Revisiting Loss Recovery for High-Speed Transmission

Hui Xie[†‡] and Tong Li[§*]
Tsinghua University[†], Qinghai University[‡], Renmin University of China[§]
Email: xdhui1981@163.com, tong.li@ruc.edu.cn

***Abstract*—The emerging applications including cloud AR/VR gaming, ultra-high definition (UHD) streaming, Metaverse, etc. imply the demand for ultra-high bandwidth transmission in the wide area network (WAN). During the past four decades, a great number of high-speed TCP variants have been proposed to improved the throughput for WAN transmission. This paper conducts a measurement study on loss recovery for high-speed transmission, and exposes that receive buffer starvation is a result of capability mismatch between loss recovery and high-speed TCP advancements such as non-loss-based congestion control. To mitigate this mismatch problem, an opportunistic random redundant retransmission (OR3) algorithm, as well as its TCP implementation TCP-OR3, is proposed. OR3 accelerates loss recovery by minimizing the maximum retransmission times of each packet. Experiment results shows that TCP-OR3 alleviates receive buffer starvation and tackles the bottleneck of the legacy approaches such as parallel TCP in the case of high-speed transmission.**

***Index Terms*—loss recovery, wide area network, parallelism, opportunistic, random redundant retransmission**

I. Introduction

The emergence of cloud computing has driven a dramatic surge in massive data transmission from end to cloud or cloud to cloud over the Internet. Large corporations may deploy dedicated links to ensure fast and reliable data transmission for part of their services. However, a vast number of other services are transmitted via the public and cheap links with high bandwidth (*e.g.,* 1 or 10 Gbps) but long round-trip time (RTT) and high packet loss ratio (PLR). This proportion is much larger for small and medium enterprises who cannot afford the expensive dedicated links. This raises a question: "*Is it possible to achieve as good a performance as the dedicated networks via non-dedicated links?*"

Considering a network with high bandwidth-delay product (BDP) and lossy links, the desire for fast and reliable data transmission in interactive scenarios is far from satisfactory. For example, data transmissions over long distances using TCP-based protocols (including FTP and HTTP) can only utilize a limited amount of bandwidths [1]. It is observed that network infrastructures with multi-Gbps delivery capacity provide only a rate of a few Mbps over cross-country and intercontinental distances (see Figure 4).

TCP performance improvement over large-BDP and lossy Internet is a longstanding problem. Since loss-based TCP's inappropriate back off results in low bandwidth utilization of wide-area transmission over the Internet. A number of high-speed and non-loss-based TCP variants, such as FAST TCP [2], Compound TCP [3], and TCP BBR [4], have solved part of the bottlenecks well. However, their recovery capability does not meet the high-speed requirement (see §III-B). Well-tuned application-level optimization such as parallel TCP [5], [6] is proved to be affective for large dataset transmission. However, predicting the optimal parameters are challenging as conditions are always changing throughout the transmission [7], [8].

In this paper, we conduct a measurement study on loss recovery for high-speed transmission. We for the first time expose that the current capability of loss recovery does not match the capability of high-speed TCP advancements such as non-loss-based congestion control. Specifically, we define the receive-buffer-bubble (RBB) as the gap in sequence space between non-contiguous blocks of data that have been received and queued at the receiver. Throughout comprehensive measurements and analysis, we find that high-speed TCP advancements result in more RBBs, since the RBBs cannot be recovered timely, they further suffer from buffer starvation. We also demonstrate that the application-level optimization such as parallel TCP cannot solve the above problem well due to its repeated turning requirement and negative effect.

To mitigate the capability mismatch between loss recovery and throughput improvement of high-speed TCP, we propose `OR3` (Opportunistic Random Redundant Retransmission), a transport-layer algorithm that improves retransmission efficiency. We further implement OR3 upon TCP called `TCP-OR3`. TCP-OR3 is a sender-side modification of TCP. Evaluation results show that TCP-OR3 alleviates receive buffer starvation and tackles the bottleneck of parallel TCP in the case of high-speed transmission.

The rest of this paper is organized as follows. Section II introduces the related work of loss recovery algorithms, accelerating protocols, and parallel TCP. Section III gives a comprehensive measurement study on the performance of high-speed TCP variants and parallel TCP. Following that, in Section IV we describe the design of OR3. Finally, we evaluate the performance of TCP-OR3 in Section V, and conclude the paper in Section VI.

