# Scenario Based Performance Analysis of Variants of TCP Using NS2 - Simulator

Yuvaraju B N
Department of Computer Science
& Engineering
N M A M Institute of Technology,
Nitte – 574 110, Karnataka, INDIA

Dr. Niranjan N Chiplunkar
Department of Computer Science
& Engineering
N M A M Institute of Technology,
Nitte – 574 110, Karnataka, INDIA

## ABSTRACT
The increasing demands and requirements for wireless communication systems especially in settings where access to wired infrastructure is not possible like natural disasters, conferences and military settings have led to the need for a better understanding of fundamental issues in TCP optimization in MANETS.TCP primarily designed for wired networks, faces performance degradation when applied to the ad-hoc scenario. Earlier work in MANETs focused on comparing the performance of different routing protocols. Our analysis is on performance of the variants of TCP in MANETS. Several different variants have been developed in order to refine congestion control in Mobile Adhoc Networks. These variants of TCP perform better under specific scenarios, our Analysis of the variants of TCP is based on three performance metrics: TCP Throughput ,Average End- to -End delay and Packet Delivery Fraction in high and low mobility. This analysis will be useful in determining the better variant among TCP Protocols to ensure better data transfer, speed, reliability and congestion control. In this paper we carry out performance study of six variants of TCP to be able to classify which variant of TCP performs better in various possible scenarios in MANETs.

**Index Terms:** Ad-hoc Networks, Congestion and Mobile Relays.

## 1. INTRODUCTION
Effectively and fairly allocating the resources of a network among a collection of competing users is a major issue. The resources of a network being shared include the bandwidth of the links and the queues on the routers or switches. Packets are queued in these queues awaiting transmission. When too many packets are contending for the same link, the queue overflows and packets have to be dropped. When such drops become common events, the network is said to be congested [1].

Networks are mainly classified into wired networks and wireless networks. In wired networks routers are separate network elements that have the sole functionality of routing the packets. Wireless networks [6] are classified into infrastructure based networks and infrastructure less networks. Ad-hoc networks [5] are infrastructure-less networks. In Ad-hoc networks, since there is no fixed infrastructure there are no separate network elements called routers and hence the mobile nodes themselves act as the routers (i.e. they are responsible for routing the packets).

Congestion control methods [4] can be router centric or host/node centric. In existing congestion control methods, the source is informed about the congestion in the network so that either it may slow down the packet transmission rate or find an alternate route which may not necessarily be an optimal route. It must be pointed out that all the congestion control methods are able to inform the source about the congestion problem because they use Transmission Control Protocol (TCP).

Our proposed method to solve the congestion problem can be implemented only in the network elements (e.g.: routers) and is independent of the underlying transport protocols. Hence this congestion control method can be used for Transmission Control Protocol (TCP) as well as User Datagram Protocol (UDP).

## 2. CONGESTION CONTROL MECHANISMS

This section describes the predominant example of end-to-end congestion control[2] in use today, that implemented by TCP. The essential strategy of TCP is to send packets into the network without a reservation and then to react to observable events that occur. TCP assumes only FIFO queuing in the network's routers, but also works with fair queuing.

### 2.1 Additive Increase/Multiplicative Decrease

TCP maintains a new state variable for each connection, called Congestion Window[3], which is used by the source to limit how much data it is allowed to have in transit at a given time. The congestion window is congestion control's counterpart to flow control's advertised window. TCP is modified such that the maximum number of bytes of unacknowledged data allowed is now the minimum of the congestion window and the advertised window.

MaxWindow = MIN (CongestionWindow, AdvertisedWindow)

EffectiveWindow = MaxWindow − (LastByteSent − LastByteAcked).

That is, MaxWindow replaces AdvertisedWindow in the calculation of EffectiveWindow.

Thus, a TCP source is allowed to send no faster than the slowest component—the network or the destination host—can accommodate.

The problem, of course, is how TCP comes to learn an appropriate value for CongestionWindow. Unlike the AdvertisedWindow, sent by receiving side of the connection, there is no one to send a suitable CongestionWindow to the sending side of TCP.

