Performance Evaluation of TCP in the Presence of UDP in Heterogeneous Networks by using Network Simulator 2

Mr Devang G. Chavda [1] Prof. Ridhdhi I. Satoniya[2]
[1]M.E. Student [2]Assistant Professor
[1,2]Department of Electronics & Communication
[1]Darshan Institute of Engineering & Technology Rajkot. [2]Gujarat Technological University, Ahmedabad.

Abstract—The performances of different TCP variants like Tahoe, Reno, New Reno, Vegas and SACK in heterogeneous network is evaluated in this paper via simulation experiment using Network Simulator 2 (NS-2). In simulation experiments, investigation was done on the performance of different TCP variants in heterogeneous wired networks.

Key Words:- NS2, RTT, DUPACK, Congestion Window, Slow Start Threshold, 3 Dupacks.

I. INTRODUCTION

Web browsers use TCP when they connect to servers on the World Wide Web, and it is used to deliver email and transfer files from one location to another.

Applications that do not require the reliability of a TCP connection may instead use the connectionless User Datagram Protocol (UDP), which emphasizes low-overhead operation and reduced latency rather than error checking and delivery validation.

II. VARIOUS TCP FLAVOURS

The TCP flavors used to recover the loss are:

A. *Tcp Tahoe*

Tahoe detects packet losses by timeouts and then retransmit the lost packets. A packet loss is taken as a sign of congestion and Tahoe saves the half of the current window as *ssthresh* value. It then set *cwnd* to one and starts slow start until it reaches the threshold value. After that it increments linearly until it encounters a packet loss. Thus it increase it window slowly as it approaches the bandwidth capacity.

– *Problem*

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.

B. *Tcp Reno*

Reno retains the basic principle of Tahoe. However it adds some intelligence over it so that lost packets are detected earlier and the pipeline is not emptied every time a packet is lost. Modification in Reno is whenever it receives 3 Dupacks it goes in to fast retransmit – fast recovery phase. Another modification in Reno is that after a packet loss, it does not reduce the congestion window to 1. The basic algorithm is presented as under:

1. Each time it receive 3 Dupacks it take that to mean that the segment was lost and we retransmit the segment immediately and enters in to 'Fast recovery'
2. Set *ssthresh* to half the current window size and also set *cwnd* to the same value.
3. For each Dupack receive increase *cwnd* by one. If the increase *cwnd* is greater than the amount of data in the pipe then transmit a new segment else wait. If there are 'w' segments in the window and one is lost, then it will receive (w-1) Dupacks. Since *cwnd* is reduced to W/2, therefore half a window of data is acknowledged before it can send a new segment. Once it retransmits a segment, it would have to wait for at least one RTT before it would receive a fresh acknowledgement. Whenever it receives a fresh ACK it reduces the *cwnd* to *ssthresh*. If it had previously received (w-1) Dupacks then at this point it should have exactly w/2 segments in the pipe which is equal to what it set the *cwnd* to be at the end of fast recovery. Thus it doesn't empty the pipe, it just reduce the flow. It continues with congestion avoidance phase of Tahoe after that.

– *Problems*

Reno performs very well over TCP when the packet losses are small. But when we have multiple packet losses in one window then Reno doesn't perform too well and its performance is almost the same as Tahoe under conditions of high packet loss. The reason is that it can only detect single packet loss. If there is multiple packet loss then the first information about the packet loss comes when it receives the 3 Dupacks. But the information about the second packet which was lost will come only after the ACK for the retransmitted first segment reaches the sender after one RTT. Also it is possible that the *cwnd* is reduced twice for packet losses which occurred in one window.

C. *Tcp Newreno*

NewReno 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 Dupacks, 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 - retransmit phase is the same as in Reno. The difference in the fast recovery phase which allows for multiple retransmissions 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 it 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 Dupacks received to zero. It exits Fast recovery when all the data in the window is acknowledged.
- *Problems*

NewReno 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 it can deduce which other segment was lost.

