A survey on effectiveness of TCP Westwood in mixed wired and wireless networks

Patel Kaushika, Rathod Jagdish M.

Abstract—Internet is expanding rapidly over the entire globe, TCP/IP is inseparable part of Internet. The increased use of wireless links for providing connectivity in remote areas and facilitating mobility in Internet brought out some serious performance issues of TCP, which was designed for wired links. TCP Westwood (TCPW) is a sender-side modification of the TCP congestion control algorithm that improves upon the performance of TCP Reno in wired as well as wireless networks. It continuously measure the bandwidth used by the connection by monitoring the rate of returning ACKs. The estimate is then used to compute congestion window and slow start threshold after a loss episode. This is to avoid over-shrinking cwnd after loss. In this paper the basic algorithm for TCP Westwood and it's characteristics like fairness friendliness are discussed. Also how error rate, RTT, buffer size and other parameters cause the effect on performance measures like throughput, congestion window utilization and ssthresh. Review of modifications done in basic algorithm based on different parameters is included with scope of improvement.

Index Terms—TCPW-TCP Westwood, ssthresh-slow start threshold, cwnd-congestion window, AI-Additive Increase, MD-Multiplicative decrease, bwe/BE- bandwidth estimation, Sack-Selective acknowledgement, RE-Rate Estimation, DUPACK-Duplicate acknowledgement, RTO-Retransmission Time Out.

1. INTRODUCTION

TCP/IP is a well known transport layer protocol, which is implemented in any network for process-to-process communication. It is a well-proven and accepted protocol suite, which has successfully ensured stable and robust network operations. TCP provides the reliable communication to web browsing and for file and e-mail Transfer. However, there are few performance issues when the conventional TCP [20] is employed in the Internet to operate over a wired/wireless network, large latency and channel noise impair performance of wireless Internet. According to several researches, it takes about 90% of all Internet traffic. TCP protocol designed and modified assuming the wired connection in Network because it provides error free connection as well as a low delay. So the congestion and server overload was considered as the main reasons for packet loss. The packet losses are detected in TCP by using a timer that triggers after the time which is twice the network's rtt (round trip time). Still in current TCP also there is no distiguation between the packet loss due to Congestion or due to corruption.[20] In network packet loss leads TCP to reduce its flow of data by reducing its congestion window (cwnd). TCP basically provides following services [20].

1. Full duplex service
2. Stream data service
3. Connection oriented service
4. Reliable services

2. TCP VARIANTS

TCP congestion control involves slow start and congestion avoidance phases. In order to improve the performance, several mitigation techniques have been suggested over standard TCP versions like NewReno and SACK TCP. The proactive schemes like, TCP Veno[21], [22] Westwood [13], [14] and TCP New Jersey[23] intend to improve flow control and avoid packet losses from estimation of certain network parameters. By improving the basic Tahoe TCP other versions invented. Tahoe TCP consist of slow start, congestion avoidance and fast retransmission algorithms. TCP Reno adds "fast recovery" to the Tahoe TCP as additional feature. TCP NewReno is a modification made in TCP Reno, where TCP sender retransmit the packet either on reception of three dupacks or expiration of retransmission timer. In case of Reno TCP three dupacks cause fast recovery to be called and fast recovery exits with new acknowledgement. Reno waits for a retransmit timer to get expired when multiple packets are lost from a window, in Reno, partial acks take TCP out of Fast Recovery by "deflating" the usable window back to the size of the congestion window. In New-Reno, partial acks do not take TCP out of Fast Recovery. Instead, partial acks received during Fast Recovery are treated as an indication that the packet immediately following the acknowledged packet in the sequence space has been lost, and should be retransmitted. Thus, when multiple packets are lost from a single window of data, New-Reno can recover without a retransmission timeout, New-Reno remains in Fast Recovery until all of the data outstanding when Fast Recovery was initiated will get acknowledged. It is selective acknowledgement which will give information about safe reaching of the packets out of order by SACK option. These TCPs are very conservative in reducing the cwnd. They consider the cause of drops as congestion only and reduce the cwnd and ssthresh to

- *Prof. Kaushika Patel is currently working as an assistant professor in Birla Vishwakarma Mahavidyalaya, India, E-mail:kdpatel@bvmengineering.ac.in*
- *Dr.Jagdish Rathod is currently working as an associate professor in Birla Vishwakarma Mahavidyalaya, India E-mail:jmrathod@bvmengineering.ac.in*

cwnd/2. TCP Westwood introduces "faster" recovery[15] to avoid over-shrinking cwnd after three duplicate ACKs by taking into account the end-to- end estimation of the bandwidth available to TCP. Therefore, modifications done to implement TCP Westwood are comparable to the ones implemented in the transition from TCP Tahoe to TCP Reno [15]. TCPW was aimed to improve performance under random or sporadic losses. This version was tested through simulation and showed considerable gain in terms of goodput in almost all scenarios.

