2015 The 5th International Workshop on Computer Science and Engineering
doi: 10.18178/wcse.2015.04.108

An Improved Slow-start and Congestion Avoidance Algorithm based on TCP Westwood

Hong Jie[1,a], Rui-Qing Wu[1,b], Nan Ding[1,c]

[1]School of Electronic Engineering, University of Electronic Science and Technology of China, Chengdu, 611731, China

[a]jieqionghong@126.com, [b]dingnan11@163.com, [c]rqwu@uestc.edu.cn

Keywords: slow-start, congestion avoidance, TCPW, bandwidth ratio.

Abstract. TCP Westwood (TCPW) utilizes the estimated link available bandwidth by measuring and averaging the rate of returning ACKs to appropriately set the slow start threshold (ssthresh) parameters, and performs well in wired/wireless networks. However, TCPW still obeys the rules of exponential and linear increment of the traditional TCP Reno and cannot adaptively adjust the congestion window (cwnd) depending on the network status during the slow-start and congestion avoidance phase, leading to more packets losses, frequent retransmissions and lower bandwidth utilization. In view of the above problems, this paper presents an improved algorithm TCPW RB, the algorithm in the slow-start and congestion avoidance phase, respectively, utilizing the buffer queue length and the bandwidth ratio factor to adaptively adjust the increments of cwnd. Simulation results show that the proposed algorithm can effectively reduce the packet losses, improve the bandwidth utilization and throughput with good fairness and friendliness.

Introduction

Along with the diversity of network structure and the complexity of network environment, the application of the conventional Transport Control Protocol (TCP) is restrained to a certain extent and TCP performance will degrade. There are three approaches to address to implement the congestion control: End-to-End (E2E) [1], Cross-layer design [2], Split-connection mechanism [3]. E2E only needs to be modified in TCP source side, not requiring the participation of intermediate nodes, and gets widespread attention with simple and easy to configure. E2E can be divided into methods of three categories: the one mechanism is based on the packet loss (New Reno, Sack) [4], the second is based on propagation time delay (TCP Vegas, Veno) [5], and the last is based on the mixed design (TCPW) [6].

TCPW uses the estimated bandwidth by monitoring the rate of returning ACKs to compute cwnd and ssthresh after a packet loss indication (three duplicate acknowledgments received or a coarse timeout expiration), to a certain extent, which improves bandwidth utilization and lowers the packet loss. However, there are some shortages for TCPW. (i) TCPW still executes the blindly exponential growth mode during slow-start phase which may generate a large number of burst data deepening the degree of network congestion. (ii) In congestion avoidance phase, TCPW inherits the linear growth of Reno which increases the frequency of network congestion at the statue closer to the network congestion and is not conducive to the maintenance of high bandwidth.

Some studies are carried out to improve TCPW [7-9]. Literature 7 proposes estimation network capacity based on RTT to carry on the adjustment to the slow start threshold. Due to the measurement of complexity and inaccuracy of RTT, the result is not effective. Literature 8 proposes to divide the congestion avoidance phase using the fixed ratio, but the fixed ratio can not correctly reflect the network status, resulting large fluctuations.

In view of the above problems, this paper presents an improved algorithm of TCPW RB, which adjusts the growth speed of cwnd according to the estimated queue length during the slow-start phase so as to avoid more packet losses and retransmission. In addition, in congestion avoidance stage, we adaptively adjust the increments of cwnd based on the bandwidth ratio factor which

indirectly reflects network status in real-time, decreasing frequent congestion, maintaining an appropriate transmission window, improving the network utilization rate.

This paper is organized as follows. Section 2 provides a detailed description of TCPW algorithm. The improved TCPW RB mechanism depicts on section 3. The results of performance are shown in section 4. Finally, Section 5 concludes the paper.

