer is restarted when an acknowledgment (ACK) that acknowledges new data is received and there is still outstanding data. The restart is conducted to guarantee that unacknowledged segments will be retransmitted after approximately RTO seconds. However, by restarting the timer on each incoming ACK,

retransmissions are not typically triggered RTO seconds after their previous transmission but rather RTO seconds after the last ACK arrived. The duration of this extra delay depends on several factors but is in most cases approximately one RTT. Hence, in most situations, the time before a retransmission is triggered is equal to "RTO + RTT".

The standardized RTO timer management is illustrated in Figure 1 where a TCP sender transmits three segments to a receiver. The arrival of the first and second segment triggers a delayed ACK (delACK) [RFC1122], which restarts the RTO timer at the sender. The RTO is restarted approximately one RTT after the transmission of the third segment. Thus, if the third segment is lost, as indicated in Figure 1, the effective loss detection time is "RTO + RTT" seconds. In some situations, the effective loss detection time becomes even longer. Consider a scenario where only two segments are outstanding. If the second segment is lost, the time to expire the delACK timer will also be included in the effective loss detection time.

```
         Sender                                     Receiver
                       ...
         DATA [SEG 1] ----------------------> (ack delayed)
         DATA [SEG 2] ----------------------> (send ack)
         DATA [SEG 3] ----X         /-------- ACK
         (restart RTO)  <----------/
                       ...
         (RTO expiry)
         DATA [SEG 3] ---------------------->
```

Figure 1: RTO restart example

During normal TCP bulk transfer the current RTO restart approach is not a problem. Actually, as long as enough segments arrive at a receiver to enable fast retransmit, RTO-based loss recovery should be avoided. RTOs should only be used as a last resort, as they drastically lower the congestion window compared to fast retransmit. The current approach can therefore be beneficial -- it is described in [EL04] to act as a "safety margin" that compensates for some of the problems that the authors have identified with the standard RTO calculation. Notably, the authors of [EL04] also state that "this safety margin does not exist for highly interactive applications where often only a single packet is in flight."

Although fast retransmit is preferrable there are situations where timeouts are appropriate, or the only choice. For example, if the network is severely congested and no segments arrive RTO-based

recovery should be used. In this situation, the time to recover from the loss(es) will not be the performance bottleneck. However, for connections that do not utilize enough capacity to enable fast retransmit, RTO-based loss detection is the only choice and the time required for this can become a performance bottleneck.

4. RTOR Algorithm

To enable faster loss recovery for connections that are unable to use fast retransmit, RTOR can be used. By resetting the timer to "RTO - T_earliest", where T_earliest is the time elapsed since the earliest outstanding segment was transmitted, retransmissions will always occur after exactly RTO seconds. This approach makes the RTO more aggressive than the standardized approach in [RFC6298] but still conforms to the requirement in [RFC6298] that segments must not be retransmitted earlier than RTO seconds after their original transmission.

This document specifies an OPTIONAL sender-only modification to TCP and SCTP which updates step 5.3 in Section 5 of [RFC6298] (and a similar update in Section 6.3.2 of [RFC4960] for SCTP). A sender that implements this method MUST follow the algorithm below:

```
When an ACK is received that acknowledges new data:

(1)  Set T_earliest = 0.

(2)  If the sum of the number of outstanding and previously unsent
     segments (prevunsnt) is less than an RTOR threshold
     (rrthresh), set T_earliest to the time elapsed since the
     earliest outstanding segment was sent.

(3)  Restart the retransmission timer so that it will expire after
     (for the current value of RTO):

     (a)  RTO - T_earliest, if RTO - T_earliest > 0.

     (b)  RTO, otherwise.
```

The RECOMMENDED value of rrthresh is four, as it will prevent RTOR from being more aggressive and potentially causing RTOs instead of fast retransmits. This update needs TCP implementations to track the time elapsed since the transmission of the earliest outstanding segment (T_earliest). As RTOR is only used when the amount of outstanding and previously unsent data is less than rrthresh segments, TCP implementations also need to track whether the amount of outstanding and previously unsent data is more, equal, or less than rrthresh segments. Although some packet-based TCP

implementations (e.g. Linux TCP) already track both the transmission times of all segments and also the number of outstanding segments, not all implementations do. Section 5.3 describes how to implement segment tracking for a general TCP implementation. To use RTOR, the calculated expiration time MUST be positive (step 3(a) in the list above); this is required to ensure that RTOR does not trigger retransmissions prematurely when previously retransmitted segments are acknowledged.