*Tong Li is the corresponding author (email: tong.li@ruc.edu.cn).

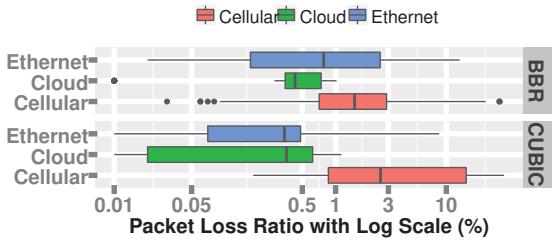


Fig. 1. Packet loss distribution of wide-area data transmission. *Cloud*, *Ethernet*, and *Cellular* refer to the scenarios of cloud to cloud, end to cloud via Ethernet links, and end to cloud via cellular links, respectively.

II. Related Work

Loss recovery algorithms. The TCP stack has proposed a series of loss detection improvements. Among them, FACK [9], RACK [10], and TLP [11] are based on the feedback scheme called selective acknowledgment (SACK) option [12]. SACK notifies the sender of at most 4 non-contiguous data blocks that have been received and queued, so the sender only need retransmit the actually lost packets. In this paper we have demonstrated that SACK may still suffer from spurious retransmissions in high-speed transmission (see §III-B). Li *et al.* proposed TCP-TACK [13] to extend the data field of ACKs, allowing ACK to carry more than 4 blocks. QUIC [14] also shares a similar idea of carrying more blocks in the ACK frames. However, both of them require dual-side modifications.

Accelerating protocols. High-speed TCP versions (such as CUBIC [15], H-TCP [16], BIC [17], and PRR [18]), and QUIC CUBIC [14] fall in the category of "loss-based" congestion controller. On the other hand, Vegas [19], FAST TCP [2], Compound TCP [3] are supposed to be "delay-based" congestion controllers. PCC [20] and Copa [21] is designed as "performance-oriented" or "learning-based" congestion controllers. BBR [4] is non-loss-based, which is proposed by Google as a "congestion-based" congestion controller.

Parallel TCP. Parallel TCP opens several TCP connections and strips the data file over multiple streams. A number of works [5]–[8] tried to find the optimal number of connections by tuning the parameters repeatedly according to historical data analysis and real-time probing or machine learning. However, both the overhead and accuracy of these technologies might be unacceptable under dynamically changing network environment. Furthermore, parallel TCP sockets introduce negative effect even if it runs with the optimal parameters.

III. Measurement Study

In this section, we examine the gap between high throughput wide-area transmission demands and loss-based TCP's inherent bottlenecks. We then show neither high-speed TCP variants nor application-level optimizations solve the problem.

A. Loss-based TCP Improperly Decreases Window

Despite multi-gigabit optical network offerings in the public Internet, most users fail to obtain even a fraction of the theoretical speeds promised by network operators. This raises a question: *Why TCP fails to fill up the pipe?*

Packet loss is ubiquitous. Figure 1 illustrates a 10-month measurement study of average packet loss of single-stream TCP on real paths through the Pantheon [22], a global-scale testbed contains measurement nodes on wired/cellular networks and in cloud datacenters across nine countries such as the UK, USA, Japan and Australia. We explore two TCP variants, TCP CUBIC (loss-based) and TCP BBR (non-loss-based), within Linux Kernel 4.10. The reason why we also investigate TCP BBR is to demonstrated that packet loss is ubiquitous in the wide-area scenarios, no matter whether the TCP variant is loss-based or non-loss-based. Specifically, the typical values of PLR are 0.5%, 1%, and 3% in the scenarios of cloud to cloud, end to cloud via Ethernet links, and end to cloud via cellular links, respectively.

Packet loss unnecessarily results in window drops. The state-of-the-art works rarely differentiate congestion and non-congestion loss explicitly. Take the loss-based congestion controller CUBIC as an example, it regards packet loss as a congestion signal. Considering other causes such as physical media errors, middlebox's drop policies, normal routing routines and packet drop attacks, decreasing the congestion window upon every packet loss has a deadly impact on TCP performance. The existence of these non-congestion factors is verified by our measurement study on the independence between these losses and the traffic generated by TCP connections in the Pantheon. Moreover, it is also a disputable question that backing off upon loss that is caused by short traffic burst in large-BDP networks.

