Simulation Based Performance Evaluation of TCP Variants along with UDP Flow

Analysis of Throughput with respect to Delay, Buffer Size and Time

Azeem Aftab[1], Azfar Ghani[2], Zain-ul-Abedin Abidi[3], Mohtashim Baqar[4]
IQRA University (Main Campus), Karachi, Pakistan[1, 2, 3]
National University of Sciences and Technology, Sector H-12, Islamabad, Pakistan[4]
Email: azeem.cheema@iqra.edu.pk[1], azfarghani@iqra.edu.pk[2], zainabidi@iqra.edu.pk[3], mohtashimbq@pnec.nust.edu.pk[4]

Abstract—**In a data communication network, several flows contend to utilize limited and shared network resources, however, this gives rise to congestion along with other network issues. So, this leads to the development of congestion control based methods/ protocols to support the deployment of real-time multimedia applications while ensuring reliability and fairness. This paper presents a simulation based study, analyzing variants of TCP i.e. TCP Tahoe, TCP Reno and TCP New Reno. In this work a simulator, network simulator-3, NS-3.25, has been used to analyze the throughput for different TCP flows in presence of a UDP flow running under a bad-sensitive and lossy environment. Further, performance is evaluated on the basis of simulation results obtained under a realistic scenario built in NS-3, emulating different background traffics, link delays, bottleneck link, limited queue size, and with different link bandwidths. The topology laid is simulated for two different queue management techniques i.e. Drop Tail (DT) and Random Early Drop (RED), operating while utilizing different congestion avoidance capabilities. Goal is to evaluate protocols in stiff conditions while evaluating each protocol based on different performance metrics. Observed experimental results have shown that TCP New Reno is more robust and has given persistent performance levels in terms of throughput in a constrained environment.**

Keywords: **NS-3; TCP Flavor; Congestion Control and Avoidance; TCP New Reno; Throughput**

I. INTRODUCTION

TCP, Transmission Control Protocol is a connection-oriented transmission protocol which ensures end-to-end delivery and reliability. It ensures reliability via mechanism of acknowledgements [1]. It requires the receiver to send an acknowledgement of the received packet to the sender. Since in reality, the network is not perfect, so we don't have 100 percent successful reception rate. A fair chunk of packets are lost, how many? Well, it depends upon how good or bad the network's performance levels are. Moreover, this infers that the loss of information or data is quite evident. To reduce or minimize the losses different flow control techniques are employed. Further, TCP has different variants dealing with the packet loss that is because of hindrance in the medium or the inability of the transceiver to ensure reliability and flow control by themselves [2]. Although, availability of routers with greater buffer sizes have played a role in reducing the packet drop rate but at the expense of increased delay, jitter, power consumption and cost [3]. However, the question on the optimum value of buffer size is still answered after more than a decade of research [4]. With high-end applications demanding network resources at full throttle, larger buffer sizes are desirable. This has driven a lot of research in the said area. In most cases, open-loop traffic models are used for modeling of networks, where the process of packet arrival and state of the queue are not related. In [5], authors used general Gaussian process to model the input traffic and derive loss probability. In [6], authors massively emphasized that the size of buffer should be equivalent to bandwidth-delay product of a link, where delay of a packet appertain to its round trip time (RTT). Similarly in related works [7]–[10], authors have investigated to formulate criteria to find an optimum size of router buffer based on network parameters.

In this work, three different TCP flavours i.e. TCP Tahoe, TCP Reno and TCP New Reno, have been compared. The comparisons are based on experimental results obtained via simulations conducted using NS-3, network simulator, version 3.25. Moreover, a lossy network environment was created via inserting propagation delays in the medium and by flooding the network with File Transfer Protocol (FTP) traffic for TCP flows and Constant Bit-Rate (CBR) traffic for UDP flow. Also, a constraint of variable buffer size was also inserted to stress test the behavior of TCP variants and in the end results were formulated on basis of observed throughput. Two congestion avoidance techniques were used in these experiments i.e. Random Early Drop (RED) and Drop Tail (DT). Further, details of TCP variants employed are briefly discussed in the following [2] [11] [12].