5. Discussion

In this section, we discuss the applicability and a number of issues surrounding RTOR.

5.1. Applicability

The currently standardized algorithm has been shown to add at least one RTT to the loss recovery process in TCP [LS00] and SCTP [HB11][PBP09]. For applications that have strict timing requirements (e.g. interactive web) rather than throughput requirements, using RTOR could be beneficial because the RTT and also the delACK timer of receivers are often large components of the effective loss recovery time. Measurements in [HB11] have shown that the total transfer time of a lost segment (including the original transmission time and the loss recovery time) can be reduced by 35% using RTOR. These results match those presented in [PGH06][PBP09], where RTOR is shown to significantly reduce retransmission latency.

There are also traffic types that do not benefit from RTOR. One example of such traffic is bulk transmission. The reason why bulk traffic does not benefit from RTOR is that such traffic flows mostly have four or more segments outstanding, allowing loss recovery by fast retransmit. However, there is no harm in using RTOR for such traffic as the algorithm only is active when the amount of outstanding and unsent segments are less than rrthresh (default 4).

Given that RTOR is a mostly conservative algorithm, it is suitable for experimentation as a system-wide default for TCP traffic.

5.2. Spurious Timeouts

RTOR can in some situations reduce the loss detection time and thereby increase the risk of spurious timeouts. In theory, the retransmission timer has a lower bound of 1 second [RFC6298], which limits the risk of having spurious timeouts. However, in practice most implementations use a significantly lower value. Initial measurements, show slight increases in the number of spurious timeouts when such lower values are used [RHB15]. However, further

experiments, in different environments and with different types of traffic, are encouraged to quantify such increases more reliably.

Does a slightly increased risk matter? Generally, spurious timeouts have a negative effect on the network as segments are transmitted needlessly. However, recent experiments do not show a significant increase in network load for a number of realistic scenarios [RHB15]. Another problem with spurious retransmissions is related to the performance of TCP/SCTP, as the congestion window is reduced to one segment when timeouts occur [RFC5681]. This could be a potential problem for applications transmitting multiple bursts of data within a single flow, e.g. web-based HTTP/1.1 and HTTP/2.0 applications. However, results from recent experiments involving persistent web traffic [RHB15] only revealed a net gain of using RTOR. Other types of flows, e.g. long-lived bulk flows, are not affected as the algorithm is only applied when the amount of outstanding and unsent segments is less than rrthresh. Furthermore, short-lived and application-limited flows are typically not affected as they are too short to experience the effect of congestion control or have a transmission rate that is quickly attainable.

While a slight increase in spurious timeouts has been observed using RTOR, it is not clear whether the effects of this increase mandate any future algorithmic changes or not -- especially since most modern operating systems already include mechanisms to detect [RFC3522][RFC3708][RFC5682] and resolve [RFC4015] possible problems with spurious retransmissions. Further experimentation is needed to determine this and thereby move this specification from experimental to proposed standard. For instance, RTOR has not been evaluated in the context of mobile networks. Mobile networks often incur highly variable RTTs (delay spikes), due to e.g. handovers, and would therefore be a useful scenario for further experimentation.

