# An Evaluation of Tail Loss Recovery Mechanisms for TCP

Mohammad Rajiullah[†], Per Hurtig[†], Anna Brunstrom[†], Andreas Petlund[‡], Michael Welzl[*]
[†]Karlstad University, [‡]Simula Research Laboratory, [*]Oslo University
{moharaji, perhurt, annabrun}@kau.se, apetlund@simula.no, michawe@ifi.uio.no

## ABSTRACT

Interactive applications do not require more bandwidth to go faster. Instead, they require less latency. Unfortunately, the current design of transport protocols such as TCP limits possible latency reductions. In this paper we evaluate and compare different loss recovery enhancements to fight tail loss latency. The two recently proposed mechanisms "RTO Restart" (RTOR) and "Tail Loss Probe" (TLP) as well as a new mechanism that applies the logic of RTOR to the TLP timer management (TLPR) are considered. The results show that the relative performance of RTOR and TLP when tail loss occurs is scenario dependent, but with TLP having potentially larger gains. The TLPR mechanism reaps the benefits of both approaches and in most scenarios it shows the best performance.

## Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*TCP*; C.2.6 [**Computer-Communication Networks**]: Internetworking—*Standards*

## Keywords

TCP; Congestion Control; Latency; Packet Loss; Recovery

## 1. INTRODUCTION

As Internet-based applications are becoming increasingly interactive, including e.g. web applications and real-time communication, low latency has become important. For example, Amazon and Shopzilla have reported significant revenue increases when decreasing latency. Amazon increased their revenues by 1% for every 100 ms reduction and Shopzilla had a 12% increase when they reduced latency from 6 to 1.2 seconds [8].

While many sources contribute to latency, the design of transport protocols like TCP [19, 4] is one important component. For instance, TCP's loss recovery mechanisms can add several round-trip times (RTTs) to the transmission time of a TCP segment. Recent large-scale measurements have found that nearly 10% of all connections incur at least one segment loss, causing an average flow to complete five times slower than those without any loss [11]. Furthermore, the authors of [11] also find that 77% of these losses are recovered through expensive retransmission timeouts (RTOs), often because segments at the tail of a burst were lost with no more new data available for transmission, preventing fast recovery.

In this paper we perform a detailed evaluation of two recently proposed loss recovery enhancements to fight tail-loss latency, RTO Restart (RTOR) [14] and Tail Loss Probe (TLP) [9, 11], as well as a new combined mechanism that applies the logic of RTOR to the TLP timer management (TLPR). Our results show that RTOR, TLP and TLPR are all able to reduce the time needed for tail loss recovery. The reduction for RTOR is typically one RTT, but can be larger if delayed acknowledgements are used. The performance of TLP depends on several factors, but the gain of TLP can be considerably larger than that of RTOR in some situations. TLPR draws the benefits of both approaches and shows the best performance in most scenarios.

The details of RTOR, TLP and TLPR are described in section (§2). §3 presents our evaluation. The gains of the mechanisms are evaluated under various network conditions, including cross-traffic scenarios and web page downloads. §4 discusses related work, and §5 concludes.

## 2. TCP LOSS RECOVERY

TCP uses two methods to recover from segment loss. If an acknowledgment (ACK) for a given TCP data segment is not received within an RTO, the sender retransmits that segment. The RTO is based on periodic RTT measurements between the sender and the receiver [17]. The RTO is conservatively calculated to not unnecessarily retransmit data that is merely delayed. This avoids possible performance degradations that can follow from unnecessarily reducing the congestion window in response to a spurious retransmission, but can sometimes be a serious problem for applications that require low latency.

To enable faster loss recovery, TCP provides a fast retransmit (FR) mechanism [4] which relies on duplicate acknowledgements (dupACKs) from the receiver. Such dupACKs are sent when an out-of-order segment arrives. To avoid mistaking mild reordering as loss, FR is only triggered after three dupACKs. However, when loss occurs and there is no new data available for transmission, the receiver cannot generate the three dupACKs needed. This may happen at the end of TCP flows or at the end of segment bursts within a connection. To alleviate this problem the early retransmit (ER) [5] algorithm dynamically adapts the number of dupACKs needed for loss detection based on the amount of outstanding data.

