# A Comparative Analysis of TCP Tahoe, Reno, New-Reno, SACK and Vegas in Homogeneous Networks

T Nishitha (*Author*)
Computer Science and Engineering
Vasavi College of Engineering
Hyderabad, India

*Abstract— In this paper, I analyzed and compared different congestion control and avoidance mechanisms which have been proposed for TCP/IP protocols, namely: Tahoe, Reno, New-Reno, TCP Vegas and SACK. Congestion is the reduced quality of service that occurs when a node carries more data than it can handle. The methods for congestion control suggest when a segment should be re-transmitted and how the sender behaves when it encounters congestion and what pattern of transmissions should it follow to avoid congestion. In this paper, I discussed how different mechanism affects the throughput and efficiency of TCP and how they compare with TCP Vegas in terms of performance.*

***Keywords—congestion control; TCP variants; performance; MANETs;***

## I. INTRODUCTION

Transmission Control Protocol (TCP) is a reliable connection oriented end-to-end protocol. TCP itself has mechanisms for ensuring reliability by requiring the receiver acknowledge the segments it receives. The network is not perfect and a small percentage of packets will be lost, either due to network error or due to the fact that there is congestion in the network and the routers are dropping packets. We assume that packet losses due to network loss are minimal and most of the packet losses are due to buffer overflows at the router**.** Thus it is important for a TCP protocol to react to a packet loss and take action to reduce congestion.

TCP uses timer for reliable transmission and starts timer whenever it sends a segment and waits for the acknowledgement. If the acknowledgement is not received from the receiver within the 'time-out' interval then it retransmits the segment.

This section describes about four intertwined congestion control mechanisms in TCP: slow start, congestion avoidance, fast retransmit and fast recovery.

### *A. Slow start and congestion avoidance*

The TCP sender employs the slow start and congestion avoidance algorithms to avoid more data to be sent in the network than it is capable of. To implement these algorithms, two flow control variables, namely, the congestion window and the advertised window are used in each TCP connection state. The TCP sender imposes the congestion window while the receiver imposes the advertised window. The minimum of the congestion window and the advertised window regulates the data transmission. Besides, the slow start threshold (ssthresh), known as a state variable, is used to decide which one is to be used among the slow start or congestion avoidance algorithms for controlling the data transmission. During the beginning of the transmission, there are many unfamiliar conditions present in the network; therefore TCP needs to gradually discover the network by assessing the bandwidth and determining the available capacity. This will eventually prevent the network from being congested with large bursts of data.

The following Fig 1.1 shows the slow start and congestion avoidance mechanisms executed by the TCP. After establishing a new connection, TCP starts the slow start mechanism and sets its congestion window size to one segment and increments its window size by one upon receiving the acknowledgement. Thus, 1 packet is sent in the first round trip time (RTT), 2 packets are sent for the second RTT, 4 packets are sent for the third RTT and continue incrementing exponentially. Hence, slow start phase is also known as the exponential growth phase where slow start increases the window size by the number of segments acknowledged. This process will be continued until either of the following situations occurs: 1) an acknowledgment is not received for some segments 2) a predetermined slow start threshold value is reached 3) the congestion window size becomes equal to the receiver's advertised window size. If either of these events takes place, TCP enters the congestion avoidance (linear growth) phase. Each time an ACK is received, congestion avoidance suggests that the congestion window size should be increased by (segment size*segment size)/congestion window. Here, segment size and congestion window is maintained in bytes.