D. *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. It also has the coarse grained timeout of Tahoe to fall back on, in case a packet loss is not detected by the modified algorithm.

SACK TCP requires that segments not be acknowledged cumulatively but should be acknowledged selectively. 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. Whenever the sender enters fast recovery, it initializes a variable pipe which is an estimate of how much data is outstanding in the network, and it also set *cwnd* to half the current size. Every time it receives an ACK it reduces the pipe by 1 and every time it retransmits a segment it increments it by 1. Whenever the pipe goes smaller than the *cwnd* window it checks which segments are un received and send them. If there are no such segments outstanding then it sends a new packet. Thus more than one lost segment can be sent in one RTT.

- Problems

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.

E. *Tcp Vegas*

Vegas is a TCP implementation which is a modification of Reno. It builds on the fact that proactive measure to encounter congestion is much more efficient than reactive ones. It tried to get around the problem of coarse grain timeouts by suggesting an algorithm which checks for timeouts at a very efficient schedule. Also it overcomes the problem of requiring enough duplicate acknowledgements to detect a packet loss, and it also suggests a modified slow start algorithm which prevents it from congesting the network. It does not depend solely on packet loss as a sign of congestion. It detects congestion before the packet losses occur. However it still retains the other mechanism of Reno and Tahoe, and a packet loss can still be detected by the coarse grain timeout of the other mechanisms fail.

The three major changes induced by Vegas are:

1) New Re-Transmission Mechanism

Vegas extend on the re-transmission mechanism of Reno. 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**[6]**. Thus it gets around the problem faced by Reno of not being able to detect lost packets when it had a small window and it didn't receive enough duplicate Ack's. To catch any other segments that may have been lost prior to the retransmission, when a non-duplicate acknowledgment is received, if it is the first or second one after a fresh acknowledgement then it again checks the timeout values and if the segment time since it was sent exceeds the timeout value then it re-transmits the segment without waiting for a duplicate acknowledgment. Thus in this way Vegas can detect multiple packet losses. Also it only reduces its window if the re-transmitted segment was sent after the last decrease. Thus it also overcome Reno's shortcoming of reducing the congestion window multiple time when multiple packets are lost.

2) Congestion avoidance

TCP Vegas is different from all the other implementation in its behaviour during congestion avoidance. It does not use the loss of segment to signal that there is congestion. It determines congestion by a decrease in sending rate as compared to the expected rate, as result of large queues building up in the routers. It uses a variation of Wang and Crowcroft;s Tri-S scheme. Thus whenever the calculated rate is too far away from the expected rate it increases transmissions to make use of the available bandwidth, whenever the calculated rate comes too close to the expected value it decreases its transmission to prevent over saturating the bandwidth. Thus Vegas combats congestion quite effectively and doesn't waste bandwidth by transmitting at too high a data rate and creating congestion and then cutting back, which the other algorithms do.

3) Modified Slow-start

TCP Vegas differs from the other algorithms during its slow-start phase. The reason for this modification is that when a connection first starts it has no idea of the available bandwidth and it is possible that during exponential increase it over shoots the bandwidth by a big amount and thus induces congestion. To this end Vegas increases exponentially only every other RTT, between that it calculates the actual sending through put to the expected and when the difference goes above a certain threshold it exits slow start and enters the congestion avoidance phase.

- Problems

TCP Vegas has the amazing property of rate stabilization in a steady state, which can significantly improve the overall throughput of a TCP flow. Unfortunately, notwithstanding this and other advantages, later research discovered a number of issues, including underestimates available network resources in some environments (e.g., in the case of multipath routing) and has a bias to new streams (i.e. newcomers get a bigger share) due to inaccurate RTT_{min} estimates.

III. Simulation Setup

The simulation tool used for this simulation experiment is ns-2. The simulation environment was done on the heterogeneous wired networks.

