QoS Evaluation of Different TCPs Congestion Control Algorithm Using NS2

Md Tarmizi[1], A. Albagul[2], Othman. O. Khalifa[2], Wahyudi[2]
[1] *Universiti Kuala Lumpur British Malaysian Institute, Gombak, 53100 Malaysia*
[2] *Faculty of Engineering, IIUM,Gombak, 53100 Malaysia*
Email (mtarmizi@bmi.edu.my albagul@iiu.edu.my)

Abstract

The success of the current Internet relies to a large extent on cooperation between the users and network. The network signals its current state to the users by marking or dropping packets. The user then strives to maximize the sending rate without causing network congestion. To achieve this, the users implement a flow control algorithm that controls the rate at which data packets are sent into the Internet. More specifically, the Transmission Control Protocol (TCP) is used by the users to adjust the sending rate in response to changing network conditions. In this paper, we focus on the degree of fairness provided to TCP connections by comparing two packet-scheduling algorithms at the router. The first one is FIFO (First In First Out, or Drop–Tail), which is widely used in the current Internet routers because of its simplicity. The second is RED (Random Early Detection), which drops incoming packets at a certain probability.

1. Introduction

Much of today's Internet traffic is carried using the Transmission Control Protocol (TCP), and consequently there has been a significant amount of research toward modeling and understanding the impact that this protocol has on file transmission times and network utilization. TCP has been and will continue to be an evolving protocol, and as such, one important task of these models is to facilitate comparisons between the different flavors of TCP. In this paper for TCP, we consider the Tahoe and Reno version, which has widely been used in the current Internet. The Vegas version, adopting a different congestion control mechanism from TCP Reno and TCP Tahoe for larger performance gain, is also considered. In this paper, for reference purposes, we will first show simulation results that FIFO cannot provide fairness among connections at all because of a bursty nature of packet losses. While the original idea of the RED algorithm is to avoid consecutive dropping of packets belonging to the same connection, it also has a capability of achieving a fair service among connections by spreading packet losses. It is next shown that RED offers better fairness than FIFO to TCP Reno connections, but it cannot keep a good fairness when the capacity of shared link becomes small compared with the total input link capacity. In TCP Vegas, on the other hand, RED offers less fairness than FIFO because of the essential incompatibility of TCP Vegas to the RED algorithm. The packet scheduling algorithms and TCP versions that we will use in this paper are not new. Our main contributions in the current paper are that the properties mentioned above are also shown through analytical results. While the model used in the analysis is very simple, the basic features of the above scheduling algorithms can be well explained. Because TCP has gone through incremental refinements, the protocol itself has grown increasingly complex, which makes analytical modeling quite difficult. In spite of a rapid growth of the Internet population and an explosive increase of the traffic demand, the Internet is still working without collapse. Of course, continuous efforts to increase the link bandwidth and router processing capacity are supporting the Internet growth behind the scenes. However, a most essential device for achieving such a success is a congestion control mechanism provided by a transport– layer protocol, i.e., TCP (Transmission Control Protocol). In TCP, each end host controls its packet transmission rate by changing the window size in response to network congestion. A key is that the TCP congestion control is performed in a distributed fashion; each end host determines its window size by itself according to the information obtained from the network. In general, there are two major objectives in the congestion control mechanism. The one is to avoid an occurrence of the network congestion, and to dissolve the congestion if the congestion occurrence cannot be avoided. The other is to provide *fair service* to connections. Keeping the fairness among multiple homogeneous/heterogeneous connections in the network is an essential feature for the network to be widely accepted. The fair service also involves detecting mis–behaving flows, which do not properly react against the network congestion and unfairly occupy the network resources (such as router buffer and link bandwidth) [1]. Current usage of the Internet is dominated by transmission control protocol (TCP) traffic such as remote terminal (e.g., Telnet), FTP, Web traffic, and electronic mail (e.g., SMTP). These TCP sources constitute 90 percent of all traffic with 50–70 percent of this TCP traffic being short-lived connections in size and lifetime (so called *mice*). Although these applications are rather *elastic* in nature, in that they can tolerate either packet delay or packet losses rather gracefully, congestion remains a major problem that leads to poor performance. If the Internet is to evolve to a high-

