Speeding Up TCP with Selective Loss Prevention

Zhenyu Zhou
Duke University
zzy@cs.duke.edu

Xiaowei Yang
Duke University
xwy@cs.duke.edu

***Abstract*—Low latency is an important design goal for reliable data transmission protocols such as TCP and QUIC. However, timeout-based loss recovery can unnecessarily increase end-to-end latency. Previous work in reducing timeout-based loss recovery latency either duplicates every packet to avoid loss or focuses on fine-tuning the timeout timers to shorten the timeout latency without causing spurious packet retransmissions. In this work, we propose a new mechanism called Selective Loss Prevention (SLP) to reduce the loss recovery latency of a reliable transport protocol.**

Through extensive trace analysis, we find that not all lost packets are equal. The loss of packets with certain flags, such as SYN and PSH, is more likely to cause timeouts than other packets. Based on this observation, we propose to selectively duplicate an "important" packet whose loss is likely to increase a connection's latency. We design an algorithm to determine when to duplicate a lost packet proactively and incorporate it into TCP's congestion control algorithm so that duplicate packets will not congest the network. We incorporate SLP into Linux's kernel and evaluate its performance. Our results show that SLP can reduce timeout-based latency caused by the loss of important packets in a connection, and its overhead is low.

I. Introduction

Many TCP-based applications demand low latency. Studies have shown that service providers might lose more than a billion dollars or 3.5% of their conversions if their sites were delayed by just one second [1], [2]. As a result, there has been much effort in reducing latency for both web and data-center applications. This work includes algorithms and systems to improve TCP's performance [3], [4], [5], [6], [7], [8], [9], newer and faster transport protocols such as SPDY [10], QUIC [11], and NDP [12], and efforts to completely bypass TCP and use RDMA over lossless Ethernet to achieve ultra-low latency [13], [14] for data center applications.

In this work, we re-examine the TCP latency reduction problem. We observe that much of the previous work focuses on improving TCP's congestion control algorithm to fully utilize any available bandwidth [15], [16], [17], [18], [19], [20]. However, from the real-world TCP traces we analyzed, we discover that TCP timeout is a main factor that increases TCP latency. A TCP sender infers a packet loss from a missing acknowledgement. However, it cannot reliably tell whether a missing acknowledgement is caused by a packet loss or by network delay [21]. If it retransmits too early, it leads to unnecessary retransmissions and may cause congestion collapse [22], [23]. If it waits too long, it hurts application performance. This problem is alleviated by fine-grained TCP timers [5] and TCP's fast retransmission technique. However, as we observe in the real-world traces (§ II-B), fast retransmission is not always triggered. Timeouts still constitute a significant fraction of a flow's loss recovery events: our analysis in § II-B shows that they account for around 10.1% of all retransmission events. For wide area networks, even if we use fine-grained timers, the timeout period can still be in the order of a few hundred milliseconds.

978-1-6654-4131-5/21/$31.00 ©2021 IEEE

In this work, we propose to use a technique called Selective Loss Prevention (SLP) to further reduce TCP latency. SLP aims to predicate the "important" packets in a flow whose losses are likely to cause timeouts, and aggressively duplicate them to prevent loss. Prior to this work, there exist two generic loss recovery mechanisms: Automatic Repeat Request (ARQ) [24], and Forward Error Correction (FEC) [25]. TCP uses ARQ for loss recovery. We investigated whether FEC can reduce the loss recovery latency in reliable transmission protocols. However, a simple analysis shows that applying FEC to all packets in a connection with a low loss rate (§ II-A) is not cost effective. This is because FEC introduces redundancy in data transmission. It introduces high packet transmission overhead but does not significantly shorten flow completion time. Unsurprisingly, QUIC, first incorporated FEC, but soon abandoned it [11]. This observation motivates us to ask this question: *Can we achieve a better cost-benefit tradeoff if we selectively apply FEC to certain packets in a connection?* We call this approach Selective Loss Prevention. In this work, we focus on the simplest form of FEC: packet duplication, and defer to study more sophisticated FEC schemes to future work.

