Performance Evaluation of Various Transmission Control Protocols in NS2

Palak Bansal, Kritika Agrawal and Ankita Gupta

Abstract Rapid transfer of data of high magnitude is a necessity to meet the demand of today's high-speed communication networks. The currently prominent implementation of TCP protocol is insufficient for the high bandwidth-latency product networks that are available to users. The current TCP Reno protocol implementation fails to utilize the available bandwidth optimally and requires an exceptionally small bit error rate (BER) for it to maintain that rate. A substantial amount of bandwidth is left unused in such networks. Moreover, congestion control is required so that the stability of the network is maintained. TCP Reno's congestion policy is not suitable in today's technology as it might take more than a few hours to recover that congestion window. Many variants of TCP protocols have been suggested which follow more aggressive approach than the current implementation like loss-based metrics algorithm and delay-based algorithm for congestion control. Loss metric based determine congestion by loss of a packet; and delay metric focuses on delay as a sign of congestion. The research work presents a comparative study and performance analysis factors in terms of congestion window size growth and fairness (throughput in terms of bytes transferred per second) of TCP Reno, Scalable TCP and TCP Cubic which are loss-based metrics algorithm, and TCP Vegas which is a delay-based algorithm for congestion control. According to the observation and analysis performed using simulation, Scalable TCP among TCP Reno, TCP Vegas and TCP Cubic performs better in terms of the congestion window corresponding with respect to time and can be considered as a good option. In terms of fairness, TCP Cubic and Scalable TCP, both are more aggressive than TCP Reno and TCO Vegas. To conclude for aggressiveness and packet loss, Scalable TCP is the best option among the TCP variants taken for performance analysis and evaluation.

P. Bansal (✉) · K. Agrawal · A. Gupta
Department of Computer Science, Jaypee Institute of Information Technology, Noida, India
e-mail: palakbansal94@gmail.com

K. Agrawal
e-mail: 1995.kritika@gmail.com

A. Gupta
e-mail: ankita.gupta.mail@gmail.com

M. Pant et al. (eds.), *Computational Network Application Tools for Performance Management*, Asset Analytics,
https://doi.org/10.1007/978-981-32-9585-8_16

Keywords Transmission control protocol · Congestion control · Network simulator · TCP Reno · Scalable TCP · TCP Vegas · TCP Cubic

1 Introduction

Large amount of data is required to be transferred quickly over high-speed networks. TCP Reno [1] is most widely accepted and implemented transport protocol. But it is not able to scale well to data rate of 100 Mbps and more. Despite large bandwidth being available, TCP Reno is not able to completely utilize it. TCP Reno was designed when the value of data rate used to be in a few kbps. Congestion control is required so that the stability of the network is maintained. TCP Reno's [2] additive-increase/multiplicative-decrease (AIMD) policy for congestion back-off is too rapid and it might take more than a few hours to recover that congestion window size as explained by Floyd [3]. Recovery time is too large for TCP Reno. Average congestion window varies with inverse square root of probability of loss [4]. TCP Reno detects congestion by packet loss. For high-speed network links, very low packet drop probability is required, which is unreasonable to achieve.

Many variants of the TCP are proposed with changes on the sender side. Those variants are more aggressive than the TCP Reno. Some of the variants follow loss-based metric, and some follow delay metric. Loss metric based determine congestion by loss of a packet; and delay metric focuses on delay as a sign of congestion. Delay metric based make full use of state of network which is given by packet delay. Some variants involving updating the congestion control condition on the sender side are TCP Vegas [5], Scalable TCP [3], TCP Cubic [6], TCP Africa [7], etc. We are simulating TCP Reno, TCP Scalable, TCP Vegas and TCP Cubic using NS2 [8] as a network simulator and analysing them using graphical analysis for this paper.

The rest of the paper is organized in the following manner: Tools used for simulating and analysis are explained in Sect. 2, followed by related work details in next Sect. 3, and then working and congestion control algorithm of TCP Reno are explained in Sect. 3.3, which is then followed by Scalable TCP, loss-based algorithm in Sect. 3.4, TCP Cubic, implemented in Linux and a loss metric based in Sect. 3.5 and TCP Vegas, delay-based algorithm in Sect. 3.6. These algorithms are then followed by their implementation and testing details in next Sect. 4. Section 5 consists of details of performance evaluation followed by conclusions and future work in Sects. 6 and 7, respectively. Finally, the last section consists of the references used for the paper.

