Exploring Congestion Control Mechanism of TCP Variants over Wired & Wireless Networks

Syful Islam, Ratnadip Kuri, Md. Humayun Kabir, Md. Javed Hossain

Abstract— A reliable end to end communication is a buzzword that is promised by the transport layer protocol TCP. TCP, a Reliable transport protocols are tuned to perform well in different networks but, packet losses occur mostly because of congestion. TCP contains several mechanisms (such as slow start, congestion avoidance, fast retransmit and fast recovery) for ensuring reliability. However, it has reached its limitation in some challenging network environments like-High speed communication, Communication over different media. Thus, it requires further analysis and development of congestion control algorithms. In this paper, we have explored the reliability and robustness of TCP variants (Tahoe, Reno, New-Reno, SACK, FACK and TCP VEGAS, HSTCP, CUBIC TCP) based on different parameters such as throughput, end-to-delay, jitter and packet drop ratio over wired and wireless networks. We have also compared and discussed different congestion control and avoidance mechanisms of TCP variants to show how they affect the throughput and efficiency of different network environments.

Index Terms—Congestion avoidance, Congestion Window, Fast-recovery, Fast-retransmit, Reliability, Slow Start, TCP variants.

1 Introduction

TCP is a reliable connection-oriented end-to-end protocol designed for the wireline networks that are characterized by negligible random packet losses. Most of the data is transmitted through TCP in today's WWW (World Wide Web). Early TCP (Transmission Control Protocol) implementation uses go-back-n mode with cumulative positive acknowledgment and requires a retransmit time-out to retransmit the lost packet. This TCP did little to minimize network congestion. Modern TCP implementations contain AIMD (additive increase/multiplicative decrease) [1] with four intertwined algorithms (slow start, congestion avoidance, fast retransmit and fast recovery) aimed at controlling network congestion while maintaining good user throughput. It ensures reliability by starting a timer whenever it sends a segment and the receiver that acknowledge the segments that it receives. If it does not receive any acknowledgment from the receiver within the 'time-out' period, it retransmits the segment again. Our paper will start by taking a brief look at different congestion avoidance algorithms and noting how they differ from each other. In this paper, we try to represent a performance comparison table to clarify the main differences among the TCP variations further.

2 Congestion Control Mechanism

2.1 Slow start

Slow-start is a mechanism used to gradually increase the amount of data in transmission and attempts to keep the segment uniformly spaced. It is one of the most critical parts of the congestion avoidance technique used by TCP as specified by RFC 5681[2]. This technique is accomplished in conjunction with other algorithms to avoid sending more data than the network is capable of transmitting, to avoid network causing congestion. In the slow start, when a connection is established, first the value of cwnd is set to 1, and after each received ACK the value is updated to (cwnd = cwnd + 1), i.e., double of cwnd for each RTT. The exponential growth of cwnd continues until a packet loss is observed, causing the value of ssthresh to be updated to (ssthresh = cwnd/2). After the packet loss, the connection starts from the slow start again with cwnd = 1, and the window is increased exponential until it equals ssthresh, the estimate for the available bandwidth in the network. At this point, the connection goes to the congestion avoidance phase where the value of cwnd is increased less aggressively with the pattern (cwnd = cwnd + 1/cwnd), implying linear instead of exponential growth. This linear increase will continue until a packet loss is detected. That is why it is also known as the exponential growth phase.

2.2 TCP's Congestion Avoidance

In Transmission Control Protocol (TCP), the congestion window is a mechanism of stopping the link between two places from being overloaded with too much traffic, i.e., it is a way to deal with packets loss. It is one of the most critical factors that determine the number of bytes that can be outstanding at any time. Congestion usually occurs when data arrives faster (a fast LAN) and send out at a lower speed [3]. The sender maintains the congestion window where the size of this window is calculated by estimating how much congestion there is between the two places. The basis of TCP congestion control mechanism lies in Additive Increase Multiplicative Decrease (AIMD), halving the congestion window for every window

- *Syful Islam, Lecturer, Dept. of Computer Science and Telecommunication Engineering, Noakhali Science and Technology University, Email: syfulcste@gmail.com*
- *Ratnadip Kuri, Lecturer, Dept. of Computer Science and Telecommunication Engineering, Noakhali Science and Technology University, Email: ratnadipkuri@gmail.com*
- *Md. Humayun Kabir, Professor & Chairman, Dept. of Computer Science and Telecommunication Engineering, Noakhali Science and Technology University, Email: hkabir269@gmail.com*
- *Md. Javed Hossain, Associate Professor, Dept. of Computer Science and Telecommunication Engineering, Noakhali Science and Technology University, Email: javednstu@gmail.com*

containing a packet loss, and increasing the congestion window by roughly one segment per RTT otherwise. If all segments are received, and the acknowledgments reach to the sender on time, some constant is added to the window size. The window keeps growing exponentially until a timeout occurs or the receiver reaches its limit (a threshold value "thresh"). After this, the congestion window increases linearly at the rate of 1/(congestion window) packets on each new acknowledgment received. When a packet is dropped, then the congestion window(W) reduced to half. After the drop, the TCP sender increases its congestion window linearly until the congestion window has reached its old value W, and another packet drop occurs. In [4] a steady-state model, the development of TCP's congestion window is depicted below:

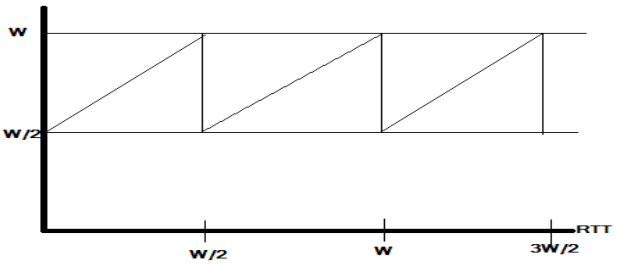

Fig.1. Development of TCP's congestion window.