3. AN OVERVIEW OF TCP WESTWOOD

S. Mascolo, C. Casetti, M. Gerla, S. S. Lee, M. Sanadidi at UCLA Computer Science Department, Los Angeles initiated research on TCP Westwood. It is mentioned that TCP Westwood (TCPW) is a sender-side modification of the TCP congestion window algorithm that improves upon the performance of TCP Reno in wired as well as wireless networks. The improvement is most significant in wireless networks with lossy links. In fact, TCPW performance is not very sensitive to random errors, while TCP Reno is very sensitive to random loss and congestion loss and it cannot discriminate between them. Hence, the tendencies of TCP Reno to over react to errors. An important distinguishing feature of TCP Westwood with respect to previous wireless TCP "extensions" is that it does not require inspection and/or interception of TCP packets at intermediate (proxy) nodes. Rather, TCPW fully complies with the end-to-end TCP design principle. The key innovative idea is to continuously measure at the TCP sender side the bandwidth used by the connection via monitoring the rate of returning ACKs. The estimate is then used to compute congestion window and slow start threshold after a congestion episode, that is, after three duplicate acknowledgments or after a timeout. The rationale of this strategy is simple: in contrast with TCP Reno which "blindly" halves the congestion window after three duplicate ACKs, TCP Westwood attempts to select a slow start threshold and a congestion window which are consistent with the effective bandwidth used at the time congestion is experienced. This mechanism is called faster recovery [15].

3.1 TCP Westwood Implementation

After 3 DUPACKS
If receiving 3 DUPACKS
Set ssthresh =(BWE*RTTmin) /seg_size;
and if cwnd > ssthresh
then set cwnd = ssthresh ;
Enter congestion avoidance
After Timeout
If RTO then
Set ssthresh = (BWE*RTTmin) /seg_size;
if (ssthresh < 2) ssthresh =2;
end if ;
cwnd = 1;
end if
enter slow start;

3.2 Bandwidth Estimation

Before a congestion episode, the used bandwidth is less or equal to the available bandwidth because the TCP source is still probing the network capacity. It is known that congestion occurs whenever the low-frequency input traffic rate exceeds the link capacity. So it is important to employ a low-pass filter to obtain the low-frequency components of the available bandwidth. The bandwidth Estimation is performed using a low-pass filter, as described by the following pseudo code

sample_BWE[k] = (acked*pkt_size*8)/(now -lastacktime)

BWE[k] =α_k* sample _BWE[k-1]+ (1-α_k)(sample _BWE[k] + sample _ BWE [k-1]))

Or in other words

$$\hat{b}_k = \alpha_k \hat{b}_{k-1} + (1-\alpha_k)\left(\frac{\hat{b}_k + \hat{b}_{k-1}}{2}\right) \quad (1)$$

$$\alpha_k = \frac{2\tau - \Delta t_k}{2\tau + \Delta t_k} \quad (2)$$

Cut off frequency=1/τ; $\hat{b}_k = \frac{d_k}{\Delta t_k}$ is the sample bandwidth

b^k the filtered bandwidth at time t_k; first order low-pass filter estimation.

α_k is the time-varying exponential filter coefficient at t_k

The approach chosen has taken care of two issues [15].

(1) The source must keep track of the number of DUPACKs it has received before new data is acknowledged.

(2) The source should be able to detect delayed ACKs and act accordingly.

3.3 TCP Westwood fairness and friendliness

TCP Westwood Fairness and Friendliness study for Lossless and lossy links has been shown in figures; here the bottleneck was created with 5 nodes with variable Reno/TCPW implemented as sources. The bottleneck bandwidth is set to 5Mbps, and the RTT is 100ms. Each TCPW connection added to the set in Fig. 1 achieves the same throughput that shows fairness. Fig. 2 illustrates two important points. First, superior performance of TCPW in high error rate environment. Second friendliness is preserved. In fact, the improvement shown by TCPW is due more to its ability to deal with wireless losses efficiently than to the "stealing" of bandwidth from Reno. [14] In presence of different error rates and propagation delay performence is evaluated [14], [15] by M.Gerla, M.Y.Sanadidi, Ren Wang and A.Zanella using network simulator NS-2, [17]. They compared throughput of Reno, Sack and TCPW as a function of error rates.

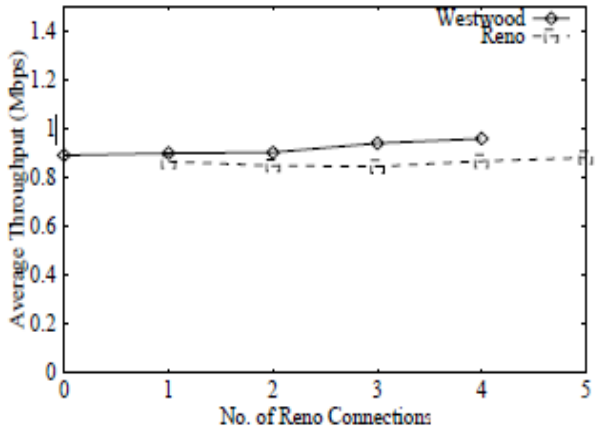


Fig. 1 Throughputs for error free link [14]

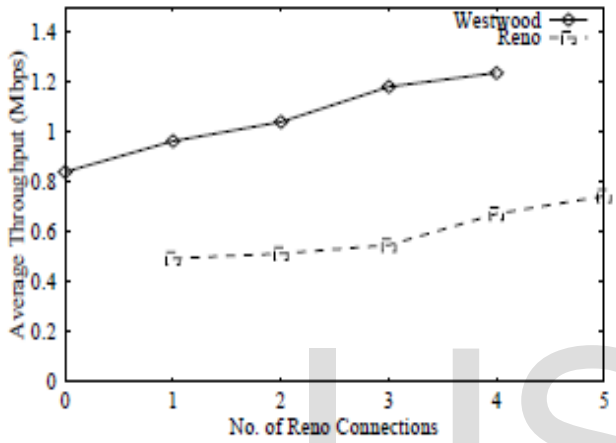


Fig. 2 Measured throughputs for lossy link [14]

The bottleneck bandwidth was 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. For small RTTs, Sack manages to recover fast enough so as to make up for its inherent lack of aggressiveness (when compared to Westwood). TCP Westwood, on the other hand, suffers from being too aggressive for small RTTs, and the resulting poorly-accurate bandwidth estimation forces it into slow start too often [14] [15].

