Internet Engineering Task Force
INTERNET DRAFT
File: draft-ietf-tsvwg-tcp-frto-01.txt

P. Sarolahti
Nokia Research Center
M. Kojo
University of Helsinki

February, 2004
Expires: August, 2004

F-RTO: An Algorithm for Detecting Spurious Retransmission Timeouts with TCP and SCTP

Status of this Memo

Abstract

Spurious retransmission timeouts (RTOs) cause suboptimal TCP performance, because they often result in unnecessary retransmission of the last window of data. This document describes the "Forward RTO Recovery" (F-RTO) algorithm for detecting spurious TCP RTOs. F-RTO is a TCP sender only algorithm that does not require any TCP options to operate. After retransmitting the first unacknowledged segment triggered by an RTO, the F-RTO algorithm at a TCP sender monitors the incoming acknowledgements to determine whether the timeout was spurious and to decide whether to send new segments or retransmit unacknowledged segments. The algorithm effectively helps to avoid additional unnecessary retransmissions and thereby improves TCP performance in case of a spurious timeout. The F-RTO algorithm can

also be applied with the SCTP protocol.

Terminology

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119].

1. Introduction

The TCP protocol [Pos81] has two methods for triggering retransmissions. Primarily, the TCP sender relies on incoming duplicate ACKs, which indicate that the receiver is missing some of the data. After a required number of successive duplicate ACKs have arrived at the sender, it retransmits the first unacknowledged segment [APS99]. Secondarily, the TCP sender maintains a retransmission timer which triggers retransmission of segments, if they have not been acknowledged within the retransmission timer expiration period. When the retransmission timer expires, the TCP sender enters the RTO recovery where congestion window is initialized to one segment and unacknowledged segments are retransmitted using the slow-start algorithm. The retransmission timer is adjusted dynamically based on the measured round-trip times [PA00].

It has been pointed out that the retransmission timer can expire spuriously and trigger unnecessary retransmissions when no segments have been lost [LK00, GL02, LM03]. After a spurious retransmission timeout the late acknowledgements of original segments arrive at the sender, usually triggering unnecessary retransmissions of whole window of segments during the RTO recovery. Furthermore, after a spurious retransmission timeout a conventional TCP sender increases the congestion window on each late acknowledgement in slow start, injecting a large number of data segments to the network within one round-trip time.

There are a number of potential reasons for spurious retransmission timeouts. First, some mobile networking technologies involve sudden delay peaks on transmission because of actions taken during a hand-off. Second, arrival of competing traffic, possibly with higher priority, on a low-bandwidth link or some other change in available bandwidth involves a sudden increase of round-trip time which may trigger a spurious retransmission timeout. A persistently reliable link layer can also cause a sudden delay when several data frames are lost for some reason. This document does not distinguish the different causes of such a delay, but discusses the spurious

retransmission timeouts caused by a delay in general.

This document describes an alternative RTO recovery algorithm called "Forward RTO-Recovery" (F-RTO) to be used for detecting spurious RTOs and thus avoiding unnecessary retransmissions following the RTO. When the RTO is not spurious, the F-RTO algorithm reverts back to the conventional RTO recovery algorithm and should have similar behavior and performance. F-RTO does not require any TCP options in its operation, and it can be implemented by modifying only the TCP sender. This is different from alternative algorithms (Eifel [LK00], [LM03] and DSACK-based algorithms [BA02]) that have been suggested for detecting unnecessary retransmissions. The Eifel algorithm uses TCP timestamps [BBJ92] for detecting a spurious timeout upon arrival of the first acknowledgement after the retransmission. The DSACK-based algorithms require that the TCP Selective Acknowledgment Option [MMFR96] with DSACK extension [FMMP00] is in use. With DSACK, the TCP receiver can report if it has received a duplicate segment, making it possible for the sender to detect afterwards whether it has retransmitted segments unnecessarily. In addition, the F-RTO algorithm only attempts to detect and avoid unnecessary retransmissions after an RTO. Eifel and DSACK can also be used in detecting unnecessary retransmissions in other events, for example due to packet reordering.