2.3 TCP's Congestion Avoidance

In Transmission Control Protocol (TCP), the Fast Retransmission is a performance enhancement technique where the duplicate acknowledgment is taken as the fundamental mechanism to reduce the time a sender waits before retransmitting a lost segment. The fast-retransmit mechanism ensures the retransmit of the packet as soon as possible if the packet is lost. The TCP sender uses a timer to recognize lost segments. If an acknowledgment is not received for a particular segment within a specified time (Round-trip delay time), the sender will assume the segment was lost in the network and will retransmit the segment. The fast-retransmit works as follows: if a TCP sender receives four acknowledgments with same acknowledge number then, there is enough evidence of packet drop with the higher sequence number. The sender will retransmit the packet before its timeout. It means that instead of waiting for the retransmit timer to expire, the sender can retransmit a packet immediately after receiving three duplicate ACKs. The concept of fast retransmission technique is depicted in Fig.2.

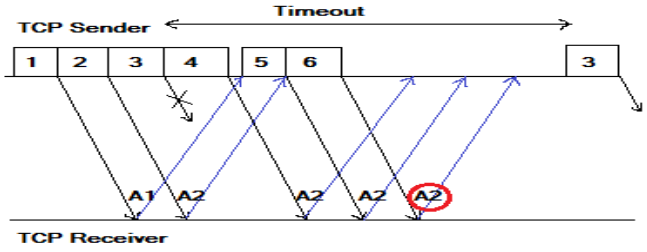

Fig.2. Representation of Fast-retransmission technique.

2.4 Fast Recovery

The Fast-Recovery algorithm is implemented together with a Fast-Retransmit algorithm that retransmits the missing packet signaled by three duplicate ACKs and wait for an acknowledgment of the entire transmit window. It is also called Fast-Retransmit/Fast-Recovery algorithm. The fast recovery is an improved version of fast retransmit and algorithms are usually implemented together [5] as follows:

a) When the third duplicate ACK in a row is received, set ssthresh to value:
ssthresh = min(cwnd/2, 2 MSS) ----------(1)
where MSS=maximum segment size.

Retransmit the missing segment. Set cwnd to ssthresh plus three times the segment size. This increases the congestion window by the number of segments that have left the network and which the other end has cached.

b) Each time another similar ACK arrives, which increase cwnd by the segment size. This also inflates the congestion window for the additional segment that has left the network. Transmit a packet, if allowed by the new value of cwnd:
cwnd = (ssthresh + no. of duplicate acks received) ------(2)

c) When the next ACK arrives, that acknowledges new data packet, set cwnd to ssthresh. This ACK is the acknowledgment of the retransmission of data from step 1, one round-trip time after the retransmission. Additionally, this ACK should acknowledge all the intermediate segments sent between the lost packet and the receipt of the first duplicate ACK. This step is congestion avoidance since TCP is down to one-half the rate it was at when the packet was lost.

3 TCP Congestion Control Algorithms In Wired Network

In this section, we performed the simulation to investigate the performance of the various congestion control algorithms (Taho, Reno, Newreno, Sack, Fack and Vegas, HSTCP, Cubic) in TCP from different aspects in a wired network. Previous work [6-7] perform comparison considering very few parameters. In our comparative analysis, we tried to investigate most of the parameter to explore actual behavior of TCP protocol over wired and wireless network. Here we use network simulator version-2.35[8-10] where TCL and OTCL scripting [11] are used for better scheduling event and controlled environment.

3.1 Comparison of Congestion window concerning time

Congestion window (CWND) of TCP changes based on the change of its basic algorithms in every TCP variant (Taho, Reno, Newreno, Sack, Fack, Vegas, HSTCP, Cubic TCP). Simulation result of congestion window describes slow start, congestion avoidance, fast retransmit and fast recovery algorithms in

TCP variants. In Fig 3 & Fig 4 we show an overall comparison of all TCP variant for low and high bandwidth network in the congestion window versus time graph.

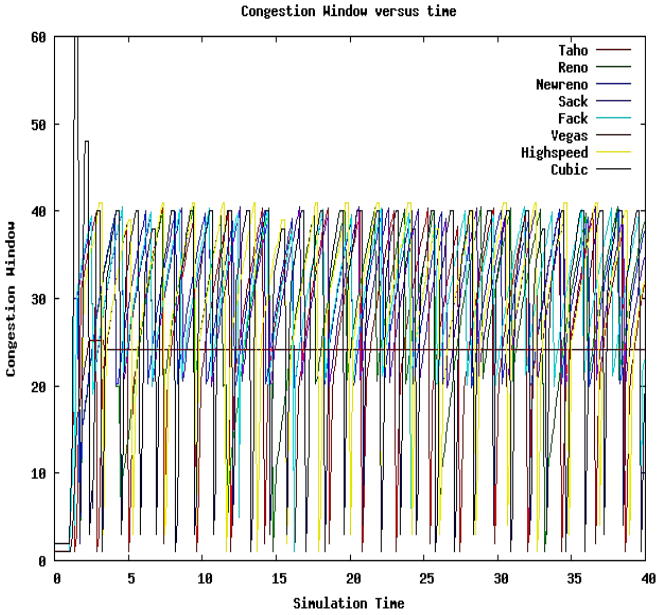

Fig 3: Overall Comparison of Congestion window versus time (20 Mb)