4. EARLIER WORK DONE

Significant efficiency improvements have been obtained by TCPW using the bandwidth estimate (BE) produced by the sampling and filtering methods. This is particularly true in environments with large leaky pipes. BE is accurate in estimating a connection fair share. However, in drop-tail routers TCP traffic has been observed to be 'bursty', i.e. sending out a full window of packets and then waiting for the acknowledgements. In this situation, competing connections may 'take turns' in injecting their windows into the network, with the result that each basically sees the bottleneck as dedicated to itself and thus tends to over-estimate its fair share. Consider an alternative available bandwidth sample, defined as the amount of data reported to be delivered by all ACKs that arrived in the last T time units, divided by T. This method is called Rate Estimation (RE). This alternative is identical to the earlier TCPW sample definition if the ACKs are uniformly spaced in time. Simulation and measurements, however, show that ACKs tend to cluster in bursts. Thus, the BE sampling method 'overestimates' the connection fair share, while providing (in the bursty case) a reasonably good estimate of the instantly available bandwidth at the bottleneck. It turns out that BE is more effective in environments where friendliness (to other TCP connections) is not of concern, for instance, in the case of single connection operation or when random error/loss is significant and TCP NewReno is unable to take its fair share [9].

4.1 Rate Estimation

$$RE_k = \frac{\sum_{tj>tk-T} dj}{T} \quad (3)$$

Where dj is the amount of data reported by ACK j. Similarly, at the previous time instant, k-1; the sample k - 1 is

$$RE_{k-1} = \frac{\sum_{tj>tk-1-T} dj}{T} \quad (4)$$

Therefore

$$RE_k - RE_{k-1} = \left(\frac{\sum_{tj>tk-T} dj}{T} - \frac{\sum_{tj>tk-1-T} dj}{T}\right) \quad (5)$$

Which, on rearrangement,

$$RE_k = RE_{k-1} + \frac{1}{T}\left(\sum_{tj>tk-T} dj - \sum_{tj>tk-1-T} dj\right) \quad (6)$$

This is a desirable feature when we are dealing with bursty TCP traffic in presence of congestion. To calculate the Rate Estimate at the instant the kth ACK is received as:

$$RE_k = \alpha_k RE_k - 1 + (1-\alpha_k)\frac{RE_k + RE_{k-1}}{T} \quad (7)$$

4.1.1 Estimation with no random errors on the paths [9]

Fig. 3 shows comparison for the BE and RE estimations when there are no random errors on the paths. Experimental set up was a bottleneck of 5Mbps shared by TCPW and TCP NewReno. The graph was obtained applying the BE and RE estimators to TCPW. The network configuration has bottleneck bandwidth=5Mbps, RTT=70ms, T=4RTT. The link is error free and no packet is dropped in initial seconds due to congestion. As seen in Fig. 3, the BE is estimating about 3.6 Mbps which is larger than its 2.5 Mbps fair share, while the RE estimator is estimating exactly the fair share value. Fig. 4 shows BE and RE estimates when there are one TCPW flow and 4 NewReno flows sharing a 10 Mbps bottleneck. The same trend as in Fig. 3 (BE overestimates while RE oscillates around fair share) was observed.

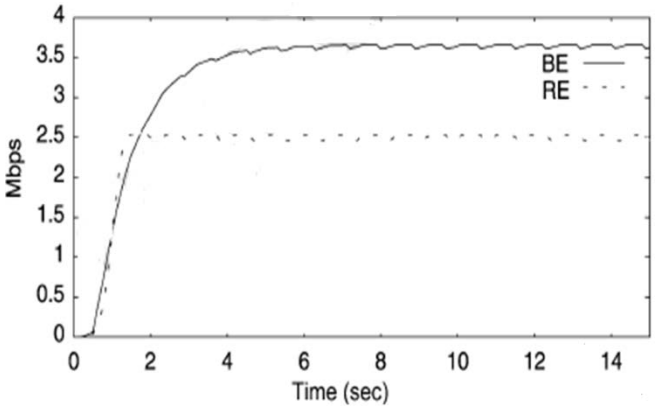


Fig. 3 BE and RE with concurrent New Reno [9]

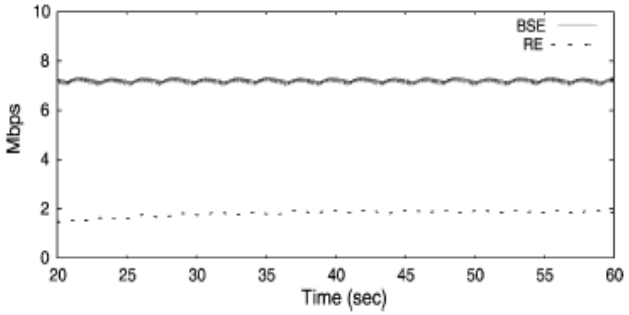


Fig. 4 BE and RE with multiple NewReno [9]