We encounter two technical challenges in this study. First, how does a TCP sender decide which packets warrant SLP? Second, if a network is congested, packet duplication may worsen the congestion status. How can we prevent redundant packets from congesting the network? To address these challenges, we first use real-world TCP traces collected by CAIDA [26] and at data center networks [27] to discover what types of packets are more likely to cause TCP timeouts. Our results suggest that head loss (i.e., the loss of a SYN packet), tail loss, and the loss of packets with PSH flags are at least ten times more likely to cause TCP timeouts than other types of TCP packets. Second, we design an algorithm to decide whether a TCP sender should duplicate a packet. The algorithm analyzes the potential latency reduction for duplicating a packet based on the current available bandwidth, the inferred loss rate, and the likelihood that the packet loss may cause a timeout. Finally, we modify TCP's congestion algorithm to take into account duplicate packets. The new congestion control algorithm will reduce

the available congestion window size of TCP when duplicate packets are sent but does not change the available receiving window size as receivers will discard the duplicate.

To evaluate the cost and effectiveness of SLP, we design a framework to incorporate SLP into the current TCP design. We also implement SLP using a Linux TCP implementation (§ II-C). We then conduct a "what-if" analysis using the wide area and data center traces mentioned above. The what-if analysis answers the question: *What are benefits and costs of SLP if we enable SLP?* Our analysis shows that duplicating SYN packets alone can shorten the flow completion times for all flows suffering SYN losses, which consist of 2.6% flows among all flows. We then run experiments to provide a preliminary evaluation of SLP's effectiveness in a real-world like environment. We compare an SLP-enabled TCP connection and a regular TCP connection between a client and a server machine using an emulated web workload. The experiments show that SLP can reduce the long tail of flow completion times caused by TCP timeouts.

As a TCP variant, we acknowledge that deploying SLP on TCP needs a considerable amount of time. Luckily, new pluggable congestion control interfaces provided by emerging user-space techniques such as QUIC [11] offer quicker iteration for SLP's deployment.

In summary, this work makes the following contributions: 1) we propose SLP to reduce TCP latency; 2) we investigate what types of packet losses are likely to cause TCP timeouts; 3) we design algorithms to incorporate SLP into TCP without causing network congestion; and 4) we provide a preliminary evaluation of the costs and benefits of SLP.

II. Design

In this section, we describe the design rationale behind SLP. We use a simple analysis to show that enabling FEC for all packets is not cost effective. We then use trace analysis to discover what types of packets are more likely to cause TCP timeouts if they are lost. Finally, we describe how to incorporate SLP into the current TCP design.

A. FEC Analysis

In this subsection, we compare the per-packet overhead of two loss recovery mechanisms: FEC and ARQ. The metric we use is the fraction of *goodput*, i.e., the number of unique packets delivered divided by the total number of packets sent. For simplicity, we assume the network randomly discards a packet with a probability p, the FEC group size is n, the number of redundant packets per group is k, and all packets have the same size.

For ARQ, each packet may be discarded with a probability p, and received reliably by a receiver with a probability $1-p$. The probability that the packet will be reliably received by a receiver after being transmitted x times is: $p^{x-1}(1-p)$. Thus, the expected number of transmission times per packet is: $\mathbb{E}[X] = \sum_{x=1}^{+\infty} xP(X=x) = \frac{1}{1-p}$. The fraction of ARQ's goodput is $\eta_{ARQ} = 1-p$.

For FEC, if we assume that no packets need to be retransmitted, the fraction of its goodput is $\eta_{FEC} = \frac{n}{n+k}$.

If the packet loss rate satisfies $p \leq \frac{k}{n+k}$, which is the condition to ensure that FEC can fully recover a lost packet without any retransmission, we have $\eta_{ARQ} \geq \eta_{FEC}$.

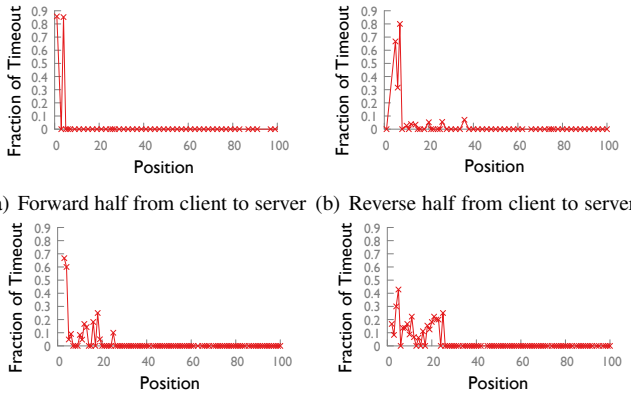
B. Important Packets

In this subsection, we use real-world packet traces to study the question whether the losses of different types of packets affect TCP performance differently. The types of packets whose losses are more likely to cause TCP timeouts are candidates for SLP.