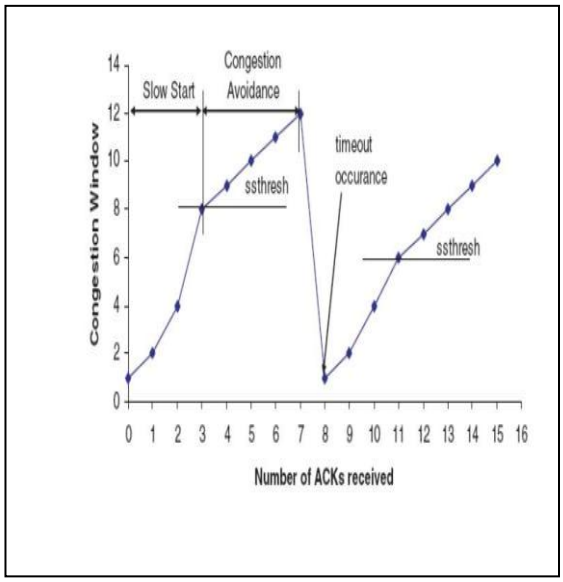


Fig 1.1: Slow Start and Congestion Avoidance

### B. *Fast Retransmission and Fast Recovery*

Whenever a packet segment is transmitted, TCP sets a timer each time and thus reliability is ensured. TCP retransmits the packet, if it does not obtain any acknowledgement within the fixed time-out interval. The reason for not getting any acknowledgement within a specific duration is due to either the packet loss or the network congestion. Therefore the TCP sender implements the fast retransmit algorithm for identifying and repairing the loss. This fast retransmit phase is applied mainly based on the incoming duplicate ACKs. As TCP is not able to understand whether a packet loss or an out-of-order segment causes the generation of the duplicate ACK, it waits for more duplicate ACKs to be received. Because in case of out-of-order segment, one or two duplicate ACKs will be received before the reordered segment is processed. On the other hand, if there are at least three duplicate ACKs in a row, it can be assumed that a segment has been lost. In that case, the sender will retransmit the missing data packets without waiting for a retransmission timer to expire. After the missing segment is retransmitted, the TCP will initiate the fast recovery mechanism until a non-duplicate ACK arrives. The fast recovery algorithm is an improvement of congestion control mechanism that ensures higher throughput even during moderate congestion. The receiver yields the duplicate ACK only when another segment is reached to it; therefore this segment is kept in the receiver's buffer and does not consume any network resources. This means, data flow is still running in the network, and TCP is reluctant to reduce the flow immediately by moving into the slow start phase. Thus, in fast recovery algorithm, congestion avoidance phase is again invoked instead of slow start phase as soon as the fast retransmission mechanism is completed.

## II. LITERATURE REVIEW

*TCP Variants*

The original design of the Transmission Control Protocol (TCP) worked reliably, but was unable to provide acceptable performance in a large and congested network. The development of the TCP has therefore been made progressively since its original incarnation in 1988. This section presents several TCP versions which have been proposed with different mechanism in order to control and avoid the network congestion.

### A. *TCP Tahoe*

The earlier versions of TCP offered a go-back-n model which used to implement the cumulative positive acknowledgment. For this purpose, retransmit timer expiration was needed in order to re-transmit the lost data. However, these former versions were unable to reduce the network congestion. Hence, for modification to earlier TCP implementations, the TCP Tahoe variant was implemented with slow-start, congestion avoidance, and fast retransmits algorithms. This version modified the round-trip time (RTT) estimator which is needed for adjusting the values of transmission timeout (RTO). In Tahoe version, when the sender accepts three duplicate acknowledgments for a single TCP segment, it assumes that a data packet is lost and hence resends the data packet regardless of the expiration of the retransmission time.

However, to identify a packet loss, the TCP Tahoe version needs a complete timeout interval or even longer sometimes due to the coarse grain timeout. In addition, upon detection of a packet loss, every time it waits until the pipeline is emptied which eventually establish a high cost in the band-width delay product links.

Problems:

The problem with Tahoe is that it take a complete timeout interval to detect a packet loss and in fact, in most implementations it takes even longer because of the coarse grain timeout. Also since it doesn't send immediate ACK's, it sends cumulative acknowledgements, therefore it follows a 'go back n' approach. Thus every time a packet is lost it waits for a timeout and the pipeline is emptied. This offers a major cost in high band-width delay product links.

### B. *TCP Reno*

Along with the implementation of the basic principles of Tahoe, the TCP Reno version adds more mechanisms so as to detect the lost packets in shorter time and also prevent the pipeline from being empty every time a packet is lost. The packet segment is assumed to be lost as soon as the duplicate acknowledgements are reached to its threshold level. Then the TCP enters the Fast

Re-transmit phase through which the lost segment is retransmitted. When the Fast Retransmit phase is completed,

TCP Reno employs the Fast Recovery algorithm which does not let the pipeline to be empty and also provides extra incoming duplicate ACKs to clock subsequent outgoing packets.