4.1.2 Estimation in presence of random errors on the path [9]

Experimental set up was as it was a bottleneck of 5Mbps shared by TCPW and TCP NewReno but random error of 0.5% was introduced. Results in Fig. 5 and Table 1 confirm that BE should be used when random error is the cause of packet losses. First, note that the fair share of TCPW is actually larger than C/2=2.5 Mbps in this case, since NewReno is inherently unable to utilize the link capacity. Through simulation result (seeTable 1), when two NewReno flows share the link, the throughput obtained by each flow is 1.4 Mbps. Thus, in this configuration, the TCPW flow can take as much as 3.6 Mbps out of the total 5 Mbps without harming NewReno performance. RE estimate settles at about 1.8 Mbps, underestimating its fair share, while BE estimates a more accurate share at 3.4 Mbps.

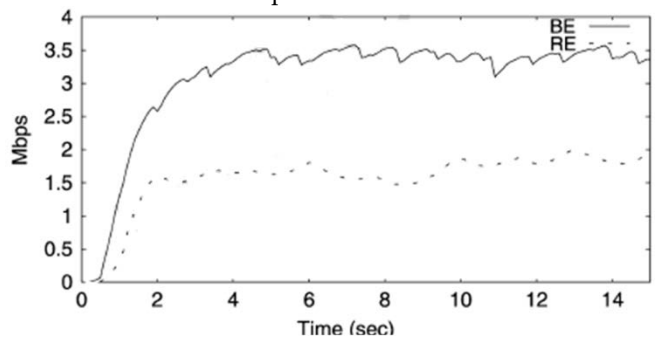


Fig. 5 BE and RE with concurrent NewReno error rate=0.5% [9]

Again, BE and RE estimates in the multiple connection Scenario was examined. Simulations run with one TCPW And four NewReno flows sharing bottleneck link of 10Mbps with random error rate of 1%. Estimates are

Table 1

Estimation and throughput (two flows) [9]

		NewReno and NewReno	TCPW BE and NewReno	TCPW RE and NewReno
Estimate (Mbps)	BE	–	3.4	–
	RE	–	–	1.8
Throughput (Mbps)	NewReno	1.4	1.4	1.4
	TCPW BE	–	2.42	–
	TCPW RE	–	–	1.76
	Total	2.8	3.82	3.16

provided by BE and RE are shown in Fig. 6; and Table 2 lists the average estimates and throughputs for the TCPW flow, and average throughputs for NewReno flows. Similar to the previous experiment, the fair share of TCPW is actually larger than C/5= 2 Mbps in this case. Through simulation (see Table 2), when five NewReno flows share the link, the average throughput obtained by each flow is 1.1 Mbps. Thus, in this configuration, the TCPW flow can take as much as 5.6 Mbps out of the total 10 Mbps without harming NewReno performance. Fig. 6 shows that RE estimates settle way below the potential fair share, while BE achieves a higher estimate. The throughput results also confirm that using BE is better in the random error case.

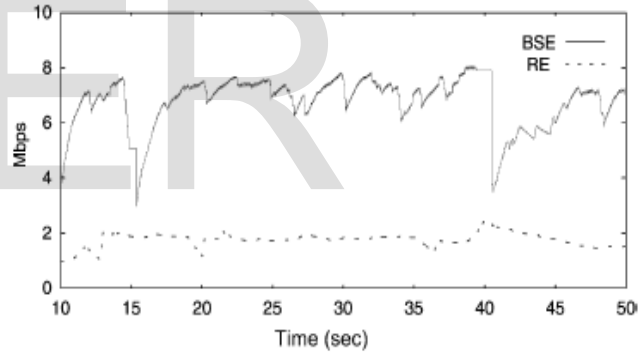


Fig. 6 BE and RE with multiple concurrent NewReno connections (error rate 1%) [9]

Table 2
Estimation and throughput (five flows) [9]

		All NewReno	TCPW BE and NewReno	TCPW RE and NewReno
Estimate (Mbps)	BE	–	6.7	–
	RE	–	–	2.1
Throughput (Mbps)	NewReno	1.1	1.04	1.12
	TCPW BE	–	3.62	–
	TCPW RE	–	–	1.78
	Total	5.49	7.77	6.27

From Table 2, TCPW BE and TCPW RE obtain 2.42 and 1.97 Mbps, respectively, while the competing TCP NewReno roughly sustains a 1.1 Mbps throughput which is equal to the fair share when five NewReno flows share the link. Thus, both TCPW BE and RE are friendly to

NewReno in this random error case. In addition TCPW BE is more efficient than TCPW RE.

In case of non-adaptive traffic like UDP the BE method is the better method to be used, and in fact, in all cases when random errors are the predominant cause of packet losses. It is important to identifying predominant cause of packet loss. And combination of rate estimation and bandwidth estimation was used for simulation experiments.