2 Tools

This section consists of the description of the tools which were used for implementing congestion control algorithms and analyzing it. Since a lot of traffic is generated in the actual network of Internet, it was necessary to imitate the actual network. To emulate the behaviour of the computer network, network simulator is being used.

2.1 Simulator

NS2 [9], second version of network simulator, is used for simulating TCP variants. It is a simulation tool that is event-driven and helps in emulating dynamic nature of actual network like Internet. Different protocols (TCP, TCP Vegas, UDP, etc.) and queues (drop tail, etc.) can be simulated using NS2, and traffic may be generated according to the protocols simulated. The network of simulation may be wired or wireless. Tcl is the script used for creating a simulation file in NS2. The Tcl script is executed with the command ns along with the name of the file and input arguments, if required. NS2 consists of mainly two languages: C++ and Object-Oriented Tool Command Language (OTcl). While simulating, internal mechanism is defined by C++ and events are configured and assembled by OTcl. TclCL is used for linking C++ and OTcl.

2.2 Simulation Results

Simulation results may be text-based as well as animation-based. To interpret the results, tools NAM and Xgraph are being used. A network object is created as output in NAM. Topology is created and coded in Tcl file is emulated using NAM, which helps to interpret flow of the packets in the network. Observations are recorded in trace file. The trace file is being used for plotting the graph and analyzing the results.

3 Related Work

TCP Reno [1], the traditional and most accepted TCP, was developed around 1990. There are various improvements proposed for TCP Reno in [2, 10], etc. Many new are variants which are proposed of TCP Reno, which changes the algorithm of congestion control and are more aggressive than TCP Reno. These variants follow different approaches for detecting congestion in the network and implementing congestion control. Some of the variants follow packet loss-based approach to detect congestion as HSTCP [11], Scalable TCP, TCP Cubic, etc. Some variants follow delay approach like TCP Vegas, TCP Fast [12]. Some protocols like TCP Africa have been proposed which follow both approaches: packet loss based and delay based to be able to utilize the benefit of both approaches. These protocols follow loss of packet approach under particular conditions and delayed approach under other conditions.

3.1 TCP Africa

TCP Africa [7] is neither a completely loss-based protocol nor delay-based protocol. It is a hybrid one. Delay metric is used by it to determine the network congestion level, and it acts accordingly. If congestion is absent, then it follows fast mode; and in congestion presence, it follows slow mode of TCP Reno. It grabs the unutilized bandwidth quickly, but it becomes slow when the next congestion event is induced. It uses a parameter α which determines the sensitivity of protocol to delay. This parameter is generally taken as greater than 1.

3.2 HSTCP

HSTCP [11] is a variation of TCP Reno which also uses packet loss as an indication of the congestion in network. Three new parameters are being used in this protocol: High_Window, which indicates the highest size of congestion window; Low_Window, which tells about the minimum congestion window required for HSTCP's new response function to work; and High_P, the probability of packet drop in HSTCP. If size of congestion window is less than or equal to Low_Window, then slow mode of TCP Reno is followed; otherwise, if size of congestion window exceeds Low_Window, then response function of high-speed TCP is followed.

3.3 TCP Reno

TCP Reno [2] congestion control algorithm consists of four components: slow start, congestion avoidance, fast retransmit, and fast recovery. Sender using TCP Reno is in any of the following phases:

- Slow start
- Congestion avoidance
- Fast retransmit and fast recovery.

Congestion is detected if either timeout occurs or triple acknowledgement is received. Whenever congestion is detected or acknowledgement is received, the next phase of Reno is decided by checking the current phase and congestion status.

Slow Start

Congestion window is increased by most Sender Maximum Sender Size (SMSS) whenever an acknowledgement is received for new packet sent. In slow start phase, congestion window is generally doubled for every round trip. When congestion window exceeds the threshold value (ssthresh), Reno goes to congestion avoidance phase.