When an RTO occurs, the F-RTO sender retransmits the first unacknowledged segment as usual. Deviating from the normal operation after a timeout, it then tries to transmit new, previously unsent data, for the first acknowledgement that arrives after the timeout given that the acknowledgement advances the window. If the second acknowledgement that arrives after the timeout also advances the window, i.e., acknowledges data that was not retransmitted, the F-RTO sender declares the RTO spurious and exit the RTO recovery. However, if either of the next two acknowledgements is a duplicate ACK, there was no sufficient evidence of spurious RTO; therefore the F-RTO sender retransmits the unacknowledged segments in slow start similarly to the traditional algorithm. With a SACK-enhanced version of the F-RTO algorithm, spurious RTOs may be detected even if duplicate ACKs arrive after an RTO.

The F-RTO algorithm can also be applied with the SCTP protocol [Ste00], because SCTP has similar acknowledgement and packet retransmission concepts as TCP. When a SCTP retransmission timeout occurs, the SCTP sender is required to retransmit the outstanding data similarly to TCP, thus being prone to unnecessary retransmissions and congestion control adjustments, if delay spikes occur in the network. The SACK-enhanced version of F-RTO should be directly applicable to SCTP, which has selective acknowledgements as a built-in feature. For simplicity, this document mostly refers to

TCP, but the algorithms and other discussion should be applicable also to SCTP.

This document is organized as follows. Section 2 describes the basic F-RTO algorithm. Section 3 outlines an optional enhancement to the F-RTO algorithm that takes leverage on the TCP SACK option. Section 4 discusses the possible actions to be taken after detecting a spurious RTO, and Section 5 discusses the security considerations.

2. F-RTO Algorithm

An RTO is spurious if there are segments outstanding in the network that would have prevented the RTO, had their acknowledgements arrived earlier at the sender. F-RTO affects the TCP sender behavior only after a retransmission timeout, otherwise the TCP behavior remains unmodified. When RTO expires the F-RTO algorithm monitors incoming acknowledgements and declares an RTO spurious, if the TCP sender gets an acknowledgement for a segment that was not retransmitted due to RTO. The actions taken in response to spurious RTO are not specified in this document, but we discuss the different alternatives for congestion control in Section 4.

Following the practice used with the Eifel Detection algorithm [LM03], we use the "SpuriousRecovery" variable to indicate whether the retransmission is declared spurious by the sender. This variable can be used as an input for a related response algorithm. With F-RTO, the outcome of SpuriousRecovery can either be SPUR_TO, indicating a spurious retransmission timeout; or FALSE, when the RTO is not declared spurious, and the TCP sender should follow the conventional RTO recovery algorithm.

A TCP sender MAY implement the basic F-RTO algorithm, and if it chooses to apply the algorithm, the following steps MUST be taken after the retransmission timer expires.

1) When RTO expires, the TCP sender SHOULD retransmit the first unacknowledged segment and set SpuriousRecovery to FALSE. Store the highest sequence number transmitted so far in variable "send_high".

2) When the first acknowledgement after the RTO arrives at the sender, the sender chooses the following actions depending on whether the ACK advances the window or whether it is a duplicate ACK.

   a) If the acknowledgement is a duplicate ACK OR it is acknowledging a sequence number equal to (or above) the value

of send_high OR it does not acknowledge all of the data that was retransmitted in step 1, the TCP sender MUST revert to the conventional RTO recovery and continue by retransmitting unacknowledged data in slow start. The TCP sender MUST NOT enter step 3 of this algorithm, and the SpuriousRecovery variable remains as FALSE.

b) If the acknowledgement advances the window AND it is below the value of send_high, the TCP sender SHOULD transmit up to two new (previously unsent) segments and enter step 3 of this algorithm. If the TCP sender does not have enough unsent data, it SHOULD send only one segment. In addition, the TCP sender MAY override the Nagle algorithm and send immediately an undersized segment if needed. If the TCP sender does not have any new data to send, the TCP sender SHOULD transmit a segment from the retransmission queue. If TCP sender retransmits the first unacknowledged segment, it MUST NOT enter step 3 of this algorithm but continue with the conventional RTO recovery algorithm. In this case acknowledgement of the next segment would not unambiguously indicate that the original transmission arrived at the receiver.

3) When the second acknowledgement after the RTO arrives at the sender, either declare the RTO spurious, or start retransmitting the unacknowledged segments.

a) If the acknowledgement is a duplicate ACK, the TCP sender MUST set congestion window to no more than 3 * MSS, and continue with the slow start algorithm retransmitting unacknowledged segments. The sender leaves SpuriousRecovery set to FALSE.

b) If the acknowledgement advances the window, i.e. it acknowledges data that was not retransmitted after the RTO, the TCP sender SHOULD declare the RTO spurious, set SpuriousRecovery to SPUR_TO and set the value of send_high variable to SND.UNA.