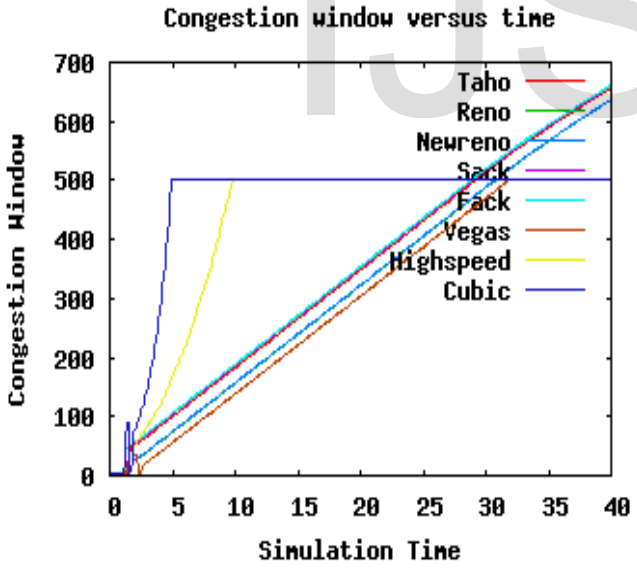

Fig 4: Overall Comparison of Congestion window versus time (2 Gb)

In TCP Vegas, Fig.3 shows that cwnd of TCP Vegas increases by the rate half than that of TCP Tahoe and New reno in slow start phase. In congestion avoidance phase cwnd is set to a constant value as cwnd is controlled according to network traffic prediction based on observed RTT values. However, in a high-speed network, TCP Vegas is quite inefficient in controlling its congestion window due to its congestion control mechanism as shown in fig 4. In High-speed TCP, Fig 4 shows the modified response function that only takes effect with higher congestion windows; it does not change TCP behavior in environments with massive congestion and therefore does not produce any new dangers of congestion drop in a high-speed network. HSTCP ensures that the response function follows a straight line on a log-log scale as does the response function for Standard TCP, for low to moderate congestion. However, in a low bandwidth network situation, it shows the opposite result compared to TCP Vegas. In Cubic TCP, Fig 4 shows that in the high-speed network it simplifies the BIC window control function and improves its TCP- friendliness and RTT fairness as BIC's growth function is too aggressive for TCP especially under short RTT or low-speed networks. As the name of the protocol represents, the window growth function of CUBIC is a cubic function regarding the surpassed time since the last loss event, whose shape is very similar to the growth function of BIC. The CUBIC function provides excellent scalability and stability. The protocol keeps the window growth rate independent of RTT, which keeps the protocol TCP friendly under short and long RTTs. The congestion epoch period of CUBIC is determined by the packet loss rate alone. We can conclude based on congestion window, that in low bandwidth network, TCP Vegas gives the better performance as it can change its congestion window based on network traffic situation but in high bandwidth network situation it completely inefficient. Again, according to the Fig 4 we can also conclude that Cubic & high-speed TCP shows its highest performance in the high-speed network where cubic TCP is best but in the low-speed network, they are completely inefficient.

3.2 Comparison of throughput (kbps) with respect to time (Second)

The throughput of TCP packets and Acknowledgement changes based on the change of its basic algorithms in every TCP variant. In Fig 5 and Fig 6 we show the comparison of throughput vs. time for TCP Taho, Reno, Newreno, Sack, Fack, Vegas, HSTCP, Cubic TCP. As we can see, initially the throughput and acknowledgment value increase abruptly and then its remain constant with respect to time for rest of the period, which indicates that packets delivery per RTT is a constant, i.e., the same number of packets are delivered by the network in a certain amount of cyclic period.

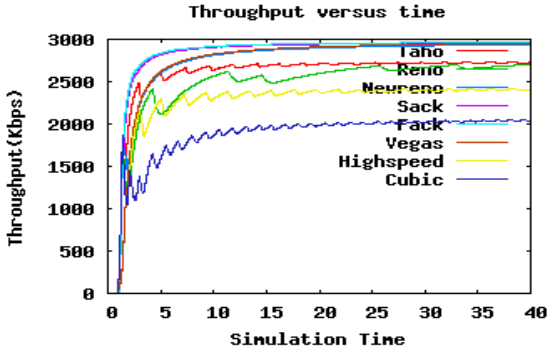

Fig 5: Overall Comparison of throughput versus time (20 Mb)