1) Packet Traces: We analyze real-world packet traces from data center networks (DCN) [27] as well as from the wide area Internet collected by CAIDA [26]. These traces include real-world traffic and their characteristics are different from active probing traffic [28].

The DCN traces were collected from multiple data centers including commercial cloud data centers, private enterprise data centers, and university campus data centers by SNMP polls. Each trace includes several days' traffic collected at randomly chosen locations within a data center. The traces only include packet headers without payload. We use the IP addresses in the packet traces to distinguish a client from a server in a given TCP connection: the client is the one actively sending the first SYN packet and the server is the one responding with a SYN/ACK.

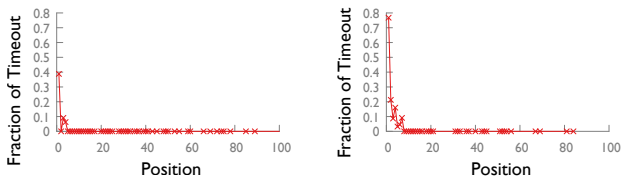
The CAIDA traces were collected at a backbone OC192 links in Chicago in 2016. However, the traces hide the authentic IP addresses by CryptoPan [29]. As a result, we cannot pair the two directions of a TCP connection. The sender of the reverse direction (from the server to the client) does not match the receiver of the forward direction due to anonymization. In our analysis, we do not distinguish the two directions.



(a) Forward half from client to server (b) Reverse half from client to server

(c) Foward half from server to client (d) Reverse half from server to client

Figure 1. **This figure shows the position analysis results for the DCN traces. The x-axis is the position index of a packet in a one-way TCP connection. We break each connection into two halves by the packet in the middle. For the first half, we index the packet from the first packet. For the second half, we index the packets from the reverse order so that the last packet always has a position 1 regardless of the connection length, and there is no overlap between the two directions. The y-axis shows the fraction of timeout-based retransmissions among all packet retransmissions. As can be seen, head losses and tail losses are more likely to cause timeouts.**

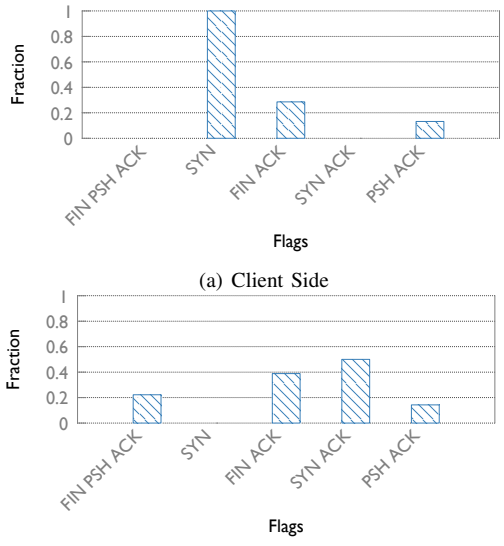


(a) Forward half (b) Reverse half

Figure 2. **This figure shows the position analysis results for the CAIDA traces.**

2) Infer Retransmission Types: To study what types of packet losses are likely to cause TCP timeouts, we first need to infer packet losses from the packet traces, and then what type of retransmission is triggered by a packet loss. To do so, we infer packet losses from retransmitted packets, which will show up as packets with duplicate sequence numbers in our traces [30]. We then use the k-means algorithm to classify retransmissions into two groups: timeout-based retransmission and fast retransmission. The classifier separates the two groups based on the packet transmission intervals. The group with the longer interval is classified as the time-out group, and the one with the shorter interval is classified as the fast retransmission group. The advantage of using k-means is that we need not manually set the threshold between the time-out group and the fast transmission group. The algorithm itself automatically infers this threshold. Wireshark uses statically configured timeout values though to infer time-outs [31], [32]. We compared our results with the results inferred by Wireshark, and found that around 3% of the data points were grouped differently.