In case the loss indication is 3 DUPACKS, ssthresh and cwnd are set as follows

If (3 DUPACK)
If(cwnd/(RE*RTTmin)/seg_size)>θ)/*congestion*/
Ssthresh=(RE*RTTmin)/seg_size
Else
Ssthresh=(BE*RTTmin)/seg_size
Endif
If (cwnd>ssthresh)
Cwnd=ssthresh
Endif
Endif
θ=1.4

In case of a packet loss indicated by a timeout expiration, ssthresh and cwnd are set as follows

If (Time-out)
Cwnd=1
Ssthresh=(BE*RTTmin)/seg_size
If(ssthresh<2)
Ssthresh=2
Endif
Endif

4.2 TCP Westwood with Agile probing

It has been showen that TCP Westwood gives significant performance improvement over high-speed heterogeneous networks. The idea of TCPW is to use Eligible Rate Estimation (ERE) methods to set the congestion window (cwnd) and slow start threshold (ssthresh) after a packet loss as discussed. ERE is defined as the transmission rate a sender ought to use to achieve high utilization and remain friendly to other TCP variants. TCP Westwood with Agile probing (TCPW-A) is a sender side enhancement of TCPW. TCPW-A perform well when it was faced with highly dynamic bandwidth, large propagation time/bandwidth, and random loss in the heterogeneous Internet. TCPW-A achieves its goal by incorporating the following two mechanisms [6]

(1) When a connection initially begins or restarts after a timeout, instead of exponentially expanding cwnd to an arbitrary preset sthresh and then going into linear increase, TCPW-A uses agile probing, a mechanism that repeatedly resets ssthresh based on ERE and forces cwnd into an exponential climb each time. The result is fast convergence to a more appropriate ssthresh value.

(2) In congestion avoidance, TCPW-A invokes agile probing upon detection of persistent extra bandwidth via a scheme called persistent non congestion detection (PNCD). While in congestion avoidance, agile probing is actually invoked under the following conditions (1) large amount of bandwidth that suddenly becomes available due to change in network conditions. This can be done via scheme called Load Gauge. (2) Random loss during slow-start that causes the connection to prematurely exit the slow-start phase. [4]

Two basic mechanisms are involved here, first mechanism is Agile Probing, which is invoked at connection start-up (including after a time-out), and after extra available bandwidth is detected. Agile Probing adaptively and repeatedly resets ssthresh based on ERE. Each time the ssthresh is reset to a value higher than the current one, cwnd climbs exponentially to the new value. This way, the sender is able to grow cwnd efficiently (but conservatively) to the maximum value allowed by current conditions without overflowing the bottleneck buffer with multiple losses. The second mechanism is "Load Gauge". Load Gauge (LG) mechanism that aimed at monitoring extra available bandwidth and invoking Agile probing accordingly. cwnd/RTTmin indicates Expected Rate in no congestion and RE is the achieved rate. To be more precise, RE is the Achieved Rate corresponding to the Expected Rate 1.5 times RTT earlier. A comparison must be used, the corresponding Expected Rate, that is (cwnd -1.5)/RTTmin. RE tracks the Expected Rate in non-congestion conditions, but flattens, remaining close to the initial Expected Rate (ssthresh / RTTmin) under congestion. The Congestion Boundary is define as

Congestion Boundary=β* Expected Rate + (1-β) * I n i t i al Expected Rate; 0<β<1,
β=0.5.
if (in Congestion Avoidance except for the initial two RTT)
{ if(RE >Congestion Boundary)
{
no_congestion_counter++;
else if (no congestion_counter > 0){
no_congestion_counter--;
if (no_congstion_counter > cwnd){
re-start Agile Probing;
}else{
no_congestion_counter = 0;
}

If the parameter β is greater than 0.5, the Congestion Boundary line gets closer to Expected Rate. We can make this algorithm more conservative by setting β > 0.5. [4] [6]

TCP Reno and its versions are widely used in current networks, however it has been actualized that their throughput deteriorates in high-speed network and wireless environments. To overcome these problems of TCP Reno versions, a number of protocols have been proposed. In these, friendliness with TCP Reno becomes very important. In the situation where TCP Reno and new protocols connections share the same bottleneck link

and throughput of either one protocol is degraded. Among of them, in the case of TCP Westwood, when the buffer size of bottleneck link router is set to bandwidth delay product, TCP Westwood achieves friendliness with TCP Reno. When buffer size was set to smaller than bandwidth delay product, throughput of TCP Reno connections degrade. On the other hand, when buffer size is set to larger than bandwidth delay product, throughput of TCP Westwood connections is degraded by TCP Reno connection. Improved version of TCP Westwood overcomes unfriendliness of TCP Westwood according to buffer size of bottleneck link router. Buffer size of the bottleneck link router was estimated by applying a bandwidth estimation technique known as RCE (Residual Capacity Estimator). Rate estimation method is used for improvement. If TCP Westwood sender detects packet losses by duplicate ACK packets, cwnd and ssthresh are updated.

If loss is detected by retransmission Timeout expiration, cwnd=1 and ssthresh was calculated by RE.