0-7803-9521-2/06/$20.00 ©2006 IEEE.

performance network providing ubiquitous services, including real time voice/video, it must understand how congestion arises and find more efficient ways to keep the network operating within its capacity. In current TCP/IP networks, TCP packet (or segment) loss, indicated by a timeout or a triple duplicated acknowledgment, is used as an indication of network congestion. Once congestion occurs, TCP controls its sending rate by limiting its congestion window size (cwnd). The data-sending rate of TCP (or the window size) is determined by the rate of incoming Acknowledgments (ACKs) to previous packets. The rate of ACK arrival is in turn determined by the presence or absence of congested link(s) along the path between a source and its destination. In steady state, the source's sending rate will match the arrival rate of the ACKs. Accordingly, TCP automatically detects congestion and regulates its sending rate. This has been referred to as TCP's *self-clocking* behavior. The three major TCP implementations are:

- Slow start and congestion avoidance: *TCP Tahoe*.
- Fast Retransmit and Fast Recovery: *TCP Reno*.
- *TCP Vegas*.

2. TCP Congestion Control Algorithms

2.1 TCP Tahoe version

In TCP Tahoe, the window size cwnd is cyclically changed as indicated in Figure 7 cwnd continues increasing until segment loss occurs. When it does occur, TCP determines that the network is congested, and throttles cwnd down to the size of one segment. TCP Tahoe has two phases in increasing cwnd; Slow Start Phase and Congestion Avoidance Phase. When an ACK segment is received by TCP at the server side at *time* $t + t_A$ is updated from cwnd (t) as follows (see, e.g., [1]); for the slow Start Phase *if* $cwnd\ (t) < ssth$;

$$Cwnd\ (t + t_A) = cwnd(t) + m$$

For Congestion Avoidance Phase ***if*** $cwnd(t) > ssth$;

$$Cwnd\ (t + t_A) = cwnd(t) + m^2/cwnd(t) \qquad (1)$$

where ssth is the threshold value at which TCP changes its phase from Slow Start Phase to Congestion Avoidance Phase. When segment loss is detected by timeout or fast retransmission algorithm, Eq.(1), cwnd(t) and ssth are updated as follows;

$$ssth = cwnd(t) / 2$$
$$cwnd(t) = m \qquad (2)$$

That is, TCP Tahoe again enters Slow Start Phase when segment loss occurs. Therefore, the dynamics of TCP Tahoe in a simplest case is; Slow Start Phase to Congestion Avoidance Phase to segment loss to Slow Start Phase [2]

2.2 TCP Reno version

TCP Reno is similar to TCP Tahoe, but uses another algorithm when segment loss occurs. In Slow Start Phase and Congestion Avoidance Phase, TCP Reno also uses Eq. (1) to update the window size, but when segment loss is detected by fast retransmission algorithm, the window size cwnd (t) is halved. That is,

$$ssth = cwnd(t) / 2$$
$$cwnd(t) = ssth$$

TCP Reno then enters Fast Recovery Phase. In this phase, the window size is increased by one segment when a duplicate ACK segment is received, and cwnd (t) is restored to ssth when the non–duplicate ACK segment corresponding to the retransmitted segment is received. Figure 7 is a typical example of the behavior of cwnd (t) [3].

2.3 TCP Vegas version

In TCP Tahoe and Reno, the window size, *cwnd*, is increased until segment loss occurs due to congestion. Then, the window size is throttled, which leads to the throughput degradation of the connection. However, it cannot be avoided because of an essential nature of the congestion control mechanism adopted in TCP Tahoe and Reno. It can detect network congestion information only by segment loss. However, it becomes a problem since the segment may be lost when the TCP connection itself causes the congestion because of its too large window size. If *cwnd* is appropriately controlled such that the segment loss does not occur in the network, the throughput degradation due to throttled window can be avoided. This is the reason that TCP Vegas was introduced. TCP Vegas employs another mechanism for detecting the network congestion. It controls *cwnd* by observing changes of RTTs (Round Trip Time) of segments that the connection has sent before. If observed RTTs become large, TCP Vegas recognizes that the network begins to be congested, and throttles *cwnd* down. If RTTs become small, on the other hand, TCP Vegas determines that the network is relieved from the congestion, and increases *cwnd*. Then, *cwnd* in an ideal situation becomes converged to the appropriate value as shown in Figure 7, and the throughput is not degraded. In Congestion Avoidance Phase, the window size is updated as [4];