3) Position Analysis: After inferring timeout-based retransmissions, we measure the fraction of timeout-based retransmissions among all retransmissions seen in the traces for packets lost at different positions in a TCP connection.



(a) Client Side

(b) Server Side

Figure 3. **This figure shows the fraction of timeouts caused by the losses of packets with different TCP flags for the DCN packet traces. The x-axis shows the flags carried by the lost packets and the y-axis shows the fraction of timeouts.**

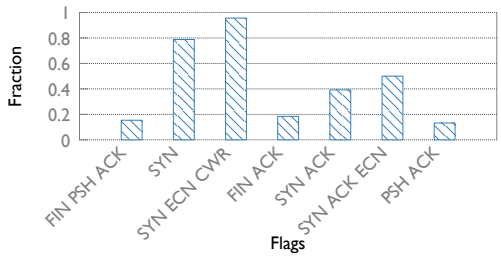


Figure 4. **This figure shows the fraction of timeouts caused by the losses of packets with different TCP flags for the CAIDA packet traces.**

We refer to this analysis as position analysis. For each connection, we break it into two halves by the packet in the middle of the connection. Since different connections have different number of packets, for the first half, we index a packet's position from the forward direction. For the second half, we index it from the reverse direction so that the last packet of a connection always has a position index 1. Because very few connections have more than 200 packets, we stop our index at 100.

Figure 1 and 2 show the results for this analysis. As can be seen, for both the data center and CAIDA packet traces, head and tail losses are more likely to cause timeouts than packet losses at other positions of a connection. Not surprisingly, more than 70% of TCP SYN packet losses caused a timeout.

4) Flag Analysis: Next, we analyze whether the losses of packets with different TCP flags have different likelihoods to cause timeouts. We refer to this study as flag analysis. Similar to the position analysis, we measure the total number of packet retransmissions for packets with different TCP flags and then the fraction of packet retransmissions triggered by timeouts among all retransmissions with the same type of TCP flag. A packet may carry multiple TCP flags. For each combination of TCP flags observed in our packet traces, we count it as a category. Figure 3 and 4 show the flag analysis results for all significant categories. We ignored the categories with $< 1\%$ packets. As can be seen, the losses of packets with FIN and SYN flags are more likely to cause timeouts, which is consistent with position analysis.

In addition, we find that the loss of a packet with the PSH flag has a significantly higher likelihood to cause timeouts than a regular data packet without any special TCP flag other than the ACK flag. We speculate this result may be due to the fact that when a TCP sender sets the PSH flag, it may not have more data to send immediately. So the loss of a PSH packet may not trigger three duplicate ACKs, which require three subsequent packets to be received by the TCP receiver. Therefore, the loss of a PSH packet is more likely to cause a timeout.

Takeaway: Our trace analysis suggests that the losses of certain packets, i.e., packets near the beginning or end of a connection, and packets carrying the PSH flag, are more likely to cause TCP timeouts than other packets. Therefore, applying SLP to these packets may shorten TCP latency without introducing much overhead.

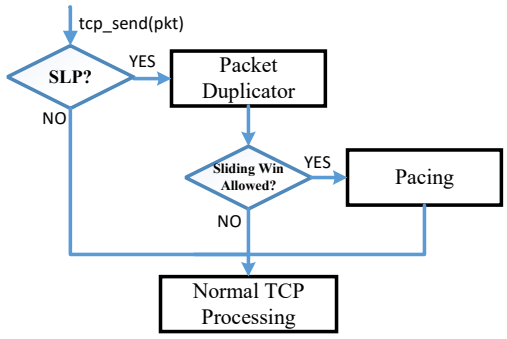


Figure 5. **This figure shows how we incorporate SLP into TCP.**

C. Architecture

Based on our trace analysis, we incorporate SLP into the existing TCP design. Figure 5 shows the key components to realize SLP. Before a TCP sender sends a packet, it invokes the SLP Arbitrator module to determine whether it should duplicate a packet (§ II-C1). If yes, it then checks whether the TCP's congestion window allows the duplication. If so, the Packet Duplicator module will duplicate the packet and update the TCP congestion window using the algorithm we soon describe in § II-C2. It then calls the Pacing module to insert a gap between the original packet and the duplicated packet, and send both packets. We use packet pacing to avoid bursty packet losses [6].