A. *TCP Tahoe*

Tahoe referred to as a TCP congestion control algorithm implemented with slow start (SS), congestion avoidance and fast retransmit algorithms. It was presented by Van Jacobson in his research paper [13]. TCP Tahoe figures that if a connection is consuming all the available bandwidth then no further packets should be infused into the network as it will trigger packet losses in the network. It ensures flow control by using acknowledgements to keep track of outgoing packets. It also maintains congestion window (CWD) to manage flow control and efficiently utilize channel capacity. Moreover, to maintain this equilibrium condition some issues need to be taken care of and they are as follows.

1) Determine the Bandwidth
2) Ensuring balance is maintained.
3) Proactive measures to congestion

1) Slow Start and Congestion Avoidance: Incoming acknowledgements trigger the TCP packet transmission. However, there occurs a problem in initiating the connection, we can only have ACKs if there is any data in the network and to insert data into the network we must have acknowledgements. To overcome this problem Tahoe gives a solution that when we start or re-start a TCP connection after packet loss then either it should build from scratch or should come through a method called 'slow-start'. It is used in simultaneity with other algorithms to send data that can be effectively transmitted through the network, so congestion can be avoided. Further, an initial burst might put stranglehold onto the network which can cause the connection to never get started. In slow start the congestion window is set to 1 initially and for the next received acknowledgement the congestion window is increased by 1. So in the first round trip time (RTT) we transmit a single packet, in the second round trip time we transmit 2 packets, then 4 and so on, which means the size of the window has been doubled after every round trip time (RTT). Thus, we increase exponentially until we met our threshold or we lose a packet which means congestion. Also, when we reach congestion we decrease our sending rate and also decrease congestion window back to one, and repeat the process. Tahoe detects packet loss by time-outs. Practically, we regularly check for time-outs because the repeated interrupts are costly, due to which it might take some time before a packet loss can be detected and retransmitted. In order to avoid congestion a technique called 'Additive Increase/Multiplicative Decrease (AIMD)' is used, this technique is also used in the best effort network or in the general internet system.

2) Problems: The problem with TCP Tahoe is that it takes a complete time-out period in order to detect a packet loss, even sometimes it takes even longer than the time-out time because of the coarse grain time-outs. Further, it sends cumulative acknowledgements rather than sending immediate acknowledgements. So whenever a packet is lost, it will have to wait until the occurrence of time-out and then the buffer is emptied, which obviously cause extraordinary delays in the connection.

B. TCP Reno

TCP Reno follows the same principle of slow start and the coarse grain re-transmit timer as in TCP tahoe. Also, to overcome the problems of TCP tahoe it includes some intelligence to detect the lost packets in earlier stages. Further, to avoid emptying pipeline whenever a packet loss occurs, in Reno we get prompt acknowledgement at whatever point a segment is received. The rationale is that whenever a duplicate acknowledgement is received, then it should have only been received if the following segment in arrangement has been deferred in the network and it reaches out of order or else the packet is lost. Further, if we get various duplicate ACKs then it implies that adequate time has passed and regardless of the fact that the segment had taken longer path, it ought to have gotten to the receiver at this point, although there is a low probability that the segment will reach the destination unaffected. For this Reno suggest 'Fast Re-Transmit' algorithm, that is, when we get three duplicate acknowledgements, it indicates that the segment was lost, so without waiting for the time-out we transmit the segment once again. In this manner, while the pipeline is nearly full, the re-transmission of the segment can be managed. Another advantage of Reno over tahoe is that the congestion window does not reduce to 1 once a packet loss occurs rather it enters into 'Fast-Re Transmit' phase [14]. Further, the said technique states that each time we receive three duplicate acknowledgements, which implies the lost of the segment, so we immediately perform re-transmission of the segment and get into 'Fast-Recovery' phase. After that we set SS threshold and CWD to half the current window size and for each received duplicate acknowledgement, CWD is incremented by one. In the event that the increment in congestion window is more than the amount of information in the channel then a new segment is transmitted else it will further wait. Also, if the total number of segments in the window is 'n' and one of them is lost, then (n-1) duplicate acknowledgements (ACKs) will be received. In addition, as congestion window is decreased to 50% of its actual, so before we send a new segment, half a window of data will be acknowledged. Also, when transmitting a segment, we would need to wait for at least one RTT before we receive a new ACK. At whatever point we get a new acknowledgement we decrease the CWD to SS threshold. Now, exactly *'n/2'* segments should be in the pipe that is equivalent to what has been set in CWD to get at the end of the fast recovery phase. In this manner, we don't empty the pipe, we simply decrease the flow. After that we proceed with congestion avoidance phase as in Tahoe.