A number of high-speed TCP variants that are not loss-based have improved throughput for wide-area transmissions. However, they suffer from new issues as discussed next.

TABLE I
Typical wide-area scenarios.

Scenarios	Cloud-light-loss (Cloud-L)	Cloud	Ethernet	Cellular	Cellular-heavy-loss (Cellular-H)
$bw(Mbps)$	1000	1000	100	10	10
$rtt(ms)$	300	300	30	256	256
$plr(\%)$	0.05	0.5	1	3	10

B. Recovery Capability Does not Match High-speed TCP

We define the *receive-buffer-bubble* (RBB) as the gap in sequence space between non-contiguous blocks of data that have been received and queued at the receiver. An RBB can be caused by packet loss or out-of-order delivery, and then it can be filled by the delayed packets or TCP's recovery mechanisms such as fast retransmission and timeout retransmission. The distribution of RBB varies significantly with network conditions. Thus, we specify the bandwidth, RTT and PLR in five typical wide-area scenarios as listed in Table I. For example in the *Cloud* scenario, the link capacity (bw) is 1 Gbps, RTT (rtt) is 300 ms and PLR (plr) is 0.5%.

High speed results in more RBBs. Figure 2 illustrates the real-time distribution of RBBs recorded in every RTT. The

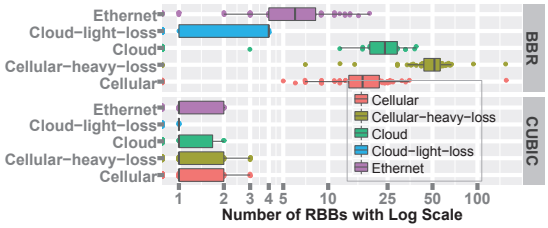
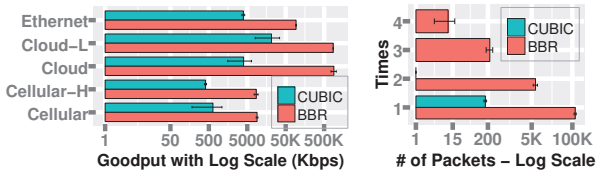


Fig. 2. Real-time distribution of RBBs in the end-host buffer of a TCP receiver under different scenarios listed in Table I. This figure plots the non-zero number of the RBBs in every RTT by means of both dot plot and box plot. Flow duration is 120 seconds, sending/receive buffer is large enough (256 MB) and LRO (Large Receive Offload) is disabled to ensure that packet size does not exceed the MTU (1500 Bytes).



(a) Average goodput (b) Retransmission times

Fig. 3. (a) Average goodput of flows under different scenarios listed in Table I. (b) Distribution of retransmission times, where $bw = 1$ Gbps, $rtt = 300$ ms, $plr = 3\%$ (Poisson pattern). Y-axis refers to the retransmission times and x-axis refers to the number of packets with log scale.

flow is measured with a duration of 120 seconds in the above five scenarios, and zero records are discarded. It is observed that the number of RBBs in TCP CUBIC does not exceed three in all scenarios, while other than in the *Cloud-light-loss* scenario, the number of RBBs in TCP BBR are persistently high (≥ 4) in the other scenarios ($plr \geq 0.5\%$).

Why BBR results in more RBBs than CUBIC? The answer is relevant with the higher throughput. As shown in Figure 3(a), TCP BBR is a high-speed TCP variant that does not regard loss as back-off signal directly, its throughput is much higher than that of loss-based CUBIC over large-BDP and lossy links. It can cause up to 150 RBBs in the receiver's queue in some worst cases. In general, the number of RBBs has its positive correlation with throughput, RTT and PLR. This issue is unremarkable for the loss-based TCP (*e.g.*, TCP CUBIC) when both throughput and number of RBBs are low. However, large number of RBBs reduces loss recovery efficiency if not carefully handled. We make an analysis as below.

TCP uses selective acknowledgement (SACK) [12] to notify the sender of at most 4 non-contiguous data blocks that have been received and queued, so the sender only need retransmit the actually lost packets. SACK option includes more than one SACK block (*i.e.*, complementary set of RBB) because the redundant blocks increase the robustness in the presence of lost ACKs. However, other TCP option deployments (*e.g.*, Timestamp option) may reduce the number of available blocks to 3, to 2 or even to 1. This raises the following concerns.