Moreover, Fast Recovery assumes whenever a duplicate ACK is attained, each time there is a single packet left in the pipe. As a result, the TCP Reno sender is capable of making sharp estimation over the amount of outstanding data in the network. Meanwhile, after entering the Fast Recovery phase, the TCP sender waits until half a window of dup ACKs are achieved, and then transmits a new data packet for each additional dup ACK. Finally, the sender leaves the Fast Recovery phase when it receives a new ACK for the new data. The variant TCP Reno can smoothly detect the single packet drop; however this version experiences difficulty in case of multiple packets dropped from the window and the performance becomes almost as like as Tahoe version. When multiple packets are dropped, the loss information of the initial packet is arrived after the reception of the duplicate ACK. On the other hand, the information about the second packet is obtained after the acknowledgement of the retransmitted initial packet is reached to the sender. Furthermore, this ACK of the retransmitted initial packet is arrived after one RTT and hence it takes longer time to process the second packet loss.

Problems:

Reno perform very well over TCP when the packet losses are small. But when we have multiple packet losses in one window then RENO doesn't perform too well and its performance is almost the same as Tahoe under conditions of high packet loss. The reason is that it can only detect a single packet losses. If there is multiple packet drop then the first info about the packet loss comes when we receive the duplicate ACK's. But the information about the second packet which was lost will come only after the ACK for the retransmitted first segment reaches the sender after one RTT. Also it is possible that the CWD is reduced twice for packet losses which occurred in one window. Suppose we send packets 1,2,3,4,5,6,7,8,9 in that order. Suppose packets 1, and 2 are lost. The ACK's generated by 2,4,5 will cause the re-transmission of 1 and the CWD is reduced to 7. Then when we receive ACK for 6,7,8,9 our CWD is sufficiently large to allow to us to send 10,11. When the re-transmitted segment 1 reaches the receiver we get a fresh ACK and we exit fast-recovery and set CWD to 4. Then we get two more ACK's for 2(due to 10,11) so once again we enter fast-retransmit and re-transmit 2 and then enter fast recovery. Thus when we exit fast recovery for the second time our window size is set to 2. Thus we reduced our window size twice for packets lost in one window. Another problem is that if the widow 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.

*C. TCP New Reno*

In case of multiple packet loss, the TCP New-Reno does not wait for the retransmission timer to be expired and hence this variant provides a dominating performance over the Reno version. In New Reno, the performance concerns about the behavior of the partial ACKs, which do not take TCP out of Fast Recovery phase while it takes TCP out from the Fast Recovery phase in Reno version. Moreover, in New-Reno, receiving partial ACKs often indicates the loss of the packets which instantly follows the acknowledged packet in the sequence space. Thus for the multiple packet losses, the New-Reno becomes able to retransmit all the packets lost from a particular window and therefore the New-Reno does not leave the Fast Recovery phase unless the acknowledgement for all outstanding data in the network is completed. However, New-Reno may experience poor performance as it takes one RTT for identifying the packet loss and therefore it is possible to infer about the information of other lost packet only when the ACK for the first retransmitted segment is received.

Problems:

New-Reno suffers from the fact that it's take 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.

*D. TCP Sack*

TCP uses a cumulative acknowledgment scheme through which only a single lost segment can be detected per round trip time. Moreover, this scheme does not allow the received packets that are not at the left edge of the receiver window to be acknowledged. Hence in order to discover the packet, the sender has to either wait for a roundtrip time or retransmit the received packet unnecessarily. Consequently, TCP loses its ACK-based clock and thus decreases the overall throughput.
In order to overcome these limitations, A SACK mechanism, combined with a selective repeat retransmission policy is developed. TCP SACK is basically an upgraded version of TCP New Reno which takes steps to solve the major problems experienced by the New Reno version. Such problems include the detection of multiple lost packets and re-transmission of more than one lost packet per RTT. With selective acknowledgments, the information about the arrived data segments can be reached successfully to the sender. As a result the sender only needs to retransmit the actual lost packet. The TCP SACK offers a significant feature so that the segments are acknowledged selectively instead of being acknowledged cumulatively. In addition, there is a block present in each ACK which monitors the acknowledgments and reports the sender of which segments have been acknowledged. For increasing and decreasing the congestion window size, the congestion control algorithms of SACK version are found almost same as Reno. The TCP SACK retains the basic properties and services of Tahoe and Reno, for instance, ensures high robustness even in the existence of the out-of-

order packets. However, when multiple packets are lost from the data window, the properties between SACK and other variants can be differentiated.