TCP does not wait for an entire window's worth of ACKs to add one packet's worth to the congestion window, but instead increments CongestionWindow by a little for each ACK that arrives. Specifically, the congestion window is incremented as follows each time an ACK arrives:

Increment = MSS × (MSS/CongestionWindow)

CongestionWindow + = Increment

That is, rather than incrementing CongestionWindow by an entire MSS bytes each RTT, we increment it by a fraction of MSS every time an ACK is received. The important concept to understand about AIMD is that the source is willing to reduce its congestion window at a much faster rate than it is willing to increase its congestion window.

## 2.2 Fast Retransmit and Fast Recovery

The mechanisms described so far were part of the original proposal to add congestion control to TCP. It was soon discovered, however, that the coarse-grained implementation of TCP timeouts led to long periods of time during which the connection went dead while waiting for a timer to expire. Because of this, a new mechanism called *fast re- transmit* was added to TCP. Fast retransmit is a heuristic that sometimes triggers the retransmission of a dropped packet sooner than the regular timeout mechanism.

# 3. VARIANTS OF TCP

## 3.1 TCP TAHOE

Tahoe[10] refers to the TCP congestion control algorithm which was suggested by Van Jacobson in his paper. TCP is based on a principle of 'conservation of packets', i.e. if the connection is running at the available bandwidth capacity then a packet is not injected into the network unless a packet is taken out as well. TCP implements this principle by using the acknowledgements to clock outgoing packets because an acknowledgement means that a packet was taken off the wire by the receiver. It also maintains a congestion window CWD to reflect the network capacity**.** Tahoe suggests that whenever a TCP connection starts or re-starts after a packet loss it should go through a procedure called 'slow-start'. The reason for this procedure is that an initial burst might overwhelm the network and the connection might never get started. The congestion window size is multiplicatively increased that is it becomes double for each transmission until it encounters congestion. Slow start suggests that the sender set the congestion window to 1 and then for each ACK received it increase the CWD by 1. So in the first round trip time (RTT) we send 1 packet, in the second we send 2 and in the third we send 4. Thus we increase exponentially until we lose a packet which is a sign of congestion. When we encounter congestion we decreases our sending rate and we reduce congestion window to one. And start over again. The important thing is that Tahoe detects packet losses by timeouts. Sender is notified that congestion has occurred based on the packet loss.

## 3.2 TCP RENO

This RENO retains the basic principle of Tahoe, such as slow starts and the coarse grain retransmit timer. 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. Reno requires that we receive immediate acknowledgement whenever a segment is received. The logic behind this is that whenever we receive a duplicate acknowledgment, then his duplicate acknowledgment could have been received if the next segment in sequence expected, has been delayed in the network and the segments reached there out of order or else that the packet is lost. If we receive a number of duplicate acknowledgements then that means that sufficient time have passed and even if the segment had taken a longer path, it should have gotten to the receiver by now. There is a very high probability that it was lost. So Reno suggests an algorithm called **'Fast Re-Transmit'.** Whenever we receive 3 duplicate ACK's we take it as a sign that the segment was lost, so we re-transmit the segment without waiting for timeout.

Thus we manage to re-transmit the segment with the pipe almost full. Another modification that RENO makes is in that after a packet loss, it does not reduce the congestion window to 1. Since this empties the pipe. It enters into an algorithm which we call **'Fast-Re-Transmit'.**

**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.
- Another problem is that if the window is very small when the loss occurs then we would never receive enough duplicate acknowledgements for a fast retransmit and we would have to wait for a coarse grained timeout. Thus is cannot effectively detect multiple packet losses.

## 3.3 New RENO

New RENO 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 duplicate packets, 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. 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 we entered fast recovery then it exits fast recovery and sets CWD to threshold value 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 duplicate ACKS received to zero. It exits Fast recovery when all the data in the window is acknowledged

***Problems:***

New-Reno suffers from the fact that it takes one RTT to detect each packet loss. When the ACK for the first retransmitted segment is received only then can we deduce which other segment was lost.

## 3.4 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. 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.

## 3.5 TCP FACK

FACK or Forward Acknowledgement is a special algorithm that works on top of the SACK options, and is geared at congestion controlling. FACK algorithm uses information provided by SACK to add more precise control to the injection of data into the network during recovery – this is achieved by explicitly measuring the total number of bytes of data outstanding in the network. FACK decouples congestion control from data recovery thereby attaining more precise control over the data flow in the network. The main idea of FACK algorithm is to consider the most forward selective acknowledgement sequence number as a sign that all the previous un-(selectively)-acknowledged segments were lost. This observation allows improving recovery of losses significantly.

