SOCIETY FOR SCIENCE AND EDUCATION
UNITED KINGDOM

TNC Transactions on Networks and Communications

VOLUME 2, ISSUE 5
ISSN: 2054 - 7420

Enhanced TCP Westwood Slow Start Phase

Mohanad Al-Hasanat, Kamaruzzaman Seman and Kamarudien Saadan
University Sains Islam Malaysia, Malaysia
mohanad.hasanat@gmail.com; {drkzaman, kamarudin}@usim.edu.my

ABSTRACT

Many end-to-end TCP implementations have been presented in the past decade. Despite that they used different methods to improve transport protocols over wireless networks; they mostly shared the same original TCP principles. TCP Westwood introduced a novel end-to-end bandwidth estimation mechanism. Nevertheless, it maintains the same slow start phase presented in TCP Reno. For the initial slow start phase, there is no safe slow start threshold value. In this paper, we propose to use the bandwidth estimation to calculate the initial slow start threshold value after the second round trip time. Furthermore, we introduce a faster state in which TCP increases the transmission rate once the link is underutilizing. As a result, the new proposed method shows better performance comparing to TCP Westwood, and TCP NewReno techniques.

Keywords: TCP Westwood; Congestion Control; Slow Start; Slow Start Threshold, TCP Enhancement.

1 Introduction

TCP– Transmission Control Protocol is the most transport protocol used over internet. Inefficient TCP performance in wireless networks motivated a wide spectrum in research community to enhance its congestion control mechanisms. Several TCP variants have been introduced over the past decade to support different network technologies [5, 6]. These mechanisms can be classified into three main categories: a link level solutions (e.g. I-TCP, M-TCP, etc.), end-to-end solutions (e.g. Explicit Congestion Notification (ECN), TCP Westwood, TCP Casablanca, etc.), and split connection solutions (e.g. Forward Error Correction (FEC), Automatic Repeat Request (ARQ), and Hybrid ARQ (HARQ), etc.).

TCP Westwood-TCPW presented a novel E2E bandwidth estimation mechanism by monitoring the rate of returning acknowledgments at the sender side [4]. TCPW inherent the basic TCP transmission control principles; flow control, congestion control, and error control mechanisms. The flow control tries to limit the transmission rate corresponding to the receiver's buffer capacity. Whereas, the congestion control mechanisms tries to limit the transmission rate by the link capacity. Therefore, TCP uses a congestion window (cwnd) to limit the number of segments the sender can transmit whenever a new acknowledgment received. TCPW starts the connection in slow start phase. During this phase the sender increments its cwnd exponentially until cwnd equal to a predefined value called slow start threshold (ssthresh). After that, a congestion avoidance phase is started, during which the sender increments its cwnd linearly. Anytime a packet loss event occurs, TCP sets the ssthresh value to one half the cwnd and trigger the slow start again. For the initial slow start there is no safe ssthresh value. If the ssthresh value is too small, then the sender will immediately stops the exponential increment of the cwnd. Thus, it will

DOI: 10.14738/tnc.25.601
Publication Date: 25rd October, 2014
URL: http://dx.doi.org/10.14738/tnc.25.601

take long time to reach the optimal cwnd size using the Additive Increase/Multiplicative Decrease (AIMD) linear incrementing. On the other hand, using a large ssthresh value will aggressively increase the cwnd. This as an effect causes more packet losses, unwanted retransmission events, and serious performance degradation.

In this paper, we propose to modify the TCPW bandwidth estimation in order to properly set the ssthresh value for the initial slow start phase. In addition, we present a faster start phase in which the sender can rapidly increase its cwnd size to shorten the slow start phase.

The rest of the paper is organized as follow; Section two presents a brief background study. We introduce our modifications in section three. Section four present a comparative simulation experiments to validate the proposed modifications. Then the conclusion is drawn in section five.

2 Background Study

2.1 Introduction

TCP Tahoe introduced the first congestion control mechanism in 1988 [1], including slow start, congestion avoidance, and fast retransmit. A new modification to the Tahoe's fast retransmit was presented in TCP Reno [2]. This modification used a fast recovery mechanism every time a fast retransmit procedure is triggered. Further modification to Reno is presented as TCP New-Reno [3]. TCP NewReno enhanced the fast retransmit mechanism in case of multiple packets lost from a single window.