One situation where ER is of no help is when the last segment(s) of a flow is lost. This phenomenon is commonly referred to as tail loss and is a significant problem for data-limited flows such as short and/or bursty data transfers [11].

To enable faster loss recovery of tail loss, RTOR and TLP are now considered for standardization and deployment.

## 2.1 RTO Restart (RTOR)

To improve recovery for tail loss, RTOR changes the RTO timer management. The current algorithm [17] restarts the timer on each incoming ACK, causing retransmissions to occur RTO seconds after the last ACK has been received, not RTO seconds after the transmission of the lost segment(s). In most cases, this adds approximately one RTT to the loss recovery time. If the ACK that triggers the restart is also delayed, then the total loss recovery time can become RTO + RTT + delACK, where delACK corresponds to the receiver's setting for delaying ACKs.

By resetting the timer to RTO $-$ $\mathrm{T_{earliest}}$, where $\mathrm{T_{earliest}}$ is the time elapsed since the earliest outstanding segment was transmitted, RTOR ensures that retransmissions will always occur after exactly RTO seconds, thus reducing the retransmission delay by an RTT in the typical case. To promote the use of FR when possible, RTOR is by default only active when the amount of outstanding segments is less than four, i.e., there is too little data outstanding to trigger FR.

## 2.2 Tail Loss Probe (TLP)

TLP tries to trigger loss recovery by e.g. ER by inducing additional (selective) ACKs (SACKs) at the receiver, even if the last segment is lost. The current version of TLP [9] transmits one probe segment after a probe timeout (PTO) if the connection has outstanding data but is otherwise idle. Equation 1 details the calculation of the PTO whenever a connection has more than one outstanding segment and Equation 2 for the case of one outstanding segment. In the below equations, SRTT is the smoothed RTT and WCDelAckT is the worst case delACK timer, recommended to be set to 200 ms in [9].

$$PTO = \max(2 * SRTT, 10ms) \quad (1)$$

$$PTO = \max(2 * SRTT, 1.5 * SRTT + WCDelAckT) \quad (2)$$

A check is also made against the RTO, and the PTO is set to the minimum of the calculated PTO and the RTO. The transmitted probe can be either a new segment, if available and the receive window permits, or a retransmission of the most recently transmitted segment. When there is tail loss the algorithm has two clear benefits: (i) the probe repairs a loss of the last segment; (ii) previously lost segments can be recovered using the probe ACK, either by relying on ER or by forward acknowledgements (FACK) [16]. If the probe is sent in the absence of loss, there is no change in the congestion control or loss recovery state of the connection. See [9, 11] for further details on TLP. In relation to [9], the newest implementation in Linux corrects a small flaw in TLP that inhibited the last segment of a flow from being retransmitted if it carried a FIN flag but no data [13].

## 2.3 TLP with Restart (TLPR)

In addition to RTOR and TLP we evaluate a new combined mechanism, TLPR, that applies the logic of RTOR to the timer management of TLP. Similar to the standard RTO timer management, the TLP algorithm restarts the PTO timer on each incoming ACK. This causes probe segments to be sent PTO seconds after the last ACK has been received, not PTO seconds after the transmission of the last segment. The TLPR algorithm instead sets the probe timer to PTO $-$ $\mathrm{T_{last}}$, where $\mathrm{T_{last}}$ is the time elapsed since the last outstanding segment was transmitted.

# 3. EVALUATION

## 3.1 Controlled Tail Loss

We start the evaluation in this section by examining RTOR, TLP, and TLPR in controlled tail loss scenarios.

*Experimental Setup.* To compare the loss recovery mechanisms, we performed a series of experiments where a sender and a receiver communicated flows of 10 segments (one full initial congestion window) over an emulated network. The end hosts were equipped with Linux 3.17 and KauNet [12] was used to emulate the network between them. KauNet is capable of exactly positioning segment loss in TCP flows and was therefore used to introduce tail loss. It was also used to apply different RTTs (10-640 ms). To compare the different loss recovery mechanisms we measured the elapsed time from connection establishment to the successful reception of the last data segment, i.e., the Flow Completion Time (FCT). To account for different receiver behaviors, we also modified the TCP implementation to support delACKs (with a timeout of 200 ms) in addition to quickACKs where each segment triggers an ACK during slow-start (which is default in Linux). We first analyse the results for a single tail loss and then expand the scope to include $n$-degree tail loss scenarios. Due to the high level of control of the experiments and the simple traffic patterns, no variation was observed in the results and there were no spurious retransmissions.

