www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 4 April, 2013 Page No. 1184-1191

A Review on Tcp Westwood Protocol For Simulated And Internet Environment

Vijay P Reshamwala*, Kaushika D Patel
Student, BVM, v. v. nagar, GTU, India
Department of Electronics, BVM, v. v. nagar, GTU, India
vijay.reshamwala528@gmail.com*; kdpatel@bvmengineering.ac.in

Date of Submission: 18 April, 2013

Abstract:
This paper discusses various versions of TCP and their congestion control algorithm changes in TCP's existing congestion control, limitation of older version of TCP variants. We discuss new version of TCP called TCP Westwood with sender side modification of the window congestion control scheme TCP Westwood continuously estimate at sender side packet rate of connection by monitoring ACK reception rate. In this paper we reviewed the comparison of performance of TCP Reno with TCP Westwood in good link and lossy link. In this paper we have discussed the fairness and friendliness issue of TCP Westwood. Also we discuss the mechanism called agile probing that improves the performance of TCP Westwood in slow start phase. This method improves the startup performance of TCP Westwood. This method improves the performance of TCP Westwood in congestion avoidance phase as well as when large amount of bandwidth that suddenly becomes available.

Keywords: Congestion Avoidance, TCP Westwood, Bandwidth Estimated, Friendliness, Router Buffer Size, random errors

1. Introduction:

The Internet is an unreliable network that cannot guarantee all data sent by host will be delivered correctly to the destination .As a result, reliable end-to-end data delivery is delegated to transport layer protocols such as the transmission control protocol.

The transmission control protocol is most widely used and established implementation of the reliable protocols but many different versions of TCP have been developed as the algorithms and techniques for increasing efficiency and performance have been refined. We examine the various versions of TCP and characterize the attributes of these protocols which contribute to their improved performance.

2. TCP Variants:

2.1TCP Tahoe:

The Tahoe TCP algorithms include Slow-Start, Congestion Avoidance, and Fast Retransmit. The idea of TCP Tahoe is to start the congestion window at the size of a single segment (the MSS) and send it when a connection is established. If the acknowledgement arrives before the retransmission timer expires, add one segment to the congestion window. This is a multiplicative increase algorithm and the window size increases exponentially. The window continues to increase exponentially until it reaches the threshold that has been set. This is the Slow Start Phase. Once the congestion window reaches the threshold, TCP slows down and the congestion avoidance algorithm takes over. Instead of adding a new segment to the congestion window every time an acknowledgement arrives, TCP increases the congestion window by one segment for each round trip time. This is an additive increase algorithm. To estimate a round trip time, the TCP codes use the time to send and receive acknowledgements for the data in one window. TCP does not wait for an entire window of data to be sent and acknowledged before increasing the congestion window. Instead, it adds a small increment to the congestion window each time an acknowledgement t arrives. The small increment is chosen to make the increase averages approximately one segment over an entire window. When a segment loss is detected through timeouts, there is a strong indication of congestion in the network. The slow start threshold is set to one-half of the current window size (the minimum of the receiver's advertised window and the sender's congestion window). Moreover, the congestion window is set to 1 segment, which forces slow start [9].

2.1.1 Demerits Of TCP Tahoe:

The problem with Tahoe is that it takes a complete timeout interval to detect a packet loss and in fact, in most implementations it takes even longer because of the coarse grain timeout. Also since it doesn't send immediate ACK's, it sends cumulative acknowledgements, therefore it follows a 'go back n 'approach. Thus every time a packet is lost it waits for a timeout and the pipeline is emptied. This offers a major cost in high band-width delay product links.

2.2 TCP Reno:

TCP Reno incorporated the fast recovery mechanism TCP Reno use same Slow-Start, Congestion Avoidance algorithm only change is that When a segment is detected by fast retransmit, the sender does Fast Recovery [9]:

In TCP Reno, TCP source behaves in the same way as that in TCP Tahoe. However, when the segment loss is detected by fast retransmission algorithm, the slow start threshold is set to half the current size of the congestion window. The congestion window size is then set to be the same as the slow start threshold plus 3 times the segment size. This is the Fast Recovery Phase, in which the window size is then increased by one segment when a duplicate ACK is received. When the non-duplicate ACK corresponding to the retransmitted segment is received, the congestion window is restored to the slow start threshold [9].

2.2.1 Demerits Of TCP Reno:

Segments not be acknowledged cumulatively but should be acknowledged selectively. The selective acknowledgment extension uses two TCP options. The first is an enabling option, "SACK-permitted", which may be sent in a SYN segment to indicate that the SACK option can be used once the connection is established. The other is the SACK option itself, which may be sent over an established connection once permission has been given by SACK-permitted. The SACK option is to be included in a segment sent from a TCP that is receiving data to the TCP that is sending that data; The SACK option is to be used to convey extended acknowledgment information from the receiver to the sender over an established TCP connection. Thus each ACK has a block which describes which segments are being acknowledged. Thus the sender has a picture of which segments have been acknowledged and which are still outstanding. [9].