5.3. Tracking Outstanding and Previously Unsent Segments

The method of tracking outstanding and previously unsent segments will probably differ depending on the actual TCP implementation. For packet-based TCP implementations, tracking outstanding segments is often straightforward and can be implemented using a simple counter. For byte-based TCP stacks it is a more complex task. Section 3.2 of [RFC5827] outlines a general method of tracking the number of outstanding segments. The same method can be used for RTOR. The implementation will have to track segment boundaries to form an understanding as to how many actual segments have been transmitted, but not acknowledged. This can be done by the sender tracking the boundaries of the rrthresh segments on the right side of the current window (which involves tracking rrthresh + 1 sequence numbers in TCP). This could be done by keeping a circular list of the segment

boundaries, for instance. Cumulative ACKs that do not fall within this region indicate that at least rrthresh segments are outstanding, and therefore RTOR is not enabled. When the outstanding window becomes small enough that RTOR can be invoked, a full understanding of the number of outstanding segments will be available from the rrthresh + 1 sequence numbers retained. (Note: the implicit sequence number consumed by the TCP FIN bit can also be included in the tracking of segment boundaries.)

Tracking the number of previously unsent segments depends on the segmentation strategy used by the TCP implementation, not whether it is packet-based or byte-based. In the case segments are formed directly on socket writes, the process of determining the number of previously unsent segments should be trivial. In the case that unsent data can be segmented (or re-segmented) as long as it still is unsent, a straightforward strategy could be to divide the amount of unsent data (in bytes) with the SMSS to obtain an estimate. In some cases, such an estimation could be too simplistic, depending on the segmentation strategy of the TCP implementation. However, this estimation is not critical to RTOR. For instance, implementations can use a simplified method by setting prevunsnt to rrthresh whenever previously unsent data is available, and set prevunsnt to zero when no new data is available. This will disable RTOR in the presence of unsent data and only use the number of outstanding segments to enable/disable RTOR. This strategy was used in an earlier version of the algorithm and works well. The addition of tracking prevunsnt was only made to optimize a corner case in which RTOR was unnecessarily disabled.

6. Related Work

There are several proposals that address the problem of not having enough ACKs for loss recovery. In what follows, we explain why the mechanism described here is complementary to these approaches:

The limited transmit mechanism [RFC3042] allows a TCP sender to transmit a previously unsent segment for each of the first two dupACKs. By transmitting new segments, the sender attempts to generate additional dupACKs to enable fast retransmit. However, limited transmit does not help if no previously unsent data is ready for transmission. [RFC5827] specifies an early retransmit algorithm to enable fast loss recovery in such situations. By dynamically lowering the number of dupACKs needed for fast retransmit (dupthresh), based on the number of outstanding segments, a smaller number of dupACKs is needed to trigger a retransmission. In some situations, however, the algorithm is of no use or might not work properly. First, if a single segment is outstanding, and lost, it is impossible to use early retransmit. Second, if ACKs are lost, early

retransmit cannot help. Third, if the network path reorders segments, the algorithm might cause more unnecessary retransmissions than fast retransmit. The recommended value of RTOR's rrthresh variable is based on the dupthresh, but is possible to adapt to allow tighter integration with other experimental algorithms such as early retransmit.

Tail Loss Probe [TLP] is a proposal to send up to two "probe segments" when a timer fires which is set to a value smaller than the RTO. A "probe segment" is a new segment if new data is available, else a retransmission. The intention is to compensate for sluggish RTO behavior in situations where the RTO greatly exceeds the RTT, which, according to measurements reported in [TLP], is not uncommon. Furthermore, TLP also tries to circumvent the congestion window reset to one segment by instead enabling fast recovery. The Probe timeout (PTO) is normally two RTTs, and a spurious PTO is less risky than a spurious RTO because it would not have the same negative effects (clearing the scoreboard and restarting with slow-start). TLP is a more advanced mechanism than RTOR, requiring e.g. SACK to work, and is often able to reduce loss recovery times more. However, it also increases the amount of spurious retransmissions noticeably, as compared to RTOR [RHB15].

TLP is applicable in situations where RTOR does not apply, and it could overrule (yielding a similar general behavior, but with a lower timeout) RTOR in cases where the number of outstanding segments is smaller than four and no new segments are available for transmission. The PTO has the same inherent problem of restarting the timer on an incoming ACK, and could be combined with a strategy similar to RTOR's to offer more consistent timeouts.

7. SCTP Socket API Considerations

This section describes how the socket API for SCTP defined in [RFC6458] is extended to control the usage of RTO restart for SCTP.