*1-Degree Tail Loss Recovery.* A representative sample of the results for a single tail loss is shown in Figure 1, grouped by RTT. For each RTT there are four bars. The bars show the normalized FCT for the different loss recovery mechanisms (baseline, RTOR, TLP, TLPR) when a single tail loss occurs. The baseline bar, which all other bars are normalized against, shows the standard Linux loss recovery including ER, but without TLP. The top graph shows the results for a receiver using delACKs and the bottom for a receiver with Linux quickACKs.

As shown in the graph, RTOR consistently reduces the FCT. When quickACKs are used, RTOR reduces the FCT with one RTT. When delACKs are used, the reduction also includes the delACK timer of the receiver. For small RTTs, shown in the upper graph at the left-hand side, the delACK timer has a larger impact on the FCT.

TLP is a much more complex mechanism, and the gain of using it depends on several factors. TLP's PTO is calculated differently depending on the number of outstanding segments, as viewed by the sender. If one segment is outstanding Equation 2 is used, otherwise Equation 1 is used. The relation between the PTO values depends on the RTT, with Equation 1 producing much smaller PTOs for small RTTs. When delACKs are used, the ACK for the last segment will be delayed by the delACK timer. If this delay combined with the RTT exceeds a previously calculated PTO (when more than one segment was outstanding), the timer will expire according to Equation 1. This will happen although only one segment is in fact lost. The con-

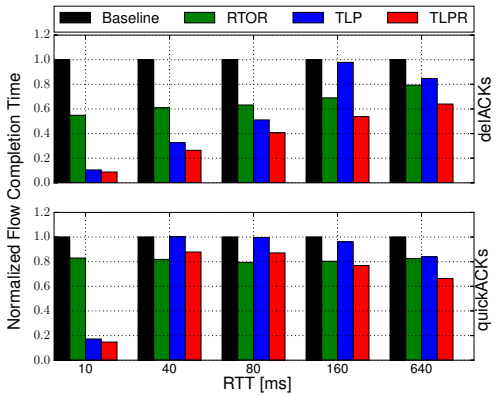



Figure 1: Normalized FCT for 10-segment-flows experiencing a loss of its last data segment.

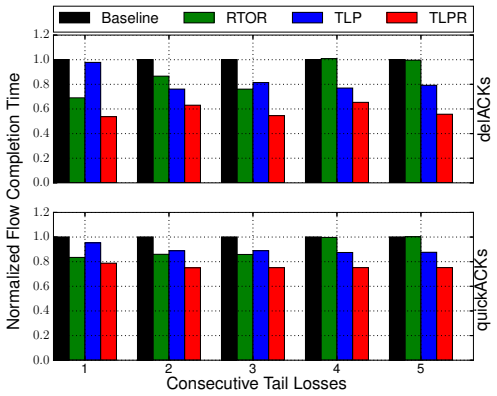



Figure 2: Normalized FCT for 10-segment-flows experiencing a loss of its $n$ last data segment(s). RTT is 80 ms.

sequence of this can be seen in the top graph, where the use of delACKs and small RTTs (10-80 ms) result in large FCT gains. When the RTT gets larger (160-640 ms), the delACK will return before the PTO fires, and Equation 2 will be used to recalculate the PTO. This is also visible for the quickACK strategy where each segment is ACKed and therefore Equation 2 is used. The exception is in the 10 ms case, where the ACK for the ninth segment does not reach the sender before the PTO fires. The gains of using RTOR and TLP are, as indicated above, different for different scenarios. Therefore the combination of them, TLPR, can in most situations reap the benefits of both components and provide the, overall, largest reductions in FCT.

*N-degree Tail Loss Recovery.* Figure 2 shows normalized FCTs for transfers having a 80 ms RTT and losing 1-5 consecutive segments from the tail. The top graph shows results using delACKs, and the bottom graph shows results with quickACKs. In general, all the mechanisms are able to reduce the FCT when the degree of tail loss increases, with the exception of RTOR when the degree of loss is larger than three. This limitation is due to the default configuration that disables RTOR when there are more than three outstanding segments. TLP and TLPR, on the other hand, try to enable fast recovery as a response to any degree of tail loss. To do this, TLP and TLPR employ FACK if four or more segments are lost in the tail and ER if 2-3 segments are lost.