The poor performance of TCP over wireless networks innovate a wide spectrum of research community to develop new solutions [8]. Many TCP variants have been presented to overcome this issue. One novel End-to-End bandwidth estimation method known as TCP Westwood [4] was presented. TCPW monitors the rate of the returning acknowledgments at the sender side to obtain an estimation of the link bandwidth. Then TCPW uses the estimation to set the slow start threshold value after a loss event occurs.

2.2 The Slow Start

Slow start algorithm used to gradually increase the number of packets in transit. The implementation of the slow start is accomplished through defining two variables to control the transmission; the congestion window (cwnd) and the receiver advertized window (rwnd). The cwnd is the number of packets the sender can send before receiving an acknowledgment (ACK). While the rwnd is the amount of packets the receiver can buffer. The sender limits the sending rate to the minimum of the cwnd and rwnd values.

To avoid congesting in the transmission links with large amount of data, TCP slowly probe the network capacity using slow start. Usually, TCP star transmission by cwnd=1 segment. During the slow start phase, the sender side increments cwnd by 1 segment for each ACK received. This exponential growth of cwnd ends when the cwnd exceeds the slow start threshold (ssthresh) value or when congestion observed. When packet loss event detects the value of ssthresh set to half of the cwnd size, the cwnd sets to 1 segment, and the TCP sender starts the slow start again. Figure 1 shows the cwnd growth during slow start phase.

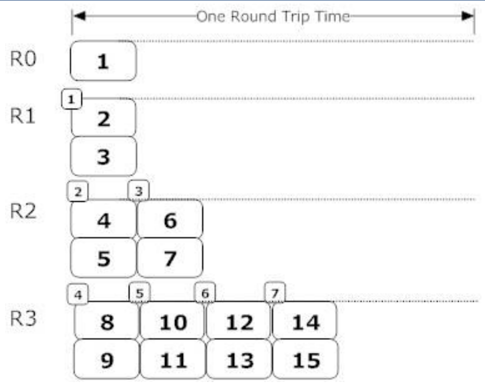

Figure 1: The Chronology of Slow Start [1]

3 The Modification

In this section we present a new method to properly set the ssthresh value in the initial slow start phase. Toward this end, we probe the link's bandwidth by counting the bytes acknowledged between two sequences ACKs at the sender side. According to the following equation:

$$ELC = \frac{Acked * SegmentSize}{t_k - t_{k-1}}$$

Where *LC* is the link capacity and *Acked* is the number of packets acknowledge within every ACK. Then we use the moving average method to update the computed ELC every ACK received according to following formula:

$$ELC = (1-\alpha)ELC_i + \alpha * ELC_{i-1}$$

Where α = 0.9.

To get the ssthresh in a form of congestion window we use the following equation:

$$ssthresh = ELC * MinRTT$$

Where RTT is round trip time (the time when a packet is sent until the ACK is received). The computed ssthresh value will provide an accurate value for initial slow start threshold according to the link capacity. As seen, this value is not a constant number; however it varies according to the connection status.

Furthermore, we proposed to use a state called the "Faster start" in which we can increase the cwnd according to a bandwidth utilization. Before sending new segment during the slow start, we check the values of the current cwnd and last round trip time. As following:

- If the last RTT is less than or equal the estimated RTT, and cwnd less than the half of ssthresh, then cwnd = cwnd + (ssthresh DIV cwnd).
- Else, cwnd =cwnd +1.

The following section shows that, the new modifications improve TCPW congestion window and the throughput.

4 Experiments and Analysis

The performance of the new modifications is assessed in this section. We use two performance metrics to evaluate the proposed modifications, throughput, and congestion window. We compare the results with TCPW and TCP NewReno. For a consistence comparison we use the same simulation scenario that had been conducted to present the original TCPW [4]. Network Simulator NS-3 is used to option the results, and gnuplot used to plot the graphs.

4.1 Simulation Setup

The topology used in this experiment is shown in Figure 2. A single source and sink connected via a gat-way (PGW).

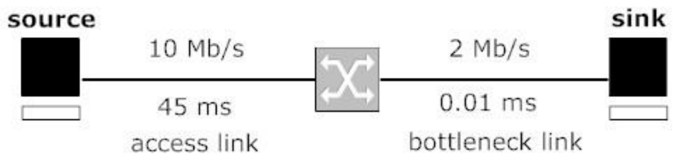

Figure 2: Simulation Topology [4].