1) Problems: Performance of Reno is very good in case of fewer packet losses. But in cases where multiple packets are not transmitted successfully in one window, then it will not perform well and practically its performance will not be different from that of tahoe, because it is only capable of detecting a single packet loss. In case of multiple packets being lost, we will first come to know about the packet loss upon receiving the duplicate acknowledgements. But the info about the second packet loss will arrive upon the completion of one *RTT* i.e. when the sender receives the acknowledgement of the first retransmitted segment. The possibility of congestion window to be reduced twice for packet losses in a single window also exists. Suppose we are sending nine packets in order. If packet 1, and 2 are lost then the acknowledgements generated by 3, 4 and 5 will cause re-transmission of 1 and the CWD will be decreased to 7. So, when we receive acknowledgements for 6,7,8,9 at that point the CWD is adequately large to allow us to send 10, 11. When the segment 1, which was retransmitted reaches the receiver it generates a new acknowledgement which exits fast-recovery and set CWD to 4. At that point we receive two more acknowledgements for 2 because of 10 and 11, which forces to re-transmit 2 after entering in fast-retransmit mode and immediately after this it enters in fast-recovery mode. Accordingly, upon exiting fast recovery again our window size is set to 2 and it cannot detect multiple packet losses viably.

C. TCP New Reno

It is a modified version of TCP Reno [15] [16]. New Reno can effectively detect multiple packet losses which was a flaw

in tahoe and Reno. Like Reno, new Reno also works in fast retransmit and recovery phase, as it receives the triple duplicate acknowledgements from the receiver. However, it is different from TCP Reno in a way that it remains in fast-recovery mode until and unless all the outstanding data, entered in fast-transmit and recovery mode is acknowledged. The new Reno overcomes the issue of reducing the congestion window multiple times that was faced by Reno. The new Reno retains the fast-transmit phase of Reno. Further, the fast-recovery phase of new Reno differs from Reno in a way that it allows multiple re-transmissions. Whenever it enters in fast-recovery mode, new Reno make notes of almost all the segments that are outstanding. The fast-recovery phase goes ahead as in Reno, however, when we receive a fresh acknowledgement (ACK) then there are two conditions: [16] [17].

1) If it acknowledges all the outstanding segments when it had entered into the fast recovery phase then it sets congestion window to SS-threshold and while exiting fast recovery mode it goes into congestion avoidance phase as in tahoe.
2) If the received acknowledgement is partial, then it figures out that the upcoming segment in queue was lost so the segment can be re-transmitted while resetting the counter of received duplicate acknowledgements to zero. Further, it exits fast-retransmit and recovery mode when all the data is acknowledged.

1) Problems: New Reno requires a complete round trip time (RTT) in order to identify a single packet loss. When we obtain acknowledgement for the first re-transmitted segment, only then we can have information about other missing segments.

II. SIMULATION RESULTS AND DISCUSSIONS

For simulations, network simulator-3 i.e. NS-3.25, was used. The simulator was operated on Linux platform, that is, Fedora 5.0 (Linux Operating System).