Overview of TCPW algorithm

TCPW is a modified version of TCP Reno and specially designed for wireless networks to solve the wireless packet loss. The main idea of TCPW is that the sender continuously computes the connection bandwidth estimation (BWE) which is defined as the share bottleneck used by the connection depending on the value of ACK time interval received at the sender side divided by the amount of data confirmed within the interval. Besides it adopts the adaptive bandwidth share estimation filtering mechanism for bandwidth samples to make the available bandwidth more accurate. When the packet loss occurs due to the random error or congestion, the sender resets the *sshtresh* and *cwnd* parameter based on the estimated bandwidth so as to derive the best value. The packet loss is suspected with a reception of three duplicates ACKs or timeout expiration. The pseudocode of TCPW algorithm is the following:

When (3 dupacks are received)
If (*cwnd*>*ssthresh*)
/*congestion avoidance phase*/
ssthresh=(*bwe***RTTmin*)/*PacketSize*;
cwnd=*ssthresh*;
If (*cwnd*<*ssthresh*)
/*slow-start phase*/
ssthresh=(*bwe***RTTmin*)/*PacketSize*;
If (*cwnd*>*ssthresh*)
cwnd=*ssthresh*;
When (a new ACK is received)
/*the traditional slow-start algorithm*/
If (*cwnd*<*ssthresh*) *cwnd*=*cwnd*++;
/*the traditional congestion avoidance algorithm*/
If (*cwnd*>*ssthresh*) *cwnd*=*cwnd* +1/*cwnd*;

where *bwe* denotes the estimated available bandwidth; RTT_{min} represents the minimum round-trip delay (*RTT*) observed over the duration of the connection; *PacketSize* indicates the length of TCP segments;

TCPW which accurately estimates the available bandwidth to reset the value of *ssthresh* makes it possible to make full use of network resources. However, TCPW in the latter part of slow-start phase will send too much data packets overrunning the capacity the network at once due to exponential growth pattern of *cwnd*. When the large amount of data flows are injected into the network at once, network could not successfully one-time deal with so many data packets and would cause transmission timeout, then increase the possibility of the network congestion. Besides, TCPW in congestion avoidance phase still adopts the traditional way of linear growth where *cwnd* will be increased by *1/cwnd* every RTT. Due to the state close to the network congestion, the mechanism of blindly linear increase is easy to cause the network congestion again quickly, and makes the frequency of network congestion, is not conducive to the stable high bandwidth, and lowers the network utilization.

Improved TCPW RB algorithm

Aimed at the problem that the sudden data flow caused by data fast growth at the latter part of slow-start stage aggravates network congestion, TCPW RB utilizes the estimated bottleneck buffer queue

length to determine the network status, then adjusts or slows down the growth rate of *cwnd*. During the congestion avoidance phase, TCPW RB dynamically adjusts *cwnd* increase based on bandwidth ratio factor. Handling duplicate ACK and timeout keep TCPW algorithm unchanged.

Enhanced slow-start mechanism. The classical Vegas [5] algorithm adjusts the size of *cwnd* according to the difference between the expected transmission rate and the actual transmission rate value. We learn from the idea, to obtain the expected sending rate by *cwnd/RTTmin*, then use the *current_bwe* estimated by TCPW RB to indicate the actual sending rate. The reason is that the estimated bandwidth which is equal to the rate at which data is delivered to the TCP receiver, reflects the currently successful transfer rate of the connection, and is more close to the actual state of the network. The buffer queue length estimation is shown in Eq.1:

$$diff=(cwnd/RTT_{min}-current\_bwe)RTT_{min}. \quad (1)$$

If the value of *diff* exceeds a certain degree, TCPW RB decreases the growth speed of cwnd in the latter part of slow-start phase so as to avoid the sudden flow resulting in more packet losses and retransmission. Otherwise, TCPWRB still maintains the exponential increase mode with increasing cwnd by 1 MSS (the maximum segment size) for every ACK received at the sender side.

Enhanced slow-start algorithm is as follows:

Step1: When a connection is established, the size of *cwnd* is initialized to 1MSS and the value of ack_flag is set to 1.

Step2: When a new ACK packet is received at the sender side, TCP sender estimates the *BWE*, that is *current_bwe*. Then we calculate the *diff* by expression (1).

Step3: TCP sender judges the network statue depending on the buffer queue length to adjust the increase pattern of *cwnd*.

i) If $diff<\beta$, TCP sender increases *cwnd* by 1MSS for every ACK received;

ii)If $diff>=\beta$and *ack_flag*==1, TCP sender increases *cwnd* by 1 MSS and sets the *ack_flag* to 0. If $diff>=\beta$but ack_flag==0, TCP sender keeps the *cwnd* unchanged and sets the *ack_flag* to 1. In other word, TCP sender increases the *cwnd* for every other ACK to slow down the growth speed.

where the β is a constant, experience value is 3. The *ack_flag* is a flag bit. We use *diff* to determine the network status, when $diff<\beta$means that the bottleneck queue groups are less and the network is not fully utilized. Therefore, the slow-start retains the original exponential growth. If not, it means that the network is fully utilized, with each of the two RTT, TCP sender increases *cwnd* by 1 MSS to slow down the growth rate of the *cwnd* during the latter part of the slow-start phase.

