Reorder Adapting TCP

Yi-Cheng Chan, Chien-Yu Chen, and Yu-Shuo Lee

Abstract Transmission Control Protocol (TCP) is the main transport protocol that provides reliable transmission in the current Internet. However, TCP still has problems that may affect its performance on various network environments. Packet reordering, which is one of the problems on TCP, refers to the out-of-order event of packets. There are many reasons that can cause packet reordering. In this chapter, we propose RA-TCP, a novel TCP variant which modifies the retransmission occasion and the congestion response of the traditional TCP mechanism, to improve the performance on packet reordering. In particular, we replace the three duplicate ACKs with a retransmission detection timer (RDT) for the trigger of fast retransmit. Our simulation studies reveal that RA-TCP performs consistently better than existing mechanisms that try to make TCP more robust to packet reordering.

Keywords TCP • Congestion control • Packet reordering • Out-of-order

1 Introduction

The congestion control of TCP adjusts the rate of data entering the network, keeping the data flow below a rate that would trigger collapse. It infers the network conditions between the sender and receiver depending on acknowledgments (ACK) and the timer. If a TCP sender detects a congestion event, it will limit the transmission rate of data entering the network by regulating the size of the congestion window (*cwnd*) and the number of unacknowledged packets allowed to be sent.

Y.-C. Chan (✉) • C.-Y. Chen • Y.-S. Lee
Department of Computer Science and Information Engineering, National Changhua University of Education, Changhua 50074, Taiwan
e-mail: ycchan@cc.ncue.edu.tw

J. Juang and Y.-C. Huang (eds.), *Intelligent Technologies and Engineering Systems*, Lecture Notes in Electrical Engineering 234, DOI 10.1007/978-1-4614-6747-2_6,

Several congestion control algorithms, including slow-start, congestion avoidance, fast retransmit, and fast recovery, have been implemented in TCP. A new TCP connection starts from slow-start phase. In slow-start, the value of *cwnd* is incremented by one whenever the sender receives an ACK, until it reaches the slow-start threshold (*ssthresh*). After *cwnd* reaches *ssthresh*, the phase of TCP sender turns into congestion avoidance.

As long as non-duplicate ACKs are received, the *cwnd* is additively increased by one every round-trip time (RTT). In the other side, the TCP sender will resend the lost packet and decrease transmission rate by setting *cwnd* to half after the arrival of three duplicate ACKs. The response of retransmission, triggering by received three duplicate ACKs, is called fast retransmit. In addition to fast retransmit, there is another method for detecting network congestion. TCP uses a retransmission timer to ensure data delivery in the absence of any feedback from the remote data receiver. The duration of this timer is referred to as RTO. When the retransmission timer expires, the TCP sender will set the *cwnd* to one and restart from slow-start phase.

A packet loss event can be detected through a TCP sender that receives three duplicate ACKs. However, packet reordering can easily trigger the three duplicate ACKs. Packet reordering refers to the network behavior where the relative order of some packets transmitted in the network in the same flow is altered when these packets are relayed in the network [1]. When packet reordering occurs, the TCP sender may mistake the out-of-order event and transmit the redundant data packets unnecessarily. Worse is that the spurious packet retransmissions keep the *cwnd* small and lead a poor TCP performance.

There are many situations that may cause packet reordering, like packet-level multipath routing, route fluttering, inherent parallelism in modern high-speed routers, router forwarding lulls, and IP fast reroute. In view of this, we propose the scheme called reorder adapting TCP (RA-TCP) to improve the TCP performance for packet reordering. RA-TCP changes the conditions which trigger the fast retransmit. Besides, according to the network situation, we use the cost function to dynamically adjust the parameters of RA-TCP.

In the following sections, we describe the design of RA-TCP and compare it with the other recent TCP versions like TCP-PR [2] and TCP-NCL [3] for packet reordering via the simulation of ns-2.

2 The Proposed Method

When the out-of-order event arises on a connection, the TCP receiver sends the duplicate ACKs. In tradition, the three consecutive duplicate ACKs are regarded as the signal of packet loss, and the TCP sender triggers the fast retransmit. Nevertheless, packet reordering may lead to the false fast retransmit. Thus the basic idea of RA-TCP is to cancel the transmission effect about duplicate ACKs.