The F-RTO sender takes cautious actions when it receives duplicate acknowledgements after an RTO. Since duplicate ACKs may indicate that segments have been lost, reliably detecting a spurious RTO is difficult in the lack of additional information. Therefore the safest alternative is to follow the conventional TCP recovery in those cases.

If the first acknowledgement after RTO covers the send_high point at algorithm step (2a), there is not enough evidence that a non-retransmitted segment has arrived at the receiver after the RTO.

This is a common case when a fast retransmission is lost and it has been retransmitted again after an RTO, while the rest of the unacknowledged segments have successfully been delivered to the TCP receiver before the RTO. Therefore the RTO cannot be declared spurious in this case.

If the first acknowledgement after RTO does not acknowledge all of the data that was retransmitted in step 1, the TCP sender reverts to the conventional RTO recovery. Otherwise, a malicious receiver acknowledging partial segments could cause the sender to declare the RTO spurious in a case where data was lost.

The TCP sender is allowed to send two new segments in algorithm branch (2b), because the conventional TCP sender would transmit two segments when the first new ACK arrives after the RTO. If sending new data is not possible in algorithm branch (2b), or the receiver window limits the transmission, the TCP sender has to send something in order to prevent the TCP transfer from stalling. If no segments were sent, the pipe between sender and receiver may run out of segments, and no further acknowledgements arrive. If transmitting previously unsent data is not possible, the following options are available for the sender.

- Continue with the conventional RTO recovery algorithm and do not try to detect the spurious RTO. The disadvantage is that the sender may do unnecessary retransmissions due to possible spurious RTO. On the other hand, we believe that the benefits of detecting spurious RTO in an application limited or receiver limited situations are not very remarkable.

- Use additional information if available, e.g. TCP timestamps with the Eifel Detection algorithm, for detecting a spurious RTO. However, Eifel detection may yield different results from F-RTO when ACK losses and a RTO occur within the same round-trip time [SKR03].

- Retransmit data from the tail of the retransmission queue and continue with step 3 of the F-RTO algorithm. It is possible that the retransmission is unnecessarily made, hence this option is not encouraged, except for hosts that are known to operate in an environment that is highly likely to have spurious RTOs. On the other hand, with this method it is possible to avoid several unnecessary retransmissions due to spurious RTO by doing only one retransmission that may be unnecessary.

- Send a zero-sized segment below SND.UNA similar to TCP Keep-Alive probe and continue with step 3 of the F-RTO algorithm. Since the receiver replies with a duplicate ACK, the sender is able to detect

from the incoming acknowledgement whether the RTO was spurious. While this method does not send data unnecessarily, it delays the recovery by one round-trip time in cases where the RTO was not spurious, and therefore is not encouraged.

- In receiver-limited cases, send one octet of new data regardless of the advertised window limit, and continue with step 3 of the F-RTO algorithm. It is possible that the receiver has free buffer space to receive the data by the time the segment has propagated through the network, in which case no harm is done. If the receiver is not capable of receiving the segment, it rejects the segment and sends a duplicate ACK.

If the RTO is declared spurious, the TCP sender sets the value of send_high variable to SND.UNA in order to disable the NewReno "bugfix" [FH99]. The send_high variable was proposed for avoiding unnecessary multiple fast retransmits when RTO expires during fast recovery with NewReno TCP. As the sender has not retransmitted other segments but the one that triggered RTO, the problem addressed by the bugfix cannot occur. Therefore, if there are three duplicate ACKs arriving at the sender after the RTO, they are likely to indicate a packet loss, hence fast retransmit should be used to allow efficient recovery. If there are not enough duplicate ACKs arriving at the sender after a packet loss, the retransmission timer expires another time and the sender enters step 1 of this algorithm.

When the RTO is declared spurious, the TCP sender cannot detect whether the unnecessary RTO retransmission was lost. In principle the loss of the RTO retransmission should be taken as a congestion signal, and thus there is a small possibility that the F-RTO sender violates the congestion control rules, if it chooses to fully revert congestion control parameters after detecting a spurious RTO. The Eifel detection algorithm has a similar property, while the DSACK option can be used to detect whether the retransmitted segment was successfully delivered to the receiver.