2.3 TCP New Reno:

New Reno is a slight modification over TCP Reno. It is able to detect multiple packet losses and thus is much more efficient that Reno in the event of multiple packet losses. Like Reno, New-Reno also enters into fast-retransmit when it receives multiple duplicate packets, however it differs from Reno in that it doesn't exit fast-recovery until all the data which was out standing at the time it entered fast recovery is acknowledged. Thus it overcomes the problem faced by Reno of reducing the cwnd multiples times. The fast-transmit phase is the same as in Reno. The difference in the fast recovery phase which allows for multiple re-transmissions in new-Reno. Whenever new-Reno enters fast recovery it notes the maximums segment which is outstanding. The fast-recovery phase proceeds as in Reno, however when a fresh ACK is received then there are two cases:

If it ACK's all the segments which were outstanding when we entered fast recovery then it exits fast recovery and sets cwnd to ssthresh and continues congestion avoidance like Tahoe.

If the ACK is a partial ACK then it deduces that the next segment in line was lost and it re-transmits that segment and sets the number of duplicate ACKS received to zero. It exits Fast recovery when all the data in the window is acknowledged [9].

2.3.1 Demerits Of TCP New Reno:

New-Reno suffers from the fact that it's taking one RTT to detect each packet loss. When the ACK for the first retransmitted segment is received only then can we deduce which other segment was lost

2.4 TCP SACK:

TCP with 'Selective Acknowledgments' is an extension of TCP Reno and it works around the problems face by TCP Reno and TCP New-Reno, namely detection of multiple lost packets, and re-transmission of more than one lost packet per RTT. SACK retains the slow-start and fast retransmits parts of Reno. SACK TCP requires that segments not be acknowledged cumulatively but should be acknowledged selectively. The selective acknowledgment extension uses two TCP options. The first is an enabling option, "SACK-permitted", which may be sent in a SYN segment to indicate that the SACK option can be used once the connection is established. The other is the SACK option itself, which may be sent over an established connection once permission has been given by SACK-permitted. The SACK option is to be included in a segment sent from a TCP that is receiving data to the TCP that is sending that data; The SACK option is to be used to convey extended acknowledgment information from the receiver to the sender over an established TCP connection. Thus each ACK has a block which describes which segments are being acknowledged. Thus the sender has a picture of which segments have been acknowledged and which are still outstanding. [9].

2.4.1 Demerits Of TCP SACK:

The biggest problem with SACK is that currently selective acknowledgements are not provided by the receiver to implement SACK we'll need to implement selective acknowledgment which is not a very easy task.

2.5 TCP Vegas:

2.5.1 New Re-Transmission Mechanism:

It keeps track of when each segment was sent and it also calculates an estimate of the RTT by keeping track of how long it takes for the acknowledgment to get back. Whenever a duplicate acknowledgement is received it checks to see if the (current time segment transmission time)> RTT estimate; if it is then it immediately retransmits the segment without waiting for 3 duplicate acknowledgements or a coarse timeout [12].

2.5.2 Modified congestion avoidance algorithm:

Instead of increasing the congestion window size blindly until losses occur, TCP Vegas tracks the changes in the throughput (or more specifically, changes in the sending rates) and then adjusts the congestion window size. It observes changes in the round-trip times of the segments that the connection has sent before. It then calculates and compares the measured throughput against the expected

throughput. If the expected sending rate is higher than the actual sending rate by a or less, TCP Vegas fears that it is not utilizing the bandwidth efficiently by occupying some router buffers and thus increases the congestion window by one. If the expected rate is higher than the actual rate by b or more, TCP Vegas assumes that congestion starts to build up and thus decreases the congestion window by one. Otherwise, the congestion window remains unchanged [12].

2.5.3 Modified slow-start:

TCP Reno doubles its window size every RTT during the slow start phase. TCP Vegas, on the other hand, doubles the window size only every other RTT during slow-start [12].

2.5.4 Demerits Of TCP Vegas:

TCP Vegas uses a conservative algorithm to decide how and when to vary its congestion window. Thus, when TCP Vegas and TCP Reno connections share a bottleneck link, Reno uses up most of the link and router buffer space. TCP Vegas, interpreting this as a sign of congestion, decreases the congestion window, which leads to an unfair sharing of available bandwidth in favour of TCP Reno.

In TCP Vegas, the parameter baseRTT denotes the smallest round-trip delay the connection has encountered and is used to measure the expected throughput. When rerouting occurs in between a connection, the RTT of a connection can change. When the new route has a longer RTT, the Vegas connection is not able to deduce whether the longer RTTs experienced are caused by congestion or route change. Without this knowledge, TCP Vegas assumes that the increase in RTT is due to congestion along the network path and hence decreases the congestion window size.