Traffic policers often cause burst losses. If all the successive ACKs reporting a specific SACK block are dropped, the sender might assume that the data in that block has not been received, and unnecessarily retransmit those packets. Large number of RBBs in the receiver's queue decrease the number of redundant notifications of a specific SACK block. For example, when the SACK option works with the Timestamp option, each SACK block will be repeated more than three times, large number of RBBs may decrease the number to exact three. Under the condition of burst loss pattern on the return path, the robustness of the SACK option decreases. Hence, spurious retransmissions become more serious.

As mentioned in §II, both TCP-TACK [13] and QUIC [14] allow ACKs to report more than 4 blocks. These advancements can significantly reduce spurious retransmissions. However, (1) both the sender-side and receiver-side have to be modified. Even without considering deployment issues, (2) this does not fundamentally solve the problem due to the large number of RBBs induced by inefficient loss recovery.

High speed wide-area transmission is buffer starving. TCP offers reliable and ordered byte-stream transmission. As a result, before being handed off to upper applications, the subsequent packets (stored temporarily in receiver's queue) of the lost packet will be stalled in the receive buffer until the RBB is filled via retransmissions. However, retransmissions might be lost again. We use k to denote the retransmission times of a specific packet. If a packet is never lost and retransmitted, its $k = 0$. Since the receive buffer required by a connection is closely related to the maximum times of retransmissions, we focus on the metric of k_{max}, the maximum value of k among all packets in a connection. Under a certain loss ratio and time period, we have Lemma 1 as below.

Lemma 1. *The higher throughput (Q), the larger the maximum times of retransmissions (k_{max}).*

Proof. To infer the relationship between k_{max} and Q, we model the problem as follows:

$$\max \quad (k) \tag{1}$$

$$\text{s.t.} \quad \rho_k = plr^k = \frac{n_k}{Q \cdot T} \tag{2}$$

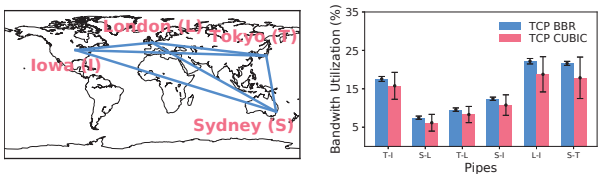
$$plr \in [0, 1) \tag{3}$$

$$n_k = 1, 2, 3, \ldots \; and \; k = 0, 1, 2, \ldots \tag{4}$$

where ρ_k is the probability for a specific packet being retransmitted k times, plr is the packet loss ratio, n_k is non-zero referring to the total number of packets being retransmitted k times and Q is the average throughput during time period T. Constraint (2) refers to the two ways of calculating ρ_k, where $Q \cdot T$ refers to the total number of packets that have been sent.

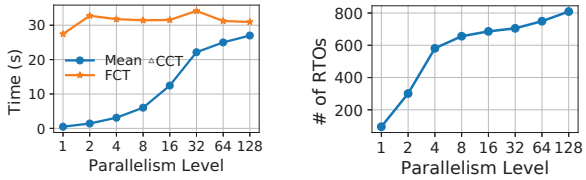
The problem turns into finding a maximum k to make n_k non-zero. Since the receive buffer required by a connection is closely related to the maximum times of retransmissions (*i.e.*, k_{max}) rather than the maximum number of packets being retransmitted k times (*i.e.*, n_k). Without loss of generality, we assume $n_k = 1$ for simplicity. Thus Equation (2) becomes $plr^{k_{max}} \propto \frac{1}{Q \cdot T}$, when $plr < 1$ and T is fixed, we infer that $max(k) \propto Q$, *i.e.*, the higher Q, the larger the k_{max}.

□



(a) Cloud-to-cloud topology (b) Bandwidth utilization

Fig. 4. Pipes are verified not full but TCP only achieves bandwidth utilization of less than 25%. Where $bw = 1$ Gbps, $rtt \in [100, 300]$ ms, and average plr is elaborated in the *Cloud* scenarios of Figure 1. The maximum receive buffer of TCP is 16 MB in the cloud end-hosts.