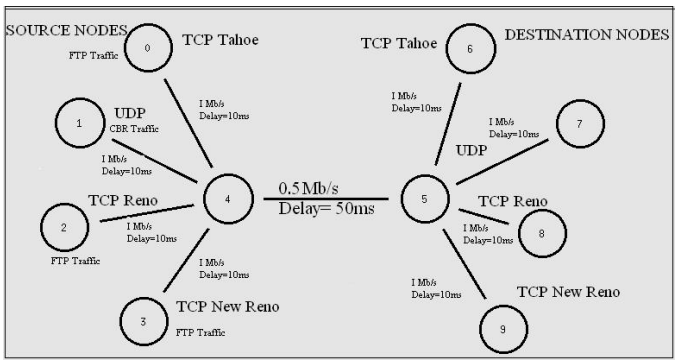

Fig. 1: Network Topology - Bottleneck Link

A. Topology and Network Details

To test and evaluate the aforementioned protocols a network was created as shown in figure 1. In this topology, one bottleneck link was created. One end of the link was connected to the four source nodes while the other end was connected to the four respective sink nodes corresponding to each one of the four source nodes. Out of the four source nodes, three were running three different TCP flavors i.e. TCP Tahoe, TCP Reno and TCP New Reno, while the fourth node was running UDP i.e. user datagram protocol. Further, file transfer protocol (FTP) traffic was used for nodes running TCP flavors while traffic with constant bit-rate (CBR) was created for node running UDP. Packet size for UDP was kept to 200 bytes while packet size for FTP traffic was kept to 1024 bytes. In addition, propagation delay for the bottle neck link was kept to 50 m-sec while for all other links it was kept to 10 m-sec. The bandwidth of the bottle neck link was kept to 0.5 Mb/sec while all the other links were working with bandwidth of 1 Mb/sec. During simulation the buffer size of *'node 4'* was varied and its effects on throughput were observed. Also, random early drop (RED) and drop tail (DT) congestion avoidance techniques were employed on source nodes and their effects were observed in terms of packet loss and in terms of consistency of throughput without having to observe much sharp discontinuities. GNU XY-Graph tool of Linux was used to plot the simulation results while throughput vs. buffer size calculations and plots were done and generated in MATLAB, respectively. The said calculations were made using the trace files generated during simulations on NS-3 i.e. using observations at each and every instant of time. Further, in GNU XY-Graph plots, red color represents TCP tahoe, green color represents UDP, blue color represents TCP Reno and yellow color represents TCP new Reno.

A. Buffer Size vs Throughput

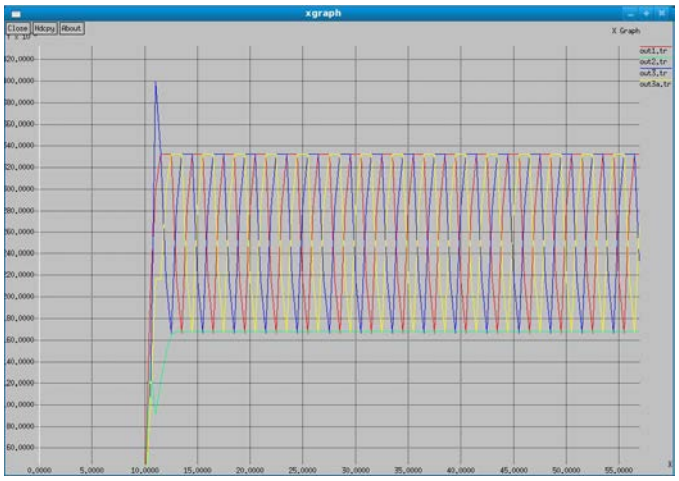

Fig. 2: Throughput with Buffer Size=10 Packets (Drop Tail)

Throughput with constant and variable buffer size was observed. Figure 2 illustrates throughput for the four respective flows when the buffer size was kept to 10 packets. It can be observed from the graph that UDP traffic settles down early as compared to other flows i.e. a near fixed throughput, this is because the packet size for UDP is small and it is a connectionless protocol. For TCP flows, TCP new Reno has few losses and decays compared to TCP Reno and TCP tahoe. Moreover, TCP Reno has second best throughput after TCP new Reno. Further, the buffer size was varied with an increment of 10 after a constant interval of 30 seconds.

Equation 1 describes the calculation of buffer size as a function of time. It can be observed from figure 3 that TCP new Reno has most consistent throughput throughout the simulation period without greater decline whereas TCP tahoe edges TCP Reno at certain instances. Also, it is of note that losses or degradation in throughput in TCP tahoe is much greater compared to TCP Reno as observed from the figure.