2.6 TCP Westwood:

TCP Westwood exploits two basic concepts: the end-to-end estimation of the available bandwidth, and the use of such estimate to set the slow start threshold and the congestion window. TCPW source continuously estimates the packet rate of the connection by properly averaging the rate of returning ACKs. The estimate is used to compute the "permissible" congestion window and slow start threshold to be used after congestion episode is detected, that is, after three duplicate Acknowledgments or after a timeout. It selects a slow start threshold and a congestion window that is consistent with the effective connection rate at the time congestion is experienced. We call such mechanism faster recovery.

3. Overview of TCP Westwood:

In TCP Westwood the sender continuously computes the Connection Bandwidth Estimate (BWE) which is defined as the share of bottleneck bandwidth used by the connection. Thus, BWE is equal to the rate at which data is delivered to the TCP receiver. The estimate is based on the rate at which ACKs are received after a packet loss indication, (i.e. reception of 3 duplicate ACKs, or timeout expiration). , the sender resets the congestion window and the slow start threshold based on BWE. This BWE varies from flow to flow sharing the same bottleneck; it corresponds to the rate actually achieved by each INDIVIDUAL flow. Thus, it is a FEASIBLE (i.e. achievable) rate by definition. Consequently, the collection of all the BWE rates, as estimated by the connections sharing the same bottleneck, is a FEASIBLE set. When the bottleneck becomes saturated and packets are dropped, TCPW selects a set of congestion windows that correspond exactly to the measured BWE rates and thus reproduce the current individual throughputs. Another important element of this procedure is the RTT estimation. RTT is required to compute the window that supports the estimated rate BWE.

3.1 TCP Westwood: Algorithm:

TCPW, congestion window increments during slow start and congestion avoidance remain the same as in Reno, i.e. they are exponential and linear, respectively. A packet loss is indicated by (a) the reception of 3 DUPACKs, or (b) a coarse timeout expiration The general idea is to use the estimated bandwidth BWE to set the congestion window (cwi*n*) and the slow start threshold (ssthresh) after a congestion episode

3.1.1 Algorithm after n duplicate ACKS [5]:

```
if (n DUPACKs are received)
ssthresh = (BWE * RTTmin) / seg_size;
if (cwin > ssthresh) /* congestion avoid. */
cwin = ssthresh;
endif
endif
```

In the pseudo-code, seg_size identifies the length of a TCP segment in bits. Note that the reception of n DUPACKs is followed by the retransmission of the missing segment, as in the standard Fast Retransmit implemented by TCP Reno. Also, the window growth after the cwin is reset to ssthresh follows the rules established in the Fast Retransmit algorithm (i.e., cwin grows by one for each further ACK, and is reset to ssthresh after the first ACK acknowledging new data). During the congestion avoidance phase we are probing for extra available bandwidth. Therefore, when n DUPACKs are received, it means that we have hit the network capacity (or that, in the case of wireless links, one or more segments were dropped due to sporadic losses). Thus, the slow start threshold is set equal to the window capable of producing the measured rate BWE when the bottleneck buffer is empty (namely, BWE*RTTmin). The congestion window is set equal to the ssthresh and the congestion avoidance phase is entered again to gently probe for new available bandwidth.. Note that after ssthresh has been set, the congestion window is set equal to the slow start threshold only if cwin > ssthresh.

3.1.2 algorithm after coarse timeout expiration [5]:

```
if (coarse timeout expires)
cwin = 1;
ssthresh = (BWE * RTTmin) / seg_size;
if (ssthresh < 2)
ssthresh = 2;
endif;
endif
```

The rationale of the algorithm above is that after a timeout, cwin and the ssthresh are set equal to 1 and BWE, respectively. Thus, the basic Reno behavior is still captured,

while a speedy recovery is ensured by setting ssthresh to the value of BWE.

3.2 Strategy for Bandwidth Estimation :

The TCPW sender uses ACKs to estimate BWE. More precisely, the sender uses the following information: (1) the ACK arrival times and, (2) the increment of data delivered to the destination. Let assume that an ACK is received at the source at time tk, notifying that dk bytes have been received at the TCP receiver. We can measure the sample bandwidth used by that connection as bk=dk/(tk–tk–1), where tk−1 is the time the previous ACK was received. Letting Δtk=tk–tk–1, then bk=dk/Δtk. Since congestion occurs whenever the low-frequency input traffic rate exceeds the link capacity, we employ a low pass filter to average sampled measurements and to obtain the low-frequency components of the available bandwidth. More precisely, we use the following discrete approximation of the low pass filter due to Tustin.

Let bk be the bandwidth sample, and b^k the filtered continuous first order low-pass filter using the Tustin estimate of the bandwidth at time tk. Let αk be the time-varying exponential filter coefficient at tk. The TCPW filter is then given by [4]

$$\hat{b}_k = \alpha_k \hat{b}_{k-1} + (1-\alpha_k)(b_k + b_{k-1})/2$$

where,

$$\alpha_k = (2\tau - \Delta t_k)/(2\tau + \Delta t_k)$$

$1/\tau$ is the filter cut-off frequency

Notice the coefficients αk depend on Δ*tk* to properly reflect the variable inter-arrival times.