$$Cwnd(t + t_A) = \begin{cases} cwnd(t) + 1, & \text{if } diff < \dfrac{\alpha}{base\_rrt} \\ cwnd(t), & \text{if } \dfrac{\alpha}{base\_rrt} \le diff \le \dfrac{\beta}{base\_rtt} \\ cwnd(t) - 1, & \text{if } \dfrac{\beta}{base\_rtt} < diff, \end{cases}$$

$$diff = \frac{cwnd\ (t)}{base\_rtt} - \frac{cwnd\ (t)}{rtt} \qquad (3)$$

where rtt is a observed round trip time, base rtt is the smallest value of observed RTTs, and α and β are some constant values. TCP Vegas has another feature in its congestion control algorithm. That is *slow* Slow Start. The rate of increasing *cwnd* in slow start phase is a half of that in TCP Tahoe and TCP Reno;

$$Cwnd(t + t_A) = cwnd(t) + m/2 \qquad (4)$$

Note that Eq. (3) used in TCP Vegas indicates that if RTTs of the segments are stable, the window size remains unchanged [5]. That can be seen by Figure 3, where the window size is converged to a fixed value in steady state.

3. TCP Tahoe with Different System Loads

3.1 Simulation Design and results

Topology 1 shows the topology of network of TCP Tahoe. From the Tcl script it run the programme that generates three graphs showing the behaviour of a TCP flow under different background traffic. TCP flow from *node0* to *node3* and sends data at max 4Mbps. The measurements are made at *node0* and at *node2* of the simulated network (see Figure 1). Throughput and packet loss are shown as a function of time. The propagation delay of each link of the simulated network is 10ms. All links have a capacity of 10Mbps. From Figure 2 it is well known that when TCP and UDP connections share the link, UDP connections tend to occupy the link. It is because the UDP connection does not adequately react against the congestion while the TCP connection does. Example behaviors of TCP and UDP connections are shown in Figure 2 where it use the model shown in Network Topology 1. From Figure 2 the UDP connection starts at 1 and continues at rate up to 0.2Mbps.

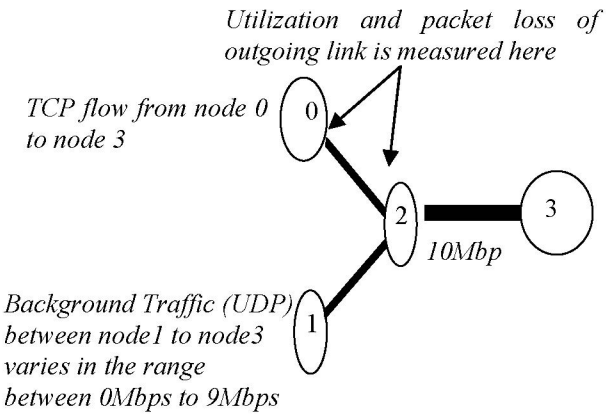

Figure 1: Network Topology 1

The bandwidth of the share link is 10Mbps. From t= 10 seconds to t = 12 seconds the UDP utilization is almost 80 % where the TCP is 20 %. The TCP flows reduce their sending rates in response to congestion, leaving the uncooperative UDP flows to use the available bandwidth. In the above result it can consider that the UDP connection keeps transmitting the packet at constant rate. From the simulation result as shown in Figure 4, at 4 second the TCP has to drop the packet (2000 Packet) because of the flow rate (8Mbps from UDP, 4 Mbps TCP) exceeds the link bandwidth. (10Mbps) After 4 seconds the TCP remain for 4Mbps until 9seconds. The packet drops only happen for UDP. After 9 seconds the graphs shows the packet for both TCP and UDP drops.

