Improvement of TCP Reno Congestion Control Protocol

Yueqiu JIANG, Junkun ZHANG, Qixue GUAN
Shenyang Ligong University, Shenyang, 110159, China
Tel.: 18905218128
E-mail: lchezjk@126.com

Received: 10 January 2014 /Accepted: 29 January 2014 /Published: 31 January 2014

Abstract: TCP Reno increases its congestion window exponentially during the phase of initial slow start. This leads a series of problems. For this reason, a piecewise mathematical function model which is symmetrical about midpoint is proposed. The growth rate of congestion window is accelerated at first and the growth rate of congestion window slows down at last during the phase of slow start so that it can smoothly transit to the phase of congestion avoidance. The algorithm also makes use of the bottleneck link based on the measurement method to calculate the network bandwidth so that the slow start threshold is set to the network bandwidth. Its congestion window, throughput, friendliness is assessed and validated by NS2 simulation. The new algorithm can effectively reduce the losses for network transmission of packets and decrease the number of network jitter.

Keywords: TCP Reno protocol, Slow start algorithm, Congestion, Function model, Threshold, NS2 simulation.

1. Introduction

Most Internet applications use the Transmission Control Protocol (TCP) for reliable, best effort transport [1]. TCP Reno is the core version of TCP protocols in Internet. It uses the slow start threshold (ssthresh) to distinguish the phase of slow start and the phase of congestion avoidance. It increases its congestion window (cwnd) exponentially in the phase of slow start and increases its cwnd with the linear growth method in the phase of congestion avoidance. Reno version appears after Tahoe version. Reno version adds fast retransmission algorithm and fast recovery algorithm in comparison with Tahoe version. Reno steps into the phase of congestion avoidance after the fast retransmit, while Tahoe steps into the phase of slow start. Reno version improves the bandwidth utilization rate. Fast retransmission algorithm uses "the message of conservation" that is properties of "the pipe model".

In recent years, the problems of congestion control have gradually caused the attention of people. For example, TCP Westwood algorithm proposes a way that sets the ssthresh by measuring the network bandwidth [2]. Scalable-TCP algorithm makes the recovery time after a congestion event independent of window size [3]. To control transmission, cwnd is subject to Additive Increase Multiplicative Decrease (AIMD) behavior [4]. Some scholars put forward a different timeout handling mechanism [5]. Some scholars put forward a way that optimizes the increase process of the phase of slow start by measuring and predicting the queue length [6]. An improved algorithm based on adaptive RTT is proposed. The key idea of this improved algorithm adjusts cwnd and ssthresh for packets losses according to RTT [7]. Some scholars propose an adaptive backoff algorithm which is based on
congestion avoidance over Binary Exponential Backoff [8]. Some scholars propose a congestion control mechanism based on accepting the threshold is put forward in this paper, which makes each DTN node dynamically adjust to its accepting threshold and accordingly to its congestion state [9]. Meanwhile, some cross-layer strategies are proposed. Some scholars propose a novel cross-layer congestion control strategy for WSN. With the help of the strategy, the transport layer of source nodes adjusts the congestion window and slow start threshold based on the received information of path bandwidth and delay and Explicit Congestion Notification (ECN), for mitigating network congestion [10]. A hop-by-hop cross-layer congestion control scheme (HCCC) built on contention-based MAC protocol is proposed. According to MAC-layer channel information including buffer occupancy ratio and congestion degree of local node, HCCC dynamically adjusts channel access priority in MAC layer and data transmission rate of the node to tackle the problem of congestion [11].

Long-lived connections and short-lived connections can usually be distinguished by the data’s transmission connection time of duration [12]. Research shows that data transmission on the Internet almost is short-lived connection (such as WEB), while long-lived connection that accounts for a small part transmits most data.

Short-lived connection often ends before it connects steadily. This suggests that the short-lived connection often only experiences the phase of slow start.