In the Fast Recovery stage of SACK version, a variable is maintained by the sender in order to measure the number of outstanding data in the network. This variable is called a pipe and it is not maintained in any of the earlier TCP versions. As long as the estimated number of outstanding packets is found below than the congestion window value, a data is transmitted or retransmitted by the sender [21]. Moreover, when the sender sends a new data or retransmits an old packet, the variable pipe is incremented by one while it is decremented by the same value upon receiving a duplicate ACK with a selective acknowledgment option. Though TCP SACK provides many advantages, it is not an easy task to implement selective acknowledgment options in TCP SACK version. Hence, currently the TCP receivers are found to be reluctant for providing the selective acknowledgment option.

Problems:

The biggest problem with SACK is that currently selective acknowledgements are not provided by the receiver To implement SACK we'll need to implement selective acknowledgment which is not a very easy task.

### *E. TCP Vegas*

Vegas is a TCP implementation which is a modification of Reno. It builds on the fact that proactive measure to encounter congestion are 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 suggest a modified slow start algorithm which prevent it from congesting the network. It does not depend solely on packet loss as a sign of congestion. It detects congestion before the packet losses occur. However it still retains the other mechanism of Reno and Tahoe, and a packet loss can still be detected by the coarse grain timeout of the other mechanisms fail.

The three major changes induced by Vegas are:

New Re-Transmission Mechanism:

Vegas extends on the re-transmission mechanism of Reno. It keeps track of when each segment was sent and it also calculates an estimate of the RTT by keeping track of how long it takes for the acknowledgment to get back. Whenever a duplicate acknowledgement is received it checks to see if the (current time segment transmission time)> RTT estimate; if it is then it immediately retransmits the segment without waiting for 3 duplicate acknowledgements or a coarse timeout. Thus it gets around the problem faced by Reno of not being able to detect lost packets when it had a small window and it didn't receive enough duplicate Acknowledgements. To catch any other segments that may have been lost prior to the retransmission, when a non-duplicate acknowledgment is received, if it is the first or second one after a fresh acknowledgement then it again checks the timeout values and if the segment time since it was sent exceeds the timeout value then it re-transmits the segment without waiting for a duplicate acknowledgment. Thus in this way Vegas can detect multiple packet losses. Also it only reduces its window if the re-transmitted segment was sent after the last decrease. Thus it also overcome Reno's shortcoming of reducing the congestion window multiple time when multiple packets are lost.

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. The details can found in. Thus whenever the calculated rate is too far away from the expected rate it increases transmissions to make use of the available bandwidth, whenever the calculated rate comes too close to the expected value it decreases its transmission to prevent over saturating the bandwidth. Thus Vegas combats congestion quite effectively and doesn't waste bandwidth by transmitting at too high a data rate and creating congestion and then cutting back, which the other algorithms do.

Modified Slow-start:

TCP Vegas differs from the other algorithms during it's 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.

## III. METHODOLOGY

In this paper, in order to perform simulation, I used Network Simulator-2 tool and I used a simple dumbbell topology shown in the Fig 3.1 below to compare the performance of TCP variants. This dumbbell topology consists of TCP senders, TCP receivers and a pair of routers. The link between the TCP senders and router 1 is called as the sender link while the link between the TCP receivers and the router 2 is called the receiver link. The sender and receiver links represent a local area network (LAN). The link between router 1 and

router 2 is called the bottleneck link. The bottleneck link represents a wide area network (WAN).