(a) FCT and ΔCCT (b) Number of RTOs

Fig. 5. Parallel TCP BBR for chunked file transmission, where $FileSize = 1$ GB, $ChunkSize = 1$ MB, $bw = 360$ Mbps, $rtt = 200$ ms, and $plr = 1\%$. (a) File completion time (FCT) and its mean chunk completion time (CCT) extra introduced by parallel TCP. (b) Average number of retransmissions caused by retransmission time out (RTO) during file transmission.

Figure 3(b) shows an example of the distribution of retransmission times. It is shown that $k_{max}^{BBR} = 4$ and $k_{max}^{CUBIC} = 2$, indicating that TCP BBR suffers more serious lost retransmissions than TCP CUBIC. Specifically, for TCP BBR, 132 "unlucky" packets have been retransmitted 3 times, indicating that the receive buffer should not be less than 3x BDPs ($\approx$ 107 MB) when TCP tries to fill these RBBs. Note that there exist 14 "unlucky" packets being retransmitted 4 times, this requires more than 143 MB receive buffer.

In general, upon a large k_{max}, the recovery latency is enlarged by the sum of retransmission delay of each RBB formed by "unlucky" packets in the sequence space. Moreover, the more times for a specific packet is retransmitted, the more receive buffer the connection occupies. Therefore, when running under large-BDP and lossy network conditions to optimize the TCP throughput, high-speed TCP might bottleneck its receive buffer which is usually small (*e.g.*, 16 MB in the *c5.xlarge* instances of Amazon EC2) in modern wide-area Linux servers. This bottleneck can be verified by our measurement study through the Pantheon. As shown in Figure 4, it is observed that buffer limitation makes TCP only achieve bandwidth utilization of less than 25% even when there is sufficient available bandwidth.

To mitigate the capability mismatch between loss recovery and throughput improvement of high-speed TCP, network operators usually raise the end-host buffer by TCP tuning. However, the issue is never solved in the case that operators only owning one side. For example, end-to-cloud services where the cloud provider is not allowed to modify the ends of all the device providers. For end-hosts, application usually turns out to be downlink traffic. Therefore, improving transmission efficiency at the sender side when the end-host's buffer is insufficient, would be a relevant contribution.

C. Application-Level Optimization Do Not Generalize

Apart from buffer size tuning, for sender-side approaches, some well-tuned application-level transmission parameters [5] such as pipelining, parallelism and concurrency are also proved to be effective for large dataset transmission. However, the optimal values of them vary significantly with the characteristics of datasets, network and end-hosts. Predicting the best combination of buffer size, pipelining, parallelism and concurrency is challenging as conditions are always changing throughout the transmission. Furthermore, even for one single parameter, incorrect configuration can either lead to congestions, under-bandwidth utilization, or unacceptable prediction overheads.

Take parallel TCP as an example. Parallelism [5], [23] is a popular method for overcoming the inadequacies of TCP under large-BDP and lossy network conditions. However, parallel TCP sockets may encounter several issues as discussed below.

Parallel TCP requires tuning repeatedly. Parallel TCP's benefit on slow links comes from being able to steal available bandwidth from the existing cross traffic. While under large-BDP and lossy network conditions, it gains performance by recovering faster than single TCP after loss-based congestion controller backs off. On the other hand, as the number of parallel sockets increases, too many simultaneous connections may lead to congestion. The congestion results in more packet loss which decreases the impact of parallelism, and the aggregate TCP bandwidth will stop increasing, or begin to decrease [5].

How many connections is sufficient to yield good throughput performance? The answer might be very hard to find. First, the network conditions such as RTT, PLR, bottleneck link capacity are always dynamically changing, and cross traffic is also unpredictable which impact the available bandwidth significantly. Second, the characteristics of datasets (*e.g.*, file size and number of files) and end-hosts (*e.g.*, file system and transmission protocol type) may also impact the optimal parallelism level. A number of works [5]–[8] tried to find the answer by tuning the parameters repeatedly according to historical data analysis and real-time probing or machine learning. However, both the overhead and accuracy of these technologies might be unacceptable under dynamically changing network environment. Furthermore, parallel TCP sockets introduce negative effect even if it runs with an optimal number of connections. We make an analysis as below.