Therefore, the slow start algorithm directly influences the transmission performance of short-lived connection.

Meanwhile, the phase of slow start is often entered into when timeout occurs. As a result long-lived connection is also influenced by the phase of slow start. Therefore, improvement of slow start has important significance to congestion control.

In this paper which is on the basis of the TCP, Reno slow start algorithm proposes a new algorithm which is in the view of TCP Reno’s defects that TCP Reno loses a large number of packets in the phase of slow start in comparison with the algorithms that some scholars proposed previously. In this paper, a piecewise mathematical function model which is symmetrical about midpoint is proposed.

The value range of ssthresh that is suitable for new algorithm is obtained by mathematical derivation. The growth rate of speed of speed is accelerated at first of the phase of slow start and the growth rate of speed of cwnd slows down at last of the phase of slow start so that it can smoothly transit to the phase of congestion avoidance in this paper. At the same time, ssthresh is set by estimating link bandwidth. At this point, the algorithm is named Reno-J.

2. TCP Slow Start Algorithm

2.1. Four Algorithm of Reno TCP Congestion Control

As known in Fig. 1, Reno algorithm can be divided into four algorithms: Slow start, congestion avoidance, fast retransmit and fast recovery [13]. Beginning transmission into a network with unknown conditions requires TCP to slowly probe the network to determine the available capacity, in order to avoid congesting the network with an inappropriately large burst of data. The slow start algorithm is used for this purpose at the beginning of a transfer, or after repairing loss detected by the retransmission timer. When receiving each response acknowledgement (ACK) segment, one formula commonly used to update cwnd during slow start is given in equation (1):

\[
cwnd = cwnd + 1
\]  

The sender steps into congestion avoidance from slow start when cwnd is exceeds ssthresh. When receiving each response ACK segment, one formula commonly used to update cwnd during congestion avoidance is given in equation (2):

\[
cwnd = cwnd + \frac{1}{cwnd}
\]  

The TCP sender uses the "fast retransmit" algorithm to detect and repair loss, based on incoming duplicate ACKs (DUPACK). The fast retransmit algorithm uses the arrival of 3 duplicate ACKs as an indication that a segment has been lost. After receiving 3 duplicate ACKs, TCP performs a retransmission of what appears to be the missing segment, without waiting for the retransmission timer to expire (RTO). When the third duplicate ACK is received, set cwnd and ssthresh to the value given in equation (3):

\[
ssthresh = cwnd / 2, cwnd = ssthresh + 3
\]  

After the fast retransmit algorithm sends what appears to be the missing segment, the "fast recovery" algorithm governs the transmission of new data until a non-duplicate ACK arrives [14]. The sender steps into not slow start but congestion avoidance again. After the retransmission timer expires, the sender steps into the phase of slow start again. One formula commonly used to update cwnd is given in equation (4):

\[
ssthresh = cwnd / 2, cwnd = 1
\]
2.2. TCP Reno Slow Start Algorithm’s Defects

1) After slow start connection is established, the sender will set cwnd to one and send one segment. When the receiver sends an acknowledged ACK to the sender, the sender sets cwnd to two and sends two packets. When the receiver sends two acknowledged ACKs to the sender, the sender sets cwnd to four and sends four packets. This makes TCP double the packets that sender sends in each of the round trip time (RTT). The congestion window size approximates the capacity of the network when the network bandwidth is the maximum at last. If timeout occurs, ssthresh is set to half of cwnd and cwnd is reset to one. What’s more, the sender steps into slow start again and the bandwidth is wasted. Slow start is intended to avoid the congestion which is caused by the burst data produced at the time of connection is established. However, the sender sends a large number of packets with the cwnd increasing exponentially. In this way it can produce a large burst data flow. The number of packet loss is too large to recover when the congestion occurs. For example, it takes the sender nRTTs to change cwnd from 1 to $2^{n-1}$. But it takes the sender only 1RTT to change cwnd from $2^{n-1}$ to $2^n$. It can be seen that the later of slow start, the faster cwnd increases. More packets are sent to the network and the large burst data flow occurs so that the network can not deal with so many packets successfully. Timeout occurs and then congestion happens. This is the slow start defect.