## 3.2 Realistic Tail Loss

In the following experiment, we evaluate RTOR, TLP and TLPR in a network with realistic background traffic, where losses occur due to congestion.

*Experimental Setup.* We used a five node dumbbell setup where the delay, bandwidth, and queue size limitations were enforced by dummynet [7]. Two of the hosts were used to produce background traffic, using the Tmix traffic generator [22]. Tmix generates synthetic traffic based on a source level characterization of TCP connections present in an actual network trace. We used a 10 minutes long Tmix input, derived from a trace collected from a real multi-gigabit Ethernet link [2]. The background traffic was the same in all experiments, and with the bandwidth and queue size configured at the bottleneck, it resulted in a bottleneck loss rate of about 1.5% and queueing delays of up to 35 ms. The foreground traffic used to evaluate the different loss recovery mechanisms consisted of flows of varying size (4, 10, or 100 segments). In the experiments, for 4 and 10 segment flow scenarios, over 95% of the experienced loss episodes[1] were tail loss, whereas, for the 100 segment flow scenario, at most 10% of the experienced loss episodes were tail loss. In addition to the queueing delay, different baseline one-way delays (owd) were configured for the foreground traffic. The owds were set to 5, 40, and 160 ms. The hosts used for foreground traffic were configured with Linux 3.17.

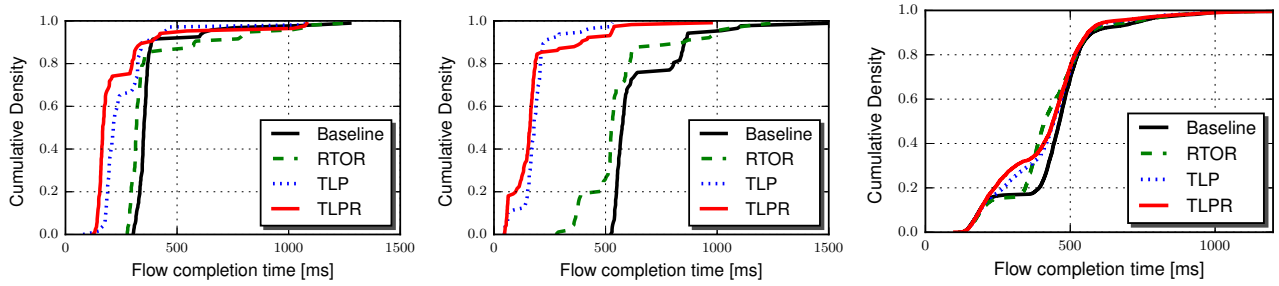
In the results we only present FCTs for flows that have at least one loss in the four last segments. The reason for this is the low loss rate at the bottleneck that obscures the effect of using different loss recovery mechanisms when loss actually does happen.

*Results.* Figure 3 shows a subset of the results. The four figures on the left show the results for four segment flows with different owds, using receivers employing quickACKs and delACKs, respectively. In all cases, RTOR saves at least one RTT as compared to the baseline. An extra saving of 200 ms is achieved when delACKs are used.

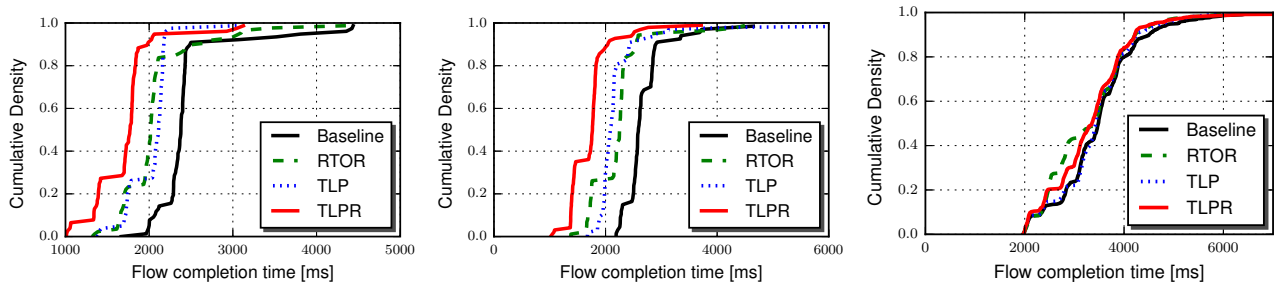
As discussed in subsection 3.1, when the RTT is small and multiple segments are lost in the tail, both TLP and TLPR recover lost segments quickly based on Equation 1. This is clearly visible in Figure 3a. FCTs are noticeably low up to approximately 70% in the CDF. However, when only the last segment is lost, Equation 2 is used and the loss recovery takes longer. FCTs therefore increase above 70% in Figure 3a. When delACKs are used, only Equation 1 is used and the relative gain of using TLP and TLPR therefore increases. This is clearly shown in Figure 3b.