Next, we describe how the SLP arbitrator selects a packet for loss prevention, and how we update the TCP's congestion control algorithm to incorporate duplicate packets.

1) SLP Arbitrator: In our design, the SLP arbitrator uses 5 parameters to select a packet for duplication. They include: 1) the estimated loss rate p; 2) the estimated available bandwidth of this connection bw; 3) the likelihood that the packet loss will lead to a timeout q; 4) the packet size pkt_size; and 5) the current TCP timeout value RTO. The SLP arbitrator uses these parameters to estimate whether a duplication will shorten or increase a flow's completion time. Specifically, if a packet is not duplicated, with probability pq, the original packet is lost and a timeout occurs. The arbitrator estimates that the connection's latency will increase on average by $t_1 = p * q * RTO$ seconds. If the packet is duplicated and the original packet is not lost, then the connection's latency will on average increase by the time it takes to transmit the duplicate packet: $t_2 = (1-p) * pkt\_size/bw$. If $t_1 \geq t_2$, it is advantageous to duplicate the packet. Otherwise, the packet should not be duplicated.

The SLP arbitrator can learn the parameters p, bw, and q from the current TCP connection or use the system's default configuration value. The SLP arbitrator can obtain the values of pkt_size and RTO from TCP. The loss rate p can be estimated from past and current connections or use a static value that reflects the average loss rate of the TCP's operating environment, e.g., a data center or the wide area network.

The probability of timeout q is the most important but also the hardest parameter to set. It is our future work to study how to accurately estimate q. Our current design sets p and q using the values we obtain from our trace analysis. We set $p = 2\%$, which is the average loss rate measured from the CAIDA traces. We set $q = 1$ for SYN packets, and 10% for PSH packets. With these settings, assuming $RTO = 1\,\text{s}$, a SYN packet whose size is less than 70 bytes even with all the TCP options will be selected for duplication if the available link bandwidth is greater than $3.35\,\text{Mbps}$. A PSH packet will be selected for duplication if the available bandwidth is greater than $7.16\,\text{Mbps}$ assuming its size is less than 1500 bytes.

2) Modified Sliding Window Algorithm: It is important that a duplicate packet does not lead to network congestion or overflow a receiver's buffer. Since TCP uses a sliding window algorithm for congestion control, flow control, and reliability, we can use the size of the sliding window to determine whether we can safely send a duplicate packet without congesting the network nor overflowing a receiver's buffer. However, a duplicate packet occupies a router's buffer but will be discarded by a receiver. Therefore, we must modify the sliding window algorithm to accommodate packet duplication.

An SLP-enabled TCP sender's sliding window size win is computed as the minimum of the receiver's advertised window size $rwnd$ and its congestion window size $cwnd$ shrunk by the number of duplicated bytes: $win = \min(cwnd - dup\_bytes, rwnd)$. The dup_bytes is set to zero when TCP starts and we soon describe how it is updated. The sender maintains three additional variables in its sequence number space: the highest ACKed sequence number $high\_acked$, the highest sequence number allowed to send by the sliding window $high\_allowed = high\_acked + win$, and a pointer pointing to the next-to-send sequence number $next$ as shown in Figure 6.

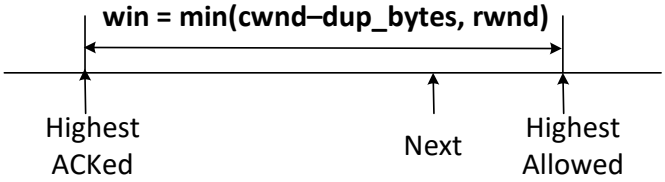


Figure 6. **Congestion Window with SLP. Three points are kept inside the sliding window for making the duplication decision.**

If the SLP Arbitrator determines that a packet $next$ with packet size pkt_size is worth duplicating, the Packet Duplicator must check whether both the congestion window size and the receiver window size allow it: $next + 2 * pkt\_size < high\_acked + cwnd$, and $next + pkt\_size < high\_acked + rwnd$. If this condition satisfies, the packet will be duplicated. It then moves $next$ pointer to $next + pkt\_size$, and sets its duplicate byte counter to the size of the duplicate packet: $dup\_bytes = pkt\_size$. The sender's sliding window shrinks correspondingly if it is bound by the congestion window size: $win = \min(cwnd - dup\_bytes, rwnd)$. The TCP sender will also store the sequence number of a duplicate packet in a variable: $dup\_seq = next$ so that it can update its sliding window size when it receives an ACK. For simplicity, our current design allows only one duplicate packet per sliding

window.