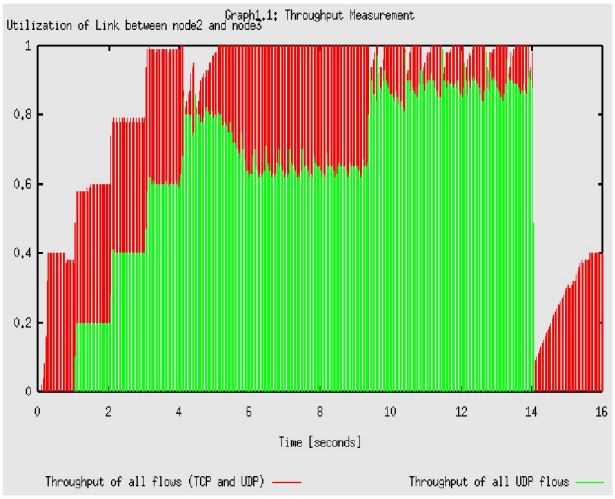

Figure 2: Throughput Measurement Utilization of Link between *node2* and *node3*

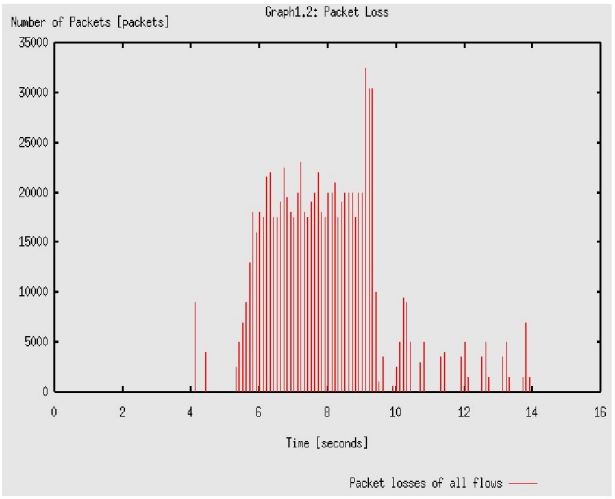

Figure 3: Packet losses of all flows

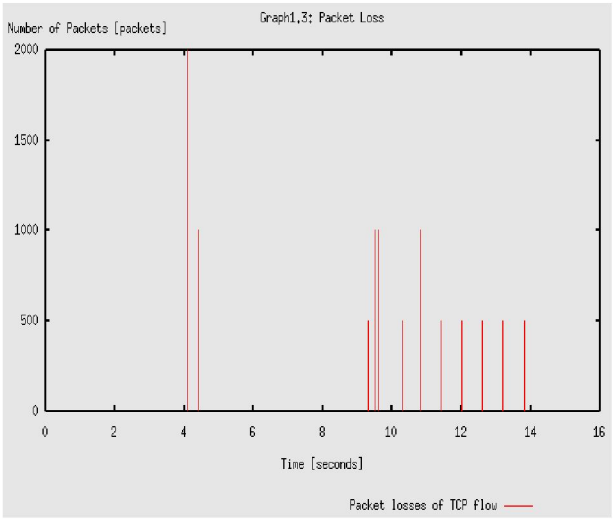

Figure 4: Packet losses of TCP flows

4. Analyze of Different TCP

4.1Simulation Design and Results

Topology 2 shows the topology of network of TCP Tahoe, TCP Reno and TCP Vegas. In this simulation it needs to analyze flows of different flavours of TCP (Tahoe, Reno, Vegas) and explain how the behaviour differs. TCP Tahoe sends data at max 4Mbps between *node0* and *node3*; TCP Reno and TCP Vegas flow sends data at max 4Mbps between *node1*, *node2* and *node3*. From the programme, the simulation graph will showing the throughput, the window size, and the queue length of TCP Tahoe, TCP Reno and TCP Vegas. The propagation delay each link of the simulated network is 10ms.