The F-RTO algorithm has a side-effect on the TCP round-trip time measurement. Because the TCP sender can avoid most of the unnecessary retransmissions after detecting a spurious RTO, the sender is able to take round-trip time samples on the delayed segments. If the regular RTO recovery was used without TCP timestamps, this would not be possible due to retransmission ambiguity. As a result, the RTO estimator is likely to be more accurate and have larger values with F-RTO than with the regular TCP after a spurious RTO that was triggered due to delayed segments. We believe this is an advantage in the networks that are prone to delay spikes.

It is possible that the F-RTO algorithm does not always avoid

unnecessary retransmissions after a spurious RTO. If packet reordering or packet duplication occurs on the segment that triggered the spurious RTO, the F-RTO algorithm may not detect the spurious RTO due to incoming duplicate ACKs. Additionally, if a spurious RTO occurs during fast recovery, the F-RTO algorithm often cannot detect the spurious RTO. However, we consider these cases relatively rare, and note that in cases where F-RTO fails to detect the spurious RTO, it performs similarly to the regular RTO recovery.

3. A SACK-enhanced version of the F-RTO algorithm

This section describes an alternative version of the F-RTO algorithm, that makes use of TCP Selective Acknowledgement Option [MMFR96]. By using the SACK option the TCP sender can detect spurious RTOs in most of the cases when packet reordering or packet duplication is present. The difference to the basic F-RTO algorithm is that the sender may declare RTO spurious even when duplicate ACKs follow the RTO, if the SACK blocks acknowledge new data that was not transmitted after RTO. The algorithm principle presented in this section is also applicable to be used with the SCTP protocol.

Given that the TCP Selective Acknowledgement Option [MMFR96] is enabled for a TCP connection, TCP sender MAY implement the SACK-enhanced F-RTO algorithm. If the sender applies the SACK-enhanced F-RTO algorithm, it MUST follow the steps below. This algorithm SHOULD NOT be applied, if the TCP sender is already in loss recovery when RTO occurs. However, it should be possible to apply the principle of F-RTO within certain limitations also when RTO occurs during existing loss recovery. While this is a topic of further research, Appendix B briefly discusses the related issues.

1) When RTO expires, the TCP sender SHOULD retransmit first unacknowledged segment and set SpuriousRecovery to FALSE. Variable "send_high" is set to indicate the highest segment transmitted so far.

2) Wait until the acknowledgement for the segment retransmitted due to RTO arrives at the sender. If duplicate ACKs arrive, store the incoming SACK information but stay in step 2. If RTO expires, restart the algorithm.

   a) if the cumulative ACK acknowledges all segments up to send_high, the TCP sender SHOULD revert to the conventional RTO recovery and it MUST set congestion window to no more than 2 * MSS. The sender does not enter step 3 of this algorithm.

   b) otherwise, the TCP sender SHOULD transmit up to two new

(previously unsent) segments, within the limitations of the congestion window. If the TCP sender is not able to transmit any previously unsent data due to receiver window limitation or because it does not have any new data to send, it MAY follow one of the options presented in Section 2. However, if the TCP sender chooses to retransmit a data segment here, SACK of that segment MUST NOT be used for declaring a spurious RTO in step (3b).

3) When the next acknowledgement arrives at the sender.

a) if the ACK acknowledges data above send_high, either in SACK blocks or as a cumulative ACK, the sender MUST set congestion window to no more than 3 * MSS and proceed with conventional recovery, retransmitting unacknowledged segments. The sender SHOULD take this branch also when the acknowledgement is a duplicate ACK and it does not contain any new SACK blocks for previously unacknowledged data below send_high.

b) if the ACK does not acknowledge data above send_high AND it acknowledges some previously unacknowledged data below send_high, the TCP sender SHOULD declare the RTO spurious and set SpuriousRecovery to SPUR_TO.

If there are unacknowledged holes between the received SACK blocks, those segments SHOULD be retransmitted similarly to the conventional SACK recovery algorithm [BAFW03]. If the algorithm exits with SpuriousRecovery set to SPUR_TO, send_high SHOULD be set to SND.UNA, thus allowing fast recovery on incoming duplicate acknowledgements.