Congestion Avoidance
In congestion avoidance phase, packet loss or timeout size of congestion window is increased by 1 full-size segment per RTT.

Congestion window is updated until acknowledgement is received for every new packet sent as:

$$\text{cwnd} = \text{cwnd} + \text{SMSS} * \text{SMSS}/\text{cwnd} \tag{1}$$

Whenever timeout occurs, the value of ssthresh is updated. In case of receiving triple acknowledgement, Reno enters fast retransmit phase.

Fast Retransmit and Recovery
A timer is being maintained for every packet sent. If a sender receives acknowledgement after timeout occurs, then it is considered that the packet is lost. But, it might take too long to realize that a packet is lost and take action accordingly. Fast retransmit phase uses duplicate acknowledgements to detect that a packet is lost. When multiple acknowledgment packets (generally 3) are received of the same sequence number, TCP considers that the packet is lost and retransmits it.

3.4 Scalable TCP

Scalable TCP [3] follows multiplicative-increase/multiplicative-decrease (MIMD) policy for updating congestion window. The phases and the congestion detection occurrences of TCP Scalable are same as that of TCP Reno. But its implementation differs from TCP Reno as follows:

When packets are acknowledged and there is no congestion, congestion window value is updated by 0.01, i.e.,

$$\text{cwnd} = \text{cwnd} + 0.01 \tag{2}$$

When congestion is detected (triple acknowledgement or timeout), then congestion window value is reduced to 0.875 of its value; that is, it is reduced by 0.125 times of its value. i.e.,

$$\text{cwnd} = \text{cwnd} - 0.125 * \text{cwnd} \tag{3}$$

Scalable TCP reduces the problem of recovery time of congestion window to a constant time rather than being dependent on the size of the congestion window, as in the case of TCP Reno.

3.5 TCP Cubic

TCP Cubic [6] is an improved variant of BIC-TCP as sometimes BIC becomes too aggressive specifically in networks with low speeds. The phases and congestion detection occurrences are similar to TCP Reno. A value $W_{\max}$ is noted when the first packet loss occurs. Cubic might be in TCP mode, concave or convex regions depending upon the value of the congestion window. If cwnd < $W_{\max,}$ then it is in concave region, and if cwnd > $W_{\max,}$ then it is in convex region.

Congestion window update is given as:

$$\text{cwnd} = C(t - K)^3 + W_{\max} \tag{4}$$

$$K = (W_{\max}\beta/C)^{1/3} \tag{5}$$

where

C cubic parameter
t time elapsed since last reduction of window
β decrease factor.

3.6 TCP Vegas

TCP Vegas [5] follows delay metric to determine the congestion of the network. Vegas is based on the idea that bytes in the transit of connection are directly related to the throughput of connection. The used bandwidth by connection is compared with the bandwidth available of the network; in order utilize available bandwidth and avoid congestion. In Vegas Base, RTT is one of the main factors used to determine the expected value of throughput.

$$\text{Throughput}_{\text{Expected}} = \text{cwnd}/\text{BaseRTT} \tag{6}$$

where cwnd is congestion window's current size.

Then, actual value of throughput is calculated by measuring the number of bytes transmitted and dividing it by RTT value. This calculation is done once per round trip time.

Value of difference (D) between actual and expected throughput is calculated.

$$D = \text{Throughput}_{\text{Expected}} - \text{Throughput}_{\text{Actual}} \tag{7}$$

Two thresholds x and y are defined such that $x < y$. If $D < x$, then congestion window is increased linearly for next RTT, and if $D > y$, then congestion window is decreased for the next RTT. If $x < D < y$, then no change in congestion window is done.