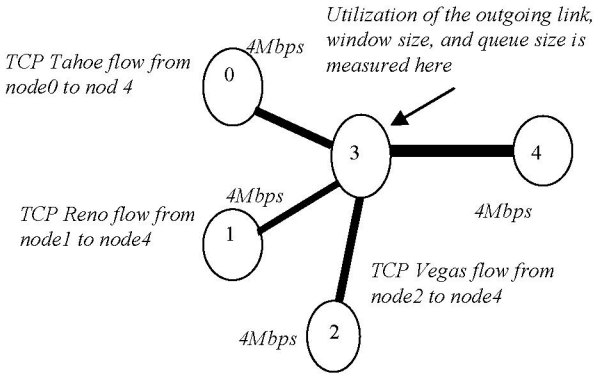

Figure 5: Network Topology 2

From Figure 6 shows the throughput measurement of Three TCP Version (Tahoe (red color), Reno (green color) and Vegas (blue color). All of the TCP (Tahoe, Reno and Vegas) have about the same throughput during the whole simulation.

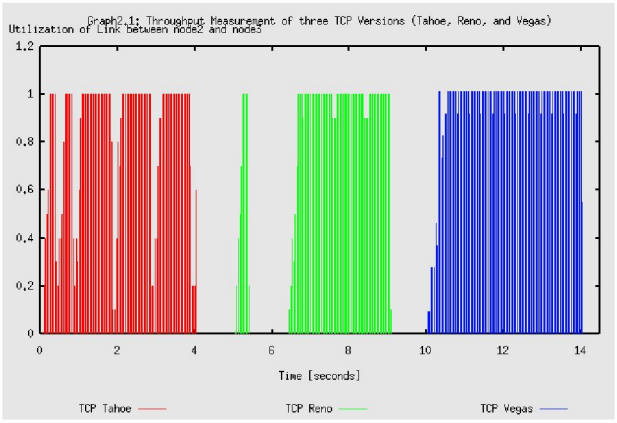

Figure 6: Throughput Measurement of TCPs

From Figure 7, it see that congestion window of TCP Tahoe drops to 0 and slow start when more than one packet are dropped in one window. TCP Tahoe is repeating to drops to 0 more compared to TCP Reno and Vegas. For TCP Reno look better than TCP Tahoe when it's only drop once to 0. A congestion window of TCP Vegas oscillates when more packets are dropped, but never goes back from slow start. TCP Vegas maintains the same window size as the value after the first packet drop and returns to higher windows size than both Tahoe and Reno. We can say that TCP Vegas performs better in the case of more than one packet is dropped in one window.

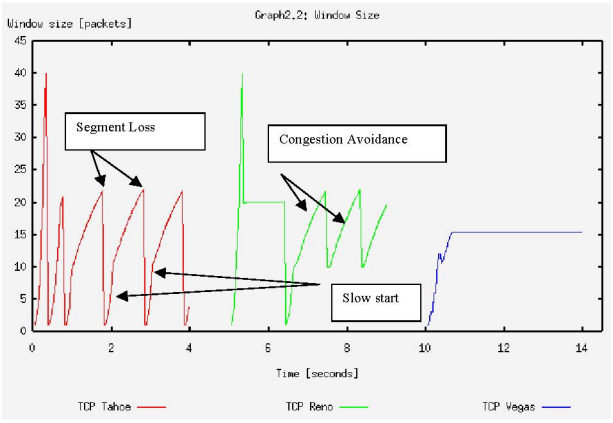

Figure 7: Window Size of TCPs

From Figure 8 it can see large oscillation number of packet in queue in TCP Tahoe and TCP Reno. From the graph it can see increasing the number of queue, both Tahoe and Reno drop down close to 0. Increasing the buffer size in the router can make the two connections share the link more fairly. The reason is that, TCP Tahoe, TCP Reno and TCP Vegas need buffer space to increase window.