In RA-TCP, the retransmission detection timer (RDT) is used to replace three duplicate ACKs for the trigger of fast retransmit. We use the parameter *mxrtt*, which refers to the possible maximum round-trip time, to set RDT. If the TCP sender waits for a corresponding ACK for the period longer than the *mxrtt*, it performs the process of fast retransmit, and we suppose the packet loss event has occurred. Each time the TCP sender receives an ACK, the *mxrtt* is updated. The equation of *mxrtt* is presented as follows:

$$mxrtt = \alpha \times srtt, \tag{1}$$

where α is a positive constant larger than 1 and *srtt* is the smoothed round-trip time estimated by sender. In traditional TCP, the *srtt* is one of the variables for computing TCP's retransmission timer [4]. When a subsequent RTT measurement R' is made, the *srtt* can be computed as follows:

$$srtt = (1 - \beta) \times srtt + \beta \times R', \tag{2}$$

where β is set to one eighth in our experiments, just the same as the description in [4]. The retransmission detection timer decides the timing of retransmission. Therefore, it is significant to set RDT's countdown period appropriately.

The parameter α denotes the proportion between *mxrtt* and *srtt*. In fact, it is impossible to choose a proper fixed value of α that is suitable for all networks. According to this, we design the mechanism that allows a TCP sender to dynamically adjust the α value in a connection. The adjustment depends on the cost estimations of transmission. We use the concept of opportunity cost to design the cost function.

To ease the cost estimation, we choose to compute the most probable cost in all cases. There are two cases for the cost estimation. First, we consider the cost of a false fast retransmit. The false fast retransmit means the retransmit is triggered for an undropped packet. In other words, the RDT's countdown period is shorter than it should be. The TCP sender will receive the corresponding ACK later. A false fast retransmit causes an unnecessary retransmission and a window reduction by half.

For detecting false fast retransmit, we add the DSACK [5] function into RA-TCP. In a connection, the TCP sender detects false fast retransmit when it received the DSACK acknowledgement which contains the information about the duplicate data packets. The TCP sender always stores the *cwnd* value before fast retransmit. When false fast retransmit is detected by DSACK, the TCP sender restores the *cwnd* to original value that was just before the occurrence of false fast retransmit.

As illustrated in Fig. 1, we suppose that a TCP connection has a current congestion window size w and a smoothed RTT that refers to R. We maintain two exponentially weighted moving average parameters, including the fast retransmit recover duration D and the maximum window size W. The former indicates the period from the inception of retransmission to the next time of retransmission. The latter represents the top value of *cwnd* in general cases. For acquiring the most

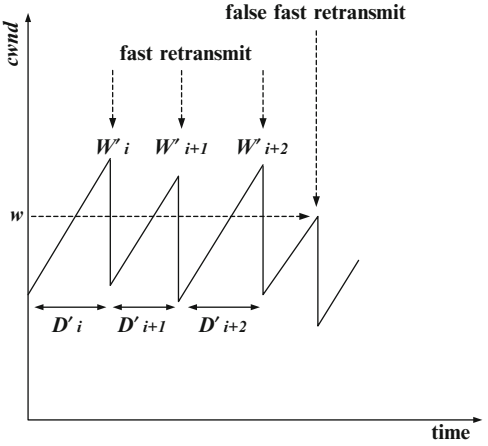

Fig. 1 A simple diagram for the variation of *cwnd* to indicate the false fast retransmit

probable estimate, we set the exponential moving average duration between the occasions of fast retransmit as D. In other words, D is the duration between two consecutive fast retransmits. Whenever a fast retransmit is triggered, we get the duration D' from the moment of last fast retransmit to current time, and the equation of D can be presented as follows:

$$D = (1 - \beta) \times D + \beta \times D'. \tag{3}$$

The W is the common max value of *cwnd*; the fast retransmit is usually triggered whenever *cwnd* is greater than or equal to W. Because the packets which the TCP sender may transmit are not more than W per RTT, we choose W to estimate the cost as large as possible. We set the length of *cwnd* as W' when the retransmission arises, and the equation of W is as follows:

$$W = (1 - \beta) \times W + \beta \times W'. \tag{4}$$

With these parameters, we can discuss the cost estimation. For example, if D equals to R, the window was halved unnecessarily for only one RTT. Based on the assumption that there is no false fast retransmit during the period of D and the congestion window size is linearly increased per RTT until the *cwnd* gets up to W, the W is the maximum value of *cwnd* commonly. Therefore, C_{ffr}, the cost of a false fast retransmit, is $W - \frac{w}{2}$. When D is longer than R, the cost is greater because the reduced window is in effect for a longer period. Note that the cost in each subsequent RTT becomes less and less as the linear increase of congestion window is progressing. The original congestion window value may be restored after $\left(W - \frac{w}{2}\right)$ RTTs. Thus, for $k = \lceil D/R \rceil$, the cost of a false fast retransmit is bounded by $\left(W - \frac{w}{2}\right) + \left(W - \frac{w}{2} - 1\right) + \cdots + \left(W - \frac{w}{2} - (k-1)\right)$, or

$$C_{\text{ffr}} \leq \sum_{i=1}^{k-1} W - \frac{i(2W - w - i + 1)}{2} \tag{5}$$

packets. Note that we limit k to $W - \frac{w}{2}$ regardless of D for disposing the cost appropriately. Because D and R are estimated as exponentially weighted moving averages, their values are not instantaneously accurate. The cost of a false fast retransmit may lie between the cost estimation for $k = \lceil D/R \rceil$ and $k = \lceil D/R \rceil$.

We also consider the cost of the delay after packet loss, C_{dpl}. The opportunity cost is introduced in idle time when the RDT's countdown period is great. In this case, the TCP sender keeps idle and does not execute the fast retransmit even though the corresponding packets are indeed dropped.

Suppose that a TCP connection has a current congestion window size w, a smoothed RTT that refers to R. Before w is halved, the TCP sender is idle because the RDT is not expired after packet loss. Compared with the normal case, the TCP sender could send less packets. In the following, we try to estimate the cost for this situation.

We maintain another parameter I to indicate the supposed idle period for this connection. Theoretically, the I is less than *mxrtt*. To compute the most probable cost, we set I to *maxrtt* in our experiments. Whenever the TCP sender receives an ACK after the non-spurious retransmission, we could estimate the cost that the TCP sender may lack for delivering during the waiting time. In addition, we also need to consider the influence of limited transmit [6].

In modern TCP, limited transmit is the mechanism which lets the sender to transmit a new packet when it received a dup-ACK before fast retransmission and time-out. To take account of limited transmit, we count the number of further duplicate ACKs that return per idle period, d, before the occurrence of the next fast retransmit. These duplicate ACKs are not part of the opportunity cost in the idle period. Hence, the cost of this idle period is

$$C_{\text{dpl}} \leq \frac{I}{R} W - d \tag{6}$$

packets. Here W is the exponentially weighted moving average of the maximum window size, and R is the smoothed RTT.

After the calculations of C_{ffr} and C_{dpl}, the adjustment of α can be performed. Above all, the S is the fundamental parameter; through it we adapt the value of α. In the results presented herein, we set S to 0.01 which is chosen to acquire appropriate adjustment of α from the numerous simulation experiments. The rule for adapting α is described as follows. As false fast retransmit occurs, RA-TCP increases α by S. This leads a longer *mxrtt*. However, the cost of delay after packet loss cannot be ignored because it also grows when the value of *mxrtt* is large. Consequently, we should consider the two opposite sides of the α adjustment. In view of this, we decrease α after an idle period that leads $C_{\text{dpl}} > C_{\text{ffr}}$. Thus, when every idle period is over and the result corresponds to the condition that C_{dpl} is large than C_{ffr}, RA-TCP decreases α by

$$\frac{C_{\text{dpl}}}{C_{\text{ffr}}} \times S. \tag{7}$$

These rules dynamically adapt α in a way that maximizes throughput for a connection experiencing reordering. False fast retransmits cause a gradual increase in the α. If the α oversteps its suitable bounds, RA-TCP also decreases α depending on S and the ratio of the two cost estimations.