Router buffer size can be calculated by

$$B = \frac{(AvgRTT * C)}{packetSize * 8} \quad (8)$$

TCP Westwood and TCP Reno are friendly to each other in case of buffer size

$$B = \frac{C * RTT\ \min}{packetSize * 8} \quad (9)$$

Residual Capacity Estimator (RCE) [5]

Residual Capacity Estimator (RCE) scheme estimates the bottleneck link capacity deducted by the uniformly distributed traffic present. The RCE scheme eliminates buffering times. The sender counts packets leaving to the receiver in retransmission time-out (RTO), and then the sender waits for correspondent returning ACK packets, where the time is set to ACKs_slot_time. Here, the sender calculates the wasted_time from the ACKs_slot_time standpoint. The sender measures an average of the interarrival time between the ACKs of ACKs_slot_time. The wasted_time is then computed as the sum of time exceeding the average in each inter arrival time of ACKs_slot_time. The exceeding gap times between ACKs are most likely a result of having periods of buffering. The sender has to compute the bottleneck link capacity by

$$C = \frac{Bits\_ACKed}{ACKs\_slot\_time - Wasted\_time} \quad (10)$$

Buffer size estimation

$$BS\_estimate = \frac{(RTT - RTT\ \min) * C}{packet\_size * 8} \quad (11)$$

$$diff\_delay = \frac{(BS\_estimate - BDP) * Packet\_size * 8}{C} \quad (12)$$

$$fair\_response = \frac{\sqrt{RTT - diff\_delay}}{\sqrt{(RTT - diff\_delay) - RTT_{\min}}} \quad (13)$$

And the ssthresh is calculated using compensated RTT_{min}

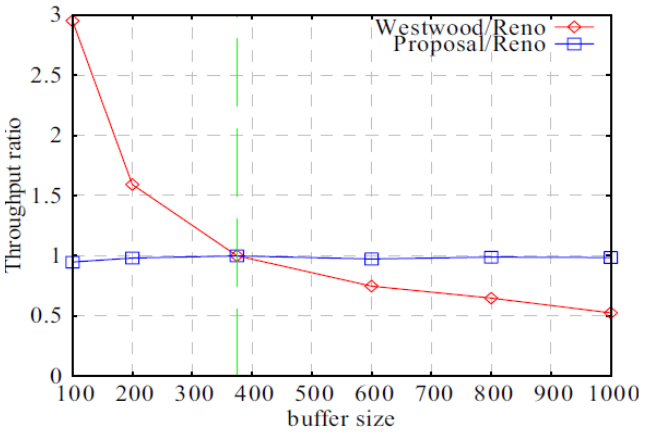
$$compensated\_RTT_{\min} = RTT - \frac{RTT}{(fair\_response)^2} = \frac{RTT}{RTT - diff\_delay} * RTT_{\min} \quad (14)$$

$$ssthresh = \frac{ERE * compensated\_RTT\ \min}{Packet\_size * 8} \quad (15)$$

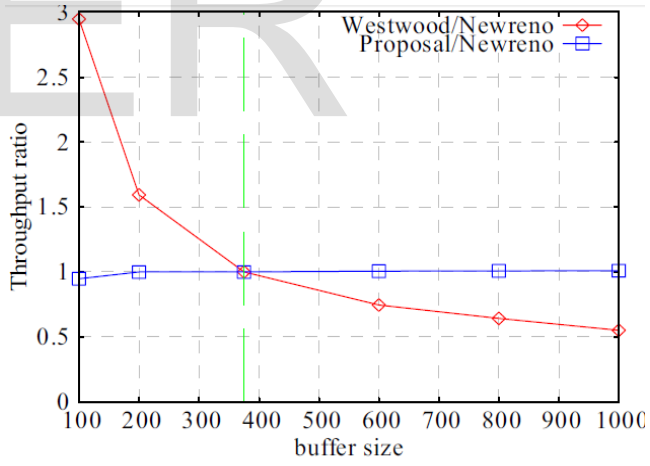
The performance was evaluated as throughput ratio [5]

$$Throughputratio = \frac{Westwood\_throughput}{legacyprotocol\_throughput} \quad (16)$$

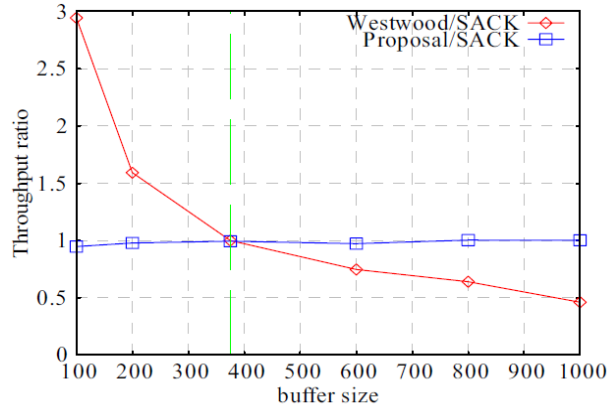
As shown in figure 7.



(a) TCP Reno



(b) TCP NewReno



(c) TCP SACK

Fig.7 (a), (b), (c) Throughput ratio between legacy protocols and TCP Westwood/proposal scheme with variant buffer size.

The appropriate ssthresh can enhance TCP performance. Assume that RTTmin is the minimum RTT. By setting the initial slow start thresh (ssthresh) of TCP to an arbitrary value, TCP performance may suffer from two potential problems as follows:

1. If ssthresh is set too high over the bandwidth of the links, the exponential increase of Congestion window size (cwnd) will generate too many packets too quickly, causing consecutive packet loss at the bottleneck router and lots timeouts.
2. If the initial ssthresh is set too low, the sender does not effectively utilize the exponential increase of the slow start state and switches prematurely to a linear increase of the congestion avoidance state. The utilization of a large bandwidth is not effective when the start up is poor. So it was required to set appropriate rate for flow.