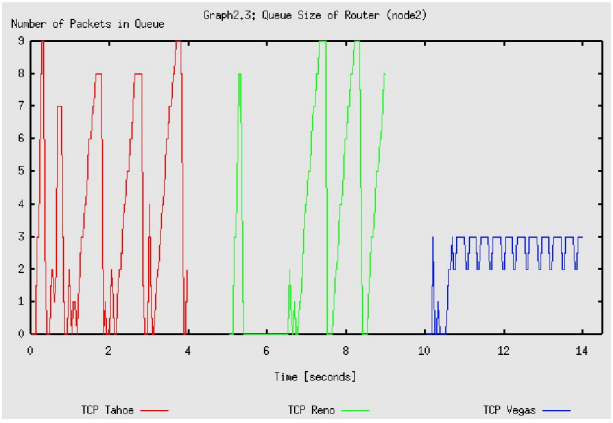

Figure 8: Queue Size of Router (*node2*) of TCPs

5. Congestion control mechanism (RED)

5.1 Simulation Design and Results

The service quality of the network can be improved if routers have congestion control mechanisms such as RED, CBQ, FQ, etc. implemented. In this simulation we need to analyze the behaviour of ECN enabled to support TCP flows if a router supports RED. From the graph shows the throughput, window size and queue size of three TCP flows (Tahoe, Reno and Vegas) changes.

We need to compare the graphs obtained with the results. The propagation delay of all links is 10ms. The results of simulations by using RED can be obtained in Figure 9, Figure 10 and Figure 11. Those figures show that RED can improve the total throughput**.** With the RED mechanism, on the other hand, the synchronization of throughput degradation can be avoided by its probabilistic packet discarding. However, the main purpose of the RED router is not to improve the fairness among connections. Even if the access link capacity is same among all connections, throughput values of those connections are much affected by the propagation delays. Compare Figure 7 and Figure 10 note again that the RED mechanism itself is not intended to establish the fairness among connections in the heterogeneous case, but it is often used as a basis of developing the new packet processing methods at the router.

6. Analysis

In this paper the connection of the network is homogeneous as shown in Network Topology. In the case of how a router with congestion control mechanism (RED) can improve service quality is shown in Network Topology. The results of simulations by using RED can be obtained in Figure 9, Figure 10 and Figure 11. Those figures show that RED can improve the total throughput. In Network Topology 2 show how the drop-tail router is used. The simulation results of the drop-tail router are shown in Figure 6, Figure 7 and Figure 8.

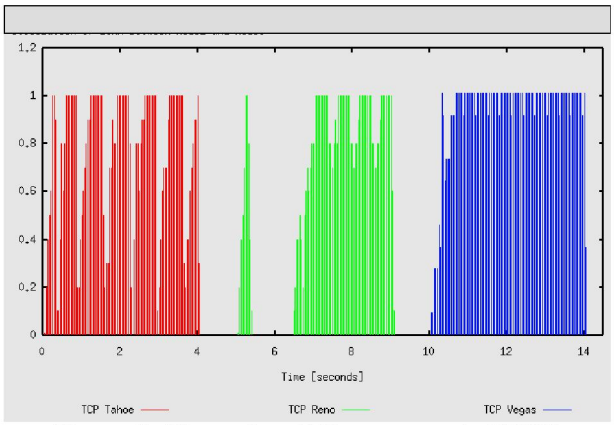

Figure 9: Throughput Measurement of TCPs

When the drop–tail discipline is used, packet losses take place continuously, and multiple TCP connections simultaneously suffer from the performance degradation by packet losses (this is shown in Figure 6, Figure 7 and Figure 8). With the RED mechanism, on the other hand, the synchronization of throughput degradation can be avoided by its probabilistic packet discarding (this is shown in Figure 9, Figure 10 and Figure 11). However, the main purpose of the RED router is not to improve the fairness among connections. Even if the access link capacity is same among all connections, throughput values of those connections are much affected by the propagation delays.