Improved congestion avoidance mechanism. Network congestion is a sustained overload the network state, and will cause the phenomenon of packet loss, latency, throughput decrease. Based on the above factors, the estimated bandwidth in the state of congestion in the network is much smaller than the normal circumstances. On the other hand, the packets losses due to the high error rate and other reasons for wireless networks is with a chance, which does not affect the RTT, the estimated bandwidth compared to the normal condition will not lead to big changes. Therefore TCPW RB adaptively adjusts the growth of the size of *cwnd* by calculating the bandwidth ratio factor during the congestion avoidance phase so as to keep the network running pretty close to its rated capacity. The bandwidth ratio factor calculated by Eq.2:

$$\delta = \frac{bwe - bwe_{avg}}{bwe_{\max} - bwe_{avg}} \quad (2)$$

$$bwe_{avg} = \frac{1}{n}\sum_{i=1}^{n} bwe_i \quad (3)$$

$$cwnd=cwnd+1/cwnd(\delta+1) \quad (4)$$

where *bwe* denotes the currently estimated bandwidth, bwe_{max} indicates the maximum bandwidth calculated by $bwe_{max}=max(bwe_i)(i=1,2,3...n)$ which means that the link is made full use of. bwe_{avg} is the value of the average bandwidth calculated by Eq.3 which means that the network is stable. TCPW RB persistently computes *bwe*, bwe_{max} and bwe_{avg}, utilizes the difference between the

estimated currently bandwidth and the average bandwidth in the range of ratio to compute the bandwidth ratio factor to real-time adjust the increase of *cwnd*. The adjustment method is shown by the above Eq.4.

Improved congestion avoidance algorithm is as follows:

Step1: If *cwnd>ssthresh*, the slow-start phase is over and the congestion avoidance phase starts. TCP sender estimates respectively the value of *bwe*, bwe_{max} , bwe_{avg} for each received ACK by Eq.3;

Step2: TCP sender calculators the bandwidth ratio factorδ by Eq.2;

Setp3: TCP sender acquires the current network status depending on the value of δand adjusts the size of *cwnd* by Eq.4;

Setp4: When a timeout occurs or 3 dupacks are received, TCP sender estimates the *bwe* to reset the value of *ssthresh*. Then the next procedure goes to the **Step1** again.

From the Eq.2, we can see that the bandwidth ratio factor $\delta \in [-1,1]$ shows the level of the currently estimated bandwidth compared with historical changes, and indirectly reflects the state of the network. When δ near 1 indicates that the current network is in a better state, *cwnd* can be continually increased by *2/cwnd*. Besides when δ approximately 0 notes that the network state is in the appropriate state, which retains *cwnd* by *1/cwnd*. However, when δ near -1 indicates that the network is in relative congestion state, *cwnd* should be set appropriately.

Simulation and Analysis

In this section, we simulate and evaluate the performance of improved TCPW RB algorithm by NS2, compared with TCPW, Reno and Vegas based on these evaluation criteria: throughput, packet losses, fairness, and friendliness.