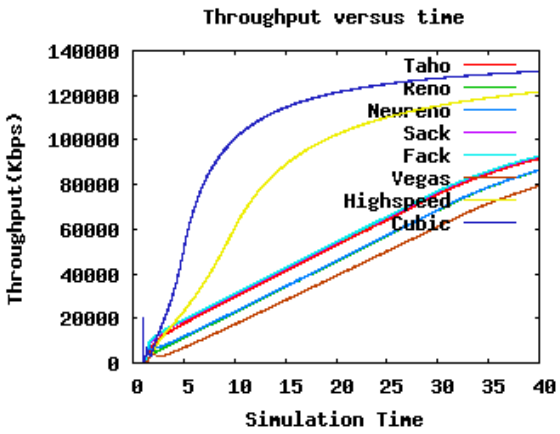

Fig.6: Overall Comparison of throughput versus time (2 Gb)

Fig 5 shows that TCP Vegas has the highest throughput, which is 2951.96 Kbps for given topology-2 because of its wise changes in slow start, congestion avoidance, and retransmission algorithms. The throughput of Newreno is 2936.81 Kbps, i.e., its performance is a little bit low as compared to Vegas due to congestion window. The TCP variants Sack and Fack have higher throughput than those of Tahoe, Reno, and Newreno. The Fack provides improved performance than that of sack in some situations. The throughput of Sack and Vegas are 2959.36 and 2964.97 respectively. However, the last & most modern variants, HSTCP & Cubic TCP show the lowest throughput value for the given topology, as it starts from slow start phase every time after retransmission.

Fig 6 shows that CUBIC TCP and HSTCP has the highest throughput for given topology-2 because of its wise changes in slow start, congestion avoidance, and retransmission algrithms although CUBIC TCP shows better result compared to HSTCP. The TCP variants Sack and Fack have higher throughput as compared to Newreno. However, The Fack provides improved performance than that of sack in some situations. The throughput of Sack and Vegas are 92596.88 and 92829.20 respectively. The throughput of Newreno is better than Vegas, Reno, Tahoe and improves performance in highspeed network and throughput is 86565.90 Kbps, i.e., its performance is a little bit low as compared to Sack due to congestion window. So finally, we can conclude that CUBIC TCP shows its very efficient performance in the high-speed network, but TCP Vegas is entirely in inefficient in challenging network situations.

3.3 Numerical Analysis

The throughput of TCP packets and Acknowledgement changes based on the change of its basic algorithms in every TCP variant. In Fig 5 and Fig 6 we show the comparison of throughput vs. time for TCP Taho, Reno, Newreno, Sack, Fack, Vegas, HSTCP, Cubic TCP. As we can see, initially the throughput.

Table-1 Comparison of Simulated data for TCP variants using different Parameters in Network Topologies.

Network Topology	BW	TCP variation Name	Result						
			Generated Packets	Received Packets	Packet dropped	Packet delivery ratio (%)	Average throughput (Kbps)	Average Ack throughput(Kbps)	End-to-end delay (ms)
Network Topology-1	2Mb	Tahoe	6634	6610	24	99.6382	262.15	8.94	49.4707
		Reno	7316	7291	25	99.6583	288.20	9.89	49.4708
		Newreno	7316	7291	25	99.6583	288.20	9.89	49.4708
		Sack1	7354	7327	27	99.6329	289.54	10.17	49.4572
		Fack	7420	7394	26	99.6496	292.31	10.20	49.4708
		Vegas	7892	7892	0	100	291.06	10.27	48.832
		HSTCP	6147	6121	26	99.577	247.39	8.33	49.6305
		Cubic	5108	5068	40	99.2169	206.43	7.30	49.6203
Network Topology-2	20Mb	Tahoe	22499	22438	39	99.7289	2725.64	95.33	5.23266
		Reno	22326	22256	41	99.6865	2703.52	94.69	5.23095
		Newreno	24250	24176	46	99.6948	2936.81	103.46	5.23158
		Sack1	24434	24362	48	99.7053	2959.36	105.30	5.23207
		Fack	24470	24408	48	99.7466	2964.97	105.54	5.23464
		Vegas	26072	26059	0	99.9501	2951.96	106.96	5.21879
		HSTCP	19554	19484	47	99.642	2406.76	83.33	5.23565
		Cubic	16577	16493	61	99.4933	2037.27	73.14	5.23474
Network Topology-3	2 Gb	Tahoe	429855	429831	14	99.9944	91757.85	1765.05	5.00416
		Reno	404986	404472	14	99.8731	86367.06	1661.55	4.99798
		Newreno	405554	405508	14	99.9887	86565.90	1666.20	5.00416
		Sack1	433808	433759	13	99.9887	92596.88	1781.70	5.00416
		Fack	434898	434847	13	99.9883	92829.20	1786.28	5.00416
		Vegas	387200	386692	0	99.8688	79398.08	1588.74	4.99754
		HSTCP	564682	564169	13	99.9092	121615.17	2316.93	5.00211
		Cubic	607428	606915	13	99.9155	130828.39	2492.42	5.00265

4 Evaluation of TCP Congestion Control Algorithms in Wireless Networks

When there is no infrastructure in the network, the mobile ad hoc network is the best choice. We need to analyses the routing mechanisms which are called the MANET routing protocols. In this section, we perform simulation in mobile ad-hoc networks to investigate the performance of the various congestion control algorithm (Tahoe, Reno, Newreno, Sack, Fack, Vegas, HSTCP, Cubic) in TCP from different aspects over two major routing protocols AODV and DSDV. In this analysis, the simulations are performed with four slightly different simulation scenarios that contain 3, 20, 30, 45 nodes respectively.