3.3 TCP Westwood fairness and friendliness:

Fair bandwidth sharing implies that all connections are provided with similar opportunity to transfer data. Friendliness is another important property of TCP protocol. TCP Westwood must be friendly to other TCP variants. That is TCP Westwood connection must be able to coexist with connections running TCP variants while providing opportunity for all connections to progress satisfactorily

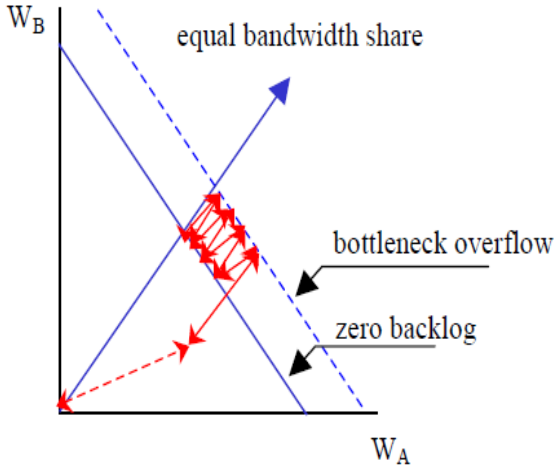

Figure. 3.1 Convergence towards the fair bandwidth sharing [5]

Consider the case of two connections with the same RTTs. Suppose, for the sake of example that the RTT is X packet Transmission times, and the bottleneck has X buffers. One connection, say A, starts first. Its window "cycles" between X and 2X each cycle terminating when buffer overflow. Later, connection B starts, first in slow start mode, and then in congestion avoidance mode. In congestion avoidance, during each cycle the A and B windows grow approximately at the same rate, i.e. one segment per RTT. Eventually, the bottleneck buffer overflows, terminating the cycle. One can show that the window at overflow is:

Wi = Ri (b/C +RTT), for i = A, B

Where, R is the achieved rate (i.e. BWE), b is the bottleneck buffer size, and C is the bottleneck link capacity.

This is a general property true for all TCP protocols, and in particular TCPW. After overflow, TCPW reduces the windows to new values Wi' as follows:

Wi'= Ri (RTT) for i = A, B

Thus, the ratios of the windows of connections A and B are preserved after overflow. Yet, the ratio WB/WA keeps increasing during congestion avoidance. Consequently, the B window and throughput ratchet up at each cycle. Equilibrium is reached when the two connections have the same windows and the same fair share of the bandwidth. The Figure 3.1 graphically illustrates the convergence to the fix point WA=WB.

It can also be applied to the case when the\ bottleneck is affected by random errors equally hitting all connections. The same method can also be used to evaluate reciprocal "friendliness" of TCPW and TCP Reno. If two connections - TCPW and Reno - are sharing the bottleneck, and the buffer size is exactly equal to the optimal window size to "fill the pipe", then the two connections spilt the bottleneck fairly. In fact, at equilibrium, each has window = X when buffer overflows. After overflow, the TCPW connection gets window = C*RTT/2 = X/2; TCP Reno simply half the current window, to X/2. Thus, friendliness is preserved Note that sizing the buffer to match the "pipe size" is a common and intuitively acceptable design choice. If the buffer is much smaller than pipe size, TCPW returns a larger CWIN than Reno, and thus tends to capture the channel. If, on the other hand, the buffer is several times larger than pipe size, Reno tends to prevail over TCPW.

Here represented simulation result illustrating the accuracy of TCP Westwood bandwidth estimation scheme

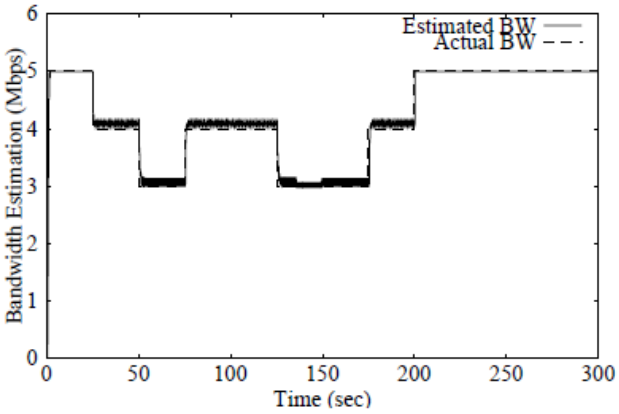

Figure. 3.2 TCPW with concurrent UDP traffic-bandwidth estimation [8]

Figure.3.2 shows a single TCP connection sharing the bottleneck link with two background UDP ON/OFF sources of varying data rates with no flow control. TCP packets are 1400 bytes in length including TCP/IP headers. TCP and UDP packets are assigned the same priority. The 5 Mbps bottleneck link has a round trip propagation time of 70 ms. Each UDP connection transmits at a constant bit rate of 1 Mbps while ON. Both UDP connections start in the OFF state; after 25s, the first UDP connection is turned ON, joined by the second one at 50s; the second connection follows an OFF-ON-OFF pattern at times 75s, 125s and 175s; at time 200s the first UDP connection is turned off as well..