2) The value of ssthresh is set to a fixed value of 20. The sender steps into the phase of congestion avoidance very early when the link can’t reach the actual bandwidth in this case [15]. Therefore the bandwidth utilization rate is relatively low and the packet can’t be transmitted quickly.

3. The Improved Slow Start Algorithm Reno-J

As mentioned above, TCP Reno slow start algorithm sets 1 packet size as a starting point, the sender increases the cwnd exponentially as equation (5) shows:

$$cwnd = 2^x$$

As known in equation (5), the slow start’s starting point is (0, 1) and the end point is ($\log_2(ssthresh)$, ssthresh). Reno-J algorithm sets $\{\log_2((ssthresh+1)/2), (ssthresh+1)/2\}$ of Reno slow start curve as piecewise and symmetric point. In the RTT interval $[0, \log_2((ssthresh+1)/2)]$, Reno-J still increases cwnd exponentially. And this can be seen in $f_1(x)$ of Fig. 2. As shown in Fig. 2, $f_2(x)$ is made in the way that $f_2(x)$ is symmetrical about $f_1(x)$ in the RTT interval $[\log_2((ssthresh+1)/2), \log_2((ssthresh+1)^2/4)]$.

The cwnd of $f_1(x)$ is proportionate to the growth rate of speed of cwnd and $f_1(x)$ is concave function when cwnd doesn’t exceed (ssthresh+1)/2.
The cwnd of $f_2(x)$ and the growth rate of speed of cwnd are inverse ratio and $f_2(x)$ is convex function when cwnd exceeds $(ssthresh+1)/2$. $f_2(x)$ can be obtained from the mathematical properties. The specific method is shown as follows: point $(a, b)$ is set as symmetric point and $f_1(x)$ and $f_2(x)$ is symmetrical about $(a, b)$. Thus equation (6) and (7) can be derived:

$$f_1(2a-x) + f_2(x) = 2b$$  \hspace{1cm} (6)

$$2^{2\log_2((ssthresh+1)/2)} + f_2(x) = ssthresh + 1$$  \hspace{1cm} (7)

Equation (8) can be derived by equation (6) and (7):

$$f_2(x) = ssthresh + 1 - \left\{(ssthresh + 1)^2 / 4\right\} \times 2^{-x}$$  \hspace{1cm} (8)

Therefore, mathematical expression of Reno-J can be shown in equation (9):

$$cwnd = \begin{cases} 2^x, x \in [0, \log_2\{(ssthresh + 1)/2\}] \\ ssthresh + 1 - \left\{(ssthresh + 1)^2 / 4\right\} \times 2^x, x \in (\log_2\{(ssthresh + 1)/2\}, \log_2\{(ssthresh + 1)^2 / 4\}] \end{cases}$$  \hspace{1cm} (9)

The Reno-J algorithm is discussed as follows:

1) The new algorithm’s main idea is the idea that the phase of slow start is divided into two phases. The one is the phase of fast increase that the sender increases cwnd very quickly in order to improve the bandwidth utilization rate. The new algorithm smoothly approaches ssthresh and slows down the growth rate of speed of cwnd in second phase.

2) As known in Fig. 2, the new algorithm’s slow start’s transmission time is longer than the original algorithm. The slow start’s transmission time can be shortened with the larger initial value of cwnd (IW). Therefore short-lived connections that transmit a small part of data benefit from the larger IW. And low bandwidth network benefits from the larger IW too. The IW is proposed to change from 1 MSS to 4 MSS [16]. For a lot of Emails (SMTP) and webpage (HTTP) which are less than 4Kb, their transmission time can be reduced by 1RTT with larger IW [17].