3 Performance Evaluation

Due to the page limitation, in this section, we only present the ns-2 simulation results on a simple wired network as shown in Fig. 2. It involves two end-systems (S1 and S2) and two routers (R1 and R2). The path between R1 and R2 models the underlying network path connecting R1 and R2. A single, long-lived TCP flow from S1 to S2 runs for 300 s. A total of ten runs are done to compute an average value of the performance metric. We take the throughput of a flow to make comparisons between four TCP variants; those are TCP SACK, TCP-PR [2], TCP-NCL [3], and the proposed RA-TCP.

To simulate packet reordering, we repeatedly change the path delay which is between R1 and R2 according to a uniform distribution. The minimum delay value between R1 and R2 is 20 ms, and the maximum value is configured from 20 to 500 ms depending on the simulation experiments. A larger maximum value will induce more variation in the path delay, thereby increasing the degree of packet reordering.

The Fig. 3 shows the throughputs of the four TCP variants when the maximum path delay varies between 20 and 500 ms. The throughput of SACK drops obviously as the maximum path delay is increased. The other three schemes have a relative good performance because they use the timers to trigger fast retransmission instead of three duplicate ACKs. However, RA-TCP's throughput outperforms that of TCP-PR and TCP-NCL when the maximum path delay is larger than 300 ms.

4 Conclusions

In this chapter, we propose a novel TCP variant, RA-TCP, as an efficient solution for packet reordering. Just like TCP-PR and TCP-NCL, we replace the three duplicate ACKs with the retransmission detection timer for the trigger of fast retransmit.

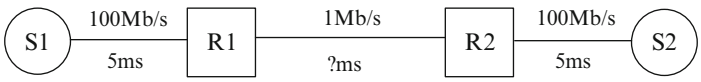

Fig. 2 A single three-hop wired network

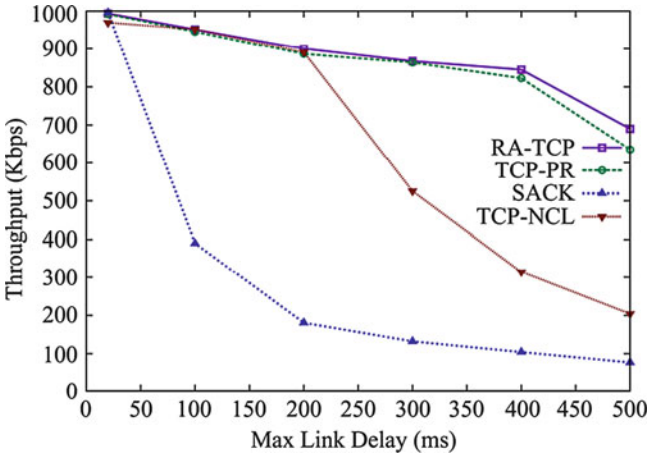

Fig. 3 Throughputs of the four TCP variants which are in different maximum path delays

For adapting the fluctuant network environment, we propose an adjustment algorithm, which refers to the cost functions of false fast retransmit and delay after packet loss, to adjust the timer fittingly. Our simulations show that RA-TCP has a significant performance improvement compared with that of SACK, TCP-PR, and TCP-NCL.

References

1. Leung K et al (2007) An overview of packet reordering in transmission control protocol (TCP): problems, solutions, and challenges. IEEE Trans Parallel Distrib Syst 18(4):522–535
2. Bohacek S, Hespanha JP, Lee J, Lim C, Obraczka K (2006) A new TCP for persistent packet reordering. IEEE/ACM Trans Netw 14(2):369–382
3. Lai C, Leung K-C, Li VOK (2010) Enhancing wireless TCP: a serialized-timer approach. IEEE INFOCOM 2010, pp 1–5
4. Paxson V, ACIRI, Allman M (2000) Computing TCP's retransmission timer, STD 7, IETF RFC 2988
5. Floyd S, Mahdavi J, Mathis M, Podolsky M (2000) An extension to the selective acknowledgement (SACK) option for TCP. IETF RFC 2883
6. Allman M, Balakrishnan H, Floyd S (2001) Enhancing TCP's loss recovery using limited transmit. IETF RFC 3042