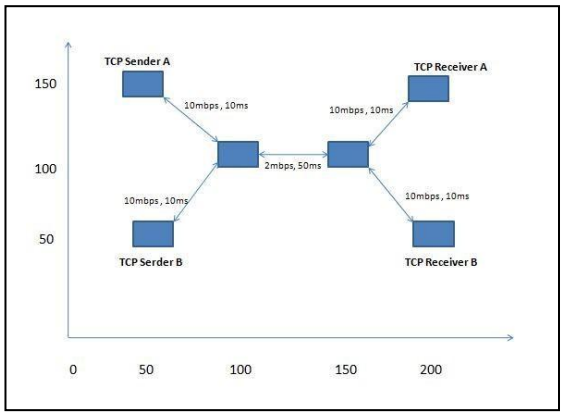


Fig 3.1 Dumbbell Topology

I used four simulation environments for the sender and receiver links as shown in Table 3.1.

Table 3.1 Simulation Environment

| Simulation Environment | TCP sender(A&B) | TCP receiver(A&B) |
|---|---|---|
| 1 | Ethernet | Ethernet |
| 2 | Ethernet | Fast Ethernet |
| 3 | Fast Ethernet | Fast Ethernet |
| 4 | Fast Ethernet | Ethernet |

The sender links and the receiver links are full wired duplex links and the bandwidth of the sender links that implement Ethernet LAN is 10Mbps and the link delay in the full duplex Ethernet LAN is set to 10ms whereas for Fast Ethernet LAN it is 100Mbps with a link delay of in 1ms. The bottleneck link is a full wired duplex link with the capacity of 2Mbps with a link delay of 50ms. The link delay of the senders, receivers and bottleneck link is set to respective value so that the resulted bandwidth delay product is the same. The reason is because we want to customize the bandwidth delay product of the bottleneck link equal to the bandwidth delay product of the sender and receiver links. The bandwidth delay product is customized to be equal to minimize the unfairness sharing of available bottleneck capacity.
The simulation parameters of the network topology are showed in Table 3.2.

Table 3.2. Simulation Parameters

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

In homogeneous network, both TCP senders uses the same TCP source varied from Tahoe, Reno, New Reno, Vegas and SACK. For the receiver side, we set TCP Sink as the TCP source. There are five cases considered in simulation experiment of homogeneous wired network as shown in Table 3.3. The performance of TCP variants in homogeneous network is evaluated in order to analyze the average throughput and average delay.

Table 3.3 Homogenous Wired Network

| Case | TCP Sender A | TCP Sender B |
|---|---|---|
| 1 | Vegas | Vegas |
| 2 | Tahoe | Tahoe |
| 3 | Reno | Reno |
| 4 | New Reno | New Reno |
| 5 | Sack1 | Sack1 |

## IV. METRICS

The metrics that is used to analyze the performance of TCP variants is Throughput, Packet Delivery Ratio and End-to-End Delay

### *A. Throughput:*

The amount of data in (bits or Bytes) successfully transferred from source to destination (or it can be measured on hop-to-hop network points), is called throughput.

Throughput = Sum of Received Packets / sum of time-out

### *B. End-to-End Delay:*

End-to-End Delay is the time that a packet spends from source to destination.

End-to-End Delay = Arrival time – Departure time(s)

### *C. Packet Delivery Ratio:*

This is the ratio of total no of packets successfully received by the destination nodes to number of packets sent by the source nodes throughout the simulation.

Packet delivery ratio(pdr) = Number of packets received / Number of packets sent

## V. SIMULATION RESULTS

To compare all the TCP variants, I have taken the dumbbell topology shown in figure and compared in both homogeneous and heterogeneous networks using different metrics.