Parallel TCP introduces negative effect. Parallel TCP uses multiple streams to transmit chunks of a file in parallel from a single-source to a single-destination. We argue that when a single stream can achieve the same throughput as the parallel TCP, parallel TCP may have negative effect due to a greater risk of tail loss. As shown in Figure 5(a), the bottleneck link capacity is set as the average throughput of a single TCP BBR stream. It is observed that the file completion time (FCT) of the single stream is slightly shorter than that of parallel TCP regardless of parallelism level. Then we explore the mean chunk completion time (CCT) extra introduced

by parallel TCP which can be calculated as $\overline{\Delta CCT} = \frac{\sum CCT_i}{FileSize/ChunkSize} - (rtt + \frac{ParallelLevel \times ChunkSize}{bw})$. It is observed that $\overline{\Delta CCT}$ increases significantly with the number of parallel TCP sockets (*i.e.*, $ParallelLevel$). Since tail loss usually causes time out, we further explore the average number of retransmissions caused by RTO, as shown in Figure 5(b). Each case with different parallelism level runs 100 times. We infer that the negative effect might be attributed to the increasing tail loss of each parallel stream.

IV. Design of OR3

As discussed above, high-speed transmission usually encounter a problem that a retransmitted packet is lost repeatedly under large-BDP and lossy network. In this paper, we propose an opportunistic random redundant retransmission (OR3) algorithm to eliminate the bottleneck of receive buffer during the loss recovery phases. OR3 follows three design principles, *i.e.*, *opportunism*, *redundancy*, and *randomization*, to minimize the maximum number of retransmissions (*i.e.*, k_{max}).

First, opportunism tackles the flow control bottleneck by allowing the sender to retransmit packets beyond the min{CWND, RWND}, where CWND is congestion window, and RWND is receive window (remaining receive buffer). The design rationale is that the receive buffer is the bottleneck of high-speed transmission in 80% of cases during loss recovery [24]. During the packet loss recovery phase, the sender is allowed to send more data than the limit of receive buffer. Since a large volume of receive buffer might be released once the opportunism retransmission succeeds, overbooking the receiving buffer in advance can improve bandwidth utilization.

Second, redundancy allows sending multiple replicas of the lost packets when a loss event is detected. OR3 applies two simple rules: (1) A packet is retransmitted with no replica if lost for the first time. (2) The more times that a packet have been lost, the more replicas are sent. For example, the i^{th} retransmission should generate $2i-1$ ($i = 1, 2, 3, ...$) replicas. The first rule is based on the fact that most of the lost packets can be successfully delivered after retransmitted once [14]. This avoids unnecessary redundancy overhead. The second rule assures that OR3 adaptively retransmits the lost packets as fast as possible.

Finally, a loss event usually contains multiple consecutive lost packets, the redundant replicas might be lost together again when burst losses occur. To improve the robustness under various packet loss pattern, randomization enables the redundant retransmitted packets to be sent out in a random number of sending cycles (one cycle equals to the interval of sending one packet at a specific pacing rate). This is useful especially in networks such as wireless networks and policy-based networks where there are burst losses.

It's worth mentioning that redundant retransmission might make the congested bottleneck more worse. Our observation, however, shows that this negative effect is negligible in the modern WAN with large-BDP links.

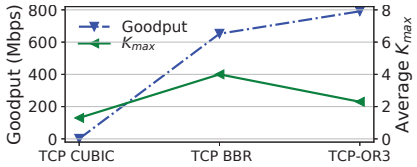
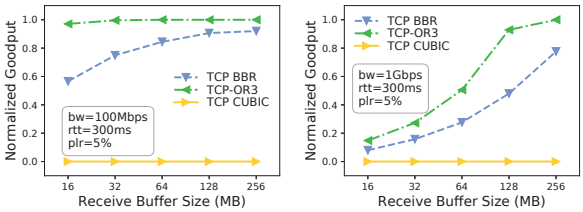


Fig. 6. k_{max}



(a) Link capacity = 100 Mbps (b) Link capacity = 1 Gbps

Fig. 7. Examples of receive buffer starvation.