When an SLP-enabled TCP sender receives an ACK, it will check whether the ACK acknowledges any duplicate byte. If it does not, it will update the highest ACKed sequence number and slide its window as in the TCP design. Otherwise, if the current ACK number ack includes n bytes from a duplicate packet: ack == $dup\_seq + n - 1$, where $n \leq dup\_bytes$, it will update the duplicate byte counter: dup_bytes -= n and update its sliding window size correspondingly: $win = \min(cwnd - dup\_bytes, rwnd)$. It will then update the duplicate sequence number to dup_seq += $n - 1$. If the ACK number completely acknowledges the duplicate sequence number $ack > dup\_seq + dup\_bytes - 1$, it will reset dup_seq and dup_bytes to zero, and correspondingly, its sliding window grows to $\min(cwnd, rwnd)$.

III. Preliminary Evaluation

In this section, we evaluate the cost and effectiveness of SLP using a what-if analysis and a simple emulation experiment. It is our future work to thoroughly evaluate SLP's performance.

A. What-If Analysis

Percentile	1%	10%	50%	90%	99%
w/ SLP (s)	0.10	0.15	0.68	9.04	19.58
w/o SLP (s)	0.81	0.86	1.37	9.77	20.26
Reduction	87.7%	82.6%	50.4%	7.5%	3.4%

Table I

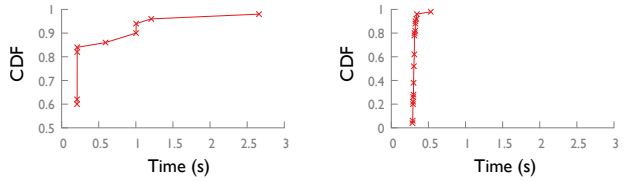
This table shows how SLP can reduce the flow completion time for flows that suffer SYN packet losses in the CAIDA packet traces.

We analyze how SLP can reduce a flow's completion time using the CAIDA packet traces. About 2.6% of all flows in the CAIDA traces suffer SYN packet losses. We assume that we duplicate SYN packets for those flows and the duplicate SYN packets are not lost. Table I shows how SLP can reduce flow completion times. Specifically, the shortest 1% of the flows with SYN losses complete in 0.81 s. With SLP, those flows could complete within 0.10 s, an 87.7% reduction. The median 50% of the flows with SYN losses complete in 1.37 s. With SLP, those flows could complete within 0.68 s, a 50.4% reduction. Overall, SLP can reduce the flow completion times for all flows with SYN losses.

B. Emulation

We conduct a preliminary emulation experiment using two Linux (Ubuntu 14.04 LTS) machines on Emulab [33], each with 12 processor cores and 16 GB of memory. We set one machine as a web client and another as a web server. We then configure the client machine to randomly discard 2% of its outgoing packets [34] using the Linux TC (Linux Traffic Control) module for all the experiments. The topology between the client and the server is not under our control. We run a modified HTTP server on the server machine. Both the client and the server's TCP stack supports SLP.

We configure the client machine to fetch a file from the server for 50 times and measure the flow completion time with and without SLP enabled. We set the file size to be the average file size from all files downloaded from the Alexa Top 100 sites [35]. We downloaded the files containing ads, html, css, flash, icon and script from these top sites and computed the average file size, which is 34 KB.



(a) Without SLP (b) With SLP

Figure 7. **This figure shows the CDF of file completion times with and without SLP. The x-axis is the file completion time, and the y-axis is the cumulative distribution of a file completion time.**

Figure 7(a) and Figure 7(b) show the file completion time distribution with and without SLP with a 2% loss rate, respectively. We find that SLP significantly reduces the long tail of the flow completion times. Before deploying SLP, although most flows complete within 0.5 s, there are several flows whose completion times are more than 1 s or even 2 s. After deploying SLP, all flows complete within 0.5 s.