The results in Figure.3.2 confirm the effectiveness of the TCP Westwood bandwidth estimation perfectly tracks the UDP fluctuations, adjusting throughput accordingly.

TCP Westwood performance in presence of link error

Here represented simulation result in figure 3.3 comparing the throughput of Reno and TCPW as a function of error rates. The bottleneck bandwidth is set to 45Mbps, and the two-way propagation time is 70ms. With no errors, the performance of TCPW and Reno is virtually identical. As error rate increases, TCPW outperforms Reno. At 1 % error rates, appropriate for wireless links, the throughput improvement is615 %. As the error rate increases further, say above 10%, even TCPW collapses, as expected.

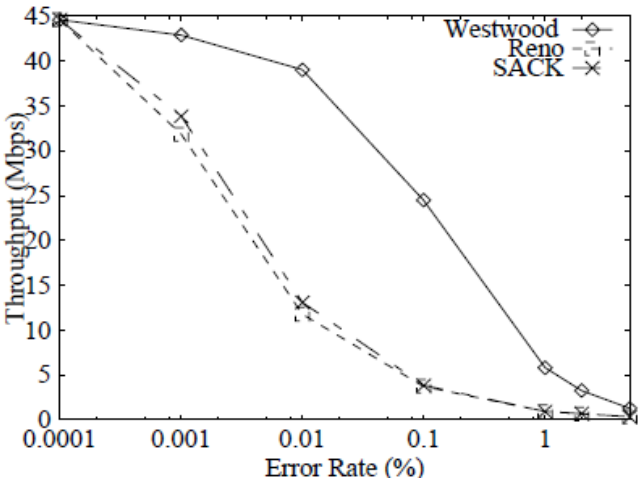

Figure. 3.3 Impact of error rates [8]

Experimental results seen in figure 3.3 confirms that new control scheme converges to fair share at steady state under uniform path conditions, also TCP Westwood handle losses caused by link errors or wireless channel more efficiently than TCP Reno.

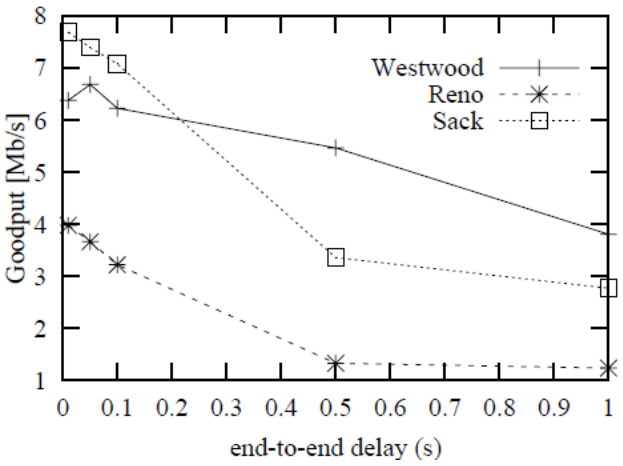

Figure. 3.4 Good put as function of one-way end-to-end delay [7]

Figure 3.4 represents simulation results that confirmed that Westwood performs better than Reno under all tested RTTS but it slightly out performed by Sack if it end-to-end delay is smaller than 0.2 seconds as depicted in Figure3.4(10 connections,45 Mb/s link capacity).

Simulation seen that for small RTTs Sack manages to recover fast enough on other hand TCP Westwood suffer from being too aggressive for small RTTS and result poorly accurate bandwidth estimation forces it into slow start state.

Internet measurements

Here researcher test TCPW in an actual Internet environment, test carried out a set of Internet experiments using the configuration depicted in Figure 3.5 The sources are at UCLA, while

Internet throughput measurements.

Destination	Italy		Taiwan		Brazil	
RTT	170 ms		250 ms		450 ms	
Protocol	TCPW	Reno	TCPW	Reno	TCPW	Reno
Throughput (KB/s)	78.66	73.93	167.38	152	22.16	15.4

Table 1 Internet throughput measurements [5]