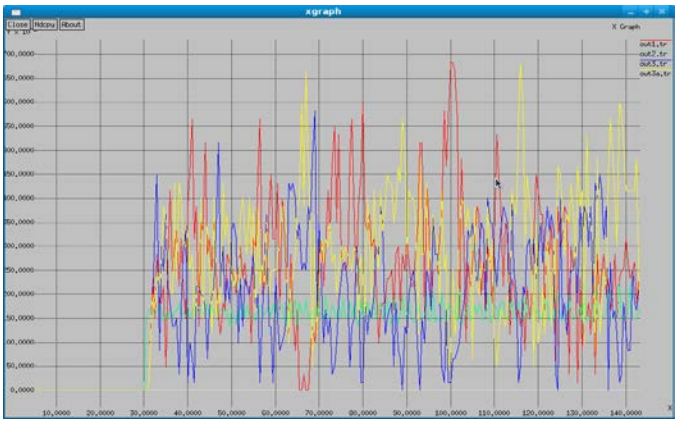

Fig. 3: Throughput with Variable Buffer Size Packets (Drop Tail)

$$\text{Buffer} = b(t) = 10 + (0{:}33t) \qquad (1)$$

where, ***'t'*** represents time and can only hold values in multiples of ***30***.

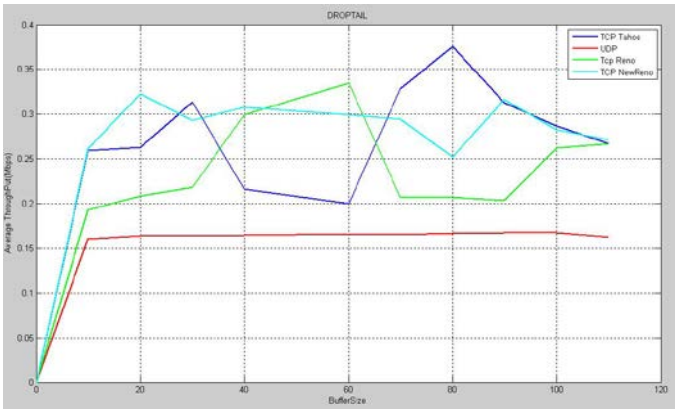

Fig. 4: Throughput vs. Buffer Size (Drop Tail)

Further, figure 4 gives a comparison of average throughput in Mb/sec with respect to buffer size. It can be observed that throughput for UDP gets to a steady state after buffer size reaches to a certain value and it remains constant irrespective of the increase in buffer size. This is due to the fact that UDP packet is of smaller size compared to TCP packet and also because UDP doesn't take care of congestion/ flow control i.e. no acknowledgements. It triggers packets at constant bit rate and that is why the throughput gets constant after certain value of the buffer size is attained. For TCP flows, the throughput increases with the increase in buffer size but due to losses and randomness in the network, variations can be observed in throughput for the respective TCP flows. Another thing of note is that TCP new Reno has most consistent and sustained throughput levels without much observed drops or decays as can be seen in figures 2, 3 and 4. Further, it can also be concluded that out of all the flows, TCP new Reno seems to be much less affected by network's randomly varying nature.

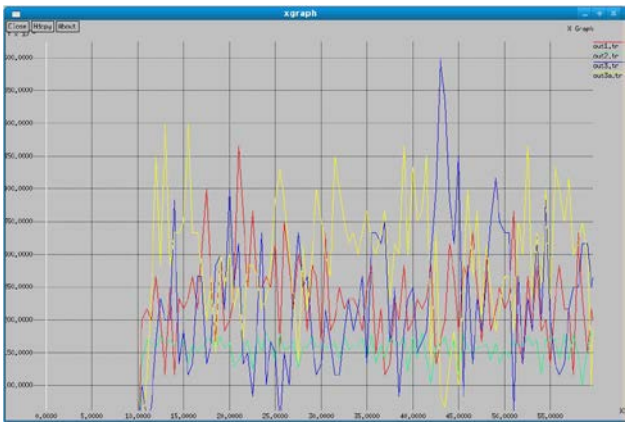

Fig. 5: Throughput with Buffer Size=10 Packets (Random Early Drop)