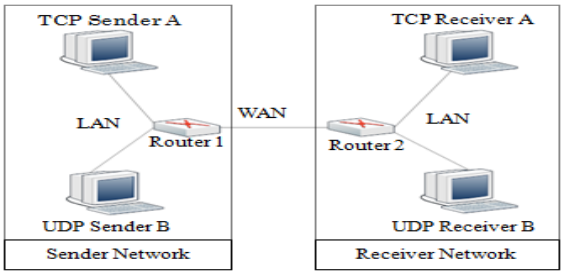

Fig. 1: Dumbbell shape wired Network

The sender links and the receiver links are a full wired duplex link. The bandwidth of the sender links that implement Ethernet LAN is 10Mbps and the link delay in the full duplex Ethernet LAN is set to 10ms. The bandwidth of the sender links that implement Fast Ethernet LAN is 100Mbps and the link delay of the full duplex wired Fast Ethernet LAN is set to 1ms. The bottleneck link is a full wired duplex link with the capacity of 2Mbps that represents current bandwidth of MyREN network. The link delay of the bottleneck link is set to 50ms. The simulation parameters of the network topology are showed in Table II.

The network topology used in the simulation experiment is shown in Figure 1. The network topology used is a dumbbell topology which consists of one TCP sender, one TCP receiver, one UDP sender, one UDP receiver, and a pair of routers. The links between the TCP/UDP sender and router 1 are called sender links while the links between the TCP/UDP receivers and the router 2 are called the receiver links. The sender and receiver link represent a local area network (LAN). The link between router 1 and router 2 is called the bottleneck link. The bottleneck link represents a wide area network (WAN). For heterogeneous wired networks, we set four simulation environments as shown in Table I.

TABLE I
Simulation Environments In Heterogeneous Wired Network

Simulation Environment	TCP Sender (A&B)	TCP Receiver (A&B)
1	Ethernet	Ethernet
2	Ethernet	Fast Ethernet
3	Fast Ethernet	Fast Ethernet
4	Fast Ethernet	Ethernet

TABLE II
Simulation Parameters

Link	Bandwidth	Delay
Bottleneck	2 Mbps	50ms
Ethernet	10 Mbps	10ms
Fast Ethernet	100 Mbps	1ms

TABLE III
Heterogeneous Wired Network

Case	TCP Sender A	Sender B
1	Tahoe	UDP
2	Reno	UDP
3	New Reno	UDP
4	SACK	UDP
5	Vegas	UDP

In heterogeneous network, for flow A, The TCP source used on the sender side in the network is varied from Tahoe, Reno, New Reno, Vegas and SACK. For the receiver side, we set TCP Sink as the TCP source. for flow B, the sending rate of the UDP source is varied. There are four cases considered in the simulation experiments of heterogeneous wired network as shown in Table III. The performance of TCP flavors versus different UDP rates evaluated in order to investigate the throughput.

IV. Results And Discussion

The performance of TCP variants against different UDP rates in heterogeneous network where the bottleneck link is shared is shown here. The throughput of TCP variants is analyzed.

TABLE IV
Throughput Results For Ethernet To Ethernet Simulation Environment

UDP rate (Kb)	Tahoe (Kbps)	Reno (Kbps)	New Reno (Kbps)	Vegas (Kbps)	Sack (Kbps)
400	1096.0	1096.0	1096.0	1027.2	1096.0
800	1093.3	1093.3	1093.3	1025.8	1093.3
1200	783.4	797.8	794.1	796.2	754.7
1600	556.67	592.7	594.2	499.1	607.6
1800	551.9	405.5	443.5	386.1	499.4