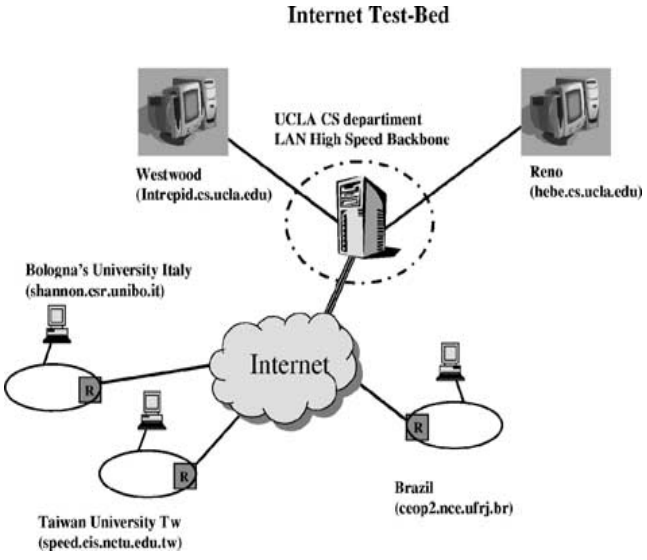

Fig.3.5. Internet measurement scenario [5]

The destinations are chosen in three different continents (Europe, South America, and Asia). The destination hosts are, of course, unaware whether the source host runs TCPW or Reno. Tests were scheduled during normal working hours at the destination sites. Experiments included either single or multiple file transfers. Throughput results were obtained by averaging repeated single file transfers. Multiple file transfer experiments were used to assess TCPW fairness. A rather large file size was used (10Mbytes) to capture only steady state behavior. A standard FTP client (ncftp-3.0.2) was used as testing software with additional code for obtaining detailed logging at 1s intervals. Here measured application throughput in terms of user data⁄s as reported by ncftp. The average throughput achieved by Reno and TCPW on the various intercontinental connections is shown in table 1. Tests were repeated about 200 times throughout the day. The results show that TCPW performs marginally better that Reno on the Italy and Taiwan connections. It performs significantly better on the Brazil connection.

TCP Westwood introduce faster recovery to avoid over shrinking cwin after three duplicate ACKs by taking into account the end-to-end estimation of the bandwidth available to the TCP.

4. Changes done in existing algorithm:

4.1 Reno Friendly TCP Westwood based on Router Buffer Estimation:

In this method, we represent an improved version of TCP Westwood to overcome unfriendliness of TCP Westwood according to buffer size of bottleneck link router. Here first investigate the friendliness of TCP Westwood through mathematical analysis using throughput model, then estimate the buffer size of the bottleneck link router by applying a bandwidth estimation technique known as *RCE* (Residual Capacity Estimator) [5], and set the parameter ssthresh.

4.1.1Router buffer problem of TCP Westwood:

TCP Westwood uses the minimum RTT (min *RTT*) to set the ssthresh, but this means that TCP Westwood does not consider RTT oscillation which happens when network begins to be congested. The fact that *RTT* relies on link delay (approximately min *RTT*) and buffering delay means that TCP Westwood performance will depend on buffer size of a bottleneck link router.

In this research, researcher investigated the friendliness between TCP Reno and TCP Westwood when they share the same bottleneck link. They firstly pointed out a problem through mathematical Analysis and simulations; the friendliness between TCP Reno and TCP Westwood is deteriorated according to buffer sizes of a bottleneck link router. That is, when the buffer size is smaller than the bandwidth delay product, throughput of TCP Reno is degraded. On the contrary, when the buffer size is larger than the bandwidth delay product, throughput of TCP Westwood is degraded by the TCP Reno connection. Here represent an improved version of TCP Westwood that achieves friendliness to TCP Reno. Key points are as follows (1) applying a bandwidth estimation technique, RCE, along with the original rate estimation technique, (2) estimating the buffer size of a bottleneck link router and deriving compensation parameters to force friendliness based on TCP throughput estimation models, and (3) updating the ssthresh parameter with the compensated RTTmin value. representd Simulation results show that represent scheme indeed achieves friendliness with TCP Reno versions without impact of router buffer sizes.

4.1.2 Throughput Ratio for Variants Buffer Sizes:

Here researcher evaluates the impact of buffer sizes on friendliness. The network topology consists of two sender hosts (S1 and S2), two receiver hosts (D1 and D2) and two routers (R1 and R2). Host S1 uses TCP Westwood scheme for data transmission, and host S2 using TCP Reno versions shares the same link between routers R1 and R2. That is, one connection of TCP Westwood scheme and another connection of TCP Reno versions compete on the bottleneck link. The bandwidth and the propagation delay of each link between the routers and sender/receiver hosts is 100[Mbps] and 5[ms]. The bandwidth and the propagation delay of the link between R1 and R2 is 50[Mbps] and 35[ms]. The total round trip delay between the sender hosts and the receiver hosts is 90[ms]. Here when assume the packet size is 1500 byte, the bandwidth delay product becomes 375[packets]. Researcher use a Tail Drop discipline for buffer management of router R1.Here researcher analyses throughput ratio in the steady state between TCP Reno versions and TCP Westwood scheme with variant buffer sizes. From the calculated ratio of average throughput of Reno and TCP Westwood it can be noted that TCP Westwood achieves friendliness to TCP Reno versions only when the buffer size of a bottleneck link router is set to 375[packets], that is the exact BDP in our simulation condition. However, when the buffer size is not set to 375[packets], friendliness with TCP Reno versions is deteriorated. other hand, it should be emphasized that TCP Westwood scheme achieves friendliness with TCP Reno versions for variant buffer sizes, because represented scheme adapts the min *RTT* according to the buffer size of a bottleneck link router.