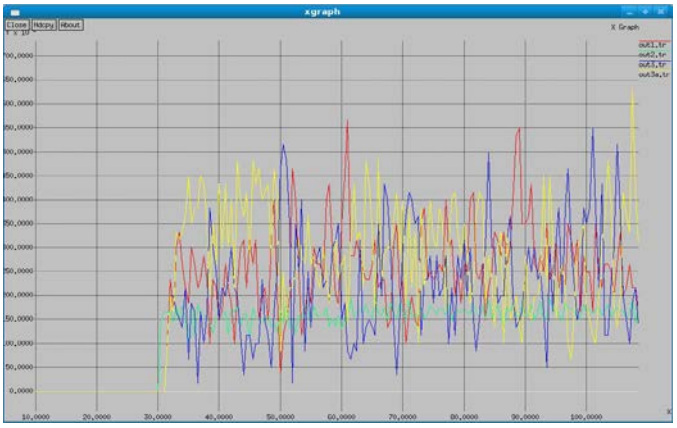

Fig. 6: Throughput with variable Buffer Size (Random Early Drop)

Similarly, simulations were conducted to observe throughput for the four respective flows while keeping same network parameters except for the congestion avoidance technique, that is, random early drop (RED). Figure 5 illustrates throughput for the four flows keeping the buffer size fixed to 10 packets. Further, performances in terms of comparison between individual flows have yielded almost similar results to what we have observed with drop tail i.e. in terms of proportion.

Further, results for variable-length buffer size were also observed while keeping random early drop (RED) as the congestion avoidance mechanism. Figure 6 shows throughput for the four respective flows when buffer size was varied i.e. with an increment of 10 after every 30 seconds. Moreover, buffer size calculation follows the same criteria as described in equation 1. It can be observed that TCP new Reno gives most consistent and sustained throughput levels. Even in this case TCP Reno comes second best to TCP new Reno whereas for TCP tahoe much lesser throughput levels were observed Also, figure 7 below illustrates the average throughputs for the respective four flows with respect to the buffer size. It can

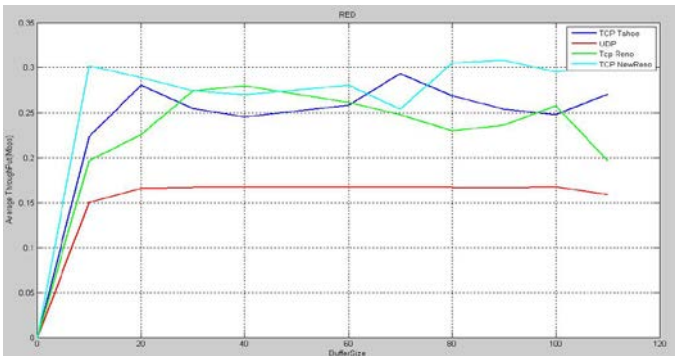

Fig. 7: Throughput vs. Buffer Size (Random Early Drop)

Also, figure 7 below illustrates the average throughputs for the respective four flows with respect to the buffer size. It can be observed from the graph that TCP new Reno outperforms TCP Reno and TCP tahoe with greater throughput and with lesser packet losses.

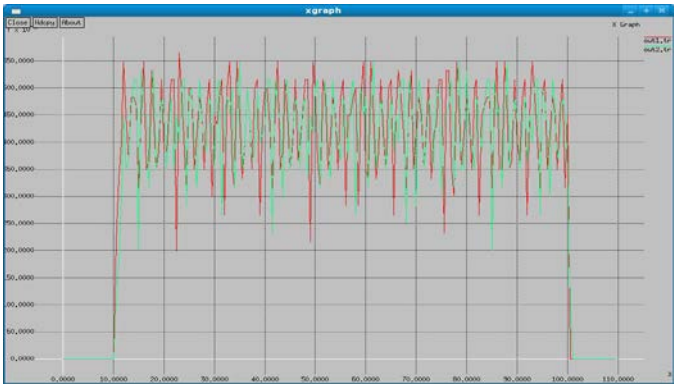

Fig. 8: Two TCP Flows (Random Early Drop)