Contrary to the scenario in Figure 3a, as the effective RTT gets larger, the difference in PTO values for using different equations becomes much smaller. TLP performs close to RTOR as is shown in the lower part of the CDFs in Figure 3d. When either the last two or three segments are lost, RTOR performs slightly better than TLP. In this case, TLP retransmits the last segment first, causing time-delayed ER to recover other lost segments. The extra delay of a quarter

[1]One or more consecutive losses in a flow




(a) 4 seg., 5 ms owd, quickACKs (b) 4 seg., 5 ms owd, delACKs (c) 100 seg., 5 ms owd, quickACKs




(d) 4 seg., 160 ms owd, quickACKs (e) 4 seg., 160 ms owd, delACKs (f) 100 seg., 160 ms owd, quickACKs

Figure 3: Flow completion time for flows with varying sizes and different baseline one-way delays.

of an RTT is seen in TLP's CDF in the figure approximately above 25%. In Figure 3e, as delACKs are used with TLP, the delACK for the third segment resets the PTO when the last segment is lost. Approximately an extra 200 ms is added to FCTs. TLP becomes slower than RTOR as shown in the lower part of the CDF in the figure. However, when either the last two or three segments are lost, TLP performs better than RTOR as shown in the same figure, because when the last two segments are lost and RTOR retransmits the third segment, the corresponding ACK is delayed.

In the above scenarios, TLPR outperforms all the other mechanisms. With TLPR, up to one RTT+200 ms can be saved in comparison with TLP (in case of delACKs), and with TLP already giving lower FCTs than RTOR in many cases.

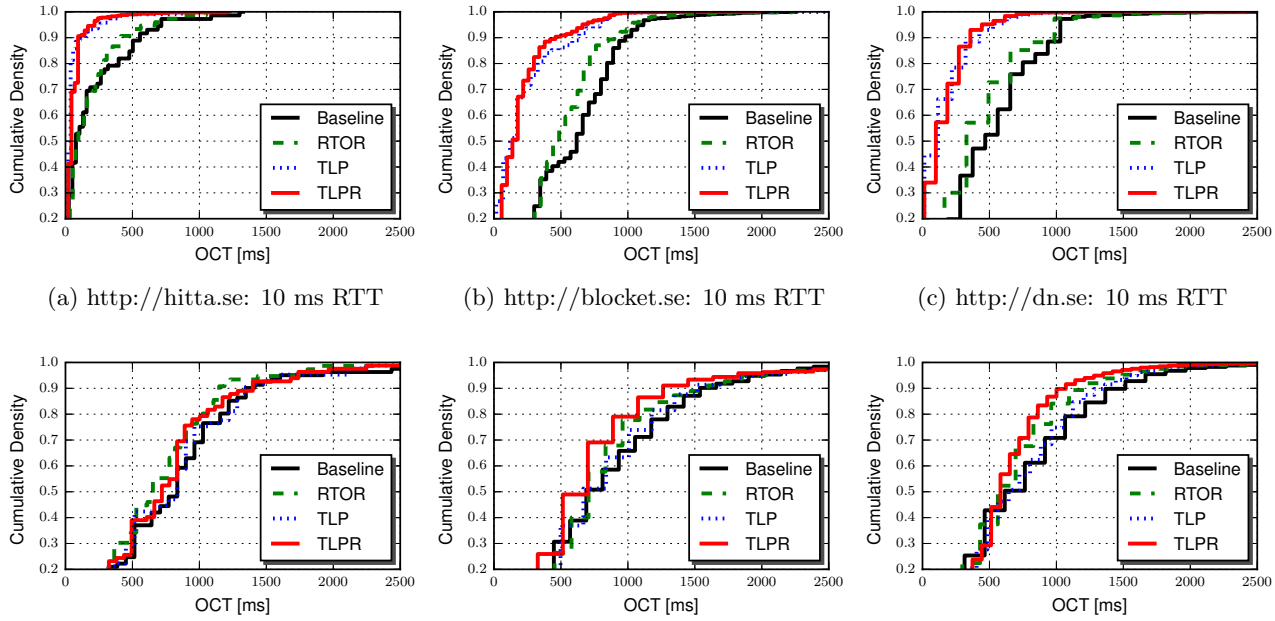
Figure 3c and Figure 3f show the results for 100 segment flows for different owds when quickACKs are employed at the receiver. The relative gain becomes smaller for larger flows as the tail becomes a smaller part of the total. For these flows the loss pattern also becomes less bursty, so the risk of having multiple losses in the tail decreases. As tail loss is more infrequent, FR and ER recover most losses and the difference between the baseline, RTOR, TLP, and TLPR becomes small. Besides, since transmitting 100 segments requires several RTTs, the RTO converges, improving the relative performance of both RTOR and baseline in Figure 3f. Moreover, TLP and TLPR require to be in the open state [9] to send the loss probes which is not always the case for the tail loss scenarios due to earlier losses in the same flows. Consequently, RTOR performs better than both TLP and TLPR in such cases, as shown in the same figure.