4. Taking Actions after Detecting Spurious RTO

Upon retransmission timeout, a conventional TCP sender assumes that outstanding segments are lost and starts retransmitting the unacknowledged segments. When the RTO is detected to be spurious, the TCP sender should not continue retransmitting based on the RTO. For example, if the sender was in congestion avoidance phase transmitting new previously unsent segments, it should continue transmitting previously unsent segments after detecting spurious RTO. In addition, it is suggested that the RTO estimation is reinitialized and the RTO timer is adjusted to a more conservative value in order to avoid subsequent spurious RTOs [LG03].

Different approaches have been discussed for adjusting the congestion control state after a spurious RTO in various research papers [SKR03, GL03, Sar03] and Internet-Drafts [SL03, LG03]. The different response suggestions vary in whether the spurious retransmission timeout

should be taken as a congestion signal, thus causing the congestion window or slow start threshold to be reduced at the sender, or whether the congestion control state should be fully reverted to the state valid prior to the retransmission timeout.

This document does not give recommendation on selecting the response alternative, but considers the response to spurious RTO as a subject of further research.

5. SCTP Considerations

The basic F-RTO or the SACK-enhanced F-RTO algorithm can be applied with the SCTP protocol. However, SCTP contains features that are not present with TCP that need to be discussed when applying the F-RTO algorithm.

SCTP association can be multi-homed. The current retransmission policy states that retransmissions should go to alternative addresses. If the retransmission was due to spurious RTO caused by a delay spike, it is possible that the acknowledgement for the retransmission arrives back at the sender before the acknowledgements of the original transmissions arrive. If this happens, a possible loss of the original transmission of the data chunk that was retransmitted due to the spurious RTO may remain undetected when applying the F-RTO algorithm. Because the RTO was caused by the delay, and it was spurious in that respect, a suitable response is to continue by sending new data. However, if the original transmission was lost, fully reverting the congestion control parameters is too aggressive. Therefore, taking conservative actions on congestion control is recommended, if the SCTP association is multi-homed and retransmissions go to alternative address. The information in duplicate TSNs can be then used for reverting congestion control, if desired [BA02].

Note that the forward transmissions made in F-RTO algorithm step (2b) should be destined to the primary address, since they are not retransmissions.

When making a retransmission, a SCTP sender can bundle a number of unacknowledged data chunks and include them in the same packet. This needs to be considered when implementing F-RTO for SCTP. The basic principle of F-RTO still holds: in order to declare the RTO spurious, the sender must get an acknowledgement for a data chunk that was not retransmitted after the RTO. In other words, acknowledgements of data chunks that were bundled in RTO retransmission must not be used for declaring the RTO spurious.

6. Security Considerations

The main security threat regarding F-RTO is the possibility of a receiver misleading the sender to set too large a congestion window after an RTO. There are two possible ways a malicious receiver could trigger a wrong output from the F-RTO algorithm. First, the receiver can acknowledge data that it has not received. Second, it can delay acknowledgement of a segment it has received earlier, and acknowledge the segment after the TCP sender has been deluded to enter algorithm step 3.

If the receiver acknowledges a segment it has not really received, the sender can be lead to declare RTO spurious in F-RTO algorithm step 3. However, since this causes the sender to have incorrect state, it cannot retransmit the segment that has never reached the receiver. Therefore, this attack is unlikely to be useful for the receiver to maliciously gain a larger congestion window.

A common case of an RTO is that a fast retransmission of a segment is lost. If all other segments have been received, the RTO retransmission causes the whole window to be acknowledged at once. This case is recognized in F-RTO algorithm branch (2a). However, if the receiver only acknowledges one segment after receiving the RTO retransmission, and then the rest of the segments, it could cause the RTO to be declared spurious when it is not. Therefore, it is suggested that when an RTO expires during fast recovery phase, the sender would not fully revert the congestion window even if the RTO was declared spurious, but reduce the congestion window to 1. However, the sender can take actions to avoid unnecessary retransmissions normally. If a TCP sender implements a burst avoidance algorithm that limits the sending rate to be no higher than in slow start, this precaution is not needed, and the sender may apply F-RTO normally.

If there are more than one segments missing at the time when an RTO occurs, the receiver does not benefit from misleading the sender to declare a spurious RTO, because the sender would then have to go through another recovery period to retransmit the missing segments, usually after an RTO.