## 3.6 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. The three major changes induced by Vegas are:

**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.

**Congestion avoidance:** TCP Vegas is different from all the other implementation in its behavior 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 crow croft's Tri-S scheme.

**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.

# 4. RESULTS AND ANALYSIS BASED ON THROUGHPUT

## 4.1 Analysis of graph of throughput versus no of nodes

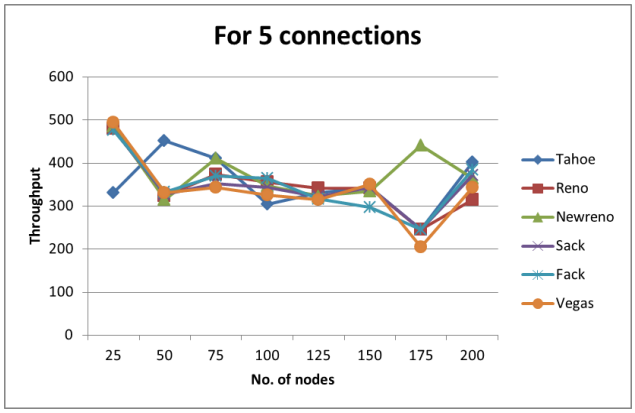


**Figure 1: Graph of No. of Nodes vs. Throughput for 5 Connections in Low Mobility**

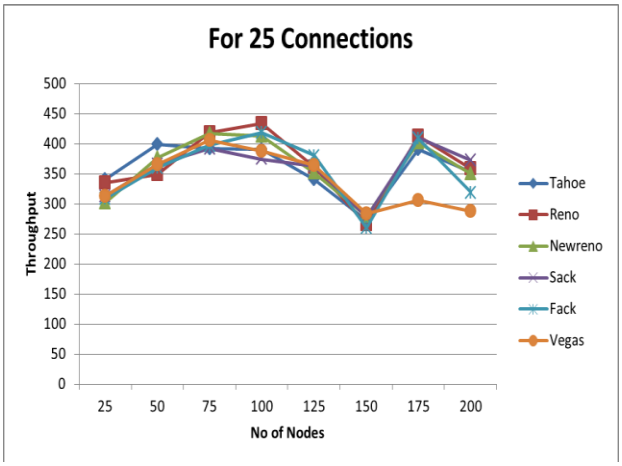


**Figure 2: graph of no. Of Nodes vs. Throughput for 25 connections in low mobility**

Figure 1 and 2 are graphs of Throughput v/s Number of nodes for 5 connections and 25 connections respectively. As we mentioned earlier, large variations are observed in the graph because TCP's performance in Mobile Ad hoc Networks is affected due to network asymmetry. Also the behavior of the underlying routing

protocol used affects the performance of TCP. The routing protocol used in our simulations is AODV which maintains only one hop information in the routing table. Due to reactive nature of AODV it may happen that data packets and ACK packets may take a different path from source to destination. This results in large variations of RTT. TCPs basic functioning depends largely on RTT and thus due to variations in RTT, we observe variations in throughput achieved as shown above in figures1 and 2.

## 4.2 Analysis of graph of throughput versus no of connections with low mobility

The below Figure 3 and 4 are graphs of Throughput v/s Number of connections for 100 nodes and 200 nodes respectively. It is observed that almost all the variants of TCP have similar performance except TCP Vegas. The performance of TCP Vegas is similar to other variants of TCP initially but later when we increase the number of connections, the performance degrades drastically. When we increase the number of connections in a network (keeping number of nodes fixed) more packets are dropped in the network due to collision. TCP Vegas has a proactive behavior that prevents the packets being dropped in the network. Due to this nature it restricts the amount of data that it transmits in the network. Thus TCP Vegas achieves low throughput as compared to other variants.

TCP SACK gives better throughput than other variants in most of the scenarios. This is because it avoids frequent retransmission of packets by sending selective acknowledgements. This mechanism is better than the mechanisms used in TCP RENO and TCP New RENO where in multiple packet losses lead to frequent retransmission of packets. Comparing figure 3 and 4 we can see that more throughput is achieved when the number of nodes are more . We can see that the range of values obtained for throughput in figure 3 is from 250 to 425 whereas in case of figure 4 it is from 300 to 500. This is because when we increase the number of nodes in a network, node density also increases.