We have also run experiments with different file sizes ranging from 2^{-2} KB to 2^{13} KB, with the file size doubling at each step. The results are similar: SLP reduces SYN timeouts and shortens the tail latency of short file completion. For long files, the file completion latency is dominated by the file size, and SLP's benefit is not as significant as for short files. Our experiments also show SLP's overhead is negligible. We observe no noticeable CPU/memory utilization changes compared to the baseline case without SLP enabled.

IV. Related Work

There has been much work in improving TCP performance. But most of them aim to improve the TCP's congestion control algorithm to fully utilize a bottleneck's bandwidth [15], [16], [17], [18], [19], [20]. In contrast, this work explores how selectively applying FEC to certain TCP packets may improve TCP's loss recovery time.

Amend et al. [36] propose to establish multiple connections at the beginning of an MPTCP connection so that the loss of one SYN packet will not slow down the connection setup time. Our work shows that duplicating the SYN packet along the same path can also avoid timeouts and shorten a TCP's connection setup time.

Flach et al. [37] introduce the idea to proactively duplicate all packets to recover from packet loss. We have shown in Section II-A that preemptively duplicates all packets may reduce the overall goodput of a network. SLP selectively duplicates important packets and can achieve a better tradeoff between low latency and high throughput.

From QUIC's specification [38], we discover that a QUIC connection sends its second handshake packet for negotiating crypto algorithm twice. This approach bears the same spirit as SLP, but to the best of our knowledge, we are the first to formulate the concept of SLP. In addition, we have designed algorithms to determine when a packet is worth duplication, and how to prevent duplicate packets from congesting the network.

There are also researchers focusing on improving the timeout mechanism. Lai et al. and Zhou et al. [39], [40] aim to tune RTO more elaborately to shorten TCP timeouts while this work aims to prevent TCP timeouts. Zhou et al. [40] point out that head and tail retransmissions contribute most to performance degradation. However, they simply treat head loss as data unavailable on the server side. Vasudevan et al. [5] show that the timeout values designed for the Internet are too coarse-grained for data center networks. They proposed to use fine-grained TCP timers to reduce the TCP timeout values for data center networks and also show the fine-grained TCP timers can reduce the latency of wide-area TCP traffic. This work is orthogonal to Vasudevan's proposal. We study how to avoid TCP timeouts using preemptive packet duplication. Our technique can shorten TCP latency with both fine-grained and coarse-grained timers.

V. Conclusion

Timeouts can significantly increase the latency of TCP-based applications. This work proposes Selective Loss Prevention, an approach that preemptively duplicates certain TCP packets to avoid TCP timeouts. We analyze TCP packet traces collected from both a backbone link and several data center networks to show that head loss, tail loss, and the loss of packets with PSH flags are more likely to cause timeouts than other packets. We propose to incorporate SLP into TCP, and design algorithms to determine when to duplicate a packet and how to prevent duplicate packets from congesting the network. We implement an SLP prototype using Linux, and conduct preliminary evaluations using what-if analysis and emulation. Our analysis shows that for the 2.6% of TCP flows that suffer SYN losses in our packet traces, SLP can significantly reduce the flow completion times. Similarly, our emulation experiments suggest that SLP can reduce the long tail of the flow completion times caused by TCP timeouts.

References

[1] K. Eaton, "How One Second Could Cost Amazon $1.6 Billion In Sales," https://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales, 2012.

[2] J. Bixby, "Case Study: The Impact of HTML Delay on Mobile Business Metrics," http://www.webperformancetoday.com/2011/11/23/case-study-slow-page-load-mobile-business-metrics/, 2011.

[3] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *ACM SIGCOMM*, 2010.

[4] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks," in *ACM SIGCOMM*, 2012.

[5] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication," in *ACM SIGCOMM computer communication review*, 2009.

[6] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the Performance of TCP Pacing," in *IEEE INFOCOM.*, 2000.

[7] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, "TCP Fast Open," in *ACM CoNEXT*, 2011.