V. Evaluation of OR3

A. Experiment Setup

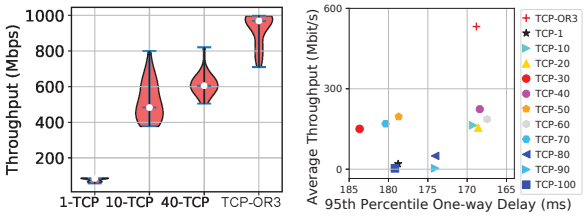
OR3 is a sender-side modification of TCP. We have implemented OR3 upon TCP, which is called TCP-OR3. TCP-OR3 only modifies the retransmission functions in the legacy TCP without changing the congestion controllers. To achieve high speed over lossy links, TCP-OR3 chooses the non-loss-based BBR [4] as its default congestion controller. We conduct contrast experiments between TCP-OR3 and other technologies in both testbed simulations and real-path measurements. If not otherwise specified, the Linux Kernel is 4.10 with DSACK, SACK and RACK (tcp_recovery) enabled, and the default TCP send/receive buffer of all schemes is set 16 MB.

For testbed simulations, we set up a testbed in which RTT, PLR and bandwidth are adjustable for performance estimation and behavior analysis under different parameter settings. The sender and receiver are connected by a Spirent Attero-X network emulator and a switch with 24 Ethernet 10/100/1000 ports, 4 x 10 Gbps NICs. Both end-hosts are with Intel Xeon CPU E5-2620, 16 GB memory, Gigabit Ethernet cards and a 10Gbps NIC. The emulator allows to set filters on both ingress and egress ports, enabling estimation on the effect of impairments to particular packets or particular types of traffic. The link between the two switches is set as bottleneck whose bottleneck buffer is 6 MB.

For real-path measurements, we integrate TCP-OR3 into the Pantheon [22] and run long-term tests in different workloads (single flow, or cross traffic). The Pantheon is a global-scale community evaluation platform for academic research on TCP variants. Measurement nodes are on wired networks and in cloud datacenters across countries. Links between endpoints are with varying bandwidth and latency, and are non-dedicated, *i.e.*, there exists wild cross traffic over the Internet.

B. Performance Results

OR3 minimizes the maximum times of lost retransmissions. In the case of buffer starvation, the utilization gains of TCP-OR3 over TCP are mainly stemming from its efficient loss



(a) Parallel TCP vs TCP-OR3 (b) Parallelism level = 1-100

Fig. 8. Parallel TCP vs TCP-OR3 in Amazon EC2.

recovery mechanism rather than its congestion control technology. To explain this clearly, as shown in Figure 6, we give an example of large-BDP and lossy wide-area data transmission, where $rtt = 300$ ms, $plr = 3\%$ in the Poisson distribution, and link capacity is 1 Gbps. We set a large enough receive buffer (512 MB) for all schemes to investigate the maximum times of retransmissions (*i.e.*, k_{max} defined in §III-B). The average k_{max} is obtained from 100 runs of each scheme. It is demonstrated that OR3 greatly increases the goodput (blue dashed line) by decreasing the k_{max} (green line).

OR3 alleviates receive buffer starvation. Figure 7(a) further shows the case of long-delay and lossy links with capacity of 100 Mbps. It is observed that TCP-OR3 still fills up the pipe when the end-host's buffer size is insufficient for TCP. This can be attributed to TCP-OR3's efficient loss recovery. When the link capacity increases to 1 Gbps, as shown in Figure 7(b), the bandwidth utilization of all schemes decreases significantly in the case of receive buffer starvation. Nevertheless, TCP-OR3 achieves a significant advantage over TCP in goodput.

OR3 tackles the bottleneck of parallel TCP. Figure 8 (a) and (b) illustrate the performance comparison between parallel TCP (with CUBIC as the default congestion controller) and single TCP-OR3. We run the tests through the Pantheon once per 20 minutes for one month on the link from AWS London to AWS Sydney. Each test lasts for 60 seconds. The link capacity is around 1 Gbps and RTT is around 300 ms. It is demonstrated that TCP-OR3 outperforms the parallel TCP in most of the cases. We further explore the performance of parallel TCP with different parallelism levels. Figure 8 reveals that TCP-OR3 enables a single stream to achieve even better performance than the optimal parallelism-level tuning over the Internet.

VI. Conclusion

This paper discusses the capability mismatch between loss recovery and throughput improvement of high-speed TCP. We therefore present OR3, as well as its TCP implementation TCP-OR3, so as to accelerate loss recovery by alleviating receive buffer starvation and tackling the bottleneck of parallel TCP. It is worth noting that the benefit of TCP-OR3 is on the basis of TCP's timely feedback from the receiver side and loss detection from the sender side. We also note that although OR3 is originally designed for high-throughput applications, it is also recommended to deploy OR3 for low-latency applications. This is because that applying OR3 accelerates tail loss recovery, which is one of the most challenging issue for tiny flow transmission in real-time applications.