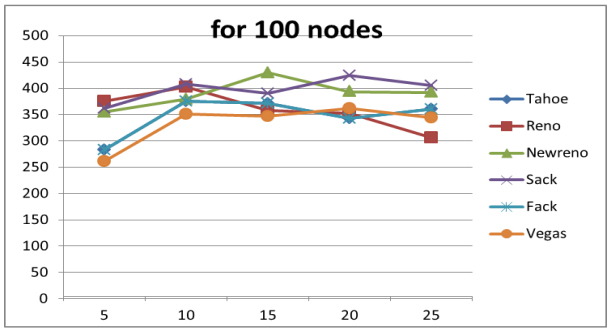


**Figure 3: Graph of No. of Connections vs. Throughput for 100 nodes in Low Mobility**

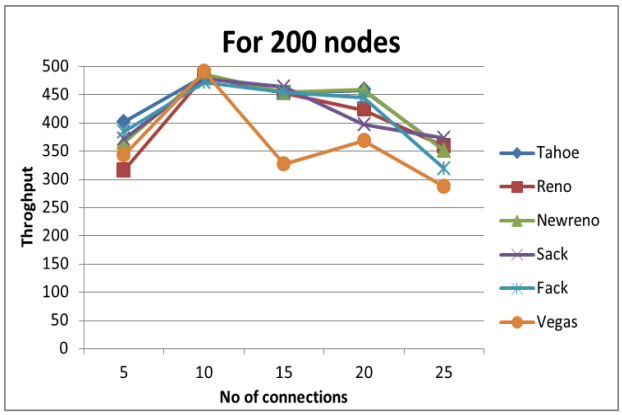


**Figure 4: Graph of No. of Connections vs. Throughput for 200 nodes in Low Mobility**

## 4.3 Analysis of graph of throughput versus no of connections with high mobility

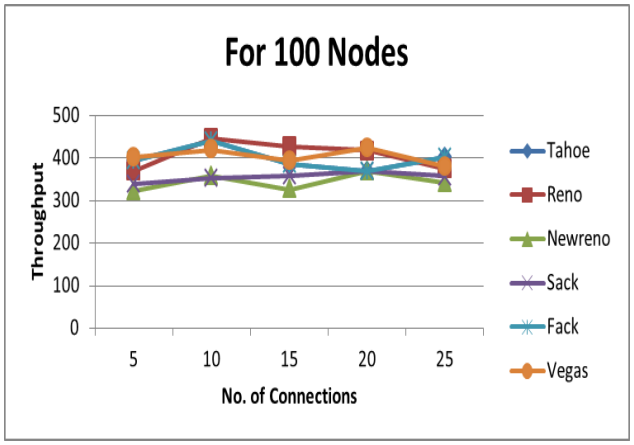


**Figure 5: Graph of No. of Connections vs. Throughput for 100 nodes in High Mobility**

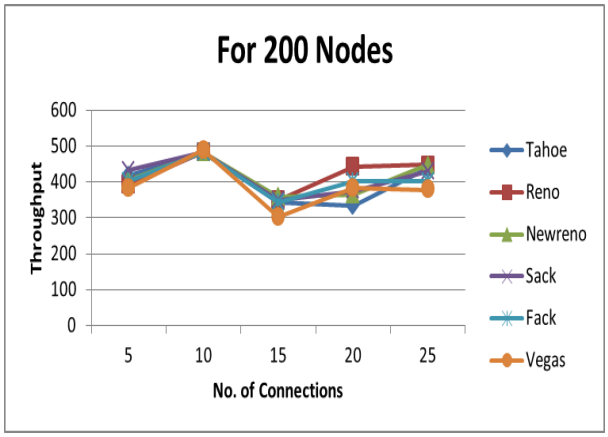

**Figure 6: Graph of No. of Connections vs. Throughput for 200 nodes in High Mobility**

The Figure 5 and 6 are graphs of Throughput v/s Number of connections for 100 nodes and 200 nodes with high mobility. It is observed that almost all the variants of TCP have similar performance except TCP Vegas. The performance of TCP Vegas is similar to other variants of TCP initially but later when we increase the number of connections, the performance degrades drastically. When we increase the number of connections in a network (keeping number of nodes fixed) more packets are dropped in the network due to collision. TCP Vegas has a proactive behavior that prevents the packets being dropped in the network. Due to this nature it restricts the amount of data that it transmits in the network. Thus TCP Vegas achieves low throughput as compared to other variants.