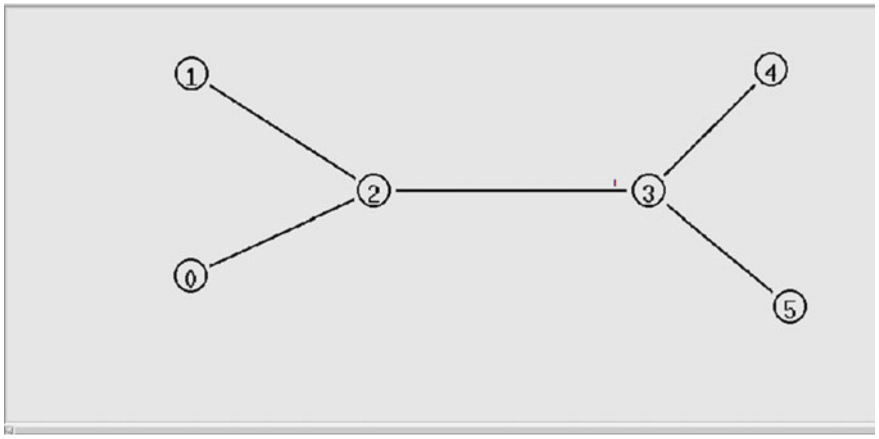

Fig. 1 Topology used for analysis of TCP variants

4 Implementation and Testing

TCP variants [13, 14] are simulated in NS2 [8] with using their default parameters for Scalable TCP, TCP Reno and TCP Vegas; for TCP Cubic, default parameters are set as beta = 0.125, max_increment = 16, fast_convergence = 1 and bic_value = 410. The topology [15] used for doing analysis is as follows (Fig. 1).

Topology consisting of six nodes is being used to simulate the network. Nodes marked as 0 and 1 are being used as TCP senders, and 4, 5 are being used as TCP receivers. Nodes 2 and 3 served as bottlenecks in the network. Links from nodes 0 and 1 to 2 have a bandwidth of 2 Gbps and delay of 10 ms. Links from 2 to 3 have a bandwidth of 1 Gbps with a delay of 200 ms. Links from 4 to 5 and 6 are similar to 0 to 1 link.

Details of simulation are stored in trace files (.tr) and their graph is plotted using Xgraph. Analysis is done in the form of congestion window size versus time and fairness [16]. For this implementation, queue size is set to 10 for connection between nodes 2 and 3, and packet size is set to 550 bytes. Simulated network is tested up to 550 bytes of packet size, and analysis is done accordingly.

5 Performance Evaluation

5.1 Congestion Window Size Growth

X-axis indicates time with interval of 10 s. *Y*-axis indicates the size of congestion window. The graph first increases and then decreases indicating packet loss. The sharp drop is due to the multiplicative-decrease policy followed by TCP Reno. After the drop, the linear increasing curve of the graph indicates that the congestion window size is increasing and no packet loss is occurring (Fig. 2).

Time is drawn on x-axis, and congestion window on y-axis indicating the congestion window size at particular instances. Time gap of 0.1 s is taken to plot the graph. The highest value achieved for congestion window is 82. The different peaks of the graph tell us about the congestion window size at that time, and growth and decrement of peaks tell us about the corresponding increment/decrement in congestion window size. Decrement indicated packet loss has occurred, and congestion is detected in the network (Fig. 3).

Congestion window increases indicating successful delivery of packets with receiving acknowledgement. Decrease tells us about packet loss through timeout or triple acknowledgement. Graph follows the policy of multiplicative increase and subsequently multiplicative decrease. Congestion window size increases multiplicatively till 55 s, then decreases and then increases again (Fig. 4).

Congestion window of TCP Vegas grows and attains a value of 33 at 5 s. Then, a packet drop takes place and the value of congestion window becomes 19. The earlier lost packet is retransmitted successfully; followed by the successful transmission of new packets which leads to an increase in the size of congestion window. Positive slope after 5 s indicates that packet loss is not taking place (Fig. 5).