In addition, another relationship has been analyzed using the results of "throughput vs. buffer size" for both random early drop (RED) and drop tail (DT). On comparison, the observed results gave clear indication that the use of random early drop (RED) as the congestion avoidance technique over drop tail (DT) would yield better throughput levels. Moreover, results were observed while keeping all the network parameters constant except for the congestion avoidance techniques. To prove this point a topology was created having two source nodes, both running TCP on them. The topology was simulated for both techniques of congestion avoidance i.e. drop tail (DT) and random early drop (RED). Simulation results observed are illustrated in figures 8 and 9. There is a clear visible difference between the two aforementioned techniques in terms of throughput levels.

With random early drop (RED), overall throughput of the system is far better compared to what has been observed with drop tail (DT). Moreover, it is quite clear from the XY-plots above that with the use of drop tail (DT) there are more losses or visible low points observed compared to random early drop (RED).

Further, table I list the expected values and variance for the packet drop rate observed with each of the variant of TCP. Moreover, the calculations were made in MATLAB using data from trace files generated via simulations on NS-3. Further, the simulation results have been averaged over 100 runs.

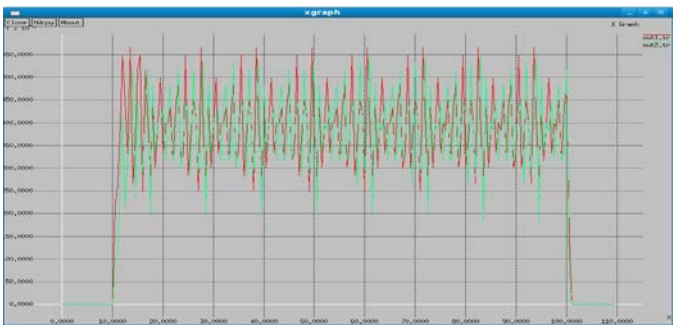

Fig. 9: Two TCP Flows (Drop Tail)

Table I: TCP vs. Packet Drop Rates - Expected Value and Variance

TCP	Expected Value	Variance
Tahoe	**9.3**	**17.32**
Reno	**7.2**	**19.23**
New Reno	**7.5**	**18.45**

III. ACKNOWLEDGEMENT

The authors would like acknowledge the efforts of faculty of engineering, sciences & technology, IQRA University and department of electronic & power engineering, PNEC-NUST for their persistent encouragement and technical support.

IV. CONCLUSIONS

This work outlines the performance evaluation of different TCP variants, namely, TCP tahoe, TCP Reno and TCP new Reno, in a constrained bottleneck network arrangement. In addition, drop tail (DT) and random early drop (RED) have been used as the two congestion avoidance mechanisms. Several runs of simulations were made using NS-3 in a lossy network environment. Moreover, throughput of the network was observed with varying buffer size and with insertion of propagation delay into the links. Also, a network with two TCP flows was created to observe throughput while using two different congestion avoidance techniques. Results observed clearly showed that TCP new Reno outperformed TCP Reno and TCP tahoe in terms of consistent throughput during the complete simulation run whereas observed simulation results for TCP Reno were second to that of TCP new Reno. Further, the simulations were averaged over 100 runs where each run lasted 400 seconds of simulation time.

Based on observations following conclusions are in order,

1) This study presents a qualitative as well as quantitative analysis of performance evaluation of different variants of TCP.

2) Packet Drop rate for TCP new Reno and TCP Reno is lesser as compared to TCP tahoe. Moreover, TCP Reno just edges TCP new Reno in terms of lesser expected packet drop rate, however, TCP new Reno has lesser variance compared to TCP Reno.

3) Random early drop (RED) outperforms drop tail (DT) in terms of achieved throughput.

4) TCP new Reno outperforms TCP tahoe and TCP Reno based on most of the key performance metrics.