From the calculated value of ratio we can see friendliness between TCP Reno and TCP Westwood is deteriorated according to buffer sizes of a bottleneck link router. That is, when the buffer size is smaller than the bandwidth delay product, throughput of TCP Reno is degraded. On the contrary, when the buffer size is larger than the bandwidth delay product, throughput of TCP Westwood is degraded by the TCP Reno connection.

4.2 Improve Slow Start threshold and congestion window:

TCP is able to work in wired, wireless as well as heterogeneous network. However, the bandwidth of such network may change frequently for many different reasons. Therefore, TCP needs to probe the extra bandwidth of a network to use the available bandwidth efficiently. Here represented scheme that improves the slow start state and the congestion avoidance state. Here represented scheme dynamically sets the slow start threshold and adjusts the congestion window in dynamic bandwidth environment.

4.2.1 Slow Start Threshold Estimation:

Here represented a slow start threshold estimation scheme that improves TCP performance during the slow start state. The ssthresh estimation dynamically adjusts the slow start threshold. The ssthresh estimation combines the expected rate and the actual rate to obtain an appropriate rate. Then the appropriate rate is used to obtain an appropriate ssthresh. The appropriate ssthresh can enhance TCP performance. Assume that RTTmin is the minimum RTT measured by the TCP source.

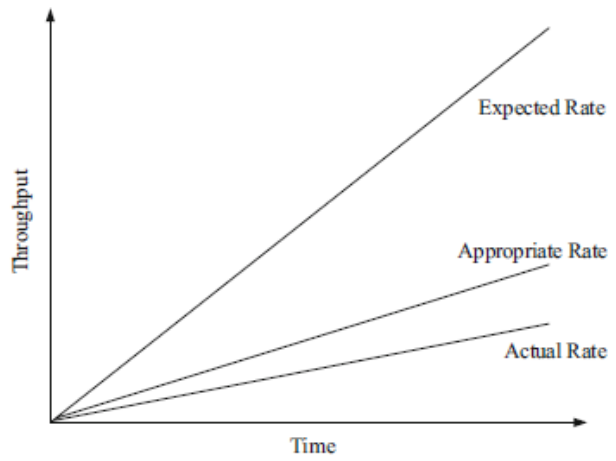

Figure. 4.1 Expected rate, appropriate rate, and actual rate with $\beta = 0.3$ [2]

We define the appropriate rate (AppR) as below, where AppR with $\beta = 0.3$ and $0 < \beta < 1$.

Expected Rate = cwnd/RTTmin
Actual Rate = cwnd/RTT
Appropriate Rate (AppR) = Expected Rate × β + Actual Rate × $(1 - \beta)$

Figure 4.1 shows the relationships among the expected rate, the actual rate, and the appropriate rate. If parameter β is close to 1, the appropriate rate would get closer to the expected rate. Therefore, the appropriate ssthresh would be set too high. On the other hand, if parameter β is close to 0, the appropriate ssthresh would be too conservative (small) to degrade TCP performance. Here set appropriate rate conservative by setting β to 0.3. If the appropriate rate is too large, ssthresh would be set too high. This would cause multiple packet loss if the exponential increase of cwnd generates too many packets too quickly.

4.2.2 Appropriate Congestion Window:

Here represented scheme, set ssthresh to Actual Rate× $(1-\beta)$ ×RTTmin/seg_size after the fast retransmission. When the timeout occurs, ssthresh is set to AppR × RTTmin/seg_size. In this state, it can detect the extra bandwidth via consecutive observation RTT (COR). COR can observe variations of RTT and determine if there are three consecutive decreases of RTT or three consecutive increases of RTT. The COR period is three ($P = 3$). For three consecutive decreases of RTT, we calculate the variation as follows [1]:

RTTdiff = RTTmax − RTTmin
Variation = RTTdiff/RTTmax,

Where RTTmax and RTTmin are the maximum and the minimum RTT measured by the TCP source, respectively. RTTdiff is the difference between RTTmax and RTTmin.