Two links, a source-PGW link labeled access link, and PGW-sink link labeled bottleneck link. The access link bandwidth is 10Mbps with propagation delay of 45ms, where the bottleneck link bandwidth is 2Mbps with propagation delay of 0.01ms. The NS3's built-in PointToPointHelper [7] is used to represent a point to point (P2P) connection between the source-PGW and the PGW-sink. To simulate the wireless lose channel we used the RateErrorModel [7] class to generate sending errors over the bottleneck link. Errors are assumed to follow random distribution. A BulkSendApplication [7] is used to generate a single traffic along the simulation period started at the source and ended at the sink. Table1 summarizes the simulation parameters.

Table 1. Simulation parameters.

Parameter	Value
Mobility	Fixed Position
Access link bandwidth	10Mb/s
Access link Propagation Delay	45 ms
Bottleneck link bandwidth	2Mb/s
Bottleneck link Propagation Delay	0.01 ms
Error model	Uniform Error Model
Packet Error Rate (PER)	0.005
Application type	Bulk Send Application
Simulation time	5 seconds

We used 5 seconds as simulation time to focus our results on the initial slow start phase.

4.2 Simulation Results

For the first experiment, we used the same parameters listed in table 1. We compared the congestion window of the new modification referred to as Petra, TCPW, and TCP NewReno. The result is plotted in Figure 3.

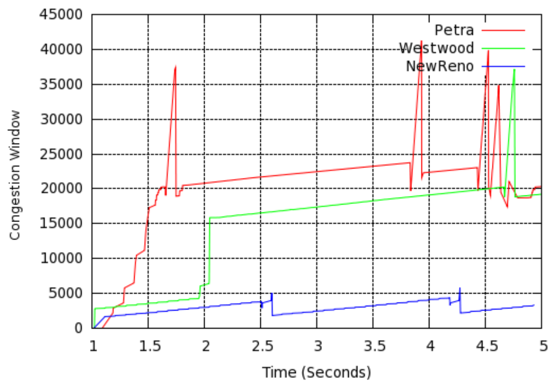

Figure 3:Congestion Window comparison.

As can be observed from this figure, a bigger cwnd values is achieved for the new modification algorithm comparing to TCPW and TCP NewReno. Moreover, we can see how fast the new modification reaches the optimal cwnd that is just about 1.75 seconds. While TCPW reached its maximum cwnd at 4.75 seconds, and TCP NewReno recorded a very small congestion window size.

Next we evaluate the total throughput achieved using the new modification, TCPW, and TCP NewReno as a function of increasing bottleneck bandwidth size. The results are plotted in Figure 4. The same network topology given above is also used.

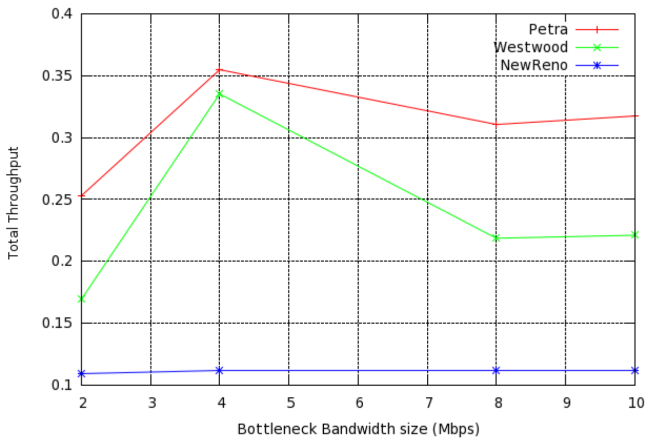

Figure 4: Throughput vs. bandwidth

It is clearly seen that a better throughput is achieved with Petra comparing to TCPW and TCP New Reno. However, weird throughput degradation is noticed after the 4Mbps bandwidth for both TCPW and Petra. One reason of this weird behavior could be due to the increasing retransmission procedures that occurred as responses to loss events. Such weird behavior did not appear for small bandwidth sizes since infrequent packet losses occurred.

The simulation experiments were extended to study the impact of various propagation delay values on the throughput. Figure 5 shows the total throughput values achieved by Petra, TCPW, and TCP New Reno over different propagation delay values started from 1ms to 60ms. The bottleneck bandwidth is set 2Mbps and the simulation period is set to 5 seconds. A BER of 0.005 is still used.