We can also observe the fact that wireless networks have hidden node and exposed node problems. Hidden node problem can be solved by Carrier Sensing, wherein a RTS (Request to Send) packet is first sent by the sender and if the transmission medium is free then a CTS (Clear to Send) packet is sent by the receiver. Once a CTS packet is sent, the two nodes can communicate with each other for the requested amount of time. However this cannot solve exposed node problem. Two other nodes which are in the transmission range of the two nodes communicating will have to wait until the transmission medium is free. This would cause the throughput to drop. This can be seen in fig 5 and fig 6 where throughput drops when the connection is increased to 15.

## CONCLUSION AND FUTURE WORK

We calculated the performance of six TCP variants; they are TCP Tahoe, TCP RENO, TCP New RENO, TCP SACK, TCP FACK and TCP Vegas. After analyzing the performance from simulated data and graphs obtained, we found that TCP Vegas is better than any other TCP variants for sending data and information due to its better Packet Delivery Fraction and Avg. end- to- end delay in both high and low mobility. This is due to fine tuning of congestion window size by taking into consideration the RTT of a packet, whereas other reactive protocols like TCP Tahoe, RENO, New RENO, SACK, and FACK continue to increase their window size until packet loss is detected. We have given the detailed behavior of all these variants of TCP under various different scenarios. We hope these results will be of some use in future study in this area helping the growing interest and resulting in the required protocol for today's high demanding world.

The future work of this project can be done in following areas:

1) The performance analysis of variants of TCP under other routing protocols like DYMO, DSR, OLSR, DSDV.
2) Expanding the range of analysis by considering other new TCP's like HS-TCP, TCP WESTWOOD etc
3) Working using the other two propagation models in ns2 - Shadowing model and Free space model
4) Considering more performance metrics like Routing Overhead, Bandwidth Delay Product, Total route requests sent, Retransmission attempts etc.

**ACKNOWLEDGMENT**

The authors would like to thank Mr. Mohit T and Ms Manasa for their contribution to the programming for some associated experiments. They would also like to thank Dr. S.Y Kulkarni, Pricipal, for his encouragement and helpful comments.

## REFERENCES

[1] Raju Kumar, Riccardo Crepaldi, Hosam Rowaihy, Albert F. Harris III, Guohong Cao, Michele Zorzi, Thomas F. La Porta "Mitigating Performance Degradation in Congested Sensor Networks", IEEE Transactions on Mobile Computing, Vol. 7, No. 6, June 2008.

[2] Jeffrey Andrews et al, "Rethinking Information Theory for Mobile Ad Hoc Networks" IEEE Communications Magazine, December 2008.

[3] Xiaoqin Chen, Haley M. Jones, A.D.S Jayalath "Congestion Aware Routing Protocol for Mobile Ad-hoc Networks", Department of Information Engineering, CECS, The Australian National University, Canberra.

[4] P. Chenna Reddy, Dr. P. ChandraSekhar Reddy. "Performance Analysis of Adhoc Network Routing Protocols".

[5] Larry L. Peterson and Bruce s. Davie."Computer Networks-A Systems Approach.Edition-3"Morgan Kaufmann Publishers.

[6] Subir Kumar Sarkar, T G Basavaraju, C Puttamadappa."Adhoc Mobile Wireless Networks-Principles, Protocols and Applications" Auerbach Publications.

[7] Marc Greis' Tutorial for the UCB/LBNL/VINT Network Simulator "ns". http://www.isi.edu/nsnam/ns/tutorial/

[8] Network Simulator - 2 (NS-2) http://mohit.ueuo.com/NS-2.html

[9] Christian Lochert, Bjorn Scheuermann, Martin Mauve."A Survey on Congestion Control in Mobile Adhoc Networks."

[10] Laxmi Subedi, Mohamadreza Najiminaini, and Ljiljana Trajković "Performance Evaluation of TCP Tahoe, Reno, Reno with SACK, and NewReno using OPNET Modeler"