Acknowledgements

We are grateful to Reiner Ludwig, Andrei Gurtov, Josh Blanton, Mark Allman, Sally Floyd, Yogesh Swami, Mika Liljeberg, Ivan Arias Rodriguez, Sourabh Ladha, and Martin Duke for the discussion and feedback contributed to this text.

Normative References

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

[BAFW03] E. Blanton, M. Allman, K. Fall, and L. Wang. A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP. RFC 3517. April 2003.

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. RFC 2018, October 1996.

[PA00] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. RFC 2988, November 2000.

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

[Ste00] R. Stewart, et. al. Stream Control Transmission Protocol. RFC 2960, October 2000.

Informative References

[ABF01] M. Allman, H. Balakrishnan, and S. Floyd. Enhancing TCP's Loss Recovery Using Limited Transmit. RFC 3042, January 2001.

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

[BBJ92] D. Borman, R. Braden, and V. Jacobson. TCP Extensions for High Performance. RFC 1323, May 1992.

[FH99] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582, April 1999.

[FMMP00] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An Extension to the Selective Acknowledgement (SACK) Option to TCP. RFC 2883, July 2000.

[GL02] A. Gurtov and R. Ludwig. Evaluating the Eifel Algorithm for TCP in a GPRS Network. In Proc. of European Wireless, Florence, Italy, February 2002

[GL03] A. Gurtov and R. Ludwig, Responding to Spurious Timeouts in TCP, In Proceedings of IEEE INFOCOM 03, March 2003.

[LG03] R. Ludwig and A. Gurtov. The Eifel Response Algorithm for TCP. Internet draft "draft-ietf-tsvwg-tcp-eifel-response-04.txt". October 2003. Work in progress.

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

[LM03] R. Ludwig and M. Meyer. The Eifel Detection Algorithm for TCP. RFC 3522, April 2003.

[SKR03] P. Sarolahti, M. Kojo, and K. Raatikainen. F-RTO: An Enhanced Recovery Algorithm for TCP Retransmission Timeouts. ACM SIGCOMM Computer Communication Review, 33(2), April 2003.

[Sar03] P. Sarolahti. Congestion Control on Spurious TCP Retransmission Timeouts. In Proceedings of IEEE Globecom 2003. December 2003.

[SL03] Y. Swami and K. Le. DCLOR: De-correlated Loss Recovery using SACK option for spurious timeouts. Internet draft "draft-swami-tsvwg-tcp-dclor-02.txt". September 2003. Work in progress.

Appendix A: Scenarios

This section discusses different scenarios where RTOs occur and how the basic F-RTO algorithm performs in those scenarios. The interesting scenarios are a sudden delay triggering RTO, loss of a retransmitted packet during fast recovery, link outage causing the loss of several packets, and packet reordering. A performance evaluation with a more thorough analysis on a real implementation of F-RTO is given in [SKR03].

A.1. Sudden delay

The main motivation of F-RTO algorithm is to improve TCP performance when a delay spike triggers a spurious retransmission timeout. The example below illustrates the segments and acknowledgements transmitted by the TCP end hosts when a spurious RTO occurs, but no packets are lost. For simplicity, delayed acknowledgements are not used in the example. The example below reduces the congestion window and slow start threshold by half after detecting a spurious RTO.

```
      ...
       (cwnd = 6,
```

```
 ssthresh < 6,
 FlightSize = 5)
1.  SEND 10 ---------------------------->
2.          <----------------------------- ACK 6
3.  SEND 11 ---------------------------->
4.                      |
                    [delay]
                        |
    [RTO]
5.  SEND 6  ---------------------------->
            <earlier xmitted SEG 6>  --->
6.          <----------------------------- ACK 7
    [F-RTO step (2b)]
7.  SEND 12 ---------------------------->
8.  SEND 13 ---------------------------->
            <earlier xmitted SEG 7>  --->
9.          <----------------------------- ACK 8
    [F-RTO step (3b)]
    [SpuriousRecovery <- SPUR_TO]
    [cwnd <- 3, ssthresh <- 3]
10.         <----------------------------- ACK 9
11.         <----------------------------- ACK 10
12.         <----------------------------- ACK 11
13. SEND 14 ---------------------------->
...
```

When a sudden delay long enough to trigger RTO occurs at step 4, the TCP sender retransmits the first unacknowledged segment (step 5). Because the next ACK covers the RTO retransmission because originally transmitted segment 6 arrives at the receiver, the TCP sender continues by sending two new data segments (steps 7, 8). Because the second acknowledgement arriving after the RTO acknowledges data that was not retransmitted due to RTO (step 9), the TCP sender declares the RTO as spurious and continues by sending new data. Because the TCP sender reduces cwnd when it detects the spurious RTO, it has to wait for some outstanding segments to leave the network before it can continue transmitting again at step 13.