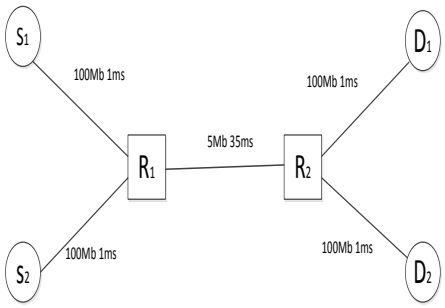

Fig.1: Network topology

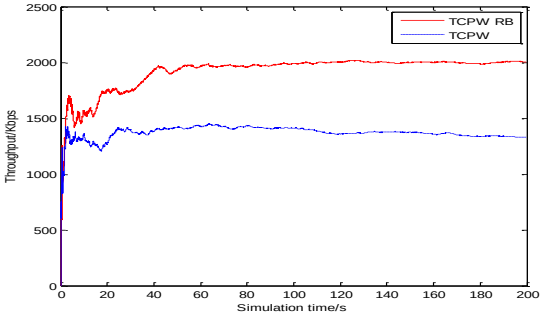

Fig.2: Throughput with 3% error rate

Fig.1 shows the network topology with a conventional dumbbell structure used for the simulation. TCP senders S_1 to S_2 are connected to the router R_1 and TCP receivers D_1 to D_2 are connected to the router R_2 respectively via a 100Mbps link with 1ms propagation delay. The link between R_1 and R_2 is the bottleneck link with 5Mbps bandwidth and 35ms propagation delay. The size of TCP packet is set to 1400 bytes and the buffer capacity is set to bandwidth delay product.

Throughput. We establish two TCP connections from the sender S to the receiver D with different error rate, ranging 1% from 5% to test the throughput performance by respectively calculating the average throughput. The statistical results are shown in Table 1. We can clearly see that the throughput of RB is significantly higher than that of Reno, Vegas and TCPW along with error rate increase. This is because RB adaptively adjusts the *cwnd* increment based on the queue length and the bandwidth ratio factor which reflect in real-time the current network state. Furthermore, during the congestion avoidance phase, RB keeps the network running with a appropriate rate close to the capacity based on the dynamics of bandwidth. Fig.2 shows the throughput comparison between TCPW and RB under 3% error rate. As seen, the throughput of RB is obviously improved a lot with respect to TCPW about 51.7% increments. The reason is that TCPW RB utilities the bandwidth factor to adjust the increase of *cwnd* in real-time in the congestion avoidance phase, and thus becomes more aware of the current network state, maintains a steady state, improves the utilization of bandwidth.

Table 1: Average throughput (Kbps) vs. error rate

Error rate	Reno	Vegas	TCPW	TCPW RB
1%	1527.8/1448.6	2135.1/2050.5	2048.5/2049.8	2340.2/2302.3
2%	974.7/1017.2	1601.3/1469.3	1633.4/1671.0	2136.1/2154.7
3%	769.7/716.0	1234.0/1138.8	1282.4/1273.3	1931.5/1884.0
4%	654.0/595.1	910.3/946.2	1046.8/998.0	1611.2/1639.0
5%	515.8/529.6	852.2/762.4	883.7/844.4	1428.2/1345.7

Packet losses. We do more detailed evaluation by the measurement of the number of the packet loss in the network to reflect the detail process of TCP connection. In practice, the number of the retransmission packets is equivalent to the packet loss, and we counts the number of retransmission packets under different bottleneck capacity. Simulation results are shown in Table 2. It is noted that the number of retransmission packets of RB compared to TCPW is decreased in a large extent and also the throughput is greatly improved. The number of retransmission packets lowered means the bandwidth utilization becomes more effective, as well as the retransmission times reduced means the free time is saved, thereby, the bandwidth utilization is improved.

Table 2: Retransmission packets and throughput (Kbps) vs. link capacity (Mbps)

Link capacity	TCPW packets /retransmission	RB packets /retransmission	TCPW throughput	RB throughput
1	8280/139	8414/46	433.9/453	548.4/380
2	16676/106	17090/51	919.1/9194	1025.8/873
3	24929/101	25486/50	1348.1/1413	1622.9/1217
4	33492/83	33772/31	1855.7/1864	2098.8/1668
5	41951/85	41939/58	2332.0/2334	2646.8/2033

Fairness and friendliness. Friendliness is a performance index measuring the degree of the influence of one TCP connection on the long-term throughput of other coexisting TCP connections. We establish two TCP connections with Reno and RB shared a 5Mbps bottleneck link. Table 3 shows the change of the throughput of TCPW RB and Reno with the error rate from 0% to 5%. Even though RB has an advantage over Reno in error-prone environments, Reno connections were not starved and keeps friendly. Fig.3 shows the throughput of Reno and RB with 2% error rate, and TCPW RB shows the friendliness to Reno.

Table 3: Throughput (Kbps) vs. error rate

Error rate	0%	1%	2%	3%	4%	5%
Reno	2799.9	1335.8	973.1	711.3	584.1	541.5
TCPW RB	1993.9	2940.0	2523.4	1997.3	1698.8	1450.2

Fairness is another important property of a TCP protocol. Multiple connections should accommodate each other and share fairly under the same TCP algorithms. We calculate the fairness index proposed in [10] to evaluate the fairness of the TCP mechanism. In the process of simulation, we increase the number of the connections of RB from 2 to 10 and n same TCP connections share the n Mbps bottleneck link. Fig.4 shows the comparison of the fairness index of three algorithms under the same error rate. Simulation result shows that the fairness of RB is better to TCPW and is little lower than the Reno, but the value of fairness index of RB is above 0.96.