5) As we know that with each passing second the size of the internet is getting larger and bigger whereas exceeding resource demands have certainly pushed the wall to absolute limits. Moreover, this increase in network size and demands has seen large amount of data being laid on to the network. Hence, this flow of large volumes of data (video, text and images) has been an aggravating cause of network congestion and so, the methods mitigating congestion are of utmost importance. Therefore, this study presented a detailed analysis of TCP congestion avoidance methods while keeping in account distinct network metrics. Moreover to make this work more meaningful, all simulations were conducted in a constrained bottleneck environment and based on analysis conclusions were drawn as in table II.

Table II: Overall Best Performing TCP Based on Key Operating Metrics

Parameters	Best TCP
Congestion Window vs. Time	**New Reno**
Packet vs. Bandwidth	**New Reno**
Packet vs. Delay	**New Reno**
Packet vs. Rate	**New Reno and Tahoe**
Throughput vs. Bandwidth	**New Reno and Reno**
Time vs. Buffer Size	**New Reno**
Packet Delivery Ratio vs. Buffer Size	**New Reno**
Packet Drop vs. Buffer Size	**Reno**

REFERENCES

[1] M. Allman, V. Paxson, and E. Blanton, "Tcp congestion control," Tech. Rep., 2009
.

[2] K. Fall and S. Floyd, "Simulation-based comparisons of tahoe, Reno and sack tcp," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, pp. 5–21, 1996.

[3] R. S. Prasad, C. Dovrolis, and M. Thottan, "Router buffer sizing for tcp traffic and the role of the output/input capacity ratio," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 5, pp. 1645–1658, 2009.

[4] A. W. Berger and Y. Kogan, "Dimensioning bandwidth for elastic traffic in high-speed data networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 8, no. 5, pp. 643–654, 2000.

[5] H. S. Kim and N. B. Shroff, "Loss probability calculations and asymptotic analysis for finite buffer multiplexers," *IEEE/ACM Transactions on Networking (TON)*, vol. 9, no. 6, pp. 755–768, 2001.

[6] C. Villamizar and C. Song, "High performance tcp in ansnet," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 5, pp. 45–60, 1994.

[7] R. Morris, "Tcp behavior with many flows," in *Network Protocols, 1997. Proceedings., 1997 International Conference on. IEEE*, 1997, pp. 205– 211.

[8] G. Appenzeller, I. Keslassy, and N. McKeown, Sizing router buffers. ACM, 2004, vol. 34, no. 4.

[9] A. Lakshmikantha, C. Beck, and R. Srikant, "Impact of file arrivals and departures on buffer sizing in core routers," *IEEE/ACM Transactions on Networking (TON)*, vol. 19, no. 2, pp. 347–358, 2011.

[10] B. Bai and J. Yan, "The role of buffer unit in routers with very small buffers," in *Computing and Networking Technology (ICCNT), 2012 8th International Conference on. IEEE*, 2012, pp. 1–4.

[11] N. Islam and S. Helal, "Performance analysis of tcp tahoe, Reno, new Reno and sack over cellular mobile system," *8th ICCIT*, pp. 786–790, 2005.

[12] M. Miyoshi, M. Sugano, and M. Murata, "Performance evaluation of tcp throughput on wireless cellular networks," in *Vehicular Technology Conference, 2001. VTC 2001 Spring. IEEE VTS 53rd*, vol. 3. IEEE, 2001, pp. 2177–2181.

[13] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM computer communication review*, vol. 18, no. 4. ACM, 1988, pp. 314–329.

[14] V. Jacobson, "Modified tcp congestion control and avoidance algorithms. end2end-interest mailing list," 1990.

[15] T. Lang, Evaluation of different TCP versions in non-wireline environments. *University of South Australia*, 2002.

[16] S. Floyd, A. Gurtov, and T. Henderson, "The newReno modification to tcp's fast recovery algorithm," 2004.

[17] L. S. Brakmo and L. L. Peterson, "Tcp vegas: End to end congestion avoidance on a global internet," *IEEE Journal on selected Areas in communications*, vol. 13, no. 8, pp. 1465–1480, 1995.