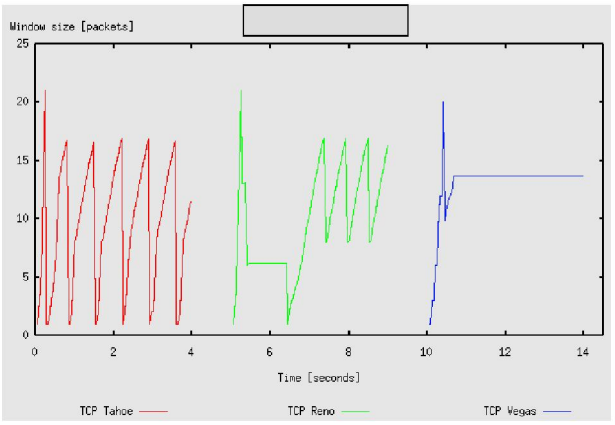

Figure 10: Window Size Measurement of TCPs

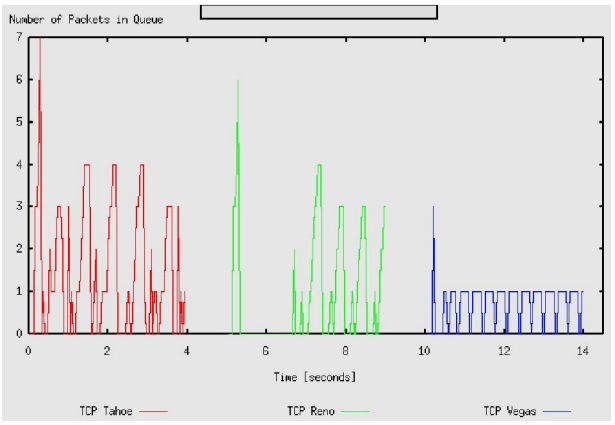

Figure 11: Queue Size Measurement of TCPs

7. Conclusions

This paper focused the issues of stability and fairness issues in congestion control mechanism of three versions of TCP, TCP Tahoe, TCP Reno TCP Vegas and UDP. The investigations are done by connections of the TCP and UDP share the bottleneck link. We have derived the following results through the mathematical analysis and the simulation experiments. From the experiment results shows that TCP Tahoe and TCP Reno can provide the fairness between connections at the expense of stability and throughput. TCP Vegas can achieve higher throughput than TCP Tahoe and Reno, but lacks in fair share of the link. TCP Vegas suffers from serious performance degradation with drop-tail routers, because of the difference of buffer occupancy at the router. RED routers can improve the fairness to some degree, but there exists an inevitable trade-off between fairness and throughput. According to the past researches, a TCP Vegas version is able to achieve higher throughput than TCP Tahoe and Reno versions, which are widely used in the current Internet. Figure 10 shows the time window size measurement of three TCP Version (Tahoe, Reno and Vegas). In TCP Vegas, the window size is converged to the fixed value and no packet loss is necessary to invoke the congestion control. TCP Vegas is expected to achieve higher throughput

than TCP Reno. Some research shows that TCP Vegas achieves 40% - 70% higher throughput, with one fifth to one – half the losses as compared to TCP Reno through simulation and implementation experiments. From the analysis and the simulation results, we find that the performance of TCP Vegas is much smaller than that of TCP Reno and Tahoe as opposed to an expectation on TCP Vegas. The RED algorithm improves the fairness to some degree, but there still be an inevitable trade-off between fairness and throughput.

8. References

[1] Go Hasegawa and Masayuki Murata, "Survey on fairness issues in TCP Congestion control mechanisms," to appear in IEICE Transactions on Communications, January 2001

[2] Go Hasegawa, Masayuki Murata and Hideo Miyahara, "Fairness and stability of congestion control mechanisms of TCP," Telecommunication Systems Journal, pp. 167-184, November 2000

[3] J. C. Hoe, "Improving the start-up behavior of a congestion control scheme of TCP," ACM SIGCOMM Computer Communication Review, vol. 26, pp. 270–280, October 1996.

[4] L. S. Brakmo, S.W.O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," Proceedings of ACM SIGCOMM'94, pp. 24–35, October 1994

[5] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global internet," IEEE Journal on Selected Areas in Communications, vol. 13, pp. 1465–1480, October 1995.