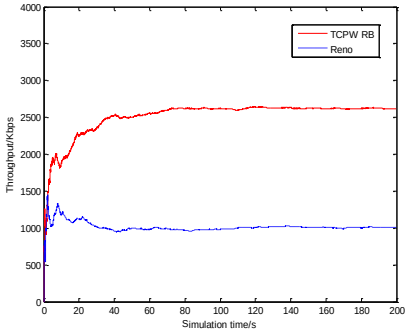

Fig.3: Friendliness with 2% error rate

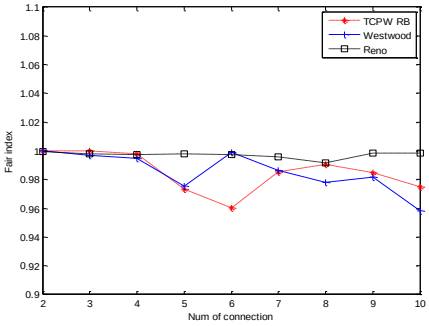

Fig.4: Fairness index

Conclusions

This paper proposes the improved algorithm called TCPW RB on the basis of TCPW, which alleviates the fast growth of the size of *cwnd* during the latter part of slow-start phase based on the estimated buffer queue length, and avoids multiple packet losses caused by the sudden flow. In addition, RB utilizes the bandwidth ratio factor to adaptively adjust *cwnd* increment during the congestion avoidance phase and makes more aware of the network state and maintains appropriate bandwidth to take full advantage of the link. Simulation results show that TCPW RB not only improves the throughput and degrades the packet losses and improves the bandwidth utilization of the network, but also maintains good fairness and friendliness compared to TCPW, Reno, Vegas.

Acknowledgements

The research work was supported by the Fundamental Research Funds for the Central Universities Project No. ZYGX2012J020.

References

[1] Fu C P, Liew S C. TCP Veno: TCP enhancement for transmission over wireless access networks. IEEE Journal of Selected Areas in Communications, 21(2), pp. 216-228. (2003).

[2] Katabi D, Handley M, Rohrs C. Congestion control for high bandwidth-delay product networks[C]//ACM SIGCOMM Computer Communication Review. ACM, 32(4), pp. 89-102. (2002).

[3] Chockalingam A, Zorzi M, Tralli V. Wireless TCP performance with link layer FEC/ARQ[C]//Communications, ICC'99 IEEE International Conference on. IEEE, 2, pp. 1212-1216. (1999).

[4] Beomjoon Kim, Dongmin Kim, and Jaiyong Lee, Lost Retransmission Detection for TCP SACK, IEEE COMMUNICATIONS LETTERS, VOL. 8, NO. 9. (2004).

[5] Samios C B, Vernon M K Modeling the throughput of TCP Vegas. ACM SIGMETRICS Performance Evaluation Review, 31(1), pp.71-81. (2003).

[6] Gerla M, Sanadidi M Y, Wang R, et al. TCP Westwood: Congestion window control using bandwidth estimation[C]//Global Telecommunications Conference, GLOBECOM'01. IEEE, 3, pp. 1698-1702. (2001).

[7] Geethu Wilson, Robin Cyriac An Enhancement to TCPW BBE by Modifying the Bandwidth Estimation Using Modifieed EWMA International Journal Of Advanced Research in Computer Science and Software Engineering (2012).

[8] Hagag S, El-Sayed A. Enhanced TCP Westwood Congestion Avoidance Mechanism (TCP WestwoodNew). International Journal of Computer Applications, 45. (2012).

[9] Lan K, Sha N. A CMT congestion window updates mechanism based on TCP Westwood[C]//Mechatronic Science, Electric Engineering and Computer (MEC), International Conference on. IEEE, pp. 2119-2122. (2011).

[10] Jain R, Chiu D M, Hawe W R. A quantitative measure of fairness and discrimination for resource allocation in shared computer system. Eastern Research Laboratory, Digital Equipment Corporation. (1984).