[8] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, Implementation and Evaluation of Congestion Control for Multipath TCP," in *NSDI*, 2011.

[9] S. Floyd, "TCP and Explicit Congestion Notification," *ACM SIGCOMM Computer Communication Review*, 1994.

[10] M. Belshe and R. Peon, "SPDY: An Experimental Protocol for A Faster Web," http://www.chromium.org/spdy/spdy-whitepaper, 2012.

[11] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar *et al.*, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *ACM SIGCOMM*, 2017.

[12] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-Architecting Datacenter Networks and Stacks for Low Latency and High Performance," in *ACM SIGCOMM*, 2017.

[13] I. D. 802.3bd MAC Control Frame for Priority-based Flow Control Project, "Superseding IEEE 802.3x Full Duplex and Flow Control," http://www.ieee802.org/3/bd/, 2010.

[14] C. DeSanti, "IEEE DCB. 802.1Qbb - Priority-based Flow Control," http://www.ieee802.org/1/pages/802.1bb.html, 2011.

[15] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Rent, and SACK TCP," *ACM SIGCOMM*, 1996.

[16] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on selected Areas in communications*, 1995.

[17] S. Floyd, A. Gurtov, and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," 2004.

[18] S. Floyd, "HighSpeed TCP for Large Congestion Windows," 2003.

[19] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, Architecture, Algorithms, Performance," *IEEE/ACM transactions on Networking*, 2006.

[20] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS*, 2008.

[21] L. Zhang, "Why TCP Timers Don't Work Well," in *ACM SIGCOMM Computer Communication Review*, 1986.

[22] A. Kesselman and Y. Mansour, "Optimizing TCP Retransmission Timeout," in *International Conference on Networking*, 2005.

[23] V. Jacobson, "Congestion Avoidance and Control," in *ACM SIGCOMM computer communication review*, 1988.

[24] A. Elhakeem, "Automatic Repeat Request," *Wiley Encyclopedia of Electrical and Electronics Engineering*, 2001.

[25] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft, "Forward Error Correction (FEC) Building Block," Tech. Rep., 2002.

[26] C. for Applied Internet Data Analysis, "The CAIDA Anonymized Internet Traces Dataset," http://www.caida.org/data/passive/passive_trace_statistics.xml, 2016.

[27] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *ACM IMC*, 2010.

[28] P. Barford and J. Sommers, "Comparing Probe- and Router-Based Packet-Loss Measurement," *IEEE Internet Computing*, 2004.

[29] T. Zseby, A. King, M. Fomenkov *et al.*, "Analysis of unidirectional ip traffic to darkspace with an educational data kit," Cooperative Association for Internet Data Analysis, Tech. Rep., 2014.

[30] J. Postel, "Transmission Control Protocol," 1981.

[31] J. Aragon, "TCP Fast Retransmit detected only within 20 ms of DupACK," https://osqa-ask.wireshark.org/questions/24168/tcp-fast-retransmit-detected-only-within-20-ms-of-dupack, 2013.

[32] W. Wang, "TCP Analyze Sequence Numbers," https://wiki.wireshark.org/TCP_Analyze_Sequence_Numbers, 2011.

[33] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, "Large-scale Virtualization in the Emulab Network Testbed," in *USENIX ATC*, 2008.

[34] M. Yajnik, S. Moon, J. Kurose, and D. Towsley, "Measurement and Modelling of the Temporal Dependence in Packet Loss," in *INFOCOM'99. IEEE*, 1999.

[35] Alexa, "Alexa Top Global Sites," http://www.alexa.com/topsites, 2013.

[36] M. Amend, E. Bogenfeld, and A. Philipp, "Techniques for Establishing A Communication Connection between Two Network Entities via Different Network Flows," 2018.

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

[38] J. Roskind, "Multiplexed stream transport over UDP," 2013.

[39] C. Lai, K.-C. Leung, and V. O. Li, "TCP-NCL: A Unified Solution for TCP Packet Reordering and Random Loss," in *IEEE PIMRC*, 2009.

[40] J. Zhou, Q. Wu, Z. Li, S. Uhlig, P. Steenkiste, J. Chen, and G. Xie, "Demystifying and Mitigating TCP Stalls at the Server Side," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, 2015.