### *A. Homogeneous Networks*

Table 5.1: Packet delivery ratio, Average end to end delay and Throughput in Homogeneous Network with TCP senders as Reno

| S.No | TCP Senders (A&B)Reno | TCP Receivers | Packet delivery ratio | Average end to end delay (ms) | Average throughput |
|---|---|---|---|---|---|
| 1. | Ethernet | Ethernet | 0.997667 | 122.144 | 1964.13 |
| 2. | Ethernet | Fast Ethernet | 0.995516 | 125.608 | 1968.35 |
| 3. | Fast Ethernet | Fast Ethernet | 0.997667 | 122.144 | 1972.13 |
| 4. | Fast Ethernet | Ethernet | 0.995516 | 125.608 | 1968.35 |

Table 5.2: Packet delivery ratio, Average end to end delay and Throughput in Homogeneous Network with TCP senders as Vegas

| S.No | TCP Senders (A&B) Vegas | TCP Receivers | Packet delivery ratio | Average end to end delay | Average throughput |
|---|---|---|---|---|---|
| 1. | Ethernet | Ethernet | 0.998816 | 21.4109 | 1984.12 |
| . | Ethernet | Fast Ethernet | 0.998816 | 22.4109 | 1991.90 |
| 3. | Fast Ethernet | Fast Ethernet | 0.998816 | 20.4209 | 1992.90 |
| 4. | Fast Ethernet | Ethernet | 0.998816 | 21.4108 | 1986.69 |

Table 5.3: Packet delivery ratio, Average end to end delay and Throughput in Homogeneous Network with TCP senders as Sack1

| S.No | TCP Senders (A&B) Sack1 | TCP Receivers | Packet delivery ratio | Average end to end delay | Average throughput |
|---|---|---|---|---|---|
| 1. | Ethernet | Ethernet | 0.997667 | 123.142 | 1802.13 |
| 2. | Ethernet | Fast Ethernet | 0.995516 | 126.923 | 1870.35 |
| 3. | Fast Ethernet | Fast Ethernet | 0.997667 | 122.614 | 1802.13 |
| 4. | Fast Ethernet | Ethernet | 0.995516 | 124.108 | 1870.35 |

Table 5.4: Packet delivery ratio, Average end to end delay and Throughput in Homogeneous Network with TCP senders as Tahoe

| S.No | TCP Senders (A&B) Tahoe | TCP Receivers | Packet delivery ratio | Average end to end delay | Average throughput |
|---|---|---|---|---|---|
| 1. | Ethernet | Ethernet | 0.997667 | 125.144 | 1902.13 |
| 2. | Ethernet | Fast Ethernet | 0.995516 | 121.608 | 1970.35 |
| 3. | Fast Ethernet | Fast Ethernet | 0.997667 | 128.144 | 1992.13 |
| 4. | Fast Ethernet | Ethernet | 0.995516 | 127.608 | 1902.35 |

Table 5.5: Packet delivery ratio, Average end to end delay and Throughput in Homogeneous Network with TCP senders as New Reno

| S.No | TCP Senders (A&B) New Reno | TCP Receivers | Packet delivery ratio | Average end to end delay | Average throughput |
|---|---|---|---|---|---|
| 1. | Ethernet | Ethernet | 0.997667 | 121.241 | 1972.13 |
| 2. | Ethernet | Fast Ethernet | 0.995516 | 125.343 | 1975.35 |
| 3. | Fast Ethernet | Fast Ethernet | 0.997667 | 126.544 | 1980.13 |
| 4. | Fast Ethernet | Ethernet | 0.995516 | 122.546 | 1975.35 |

## VI. CONCLUSION

In this paper, I have compared various TCP variants by considering the homogeneous networks by using various metrics such as packet delivery ratio, Average end to end delay and Average throughput. The results show that TCP Vegas has maximum throughput, minimum delay and maximum packet delivery ratio when compared to other TCP variants.

## *References*


[1] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno, and SACK TCP," *Computer Communication Review*, USA, vol. 26, no. 3, pp. 5-21, July 1996.

[2] L. S. Brakmo, S. W. O'Malley, L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," In the Proceedings of the ACM SIGCOMM, vol. 24,pp. 24-35, Oct 1994.

[3] V. Jacobson, "Congestion Avoidance and Control", In the Proceddings of the SIGCOMM'88 Symposium, pp. 314-329, Aug. 1988.

[4] M.Allman, V.Paxson and W.Stevens, "TCP Congestion Control", Request for Comment 2581, Internet Engineering Task Force, April 1999.

[5] L. S. Brakmo, L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet", In the Proceedings of IEEE Journal On Selected Areas In Communications, vol.13, no. 8, pp. 1465- 1480, Oct. 1999.