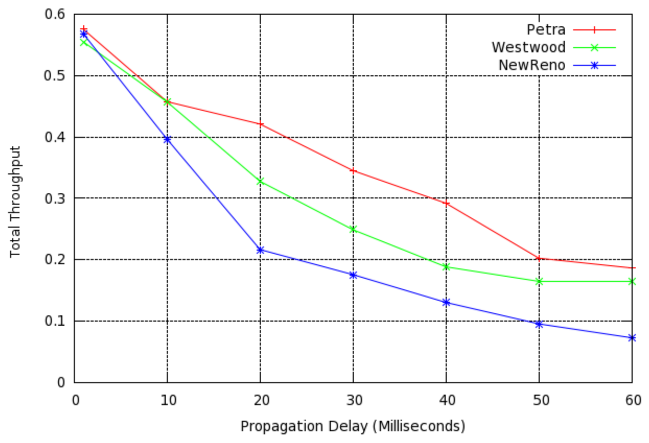

Figure 5: Throughput vs. Propagation delay

Figure 5 shows better throughput values recorded using the new modification over the other implementations. As expected, the throughput is decreased as the propagation delay is increased.

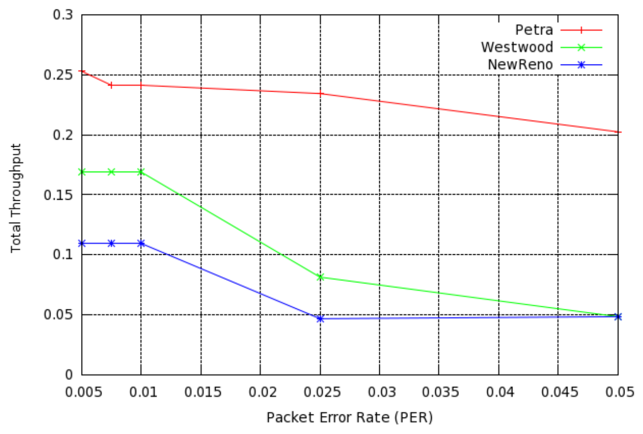

Figure 6: Throughput vs. Packets Error Rate.

In figure 6 we plotted the throughput recorded by the three implantations. We used various values of PER over the same bandwidth size and propagation delay 2Mbps and 45Ms respectively. As we can be seen, the new modification outperformed other implementations significantly for the entire range of PER.

5 Conclusion

In this paper, new modifications to TCPW slow start phase, referred to as Petra, were introduced. One modification suggested using bandwidth estimation in order to set the initial value of the slow start threshold. The other modification is represented by using a faster start to rapidly increase the congestion window size according to the link status. Simulation results in this paper slowed that, Petra improved TCP performance in terms of throughput and congestion window size.

In future work, we could further extend the evaluation process to study the impact of more performance metrics including Jitters, delay, and packet loss. Moreover, we could compare Petra to other TCP implementations, in addition to investigating the fairness and the friendless of the new modifications.

REFERENCES

[1]. Van Jacobson and M. J. Karels, *Congestion Avoidance and Control*. ACM Computer Communication, 1988. 18: p. 314-329.

[2]. V. Jacobson. *Modified TCP Congestion avoidance algorithm*, end2end-interest mailing list, April 30, 1990.

[3]. Floyd, S., et al., *The NewReno Modification to TCP's Fast Recovery Algorithm.* RFC 3782, April 2004.

[4]. S. Mascolo, C., et al., *TCP westwood: Bandwidth estimation for enhanced transport over wireless inks.* ACM SIGMOBILE, 2001. p. 287-297.

[5]. H. Xie, A., et al., *A Novel Cross Layer TCP Pacing Protocol for Multi-hop Wireless Networks*. IEEE Wireless Communications and Networking Conference, 2013.

[6]. C. Hu, X., et al., *WiTracer: A Novel Solution to Improve TCP Performance overWireless Network.* IEEE, 2013.

[7]. The ns-3 Network Simulator Doxygen Documentation. http://www.nsam.org/doxygen/group_tcp.html, December 2013.

[8]. M. Al-Hasanat, K., et al.. Enhanced TCP Westwood Congestion Control Mechanism over Wireless Networks. In International Conference on Advanced Technology & Sciences. 12-15 August, 2014. Antalya, T