A.2. Loss of a retransmission

If a retransmitted segment is lost, the only way to retransmit it again is to wait for the RTO to trigger the retransmission. Once the segment is successfully received, the receiver usually acknowledges several segments at once, because other segments in the same window have been successfully delivered before the retransmission arrives at the receiver. The example below shows a scenario where retransmission (of segment 6) is lost, as well as a later segment (segment 9) in the same window. The limited transmit [ABF01] or SACK TCP [MMFR96]

enhancements are not in use in this example.

```
      ...
       (cwnd = 6,
       ssthresh < 6,
       FlightSize = 5)
          <segment 6 lost>
          <segment 9 lost>
      1.  SEND 10 ---------------------------->
      2.          <----------------------------- ACK 6
      3.  SEND 11 ---------------------------->
      4.          <----------------------------- ACK 6
      5.          <----------------------------- ACK 6
      6.          <----------------------------- ACK 6
      7.  SEND 6  --------------X
          <segment 6 lost>
          [ssthresh <- 3, cwnd <- ssthresh + 3 = 6]
      8.          <----------------------------- ACK 6
                              |
                              |
          [RTO]
          [ssthresh <- 2]
      9.  SEND 6  ---------------------------->
      10.         <----------------------------- ACK 9
          [F-RTO step (2b)]
      11. SEND 12 ---------------------------->
      12. SEND 13 ---------------------------->
      13.         <----------------------------- ACK 9
          [F-RTO step (3a)]
          [SpuriousRecovery <- FALSE]
          [cwnd <- 3]
      14. SEND 9  ---------------------------->
      15. SEND 10 ---------------------------->
      16. SEND 11 ---------------------------->
      17.         <----------------------------- ACK 11
      ...
```

In the example above, segment 6 is lost and the sender retransmits it after three duplicate ACKs in step 7. However, the retransmission is also lost, and the sender has to wait for the RTO to expire before retransmitting it again. Because the first ACK following the RTO acknowledges the RTO retransmission (step 10), the sender transmits two new segments. The second ACK in step 13 does not acknowledge any previously unacknowledged data. Therefore the F-RTO sender enters the slow start and sets cwnd to 3 * MSS. Congestion window can be set to three segments, because two round-trips have elapsed after the RTO. After this the receiver acknowledges all segments transmitted prior to entering recovery and the sender can continue transmitting new

data in congestion avoidance.

A.3. Link outage

The example below illustrates the F-RTO behavior when 4 consecutive packets are lost in the network causing the TCP sender to fall back to RTO recovery. Limited transmit and SACK are not used in this example.

```
      ...
       (cwnd = 6,
       ssthresh < 6,
       FlightSize = 5)
          <segments 6-9 lost>
      1.  SEND 10 ---------------------------->
      2.          <---------------------------- ACK 6
      3.  SEND 11 ---------------------------->
      4.          <---------------------------- ACK 6
                               |
                               |
          [RTO]
          [ssthresh <- 3]
      5.  SEND 6  ---------------------------->
      6.          <---------------------------- ACK 7
          [F-RTO step (2b)]
      7.  SEND 12 ---------------------------->
      8.  SEND 13 ---------------------------->
      9.          <---------------------------- ACK 7
          [F-RTO step (3a)]
          [SpuriousRecovery <- FALSE]
          [cwnd <- 3]
      10. SEND 7  ---------------------------->
      11. SEND 8  ---------------------------->
      12. SEND 9  ---------------------------->
      13.         <---------------------------- ACK 14
```

Again, F-RTO sender transmits two new segments (steps 7 and 8) after the RTO retransmission is acknowledged. Because the next ACK does not acknowledge any data that was not retransmitted after the RTO (step 9), the F-RTO sender proceeds with conventional recovery and slow start retransmissions.