4. The Measurement Based on Bottleneck Bandwidth

Abstract-Available bandwidth is of great importance to network Quality of Service assurance, network load balancing, streaming media rate control, routing, and congestion control [18]. At present, some scholars propose the available bandwidth estimation strategy based on the Network Allocation Vector for Wireless Sensor Networks. According to the size of the average contention window, network nodes predict the probability of collision in process of frame transmission, and then estimate the number of retransmission [19]. $B_{eq}$ is proposed in order to effectively describe the link condition [20]. As shown in Fig. 3, R1, R2,...,Rn are the intermediate routers and B1, B2,...,Bn+1 are every link’s bandwidths. Sender and receiver connect directly by a link which bandwidth is $B_{eq}$ so that sender and receiver like a “pipe”. Parameter $B_{eq}$ can be estimated by the method as follows: sender continuously sends two TCP packets P1, P2 whose sizes are L1 and L2 to Receiver. The condition that the network congestion is consistent can be assumed. The end-to-end delays are $Delay_{p1}$ and $Delay_{p2}$ respectively. Acknowledgement ACKs are $ACK_{1}$ and $ACK_{2}$ respectively and delays are $ACK_{1 Delay}$ and $ACK_{2 Delay}$ respectively. So equation (10) and (11) are defined as follows:

$$RTT_1 = Delay_{p1} + Delay_{ACK_1}$$  \hspace{1cm} (10)

$$RTT_2 = Delay_{p2} + Delay_{ACK_2}$$  \hspace{1cm} (11)

Since the response ACKs without data almost have the same size and they are much smaller than TCP packets. $Delay_{ACK_1} \approx Delay_{ACK_2}$. So equation (12) is defined as follows:

$$RTT_1 - RTT_2 = (Delay_{p1} - Delay_{p2}) + (Delay_{ACK_1} - Delay_{ACK_2})$$

$$= (L1 - L2) \times \left( \sum_{j \in R_i, R \in A} B_j + \sum_{j \in R_j, R \in B} B_j \right)$$

$$= (L1 - L2) \times \frac{1}{B_{eq}}$$  \hspace{1cm} (12)

Equation (13) can be derived by equation (12):

$$B_{eq} = 1/\left( \sum_{i \in A, B} \frac{1}{B_i} + \sum_{j \in B_j, R \in B} \frac{1}{B_j} \right)$$  \hspace{1cm} (13)

$R_i$ belongs to $A$ that are collections of routers which don’t have the bandwidth allocation function. $B_i$ is the actual bandwidth that $R_i$ distributes to the TCP connection on the downstream link. $R_j$ belongs to $B$ that are collections of routers which have the bandwidth allocation function. $B_j$ is the actual bandwidth that $R_j$ distributes to the TCP connection on the downstream link.

As known in equation (13), $B_{eq}$ is equal to reciprocal of summation of every bandwidth’s reciprocal. $ssthresh$ can be set as $\frac{B_{eq} \times BaseRTT}{MSS}$, so the ssthresh is set more accurately and the bandwidth utilization is improved. BaseRTT is the minimum
RTT for the segment which is 1 MSS passes through the TCP link in Fig. 3.

Fig. 3. TCP link.

5. Algorithm Validation

1) The follows can be seen by Fig. 4 and Fig. 5:
   \[ a = \log_2((ssthresh + 4)/2) - 2 = \log_2((ssthresh + 4)/8) \]
   \[ 2a = \log_2((ssthresh + 4)^2)/64 \]
   \[ b = \log_2(ssthresh) \]
   If the new algorithm wants to shorten the transmission time, it can set \( b > 2a \), so inequality (14) can be derived:
   \[ \log_2((ssthresh + 4)^2)/64 < \log_2(ssthresh) \]
   (14)

   Inequality (15) can be derived by inequality (14):
   \[ 0 < ssthresh < 56 \]
   (15)