Please note that this section is informational only.

7.1. Data Types

This section uses data types from [IEEE.1003-1G.1997]: uintN_t means an unsigned integer of exactly N bits (e.g., uint16_t). This is the same as in [RFC6458].

7.2. Socket Option for Controlling the RTO Restart Support (SCTP_RTO_RESTART)

This socket option allows the enabling or disabling of RTO Restart for SCTP associations.

Whether RTO Restart is enabled or not per default is implementation specific.

This socket option uses IPPROTO_SCTP as its level and SCTP_RTO_RESTART as its name. It can be used with getsockopt() and setsockopt(). The socket option value uses the following structure defined in [RFC6458]:

```
struct sctp_assoc_value {
  sctp_assoc_t assoc_id;
  uint32_t assoc_value;
};
```

assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets, this parameter indicates upon which association the user is performing an action. The special sctp_assoc_t SCTP_{FUTURE|CURRENT|ALL}_ASSOC can also be used in assoc_id for setsockopt(). For getsockopt(), the special value SCTP_FUTURE_ASSOC can be used in assoc_id, but it is an error to use SCTP_{CURRENT|ALL}_ASSOC in assoc_id.

assoc_value: A non-zero value encodes the enabling of RTO restart whereas a value of 0 encodes the disabling of RTO restart.

sctp_opt_info() needs to be extended to support SCTP_RTO_RESTART.

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

This document discusses a change in how to set the retransmission timer's value when restarted. Therefore, the security considerations found in [RFC6298] apply to this document. No additional security problems have been identified with RTO Restart at this time.

10. Acknowledgements

The authors wish to thank Michael Tuexen for contributing the SCTP Socket API considerations and Godred Fairhurst, Yuchung Cheng, Mark Allman, Anantha Ramaiah, Richard Scheffenegger, Nicolas Kuhn,

Alexander Zimmermann, and Michael Scharf for commenting on the draft and the ideas behind it.

All the authors are supported by RITE (http://riteproject.eu/), a research project (ICT-317700) funded by the European Community under its Seventh Framework Program. The views expressed here are those of the author(s) only. The European Commission is not liable for any use that may be made of the information in this document.

11. Changes from Previous Versions

RFC-Editor note: please remove this section prior to publication.

11.1. Changes from draft-ietf-...-05 to -06

- o Added socket API considerations, after discussing the draft in tsvwg.

11.2. Changes from draft-ietf-...-04 to -05

- o Introduced variable to track the number of previously unsent segments.
- o Clarified many concepts, e.g. extended the description on how to track outstanding and previously unsent segments.
- o Added a reference to initial measurements on the effects of using RTOR.
- o Improved wording throughout the document.

11.3. Changes from draft-ietf-...-03 to -04

- o Changed the algorithm to allow RTOR when there is unsent data available, but the cwnd does not allow transmission.
- o Changed the algorithm to not trigger if RTOR <= 0.
- o Made minor adjustments throughout the document to adjust for the algorithmic change.
- o Improved the wording throughout the document.

11.4. Changes from draft-ietf-...-02 to -03

- o Updated the document to use "RTOR" instead of "RTO Restart" when refering to the modified algorithm.

- Moved document terminology to a section of its own.

- Introduced the rrthresh variable in the terminology section.

- Added a section to generalize the tracking of outstanding segments.

- Updated the algorithm to work when the number of outstanding segments is less than four and one segment is ready for transmission, by restarting the timer when new data has been sent.

- Clarified the relationship between fast retransmit and RTOR.

- Improved the wording throughout the document.

11.5. Changes from draft-ietf-...-01 to -02

- Changed the algorithm description in Section 3 to use formal RFC 2119 language.

- Changed last paragraph of Section 3 to clarify why the RTO restart algorithm is active when less than four segments are outstanding.

- Added two paragraphs in Section 4.1 to clarify why the algorithm can be turned on for all TCP traffic without having any negative effects on traffic patterns that do not benefit from a modified timer restart.

- Improved the wording throughout the document.