VII. Acknowledgement

This work is partially supported by the National Natural Science Foundation of China (No.62162053No. 61762074, No. 62062059) and a grant (No.SKL-IOW-2020TC2004-01) from Tsinghua University.

References

[1] NERSC, "Application tuning to optimize international astronomy workflow from NERSC to LFI-DPC at INAF-OATs," https://fasterdata.es.net/data-transfer-tools/case-studies/nersc-astronomy/, 2017.

[2] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "Fast tcp:motivation, architecture, algorithms, performance," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246–1259, 2006.

[3] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound tcp approach for high-speed and long distance networks," in *Proceedings of IEEE INFOCOM*, 2006, pp. 1–12.

[4] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: congestion-based congestion control," *ACM Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[5] T. J. Hacker, B. D. Athey, and B. Noble, "The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network," in *Proceedings of IEEE IPDPS*, 2002, pp. 1–10.

[6] L. Jason, G. Dan, T. Brian, A. Bill, B. Joe, B. John, and T. Steve, "Applied techniques for high bandwidth data transfers across wide area networks," in *Proceedings of IEEE CHEP*, 2001, pp. 428–431.

[7] G. Hasegawa, T. Terai, T. Okamoto, and M. Murata, "Scalable socket buffer tuning for high-performance web servers," in *Proceedings of IEEE ICNP*, 2001, pp. 281–289.

[8] K. M. Choi, E. N. Huh, and H. Choo, "Efficient resource management scheme of tcp buffer tuned parallel stream to optimize system performance," in *Proceedings of EUC*, 2005, pp. 683–692.

[9] M. Mathis and J. Mahdavi, "Forward acknowledgement: refining tcp congestion control," *Acm Sigcomm Computer Communication Review*, vol. 26, no. 4, pp. 281–291, 1996.

[10] Y. Cheng and N. Cardwell, "Rack: a time-based fast loss detection algorithm for tcp," *Work in progress, IETF*, 2016.

[11] Y. Cheng, N. Cardwell, N. Dukkipati, and M. Mathis, "Tail loss probe (tlp): an algorithm for fast recovery of tail losses," *Work in progress, IETF*, 2013.

[12] M. M., M. J., F. S., and R. A., "RFC 2018: TCP selective acknowledgment options," *IETF*, 1996.

[13] T. Li, K. Zheng, K. Xu, R. A. Jadhav, T. Xiong, K. Winstein, and K. Tan, "Tack: Improving wireless transport performance by taming acknowledgments," in *ACM SIGCOMM*, 2020, pp. 15 – 30.

[14] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of ACM SIGCOMM*, 2017, pp. 183–196.

[15] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[16] D. Leith and R. Shorten, "H-tcp: Tcp for high-speed and long-distance networks," in *Proceedings of PFLDnet*, 2004, pp. 1–16.

[17] L. Xu, "Binary increase congestion control for fast long-distance networks," in *Proceedings of IEEE INFOCOM*, 2004, pp. 2514–2524.

[18] M. Mathis, N. Dukkipati, and Y. Cheng, "Rfc 6937: Proportional rate reduction for tcp," *IETF*, 2013.

[19] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "Tcp vegas: new techniques for congestion detection and avoidance," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 4, pp. 24–35, 1994.

[20] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "Pcc: re-architecting congestion control for consistent high performance." in *Proceedings of USENIX NSDI*, 2015, pp. 395–408.

[21] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *Proceedings of USENIX NSDI*, 2018, pp. 329–342.

[22] W. Keith and Y. Yan, "Pantheon of Congestion Control," http://pantheon.stanford.edu/, 2018.

[23] F. Zhang, J. Zhai, X. Shen, O. Mutlu, and X. Du, "Poclib: A high-performance framework for enabling near orthogonal processing on compression," *IEEE TPDS*, vol. 33, no. 2, pp. 459–475, 2022.

[24] L. Ke and J. Lee, "Improving tcp performance over mobile data networks with opportunistic retransmission," in *IEEE WCNC*, 2013.