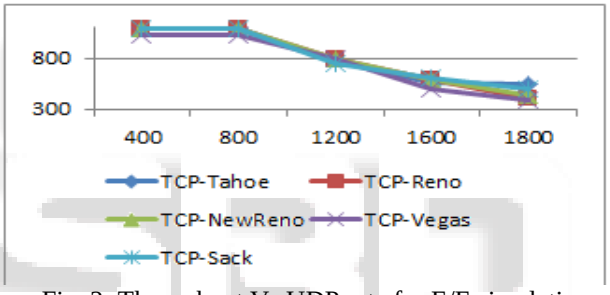

Fig. 2: Throughput Vs UDP rate for E/E simulation Environment

TABLE V
Throughput Results For Ethernet To Fast Ethernet Simulation Environment

UDP rate (Kb)	Tahoe (Kbps)	Reno (Kbps)	New Reno (Kbps)	Vegas (Kbps)	Sack (Kbps)
400	1263.4	1263.5	1263.5	1189.0	1263.5
800	1206.1	1206.1	1206.1	1156.5	1206.1
1200	909.9	830.3	869.5	863.4	862.3
1600	692.0	592.0	687.01	340.9	636.2
1800	539.6	541.5	522.28	243.7	537.1

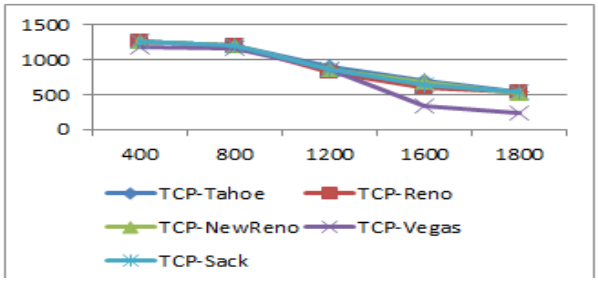

Fig. 3: Throughput Vs UDP rate for E/F simulation Environment

TABLE VI
Throughput Results For Fast Ethernet To Ethernet Simulation Environment

UDP	Tahoe	Reno	New	Vegas	Sack

rate (Kb)	(Kbps)	(Kbps)	Reno (Kbps)	(Kbps)	(Kbps)
400	1263.6	1263.6	1263.6	1189.1	1263.6
800	1206.0	1206.0	1206.0	1156.5	1206.0
1200	901.6	803.9	901.6	857.0	848.9
1600	693.4	677.8	681.9	310.0	665.1
1800	575.5	584.0	501.0	245.1	549.4

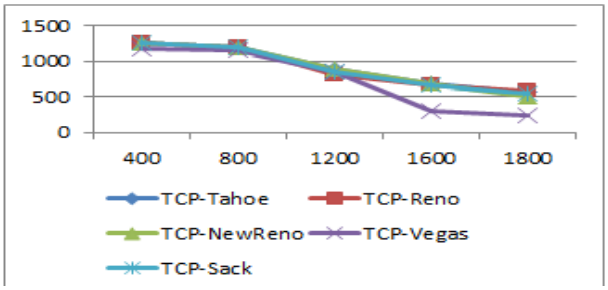

Fig. 4: Throughput Vs UDP rate for F/E simulation Environment

TABLE VII
Throughput Results For Fast Ethernet To Fast Ethernet Simulation Environment

UDP rate (Kb)	Tahoe (Kbps)	Reno (Kbps)	New Reno (Kbps)	Vegas (Kbps)	Sack (Kbps)
400	1484.9	1486.8	1486.8	1404.2	1484.9
800	1311.7	1391.5	1391.5	1254.8	1311.7
1200	999.6	1138.3	1085.0	954.3	950.2
1600	758.4	872.2	975.5	702.5	737.9
1800	653.0	791.1	868.2	273.9	675.6