For the implementation considered, congestion window size of TCP Vegas does not vary much as compared to TCP Reno; the difference is really less. No packet loss after 10 s for the simulation taken could be one of the reasons for this observation. Values of congestion window of Scalable TCP are much higher than that of TCP Reno; even when there is a packet loss in scalable and congestion window decreases, there is no packet loss at the same time on TCP Reno. This shows the aggressiveness

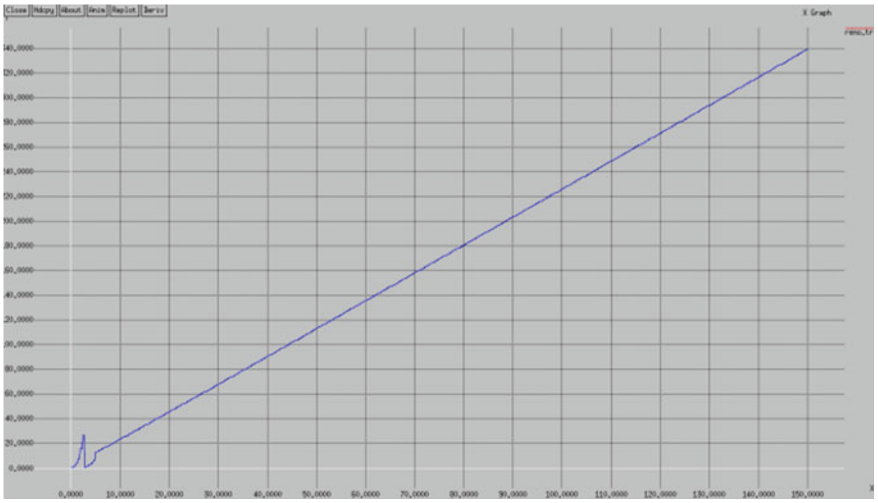

Fig. 2 Congestion window versus time for TCP Reno in NS2

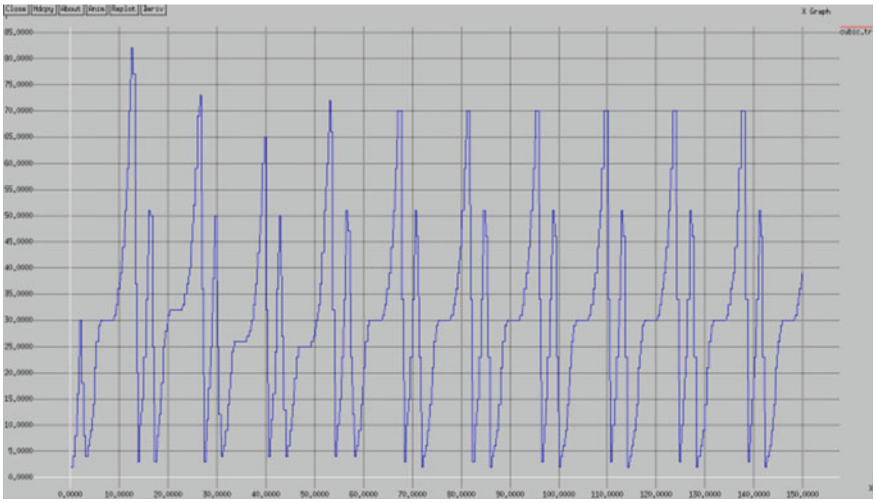

Fig. 3 Congestion window versus time for TCP Cubic in NS2

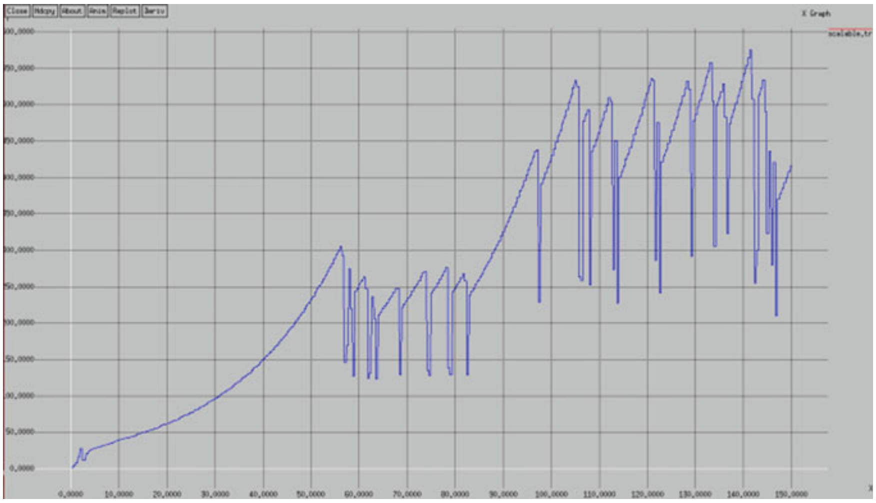

Fig. 4 Congestion window versus time for TCP Scalable in NS2

of the Scalable TCP. Values of congestion window for TCP Cubic are less than TCP Reno, but they increase at almost by same level despite TCP Cubic window having less value (Table 1).