In the experiments, we have also calculated the fraction of spurious retransmissions in relation to all segments. The fractions are $2.18 \times 10^{-4}$ for the baseline, $3.67 \times 10^{-4}$ for RTOR, $3.49 \times 10^{-4}$ for TLP, and $3.48 \times 10^{-4}$ for TLPR. Our calculations include the case where lost ACKs lead to spurious retransmissions.

## 3.3 Web Page Experiments

So far we have only considered synthetic traffic. Below we focus on the web, being one of Internet's most used applications.

*Experimental Setup.* To perform controlled experiments but still use real web sites we recorded a large number of sites from the Swedish version of the Alexa Top 500 list [1] to a local web server. For our web experiments three representative sites were chosen: a lightweight telephony search engine consisting of 12 objects with a mean size of 24 Kbytes; a medium-sized classified site with 44 objects with a mean size of 52 Kbytes; and a larger newspaper site with 164 objects with a mean size of 25 Kbytes. We downloaded each site from the web server over an emulated network. The download process mimics the common web browser behaviour of first retrieving the HTML document, and then all resources (images, style sheets and javascripts) associated with the document. The resource download was performed using six parallel connections per sub domain, using persistent TCP connections. Experiments were performed with the receiver using delACKs. While previous experiments have only considered traffic patterns with a single burst, this experiment is focused on bursty persistent connections. The network conditions were emulated using KauNet [12] and included a range of RTTs (10-160 ms) and a 10 Mbps bandwidth. To create realistic segment loss we used a Gilbert-Elliot model to create correlated loss patterns similar to what is described




(a) http://hitta.se: 10 ms RTT (b) http://blocket.se: 10 ms RTT (c) http://dn.se: 10 ms RTT

(d) http://hitta.se: 160 ms RTT (e) http://blocket.se: 160 ms RTT (f) http://dn.se: 160 ms RTT

Figure 4: Completion time for web objects, with a receiver that uses delACKs.

in [11]. The average loss probability was configured to be approximately 1.5%. For each download, we calculated the average object completion time (OCT) for all objects in flows where at least one object experienced a segment retransmission in one of the four last segments of the object. This way, we reduce the amount of noise in the measurements while still capturing any effects of spurious retransmissions.

*Results.* The results for a subset of the experiments are shown in Figure 4. The graphs in the top row of the figure show OCTs for the selected web sites for an RTT of 10 ms and the graphs in the bottom row for an RTT of 160 ms.