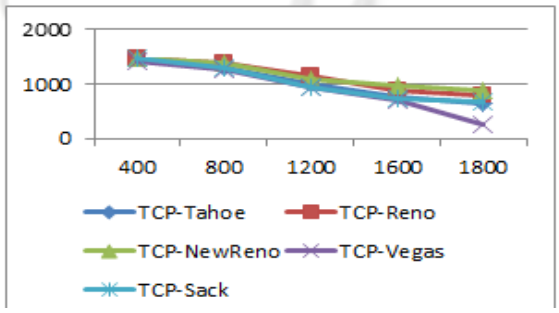

Fig. 5: Throughput Vs UDP rate for F/F simulation Environment

For all the simulation experiments in the heterogeneous wired network, TCP Vegas is unable to achieve fair bandwidth allocation when sharing the bottleneck link with UDP. TCP Vegas receives lower throughput compared with TCP Tahoe, TCP Reno, TCP NewReno and TCP SACK. The congestion controlling mechanisms implemented in TCP NewReno are slightly the same as TCP Reno except that TCP NewReno is able to detect multiple packet losses. Hence the results of TCP Reno and TCP NewReno do not vary much. In order to fully utilize the available bandwidth of the bottleneck link, TCP Reno and TCP NewReno continue to increase the window size until packet losses occur. As window size increases, the buffer size of TCP Reno and TCP NewReno in the bottleneck link also increases. The larger buffer size keeps more packets and hence dominates most of the available bandwidth in the bottleneck link. Thus, TCP Reno and TCP NewReno connections received higher throughput as the buffer size is larger. On the other hand, TCP Vegas congestion avoidance mechanisms detect congestion at early stage. TCP Vegas detects congestion faster than TCP Reno and TCP NewReno. TCP Vegas congestion avoidance mechanism reduces the window size in order to maintain a smaller buffer queue size. The smaller buffer size minimizes the packet losses due to buffer overflows. Thus, the TCP Vegas uses smaller bandwidth capacity in the bottleneck link. This concludes why throughput of TCP Vegas is significantly low when sharing the bottleneck link with UDP. The simulation result in the heterogeneous wired network is tabulated in Table IV, V, VI and VII.

V. CONCLUSION

This simulation experiment analyse the performances of TCP variants versus different UDP rates in term of throughput in heterogeneous wired networks.
From our simulation results,

- For Ethernet to Ethernet simulation environment TCP Tahoe achieves higher throughput.
- For Ethernet to Fast Ethernet simulation environment TCP Tahoe and Reno (at UDP rate 1800kbps) achieves higher throughput.
- For Fast Ethernet to Ethernet simulation environment TCP Tahoe and Reno (at UDP rate 1800kbps) achieves higher throughput.
- For Fast Ethernet to Fast Ethernet simulation environment TCP New Reno achieves higher throughput at higher UDP rate.

VI. FUTURE WORK

This simulation experiments analyse the performance of TCP variants versus different UDP rates in term of throughput in heterogeneous networks connection. The results of this simulation experiments prepares a comparison medium for the performance evaluation of enhancement in TCP in networks.

ACKNOWLEDGEMENT

I would like to thank Darshan Institute of Engineering and Technology, Rajkot for the support on my research and this research paper.

REFERENCES

[1] Jacobson, V. "Congestion Avoidance And Control". In Proceedings Of Sigcomm'88 (Stanford, Ca, August 1988), Acm.
[2] Alexander Afanasyev, Neil Tilley, Peter Reiher, And Leonard Kleinrock, "Host-To-Host Congestion Control For Tcp", Ieee Communications Surveys & Tutorials, March 2010.
[3] S.Floyd, T.Henderson "The New-Reno Modification To Tcp's Fast Recovery Algorithm" Rfc 2582, April 1999..
[4] Yakubu Friday, "Performance Analysis Of Tcp In A Multiple Reliable Connections Environment With Udp Traffic Flows Using Opnet Simulator", Ahmadu Bello University, Zaria Nigeria, March, 2011.
[5] Dr. Neeranjan Bhargava, Dr. Ritu Bhargava, Anchal Kumawat, Bharat Kumar, "Performance Of Tcp-Throughput On Ns2 By Using Different Simulation Parameters" International Journal Of Advance Computer Research(Issn(Print):2249-7277, Issn (Online) : 2277-7970), Volume-2 Number-4 Issue-6 December-2012.

[6] Ajay Singh, Dr. Pankaj Dashore, "A Comparative Study Of Udp And Tcp By Using Ns2", International Journal Of Computer Science And Information Security (Ijcsis) Volume (1): Issue(1), June 2013.