2) The total segments that Reno algorithm sends when cwnd reaches the ssthresh in the phase of slow start are \( T1 \):

   Since the new algorithm is symmetrical about the point \( (\log_2((ssthresh + 4)/8), (ssthresh+4)/2) \). So S1 is symmetric to S2 and their areas are equal. The total segments that Reno-J algorithm sends when cwnd reaches the ssthresh in the phase of slow start are equivalent to the shadow trapezoid area \( T2 \) in Fig. 5. Equation (18) can be derived:
   \[ T2 = \{ (4 + ssthresh)/2 \} \times \{ \log_2((ssthresh + 4)/2) - 2 \} \times 2 \]
   (18)

Equation (17) is derived by putting \( b \) into \( T1 \):
   \[ T1 = 2ssthresh - 1 \]
   (17)

   If the throughput of the new algorithm is wanted to be better than TCP Reno algorithm’s, \( T1 \) is smaller than \( T2 \). Inequality (19) can be derived by equation (17) and (18):
   \[ ssthresh \geq 21 \]
   (19)

3) As known by inequality (15) and (19), the new algorithm is better than Reno in slow start’s transmission and throughput when ssthresh is in the interval \([21, 56)\). What’s more, the new algorithm can step smoothly from the phase of slow start into the phase of congestion avoidance.

6. The Simulation Results and Analysis

In this paper, the new algorithm is simulated and verified by NS2 that is developed by UC Berkeley LNBL research team. This paper adopts the topology 1 in the Fig. 6 to simulate in order to compare cwnd and throughput of two algorithms. The topology sets \( s0 \) as sender and \( r3 \) as receiver. The link bandwidths from \( s0 \) to router \( r1 \) and from router \( r2 \) to \( r3 \) are both set to 10 mb/s. The bottleneck link bandwidths between \( r1 \) and \( r2 \) are set 1mb and 1.5mb respectively to compare. BaseRTTs measure 23.957 ms and 26.012 ms respectively. \( B_{eq} \) can be measured by equation (13). When the bottleneck link bandwidth is 1 MB, \( B_{eq} = \frac{1}{10} + \frac{1}{10} + \frac{1}{1.2} = \frac{1}{12} \).
ssthresh=\frac{1}{1.2}\times23.957=20.\text{ When the bottleneck link bandwidth is 1 MB, } B_{\text{eq}} = \frac{1}{10} + \frac{1}{10} + \frac{1}{1.5} = \frac{30}{26},\text{ ssthresh}=\frac{30}{26}\times26.012=30.\text{ Router cache is a size of 27 packets and every packet is 1024. The sender sends 1 MB/s FTP one-way flow of data continuously. Routing queue management uses the Drop - Tail strategy.}

**Fig. 6. The topology 1.**

**6.1. Congestion Window**

It can be seen from Fig. 7 that whether the bottleneck link bandwidth is 1 MB or 1.5 MB, Reno slow start algorithm’s cwnd all reaches ssthresh which is about 20.

**Fig. 7. The comparison of Reno algorithm’s cwnd when bottleneck link bandwidths are 1 Mb and 1.5 Mb respectively.**

However, cwnd doesn’t increase with the increase of bottleneck link bandwidth. Because of the ssthresh is fixed to 20 packet size, the sender steps from slow start into congestion avoidance prematurely and reduces the transmission rate of data segments. What’s more, the sender steps from slow start into congestion avoidance unnaturally. Therefore, under the condition of the bandwidth growth, the Reno algorithm’s network utilization is low. It can be seen from Fig. 8 that Reno-J algorithm whose bottleneck link bandwidth is 1.5 MB makes cwnd comparatively reach about 30. So Reno-J algorithm can adapt to the change of the network environment. The new algorithm can appropriately adjust the growth rate of speed of cwnd according to the size of the bandwidth delay product so that the network is fully and appropriately used. What’s more, the improved slow start algorithm, can smoothly step into the phase of congestion avoidance. Thus the new algorithm relatively reduces the incidence of burst data and avoids the unnecessary packet discarding. As shown in Fig. 8, Reno-J slow start function curve is symmetrical about cwnd’s midpoint. Since Reno-J only changes TCP Reno’s slow start algorithm. It still uses the fast retransmit and fast recovery. Therefore it has the same properties with TCP Reno after slow start.