- Replaced and updated some references.

11.6. Changes from draft-ietf-...-00 to -01

- Improved the wording throughout the document.

- Removed the possibility for a connection limited by the receiver's advertised window to use RTO restart, decreasing the risk of spurious retransmission timeouts.

- Added a section that discusses the applicability of and problems related to the RTO restart mechanism.

- Updated the text describing the relationship to TLP to reflect updates made in this draft.

- Added acknowledgments.

12. References

12.1. Normative References

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.

[RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, April 2003.

[RFC3708] Blanton, E. and 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.

[RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, February 2005.

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

[RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP", RFC 5682, September 2009.

[RFC5827] Allman, M., Avrachenkov, K., Ayesta, U., Blanton, J., and P. Hurtig, "Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)", RFC 5827, May 2010.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

12.2. Informative References

[EL04] Ekstroem, H. and R. Ludwig, "The Peak-Hopper: A New End-to-End Retransmission Timer for Reliable Unicast Transport", IEEE INFOCOM 2004, March 2004.

[FDT13] Flach, T., Dukkipati, N., Terzis, A., Raghavan, B., Cardwell, N., Cheng, Y., Jain, A., Hao, S., Katz-Bassett, E., and R. Govindan, "Reducing Web Latency: the Virtue of Gentle Aggression", Proc. ACM SIGCOMM Conf., August 2013.

[HB11] Hurtig, P. and A. Brunstrom, "SCTP: designed for timely message delivery?", Springer Telecommunication Systems 47 (3-4), August 2011.

[IEEE.1003-1G.1997]
Institute of Electrical and Electronics Engineers, "Protocol Independent Interfaces", IEEE Standard 1003.1G, March 1997.

[LS00] Ludwig, R. and K. Sklower, "The Eifel retransmission timer", ACM SIGCOMM Comput. Commun. Rev., 30(3), July 2000.

[P09] Petlund, A., "Improving latency for interactive, thin-stream applications over reliable transport", Unipub PhD Thesis, Oct 2009.

[PBP09] Petlund, A., Beskow, P., Pedersen, J., Paaby, E., Griwodz, C., and P. Halvorsen, "Improving SCTP Retransmission Delays for Time-Dependent Thin Streams", Springer Multimedia Tools and Applications, 45(1-3), 2009.

[PGH06] Pedersen, J., Griwodz, C., and P. Halvorsen, "Considerations of SCTP Retransmission Delays for Thin Streams", IEEE LCN 2006, November 2006.

[RFC6458] Stewart, R., Tuexen, M., Poon, K., Lei, P., and V. Yasevich, "Sockets API Extensions for the Stream Control Transmission Protocol (SCTP)", RFC 6458, December 2011.

[RHB15] Rajiullah, M., Hurtig, P., Brunstrom, A., Petlund, A., and M. Welzl, "An Evaluation of Tail Loss Recovery Mechanisms for TCP", ACM SIGCOMM CCR 45 (1), January 2015.

[RJ10] Ramachandran, S., "Web metrics: Size and number of resources", Google http://code.google.com/speed/articles/web-metrics.html, May 2010.

[TLP] Dukkipati, N., Cardwell, N., Cheng, Y., and M. Mathis, "TCP Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses", Internet-draft draft-dukkipati-tcpm-tcp-loss-probe-01.txt, February 2013.

Authors' Addresses

Per Hurtig
Karlstad University
Universitetsgatan 2
Karlstad 651 88
Sweden

Phone: +46 54 700 23 35
Email: per.hurtig@kau.se

Anna Brunstrom
Karlstad University
Universitetsgatan 2
Karlstad 651 88
Sweden

Phone: +46 54 700 17 95
Email: anna.brunstrom@kau.se

Andreas Petlund
Simula Research Laboratory AS
P.O. Box 134
Lysaker 1325
Norway

Phone: +47 67 82 82 00
Email: apetlund@simula.no

Michael Welzl
University of Oslo
PO Box 1080 Blindern
Oslo N-0316
Norway

Phone: +47 22 85 24 20
Email: michawe@ifi.uio.no