The appropriate rate defined as (AppR) where AppR with $\beta = 0.3$ and $0 < \beta < 1$

$$Expectedrate = \frac{cwnd}{RTT\ \min} \quad (17)$$

$$Actualrate = cwnd / RTT \quad (18)$$

$$AppropriateRate(AppR) = \frac{ExpectedRate * B + ActualRate * (1 - B)}{RTT\ \min} \quad (19)$$

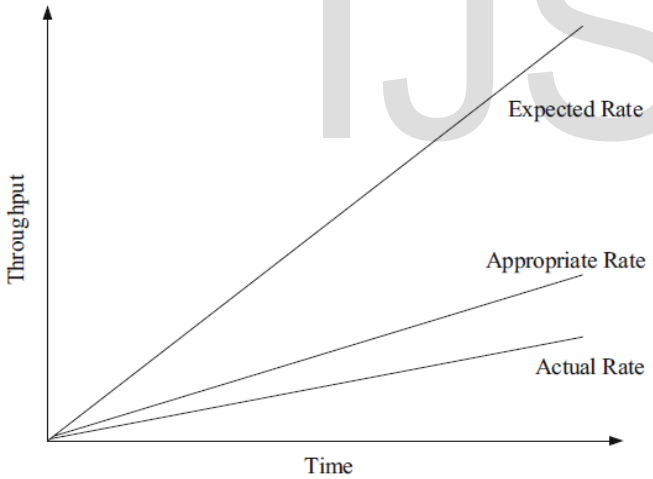


Fig. 8 Expected rate, appropriate rate, and actual rate with β = 0.3

Figure 8 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. In this paper, we make the 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. When a sender receives an ACK in the slow-start state, the pseudo code of the algorithm is as given. [3]

```
if (cwnd < ssthresh) /* slow start */
ssthresh = AppR × RTTmin/seg_size;
if (cwnd > sthresh) /*congestion avoidance */
cwnd = ssthresh;
endif
endif

if (timeout expires) /*slow start */
cwnd = 1;
ssthresh = AppR × RTTmin/seg_size;
if (ssthresh < 2)
ssthresh = 2;
endif
endif
```

When a timeout or a fast retransmission is in the congestion avoidance state, the pseudo code of the algorithm is as follows

```
If (fast retransmission executes)
ssthresh = Actual Rate × (1−β) × RTTmin/seg_size;
cwnd = ssthresh;
endif
if (timeout expires)
cwnd = 1;
ssthresh = AppR × RTTmin/seg_size;
if (ssthresh < 2)
sthresh = 2;
endif
endif
```

Appropriate Congestion Window Calculation

RTTdiff = RTTmax − RTTmin

Variation = RTTdiff/RTTmax,

For three consecutive decreases of RTT, three cases are defined of the next cwnd below.

cwndnext = cwndcur + 1, if Variation < 1/3

cwndnext =cwndcur + 3, if 1/3 ≤ Variation < 2/3

cwndnext =cwndcur + 5, if Variation ≥ 2/3

For three consecutive increases of RTT, cwndnext = _ cwndcur + 1, if Variation < 1/2

cwndnext = cwndcur , if Variation ≥ 1/2.

The discussed scheme is summarized in Fig. 9. This

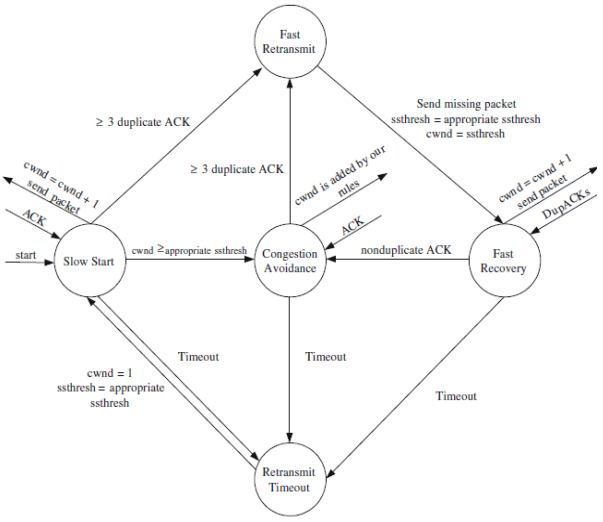


Fig.9 Congestion control diagram for the discussed scheme [3]

scheme helps a sender intelligently solve the issues of high dynamic bandwidth and long delay.
Another contribution of this work was the proposed observation RTT mechanism, which uses an aggressive congestion avoidance strategy. It observes variations of RTT and adjusts the cwnd effectively to improve performance.

CONCLUSIONS

TCP Westwood estimates bandwidth and adjusts the cwnd and ssthreh after loss detection. Sets bandwidth to the measured rate currently experienced by the connection, rather than using the conventional MD scheme. Literatures say experimental studies reveal improvements in throughput performance, as well as in fairness. In addition, friendliness with TCP Reno was observed in a set of research paper showing that TCP Reno connections are not starved by TCPW connections. Most importantly, TCPW is extremely effective in mixed wired and wireless networks. TCPW handles the losses caused by link errors. In presence of errors TCP Westwood outperforms without stealing large fraction of competing TCP. The BE sampling method 'overestimates' the connection fair share at bottleneck in case of lossless networks with bursty traffic. In general BE is more effective in environments where friendliness (to other TCP connections) is not of concern.