**Fig. 8. The comparison of Reno-J algorithm’s cwnd when bottleneck link bandwidths are 1 Mb and 1.5 Mb respectively.**

**6.2. Comparison of Throughput**

Reno and Newreno are the core versions of congestion control algorithms. Fig. 9 is a comparison of throughput between Reno-J and TCP Reno. Fig. 10 is a comparison of throughput between Reno-J and Newreno. As shown in Fig. 9 and Fig. 10, Reno-J algorithm reduces to send the burst data to network and reduces the number of timeout. Reno-J takes the situation that many packets lose in a range of a window into account. Adopting the new algorithm, the throughput of Reno-J is better than Reno’s and Newreno’s. Reno-J’s first stable throughput can reach 1418 Kbps. The average throughput of Newreno is 1282.3 Kbps and the average throughput of Reno is 1238.9 Kbps. The average throughput of Reno-J is 1415.8 Kbps. The number of throughput’s jitter of Reno-J is less than Reno’s. The network transmission rate of Reno-J is stability. The utilization rate of network bandwidth is improved with Reno-J.
6.3. Friendliness

This paper adopts the topology in the Fig. 11 to simulate in order to verify the friendliness between Reno and Reno-J. The topology sets s2, s4 as sender and r3, r5 as receiver. The link bandwidths from s2 to router r0 and from router r1 to r3 are set to 15 mb. The link bandwidths from s4 to router r0 and from router r1 to r5 are set to 15 mb. Their transmission delays are 1 ms. The bottleneck link bandwidth between r1 and r2 is set 1.5 mb. The value of ssthresh is set 30. Router cache is size of 27 packets and every packet is 1024. The sender sends 1 MB/s FTP one-way flow of data continuously. Routing queue management uses the Drop-Tail strategy.

Friendliness refers to the condition that the two protocols, on average, share the network bandwidth instead of competing for network bandwidth. Thus their throughputs are roughly consistent under the same conditions. Fig. 12 is the comparison of throughput between Reno and Reno-J. Obviously, the Reno-J is friendly to Reno. According to Fig. 12, Reno-J doesn’t make the performance of TCP Reno decreased obviously.

7. Conclusion

1) This paper aims at the defects of Reno’s slow start that the sender sends much burst data and loses many packets in the late of the phase of slow start. The Reno-J algorithm is proposed in this paper. The new algorithm is better than Reno in slow start’s transmission time and throughput when ssthresh is in the interval [21, 56) through deriving by the mathematical operations. Reno – algorithm which sets the ssthresh value based on link bandwidth is more adapt to the network situation. The simulation results prove that the Reno-J’s slow start mechanism can effectively reduce the network transmission of packet discarding, reduce the burst data, and maintain the stability of network. The new algorithm reduces the risks of overflowing of router caches. The congestion window, throughput and friendliness all achieve the desired effects.
2) The algorithm that only needs to modify TCP sender port’s algorithm and doesn’t need to make changes to bottlenecks in the router can achieve improvements of end-to-end TCP performance. Reno and Reno-J can coexist friendly, so Reno-J can be applied to reality.

Acknowledgment

We would like to express our gratitude thanks to the Communication and Network Institute of Shenyang Ligong University for their invaluable suggestions to improve this paper. This work is supported by Liaoning Province Colleges and Universities Excellent Talents Support Program (Grant No. LR201034), CALT Innovation Universities Excellent Talents Support Program (Grant No. 20130423), and also supported by the Liaoning Province Educational Office Foundation of China (Grant No. L2011038).

References