4.1 Comparison of Congestion window concerning time

Currently, all implementations of the TCP congestion algorithm assume that congestion causes timeouts but, also transmission errors. This study stands when applied to wired networks as they are relatively reliable and show very low errors. However, this concept does not stand for wireless network as they suffer from high error and packet loss rates. For this reason, any packet loss in wireless transmission is falsely considered by the TCP protocol as due to congestion which triggers the congestion algorithm to reduce the window size to one segment and consequently reducing transmission speed and packet throughput. As the TCP congestion algorithm was not initially designed to support the error-prone wireless network, but the very reliable wired network, it is impossible for the sender to differentiate between congestion loss and error loss. As a result, in timeout situations over wireless networks, the TCP often makes the wrong decision by slowing down the burst of packets while it should instead retransmit lost packets. Congestion window (CWND) of TCP in wireless network changes based on the change of its basic algorithms in every TCP variant (Tahoe, Reno, Newreno, Sack, Fack, Vegas, HSTCP, Cubic). Simulation result of congestion window describes slow start, congestion avoidance, fast retransmit and fast recovery algorithms in TCP variants.

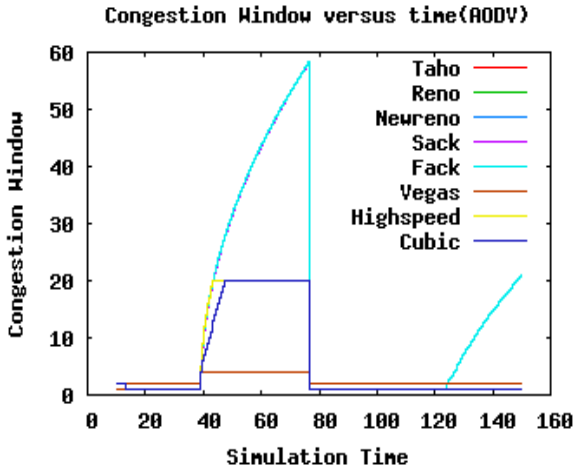

Fig 7: Overall Comparison of Congestion window versus time using AODV (3 nodes)

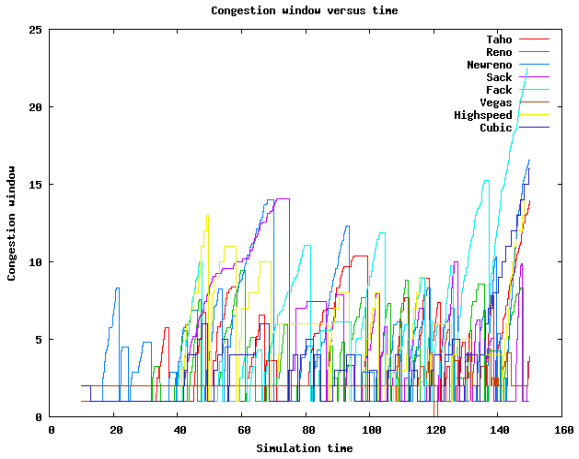

Fig 8: Overall Comparison of Congestion window versus time using AODV (45 nodes)

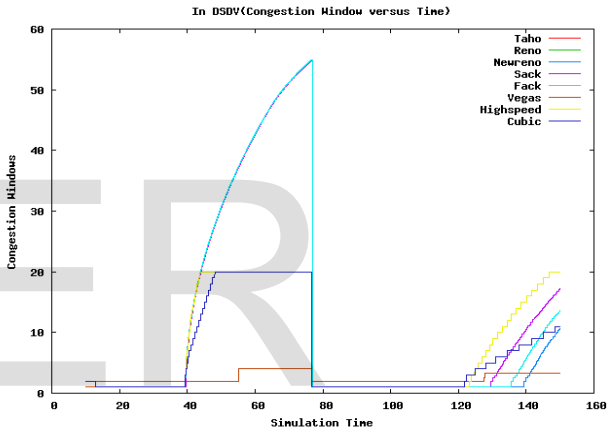

Fig 9: Overall Comparison of Congestion window versus time using DSDV (3 nodes)

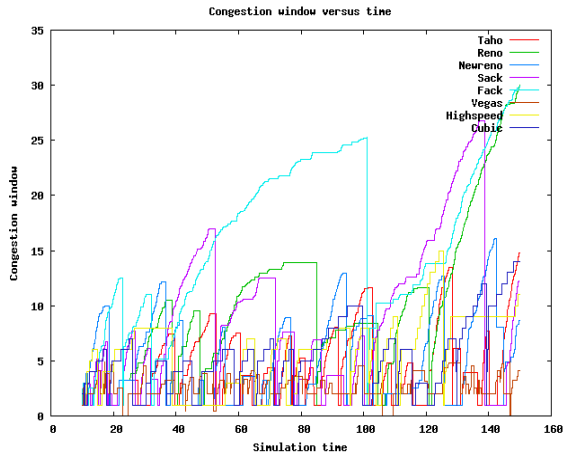

Fig 10: Overall Comparison of Congestion window versus time using DSDV (45 nodes)

Finally, Fig 7, 8, 9 & 10 shows an overall comparison of the congestion window versus time among all the TCP variants in different network situation discussed above. We can conclude based on congestion control, that in case of small number of nodes, TCP Taho, Reno, Newreno gives highest performance and in network scenarios where the number of nodes more significant than 30, CUBIC TCP is best suited when AODV routing protocol is used, and TCP Newreno is best suited when DSDV routing protocol is used as it can change its congestion window based on network traffic situation but in high bandwidth network situation it completely inefficient. Again, according to the figure above we can also conclude that Cubic & high-speed TCP shows its highest performance in network scenarios where the number of the node is higher than 30 and the routing protocol is AODV.