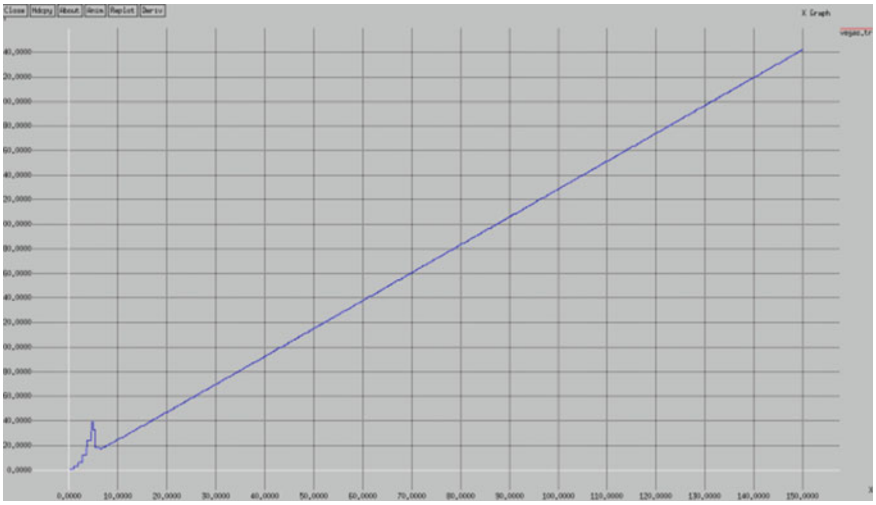

Fig. 5 Congestion window versus time for TCP Vegas in NS2

Table 1 Congestion window at different time elapsed

Time (in s)	TCP Reno	TCP Vegas	TCP Cubic	Scalable TCP
5	13.07	33	21	28
10	23.48	24.7	36	39
20	46	47.35	31	62
30	67.53	69.24	50	95
40	90.27	92.18	65	149
50	113.12	115.14	27	233
60	136.01	138.11	21	253
70	157.82	160.1	23	230
80	180.65	183.09	44	248
90	203.52	206.08	30	326

5.2 *Fairness*

For analyzing fairness [16] of TCP variants with TCP Reno, link from node 0 to node 4 is simulated using TCP Cubic, Scalable TCP, TCP Vegas one by one, and the link from node 1 to node 5 is simulated using TCP Reno.

For evaluating fairness, throughput in terms of bytes transferred per second is taken as a parameter. Throughput or bytes transferred per second is plotted on *y*-axis against time in seconds is plotted on *x*-axis.

The cubic curve (red curve) depicts the throughput variation with time for TCP Cubic and Threno (green curve) depicts the variation for TCP Reno. Initially, both start with throughput at low value and later on (at 5 s) value gets increased, varying TCP Cubic's value more than TCP Reno. 171 packets are dropped during this scenario (Fig. 6).

In Fig. 7, Scalable TCP's throughput values are plotted by curve scalable (red curve) and TCP Reno's throughput is plotted by Reno (green curve). Throughput of Scalable TCP increases more than TCP Reno. While simulating scenario is depicted in Fig. 7, 61 packets are dropped.