Looking at the top row of the figure, we see that in scenarios with low RTTs all mechanisms perform better than the baseline and TLP and TLPR clearly gives the largest gains, reducing OCTs with more than 50% in many cases. This is consistent with the results in subsection 3.1, also showing large performance benefits for TLP and TLPR given low RTTs. This is due to the calculation of TLP's probe timer (PTO), which is set more aggressive than the RTO for very low RTTs. In scenarios where the RTT is larger (bottom row of Figure 4) the performance difference between the mechanisms is less pronounced. TLPR still performs best among the mechanisms, but RTOR is, on average, better than TLP. For example, if we consider the newspaper site, shown in Figure 4f, approximately 84% of all OCTs are less than 1 s for RTOR. For TLP, the corresponding fraction is only 74% and for baseline about 70%. The reason for TLP to perform worse than RTOR is due to its interaction with the delACK timer.

The fraction of spurious retransmissions in the experiments (in relation to all segments) are $4.8 \times 10^{-5}$ for the baseline, $5.9 \times 10^{-5}$ for RTOR, $1.8 \times 10^{-3}$ for TLP, and $2.1 \times 10^{-3}$ for TLPR.

### 3.4 Discussion

As seen in our experiments, RTOR reduces FCTs with at least one RTT as compared to the baseline. The loss recovery time can, however, never go below the RTO. As the PTO used by TLP can be smaller than the RTO, TLP can offer larger performance improvements for some scenarios. This is particularly visible for small RTTs in our experiments. One of the main reasons that RTOR sometimes performs better than TLP is that the PTO is reset by incoming ACKs. TLPR removes this problem and also performs the best in almost all evaluated scenarios.

We can also see from our evaluation that the largest relative gains in FCT are offered for small flows. For larger flows, all mechanisms including the baseline provide similar performance. In this case, the relative impact of the mechanisms becomes smaller as the tail constitutes a smaller fraction of the flow and tail loss is also less prevalent.

As RTOR, TLP, and TLPR all reduce the time a sender waits before it retransmits a segment they inevitably increase the risk of a spurious retransmission. However, the amount of spurious retransmissions observed in our experiments is small, with 0.21% for TLPR in the Web scenario as the largest value. In the end, use of the mechanisms is a trade-off between the performance gains offered and the extra network load caused during times of contention.

## 4. RELATED WORK

There are two main approaches to improve the loss recovery for short or application-limited flows: enable FR more often or improve the RTO mechanism so that less time is spent to detect losses. One of the earliest attempts to enable FR during loss recovery when a sender cannot receive three dupACKs due to the congestion window being too small is limited transmit [3] (LT). LT allows a TCP sender to send

new data beyond this limitation for each of the first two dupACKs to cause a FR. However, LT does not help when there are no unsent data or if the receiver buffer is full. Similar to the ER algorithm discussed in §2, Petlund et al. [18] suggested to use FR on reception of the first dupACK to avoid RTOs for application-limited flows.

None of the above mentioned solutions help when all outstanding segments are lost or the only outstanding segment is lost, which can only be detected via a timeout. Several prior works [6, 20, 21, 10, 23] showed deficiencies in calculating the RTO and proposed improvements. The work in [21] investigated the effect of the large initial RTO whereas work in [6] showed that TCP's loss recovery performance is largely influenced by the minimum RTO value. Spurious retransmission is, however, more likely to happen if a lower value for the minimum RTO is used. Work in [20] proposed an adaptive algorithm that selectively extends the minimum RTO to avoid spurious retransmissions. The authors of [10] proposed a new algorithm to calculate the RTO that was shown to have quicker loss detection and be more responsive to sudden changes of the RTT.

A solution similar to RTOR is proposed in [23], although the solution is based on an estimate of the implicit RTT. Previously, the work in [15] mentioned the problem of adding an implicit RTT to the RTO and suggested that it should be removed. However, the same author later mentioned the implicit RTT in [10] as a safety margin against spurious retransmissions and recommended not to remove it.