4.2 Comparison of throughput (kbps) with respect to time (Second)

4.2.1 Comparison of Throughput In case of AODV routing protocol

Fig 11 & Fig 12 given below show the comparison of throughput and versus time for TCP Tahoe, Reno, Newreno, Sack, Fack, Vegas, HCTCP, CUBIC in case AODV routing protocol. The simulation shows quite a different result from section 3.2.

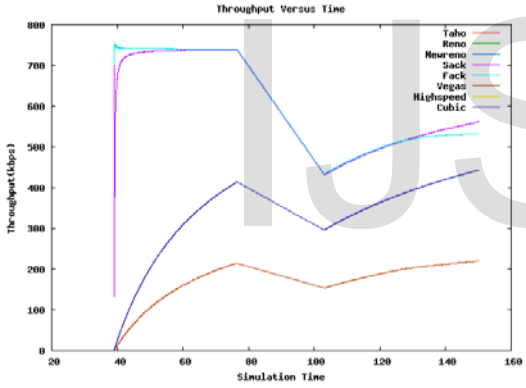

Fig 11: Overall Comparison of Congestion window versus time using AODV (3 nodes)

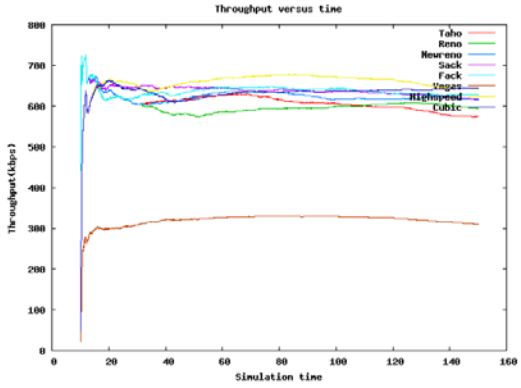

Fig12: Overall Comparison of Congestion window versus time using AODV (45 nodes)

Fig 12 shows that CUBIC TCP indicated by Blue line in the comparison graph has the highest throughput, which is 644.11Kbps for given topology-4 with 45 nodes because of its wise changes in slow start, congestion avoidance, and retransmission algorithms. HSTCP which is indicated by Yellow line is next to HSTCP in case of 45 nodes. The TCP variants Sack and Fack have higher throughput than Newreno. The Fack provides improved performance in throughput (654.70kbps) than that of sack in some situations when the number of the node is 30. The throughput of Sack and cubic are 623.45, 626.28kbps respectively. However, they improve throughput in some particular situations. The throughput of Newreno is better than Vegas, Tahoe and improves performance in high-speed network and throughput is 650.90 Kbps, i.e., its performance is a little bit low as compared to Sack due to congestion window when the number of the node is 20. The throughput of Reno in case of AODV routing protocol is 671.43kbps when the number of the node is less than or equal to 20. So finally, we can conclude that CUBIC TCP shows its very efficient performance in the network when the number of the node is higher than 30 but TCP Tahoe, Reno, Newreno gives an efficient performance when some nodes are less than 10 and is completely inefficient in challenging network situation.

4.2.2 Comparison of Throughput In case of DSDV routing protocol

Fig 13 and Fig 14 given below show the comparison of throughput and acknowledgment vs. time for TCP Tahoe, Reno, Newreno, Sack, Fack, Vegas, HCTCP, CUBIC in case of DSDV routing protocol. The simulation shows quite a different result from section 4.2.1.

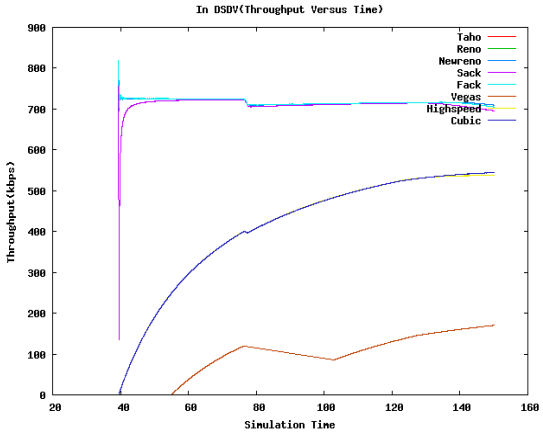

Fig 13: Overall Comparison of Congestion window versus time using DSDV (3 nodes).

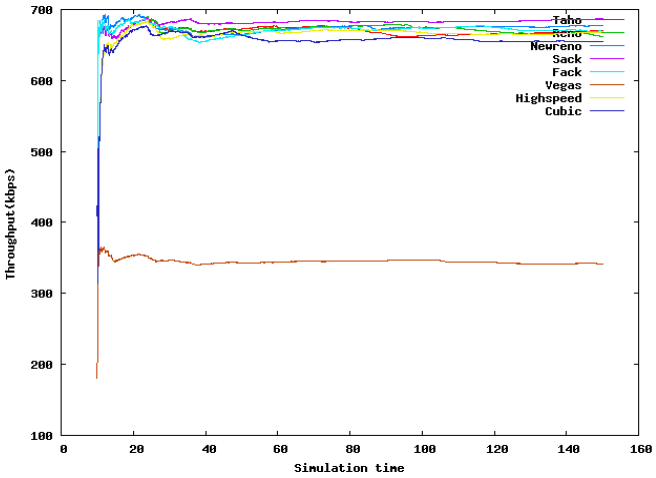

Fig 14.: Overall Comparison of Congestion window versus time using DSDV (45 nodes).