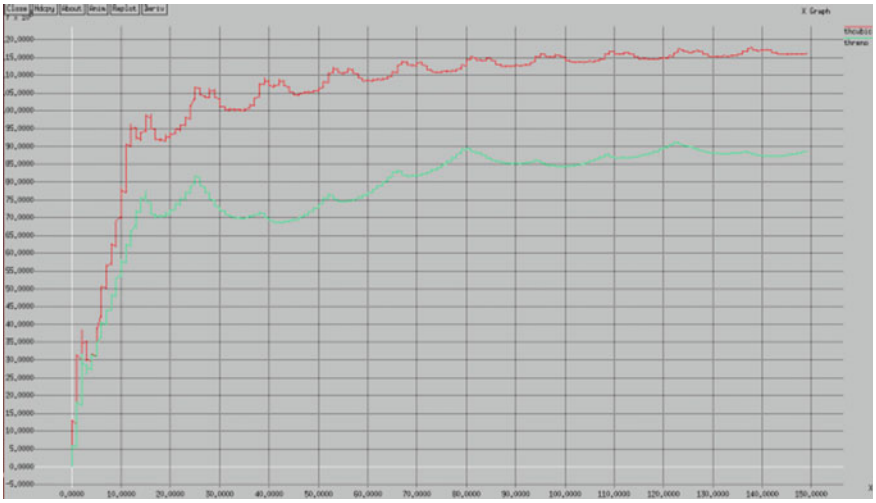

Fig. 6 Bytes transferred per second versus time for TCP Cubic and Reno

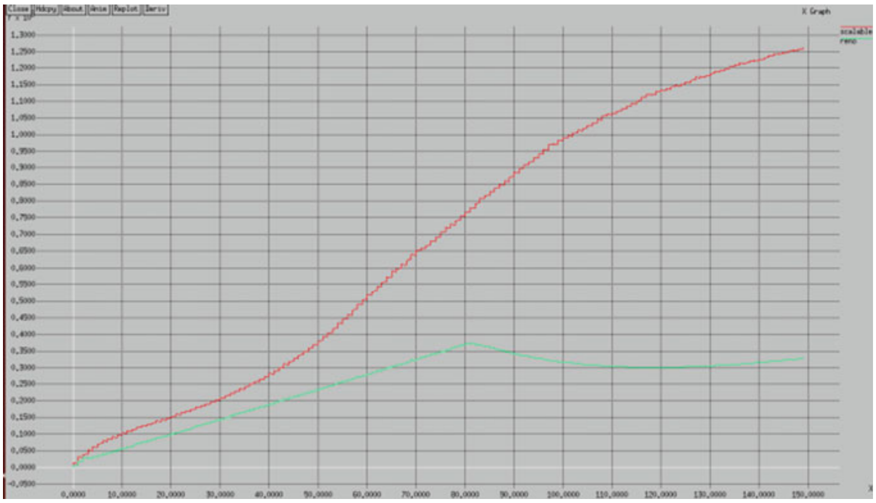

Fig. 7 Bytes transferred per second versus time for TCP Scalable and Reno

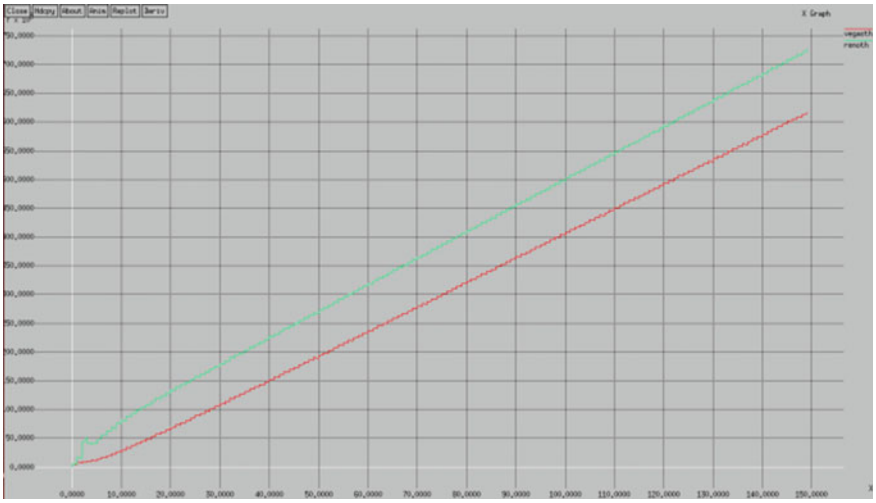

Fig. 8 Bytes transferred per second versus time for TCP Vegas and Reno

In Fig. 8, renoth (green curve) indicates the throughput values with time for TCP Reno and vegasth (red curve) indicates values for TCP Vegas. Throughput values for both the TCP Reno are TCP Vegas which are nearby initially, and then, TCP Reno's throughput increases more than TCP Vegas. Twenty packets are dropped while transferring data in this scenario.