Here dynamically adjust *cwnd* in the congestion avoidance state according to the degree of variation of RTT. For three consecutive decreases of RTT, there define three cases of the next cwnd below [2].

$$\text{cwndnext} = \begin{cases} cwndcur + 1, \text{if Variation} < 1/3 \\ cwndcur + 3, \text{if } 1/3 \leq \text{Variation} < 2/3 \\ cwndcur + 5, \text{if Variation} \geq 2/3 \end{cases}$$

For three consecutive increases of RTT, calculate the variation in a same way as we calculated the variation for three consecutive decreases of RTT. For three consecutive increases of RTT, we define two cases of the next cwnd below [2].

$$\text{cwndnext} = \begin{cases} cwndcur + 1, \text{if Variation} < \frac{1}{2} \\ cwndcur, \text{if Variation} \geq \frac{1}{2} \end{cases}$$

4.3 Agile Probing:

Here represented [2] Agile Probing scheme improve the performance of TCP during startup and over large leaky pipe with the help of persistent non congestion detection technique. Agile probing and PNCD are cooperated into TCP Westwood to overcome the “slow” slow-start and inefficient window increase. In slow-start, agile probing is always used, while in congestion avoidance, it is invoked only after PNCD detects persistent non congestion.

4.3.1Agile Probing Mechanism:

Agile Probing uses Eligible Rate Estimation scheme to adaptively and repeatedly reset the value of ssthresh. During agile probing, when the current ssthresh is lower than Eligible Rate Estimation, the sender resets ssthresh higher accordingly, and increases cwnd exponentially. Otherwise, cwnd increases linearly to avoid overflow. In this way, agile probing probes the available network bandwidth for this connection, and allows the connection to eventually exit slow-start close to an ideal Window corresponding to its fair share of bandwidth.

4.3.2Persistent non congestion detection:

Here presented [2] a PNCD mechanism that aims at detecting extra available bandwidth and invoking agile probing accordingly. In congestion avoidance, a connection monitors the congestion level constantly. If a TCP sender detects persistent non congestion conditions, which indicates that the connection may be eligible for more bandwidth, the connection invokes agile probing to capture such bandwidth and improve utilization of available bandwidth.

5. Conclusion and future research:

In this paper we represent the brief over view of older version of TCP variants and their demerits. Here we also represent new version of TCP called TCP Westwood that improve the performance of TCP in lossy link where packets are drops due to link error .we also seen the fairness and friendliness issued of TCP Westwood. In presence of error

rate it is difficult to establish the fairness between TCP Westwood and other coexisting TCP Variants.

It was surveyed that TCP Westwood cannot efficiently use large amount of bandwidth that suddenly becomes available due to change in network conditions, random loss during slow-start that causes the connection to prematurely exit the slow-start phase.

There is a scope of improvement in refinement of bandwidth estimation and filtering method in order to improve the TCP Westwood friendliness in presence of error rate. ssthresh can be adjusted adaptively so the slow start phase can increase dynamically. It also improves the performance of TCP Westwood in congestion avoidance phase when large amount of bandwidth that suddenly become available. Improve the performance of TCP Westwood in wireless network where main reason for packet loss is link failure.

REFERENCES

[1] Shimaa Hagag, Ayman EI-Sayed(IEEE Senior Member) ,"Enhanced TCP Westwood Congestion Avoidance Mechanism(TCP WestwoodNew)",International Journal Of Computer Application, May-2012

[2] Neng-Chung Wang, Jong-Shin Chen, Yung-Fa Huang, Chi-LunChiou "Performance Enhancement Of TCP in Dynamic Bandwidth Wired and Wireless Network" Wireless Pers Commun, Springer Science+Business Media, March-2008

[3] Kenshin Yamada, Ren Wang, M. Y. Sanadidi, Mario Gerla "TCP With Sender-Side Intelligence to Handle Dynamic, Large, Leaky Pipes" IEEE Communication Society, Feb-2005

[4] Kazumi Kaneko, Jiro Katto "Reno Friendly TCP Westwood Based On Router Buffer Estimation"International Conference on Autonomic and Autonomous System and International Conference on Networking and Services, IEEE Computer Society, 2005

[5] Claudio Casetti, Mario Gerla, Saverio Mascolo, M.Y.Sanadidi,Ren Wang "TCP Westwood: End-to-End Congestion Control For Wired/Wireless Networks" Kluwer Academic Publisher, 2002.

[6] Ren Wang, Massimo Valla, M.Y.Sanadidi, Mario Gerla "Adaptive Bandwidth Share Estimation in TCP Westwood" UCLA Computer Science Department, Los Angeles, CA 90005, USA.

[7] S.Mascolo, C.Casetti, M.Gerla, S.S. Lee, M.Sanadidi "TCP Westwood: Congestion control with faster recovery"

[8] Mario Gerla, M.Y.Sanadidi, Ren Wang, Andrea Zanella "TCP Westwood: Congestion Window Control Using Bandwidth Estimation"

[9] Kevin Fall and Sally Floyd "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP"

[10] M.Allman, V.Paxson "TCP Congestion Control, RFC-2581" Network Working Group, April-1999

[11] S. Floyd , T. Henderson,U.C. Berkeley," The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 2582,April 1999

[12] Lawrence S. Brakmo, Sean. O'Malley, Larry L.Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance",Department of Computer Science The University of Arizona Tucson,AZ 85721, February 16,1994

[13] http://www.cs.ucla.edu/~nrl/hpi/tcpw/tcpw_ns2