Fig 14 shows that TCP Newreno indicated by Blue line in the comparison graph has the highest throughput, which is 677.34Kbps for given wireless network topology-4 because of its wise changes in slow start, congestion avoidance, and re-transmission algorithms. Tahoe(669.97kbps), Fack (669.41kbps) which are indicated by Red and Violate line is next to TCP Newreno. The TCP variants Sack and Cubic, HSTCP have lower throughput than Newreno when the number of the node is 45. However, Sack improves throughput in some particular situations. The throughput of Reno and Cubic TCP are improved in case of nodes less than equal to 30, and the throughputs are 689.50, 684.18kbps respectively. The throughput of TCP Reno is also best 703.06kbps than NewReno, Tahoe, Sack, Fack and improves performance when the number of nodes is less than or equal to 20. So finally, we can conclude that TCP Newreno shows its very efficient performance in the network where the number of the node is 45 and the routing protocol is DSDV but TCP Tahoe, Reno is efficient in network situation where the number of the node is less than 10.

4.3 Numerical analysis of TCP Over MANET

In the end simulation for each TCP variants, we have calculated the performance parameter which will further help to identify which TCP version is best suited for which situation. From the table below, we can summarize that CUBIC TCP is best suited for a network where the number of the node is higher than 45 and routing protocol is AODV and TCP Newreno is best for a network where the number of the node is higher than 45 and Routing protocol is DSDV. Again, TCP Reno performs its best when in both of AODV and DSDV routing protocol when the number of the node is less than or equal to 20 nodes.

Table-2: Simulated Data Table of Network Topology-1,2,3 & 4 using TCP variants using AODV

Routing Protocol	TCP Variants	Throughput(kbps) Number of Nodes			
		3 Nodes	20 Node	30 Nodes	45 Nodes
AODV	Taho	561.21	664.67	623.91	575.27
	Reno	561.21	671.43	625.14	594.66
	Newreno	561.21	650.05	604.04	617.35
	Sack	561.21	618.11	623.45	615.35
	Fack	532.45	658.36	654.70	628.10
	Vegas	219.76	304.84	316.68	310.22
	HSTCP	442.80	598.68	604.18	641.34
	Cubic	442.84	609.28	626.57	644.11

Table-3: Simulated Data Table of Network Topology-1,2,3&4 using TCP variants in case of DSDV

Routing Protocol	TCP Variants	Throughput(kbps) Number of Nodes			
		3 Nodes	20 Nodes	30 Nodes	45 Nodes
DSDV	Taho	709.71	701.44	675.82	669.97
	Reno	709.71	703.06	689.50	662.42
	Newreno	709.71	693.95	604.04	677.34
	Sack	695.55	687.39	680.65	686.77
	Fack	705.23	693.72	654.70	669.41
	Vegas	170.34	353.19	316.68	341.62
	HSTCP	536.90	691.80	644.11	668.39
	Cubic	544.30	685.85	684.18	654.20

Table-4: Comparison of TCP variants based on their main Property (Congestion Control Algorithm)

Properties / TCP Variants	Slow Start	Congestion Avoidance	Fast-Retransmit	Fast-Recovery	Advantage & Problem
Taho	The sender sets the CWND to 1 and then for each ACK received, it increases the CWD by 1 in each RTT & lasts till the cwnd reaches ssthresh.	Uses AIMD technique where cwnd is set to half of the current window size than on each ACK for new data cwnd is increased by (1/cwnd).	On receiving three duplicate ACKs, it retransmits packet and set ssthresh to half of cwnd and then enters in slow start phase by setting cwnd to one segment.	----------	Takes a complete timeout interval to detect a packet loss.
Reno (Modification of Taho)	Same as Taho	Same as Taho	Same as TCP Taho but does not return to slow start phase and effectively waits until half of a window of duplicate ACKs have been received.	The number of duplicate ACKs inflates the current CWND.	It performs well when the packet losses are small, but on multiple packet losses, its performance is as weak as TCP Taho.
Newreno (Modification of Reno)	Same as Reno	Same as Reno	Same as Reno	Here the new reno "partial ACK" does not deflate the usable window back to the size of the CWND and it does not exit fast-recovery until all the outstanding data is acknowledged.	Overcome the problem of Reno, but it suffers from the fact that it takes one RTT to detect each packet loss.
Sack (Selective Acknowledgement)	Same as New Reno	Same as New Reno	Same as New Reno but the sender only retransmits data when the estimated number of packets in the path is less than the CWND	It maintains a variable called pipe that represents the estimated number of packets outstanding in the path and handle multiple packet loss.	It Overcomes the problem of Newreno, but the biggest problem with SACK is that currently selective acknowledgments are not provided by the receiver
Fack (Forward Acknowledgement)	Same as Sack	Same as Sackbut decouples congestion control from data recovery thereby attaining more precise control over the data flow in the network.	Same as Sack	It uses the forward SACK sequence number as a sign that all the previous un-(selectively)-acknowledged segments were lost which observation allows improving recovery	It accomplishes more precise control over the data flow in the network but in the experimental stage.