A.4. Packet reordering

Since F-RTO modifies the TCP sender behavior only after a retransmission timeout and it is intended to avoid unnecessary retransmits only after spurious RTO, we limit the discussion on the

effects of packet reordering in F-RTO behavior to the cases where packet reordering occurs immediately after the RTO. When the TCP receiver gets an out-of-order segment, it generates a duplicate ACK. If the TCP sender implements the basic F-RTO algorithm, this may prevent the sender from detecting a spurious RTO.

However, if the TCP sender applies the SACK-enhanced F-RTO, it is possible to detect a spurious RTO also when packet reordering occurs. We illustrate the behavior of SACK-enhanced F-RTO below when segment 8 arrives before segments 6 and 7, and segments starting from segment 6 are delayed in the network. In this example the TCP sender reduces the congestion window and slow start threshold in response to spurious RTO.

```
      ...
       (cwnd = 6,
       ssthresh < 6,
       FlightSize = 5)
      1.  SEND 10 ---------------------------->
      2.          <---------------------------- ACK 6
      3.  SEND 11 ---------------------------->
      4.                      |
                          [delay]
                              |
          [RTO]
      5.  SEND 6  ---------------------------->
                  <earlier xmitted SEG 8>  --->
      6.          <---------------------------- ACK 6
                                                 [SACK 8]
          [SACK F-RTO stays in step 2]
      7.          <earlier xmitted SEG 6>  --->
      8.          <---------------------------- ACK 7
                                                 [SACK 8]
          [SACK F-RTO step (2b)]
      9.  SEND 12 ---------------------------->
      10. SEND 13 ---------------------------->
      11.         <earlier xmitted SEG 7>  --->
      12.         <---------------------------- ACK 9
          [SACK F-RTO step (3b)]
          [SpuriousRecovery <- SPUR_TO]
          [ssthresh <- 3, cwnd <- 3]
      13.         <---------------------------- ACK 10
      14.         <---------------------------- ACK 11
      15. SEND 14 ---------------------------->
      ...
```

After RTO expires and the sender retransmits segment 6 (step 5), the receiver gets segment 8 and generates duplicate ACK with SACK for

segment 8. In response to the acknowledgement the TCP sender does not send anything but stays in F-RTO step 2. Because the next acknowledgement advances the cumulative ACK point (step 8), the sender can transmit two new segments according to SACK-enhanced F-RTO. The next segment acknowledges new data between 7 and 11 that was not acknowledged earlier (segment 7), so the F-RTO sender declares the RTO spurious.

Appendix B: Applying SACK-enhanced F-RTO when RTO occurs during loss recovery

We believe that slightly modified SACK-enhanced F-RTO algorithm can be used to detect spurious RTOs also when RTO occurs while an earlier loss recovery is underway. However, there are issues that need to be considered if F-RTO is applied in this case.

The original SACK-based F-RTO requires in algorithm step 3 that an ACK acknowledges previously unacknowledged non-retransmitted data between SND.UNA and send_high. If RTO takes place during earlier (SACK-based) loss recovery, the F-RTO sender must only use acknowledgements for non-retransmitted segments transmitted before the SACK-based loss recovery started. This means that in order to declare RTO spurious the TCP sender must receive an acknowledgement for non-retransmitted segment between SND.UNA and RecoveryPoint in algorithm step 3. RecoveryPoint is defined in conservative SACK-recovery algorithm [BAFW03], and it is set to indicate the highest segment transmitted so far when SACK-based loss recovery begins. In other words, if the TCP sender receives acknowledgement for segment that was transmitted more than one RTO ago, it can declare the RTO spurious. Defining an efficient algorithm for checking these conditions remains as a future work item.

When spurious RTO is detected according to the rules given above, it may be possible that the response algorithm needs to consider this case separately, for example in terms of what segments to retransmit after RTO, and whether it is safe to revert the congestion control parameters in this case. This is considered as a topic of future research.

Authors' Addresses

Pasi Sarolahti
Nokia Research Center
P.O. Box 407
FIN-00045 NOKIA GROUP
Finland

Phone: +358 50 4876607
EMail: pasi.sarolahti@nokia.com
http://www.cs.helsinki.fi/u/sarolaht/

Markku Kojo
University of Helsinki
Department of Computer Science
P.O. Box 26
FIN-00014 UNIVERSITY OF HELSINKI
Finland

Phone: +358 9 1914 4179
EMail: markku.kojo@cs.helsinki.fi