SCOPE OF IMPROVEMENT

Refinement in bandwidth estimation and filtering method can be done in order to improve TCP Westwood's performance. Present implementation calculate congestion window and ssthresh base on estimated bandwidth only on loss occurrences, this can be done on each packet reception. This can improve bandwidth utilization. Improvement in Friendlyness has scope. Based on the network load cwnd can be adjusted. Bandwidth Delay Product and buffer size comparison can be done. RTT can also be useful parameter in improving the performance. Modified algorithm can be implemented to validate on simulator and on real test beds for wired/wireless networks.

Acknowledgment

The author is thankful to Principal and Head of Electronics and Communication Department, CHARUSAT University,Gujarat- India for their support and encouragement during the research endeavour. We would also like to thank Principal and Head of Electronics Department, Birla Vishwakarma Mahavidhyalaya, Gujarat-India for all cooperation during the research work.

REFERENCES

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

[2] Kou Lan, Niu Sha"*A CMT Congestion Window Updates Mechanism Based on TCP Westwood*",IEEE International Conference on Mechatronic Science, Electric Engineering and Computer, August-2011, held at Jilin China.

[3] Neng-Chung Wang, Jong-Shin Chen, Yung-Fa Huang, Chi LunChiou "*Performance Enhancement Of TCP in Dynamic Bandwidth Wired and Wireless Network*", Springer Science+Business Media, Wireless Personal Communications: An International Journal archive Volume 47 Issue 3, November 2008 Pages 399 – 415.

[4] Kenshin Yamada, Ren Wang, M. Y. Sanadidi, Mario Gerla "*TCP With Sender-Side Intelligence to Handle Dynamic, Large, Leaky Pipes*" IEEE Journal on Selected Areas in Communications,Volume: 23 , Issue: 2 Page(s): 235 - 248 Feb-2005.

[5] 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.

[6] Kenshin Yamada, Ren Wang, M. Y. Sanadidi, Mario Gerla "*TCP Westwood with Agile Probing: Dealing with Dynamic, Large, Leaky Pipes*", in proceeding of: Communications, 2004, IEEE International Conference.

[7] S. Floyd, T. Henderson, A. Gurtov, "*The NewReno modification to TCP's fast recovery algorithm*", IETF RFC, 3782, April-2004.

[8] S. Floyd, T. Henderson, A. Gurtov, Y. Nishida "*The NewReno Modification to TCP's Fast Recovery Algorithm*", Internet Engineering Task Force RFC-6582 2004.

[9] M. Gerla, B.K.F. Ng, M.Y. Sanadidi, M. Valla, R. Wang "*TCP westwood with adaptive bandwidth estimation to improve efficiency/friendliness tradeoffs*", Journal of Elsevier Computer Communication volume 27 (1) (2004) 41_58.

[10] S. Floyd, T. Henderson, A. Gurtov "*The New Reno Modification to TCP's Fast Recovery Algorithm,RFC-3582*",Networking Working Group,April-2004.

[11] S. Floyd, T. Henderson, A. Gurtov, Y. Nishida "*The New Reno Modification to TCP's Fast Recovery Algorithm,RFC-6582*", Internet Engineering Task Force, 2004.

[12] Claudio Casetti, Mario Gerla, Saverio Mascolo, M.Y.Sanadidi, Ren Wang "*TCP Westwood: End-to-End Congestion Control For Wired/Wireless Networks*", Wireless Networks 8, 467–479, 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

[13] Ren Wang, Massimo Valla, M. Y. Sanadidi, Mario Gerla "*Adaptive Bandwidth Share Estimation in TCP Westwood*", Proc. IEEE Globecom 2002, Taipei, Taiwan, R.O.C., November 17-21, 2002.

[14] Mario Gerla, M. Y. Sanadidi, Ren Wang, and Andrea Zanella "*TCP Westwood: Congestion Window Control Using Bandwidth Estimation*", Proceedings of IEEE Globecom 2001, Volume: 3, pp 1698-1702, San Antonio, Texas, USA, November 25-29, 2001.

[15] S. Mascolo, C. Casetti, M. Gerla, S. S. Lee, M. Sanadidi "*TCP Westwood: Congestion control with faster recovery*",UCLA Computer Science Department, Los Angeles, CSD TR200017.

[16] K. Fall and S. Floyd, "*Simulation-based comparisons of Tahoe, Reno, and SACK TCP*", Newsletter , ACM SIGCOMM Computer Communication Review.

[17] *Ns-2, network simulator (ver.2).*LBL, URL: http://www-mash.cs.berkeley.edu/ns.

[18] *TCP Westwood modules for ns-2*:URL: http://www1.tcl.polito.it/casetti/tcp-westwood.

[19] J. B. Postel, "*RFC-793: Transmission Control Protocol"*, IETF, September 1981.

[20] A Book on "*TCP/IP Protocol Suite*" by Behrouz A. Forouzen.

[21] C. P. Fu, "TCP Veno: End-to-End Congestion Control Over Heterogeneous Networks," Ph.D. dissertation, The Chinese University, Hong Kong, 2001.

[22] Cheng Peng Fu, Soung C. Liew, TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks, IEEE Journal On Selected Areas In Communications, Vol. 21, No. 2, February 2003

[23] Kai Xu, Ye Tian, and Nirwan Ansari, "TCP-Jersey for Wireless IP Communications", IEEE Journal On Selected Areas In Communications, Vol. 22, No. 4, May 2004 747-756

[24] TCP Westwood Modules [Online]. Available: http://www1.tlc.polito.it/casetti/tcp-westwood