				of losses significantly	
Vegas (Modification of Reno)	It offers a modified slow start algorithm which prevents it from being congested the network [10].	It determines congestion by a decrease in the sending rate as compared to the expected rate.	Modification of Reno where if the (current segment transmission time)> RTT; it then immediately retransmits the segment without waiting for three duplicate acknowledgments [8]	----------	Overcome most of the problem of the above TCP variants and very efficient in data transmission.
HSTCP	----------	Loss-based TCP congestion control and only takes effect with higher congestion windows.	----------	----------	It does not modify TCP behavior in environments with heavy congestion, and therefore does not introduce any new dangers of congestion collapse
CUBIC TCP	----------	It uses a Cubic function instead of a linear window increase of the current TCP standards.	----------	----------	Improved scalability and stability under fast and long-distance networks

5 Conclusion

In this work, the analysis is done on TCP variants named TCP Tahoe, Reno, Newreno, Sack, Fack, Vegas, HSTCP, and Cubic TCP in both wired and wireless network. Here we have evaluated the performance characteristics (Congestion window, Throughput, Delay, Jitter, Packet delivery ratio, End-to-end delay) of various TCP congestion control schemes under the wired network conditions with bottleneck end-to-end link capacities. We tried to find some critical cases in which TCP Reno, TCP New Reno, TCP Vegas, Cubic TCP, HSTCP make some performance improvements compared to all simulated TCP variants. From simulated data and graphs obtained, we tried to find which TCP variants are better for challenging network situations. In low bandwidth network, we find that both TCP Vegas and TCP SACK make some performance improvements to TCP Reno. TCP Vegas achieves higher throughput than Reno and SACK for large loss rate. TCP SACK is better when more than one packet dropped in one window. TCP Vegas causes much fewer packets retransmissions than TCP Reno and SACK. We have also shown that TCP Vegas does lead to a fair allocation of bandwidth for different delay connections. Both TCP Reno and SACK bias against long delay connections. But TCP Vegas shows very low throughput in case of high bandwidth network. In high bandwidth network, Cubic TCP achieves higher throughput compared to all other TCP variants. But one drawback of Cubic TCP is its low throughput in low bandwidth network. We have also concentrated on the behavior of TCP's reliability in the mobile ad-hoc network. In MANET, TCP Newreno, Cubic TCP achieve the highest throughput when the routing protocol is DSDV and AODV respectively, and a number of the node is higher than 30. Finally, we have prepared a detail comparison table and some suggestion in improving the drawback of TCP variants and try to comment which variation is better for which network situation.

ACKNOWLEDGMENT

The authors would like to thanks the department of computer science and Telecommunication Engineering for supporting the research.

REFERENCES

[1] Jian (Jason) and Yi Zhen. "Comparison of different TCP congestion control mechanism.", Link: *http://www2.ensc.sfu.ca*, Access date: 10.04.2017.W.-K. Chen, *Linear Networks and Systems*. Belmont, Calif.: Wadsworth, pp. 123-135, 1993. (Book style)

[2] M. Allman, V. Paxson, E. Blanton. "*TCP Congestion Control*", RFC5681, September 2009

[3] Digvijaysinh B Kumpavat , Prof. Paras S Gosai and Vyomal N Pandya. "Comparison of TCP congestion control mechanisms Tahoe, Newreno and Vegas". *IOSR Journal of Electronics and Communication Engineering*, Volume 5, Issue 6, PP 74-79

[4] S. Floyd. "Promoting the Use of End-to-End Congestion Control in the Internet". *IEEE/ACM Transactions on Networking*, Volume 7, Issue 4.

[5] W. Stevens. "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms". *RFC 2001*, January 1997.

[6] Md. Shohidul Islam, M.A Kashem, W.H sadid, M.A Rahman, M.N Islam, S. Anam. "TCP Variants and Network Parameters: A Comprehensive Performance Analysis", *International Multiconference of Engineerings and Computer Scientists 2009*, Date: March 18-20 2009, pp:351-353.

[7] Lawrence S. Brakmo and Larry L. Peterson. "TCP Vegas: End to End Congestion Avoidance on a Global Internet", *IEEE Journal on Selected Areas in Communication*, Volume 13, No 8, pp:1465-1480.

[8] Hui (Hilary) Zhang and Zhengbing Bian, Evaluation of different TCP congestion control algorithms using NS-2, Link:*http://www2.ensc.sfu.ca/~ljilja/ENSC835/Spring02/Projects/bian_zhang.hilary/Hilary_and_Bian_Report.pdf*, Access Date: 5-07-2017.

[9] The Network Simulator - ns-2, Link: *http://www.isi.edu/nsnam/ns*, Date Access: 7.06.2017.

[10] Paul Meeneghan and Declan Delaney, *An Introduction to NS, Nam, and Otcl scripting*, Department of Computer Science, National University of Ireland, Maynooth, 2004.

[11] TCL and OTCL programming, Link: *http://www.mathcs.emory.edu/~cheung/Courses/558/Syllabus/A2-Tcl/Tcl-1-vars.html*, Access Date: 6.04.2017