6 Conclusion

According to the network taken for simulation, observation, and analysis, Scalable TCP among TCP Reno, TCP Vegas and TCP Cubic perform better in terms of the congestion window corresponding with respect to time. If the evaluation metric of congestion window with time is being considered, then Scalable TCP can be considered as a good option.

When performance metric of fairness is taken, aggressiveness and packet drop, both are taken into consideration. TCP Cubic and Scalable TCP both are more aggressive than TCP Reno, but packet loss is more when TCP Cubic is taken along with TCP Reno for evaluating fairness. Packet drop is lowest in scenario when TCP Vegas is simulated along with TCP Reno in the same network, but TCP Vegas is less aggressive than TCP Reno. To moderate both the terms, aggressiveness and packet loss, Scalable TCP is the best option among the TCP variants taken.

According to both the metrics, congestion window variation and fairness, Scalable TCP performs finest and can be considered as an alternative option for TCP Reno.

7 Future Work

This analysis can be extended to other protocols as TCP Fast, TCP Africa, etc. TCP is currently not implemented in NS2; thus, it can be implemented in C++ and then linked to NS2 followed by its simulation. Other performance metrics may be considered for evaluation as safety, reliability, etc., for complete analysis of an alternative option for TCP Reno.

References

1. J., Van, Congestion avoidance and control. ACM SIGCOMM Comput. Commun. Rev. **18**(4). ACM (1988)
2. A. Mark, V. Paxson, E. Blanton, TCP congestion control. No. RFC 5681 (2009)
3. T. Kelly, Scalable TCP: improving performance in highspeed wide area networks. ACM SIGCOMM Comput. Commun. Rev. **33**(2), 83–91 (2003)
4. P. Jitendra, et al., Modeling TCP throughput: a simple model and its empirical validation. ACM SIGCOMM Comput. Commun. Rev. **28**(4), 303–314 (1998)
5. L.S. Brakmo, S.W. O'Malley, L.L. Peterson, *TCP Vegas: New Techniques for Congestion Detection and Avoidance*, vol. 24(4) (ACM, 1994)
6. S. Ha, I. Rhee, X. Lisong, CUBIC: a new TCP-friendly high-speed TCP variant. ACM SIGOPS Operat. Syst. Rev. **42**(5), 64–74 (2008)
7. R. King, R. Baraniuk, R. Riedi, TCP-Africa: an adaptive and fair rapid increase rule for scalable TCP, in *Proceedings IEEE INFOCOM 2005 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3. (IEEE, 2005)
8. Tutorial for Linux in NS2, http://netlab.caltech.edu/projects/ns2tcplinux/ns2linux/tutorial/. Last accessed 2017/05/27
9. Introduction to NS2, http://www.ns2blogger.in/p/introduction-to-ns2.html. Last accessed 2017/05/03
10. S. Floyd, H. Balakrishnan, M. Allman, Enhancing TCP's loss recovery using limited transmit (2001)
11. S. Floyd, High speed TCP for large congestion windows (2003)
12. D.X. Wei, et al., FAST TCP: motivation, architecture, algorithms, performance. IEEE/ACM Trans. Network. **14**(6), 1246–1259 (2006)
13. A. Sawarkar, H. Saraswat, Performance analysis of TCP variants. Int. J. Comput. Sci. Netw. Secur. (IJCSNS) **16**(4), 102 (2016)
14. K., Ravi, G. Singh, Various TCP options for congestion evasion. Feedback **4**(4) (2015)
15. P. Subramanya et al., Performance evaluation of high speed TCP variants in dumbbell network. IOSR J. Comput. Eng. **16**(2), 49–53 (2014)
16. A Quantitative Measure of Fairness and Discrimination for Resource Allocation, http://www1.cse.wustl.edu/~jain/papers/ftp/fairness.pdf. Last accessed 2017/06/30