# 5. CONCLUSIONS

Several studies [9, 11] mention the prevalence of tail loss in TCP flows. In this paper, we have carried out a thorough evaluation of two recently proposed mechanisms RTOR and TLP as well as a new mechanism called TLPR that reduce TCP tail loss latency. TLPR applies the logic of RTOR to the TLP timer. We evaluated the mechanisms in three different scenarios: one scenario with controlled tail loss, one where loss was created by realistic background traffic, and a scenario involving web page downloads. The results from the experiments show that RTOR reduces tail loss recovery time by one RTT as compared to standard TCP, and possibly more if delACKs are used. Several factors such as loss patterns, RTTs and ACK mechanisms at the receiver influence the performance of TLP. While our evaluation shows larger gains of TLP as compared to RTOR in some scenarios, the new mechanism TLPR integrates the benefits of the two mechanisms and performs the best in most cases.

Since RTOR, TLP, and TLPR reduce retransmission times, they also increase the risk of spurious retransmissions. While we have observed some spurious retransmissions in our traces, no performance implications have been observed for the workloads and network configurations used in the paper.

# 6. ACKNOWLEDGMENTS

This work was funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed are solely those of the author(s).

# 7. REFERENCES

[1] Alexa top 500 global sites. `http://goo.gl/K4WqQH`.

[2] Tmix input files. `http://goo.gl/am4Gda`.

[3] M. Allman et al. Enhancing TCP's loss recovery using limited transmit. *RFC 3042*, January 2001.

[4] M. Allman et al. TCP congestion control. *RFC 5681*, September 2009.

[5] M. Allman et al. Early retransmit for TCP and stream control transmission protocol (SCTP). *RFC 5827*, April 2010.

[6] M. Allman and V. Paxson. On estimating end-to-end network path properties. In *Proc. of the ACM SIGCOMM*, Cambridge, USA, August 1999.

[7] M. Carbone and L. Rizzo. Dummynet revisited. *ACM SIGCOMM CCR*, 40(2):12–20, April 2010.

[8] P. Dixon. Shopzilla's site redo - you get what you measure. In *Proc. of Velocity*, June 2009.

[9] N. Dukkipati et al. Tail loss probe (TLP): An algorithm for fast recovery of tail losses. Internet draft, work in progress, IETF, February 2013. draft-dukkipati-tcpm-tcp-loss-probe-01.

[10] H. Ekström and R. Ludwig. The peak-hopper: A new end-to-end retransmission timer for reliable unicast transport. In *Proc. of the IEEE INFOCOM*, Hong Kong, March 2004.

[11] T. Flach et al. Reducing web latency: the virtue of gentle aggression. In *Proc. of ACM SIGCOMM*, Hong Kong, China, August 2013.

[12] J. Garcia et al. KauNet: Improving reproducibility for wireless and mobile research. In *Proc. of MobiEval*, San Juan, USA, June 2007.

[13] P. Hurtig. tcp: fixing TLP's FIN recovery. Linux kernel patch, June 2014. `http://goo.gl/fAVKpL`.

[14] P. Hurtig et al. TCP and SCTP RTO restart. Internet draft, work in progress, IETF, October 2014. draft-ietf-tcpm-rtorestart-04.

[15] R. Ludwig and R. H. Katz. The Eifel algorithm: making TCP robust against spurious retransmissions. *ACM SIGCOMM CCR*, 30(1), January 2000.

[16] M. Mathis and J. Mahdavi. Forward acknowledgment: Refining TCP congestion control. In *Proc. of ACM SIGCOMM*, Stanford, CA, USA, August 1996.

[17] V. Paxson et al. Computing TCP's retransmission timer. *RFC 6298*, June 2011.

[18] A. Petlund et al. TCP mechanisms for improving the user experience for time-dependent thin-stream applications. In *Proc. of IEEE LCN*, Montreal, Canada, October 2008.

[19] J. Postel. Transmission control protocol. *RFC 793*, September 1981.

[20] I. Psaras and V. Tsaoussidis. On the properties of an adaptive TCP minimum RTO. *Elsevier Computer Communications*, 32(5), March 2009.

[21] N. Seddigh and M. Devetsikiotis. Studies of TCP's retransmission timeout mechanism. In *Proc. of the IEEE ICC*, volume 6, Helsinki, Finland, 2001.

[22] M. C. Weigle et al. Tmix: A tool for generating realistic TCP application workloads in ns-2. *ACM SIGCOMM CCR*, 36, July 2006.

[23] Z. Wen and K. L. Yeung. On RTO timer implementation in TCP. In *Proc. of the ISDA*, Cairo, Egypt, December 2010.