A Comparison of Queueing Algorithms Over TCP Protocol

Mahmud Mansour *mm.mansour@gmail.com*
Faculty of Information Technology
Department of Network Engineering
Tripoli University
Tripoli, Libya

Ahmed Hmeed *A.Hmeed@hotmail.com*
Faculty of Information Technology
Department of Network Engineering
Tripoli University
Tripoli, Libya

Abstract

Network congestion control is one of the most key problems in network study. With the expansion of network size, the continuous increase of network bandwidth and increasing diversification of networking forms, congestion control has encountered some new problems that require a solution.

If a packet number that reaches the network is greater than the processing capacity of network, network performance would drop dramatically, resulting in an inevitable congestion. In order to avoid congestion, people use congestion control algorithm in the network.

This paper studies different versions of TCP source algorithms, such as Reno and Vegas, and investigate the impact of various Queuing management algorithms on the self-similarity of network traffic. We compare the performance of Reno and Vegas using various queue management algorithms, namely Droptail, Fair Queueing (FQ), Deficit Round Robin (DRR) and Random Early Detection (RED) using NS-2 network simulators. The characteristics of different algorithms are also discussed and compared based on the basis of packet loss, fairness and throughput metric.

Keywords- Congestion Control, TCP Reno, TCP New Reno, TCP Vegas, Transmission Control Protocol /Internet Protocol (TCP/IP).

1. INTRODUCTION

Internet has experienced exploding growth since its emergence. Internet traffic keeps growing at an exponential rate of almost doubling itself each year; and this trend is expected to continue [1]. Various and vast amount of Internet-based applications and services emerge with this growth and surveys reveal trend towards them. More and more people come to depend on them, and all kinds of business processes are built around them. And along with the emergence of Next Generation Network (NGN) services Internet is entering every home and business groups, and Internet-based applications and services are pervading everyday life.

In return, people and business groups unceasingly bring forth all kinds of new demands. They not only ask for diversified applications and services to satisfy their needs, but also demand for better quality of service (QoS). These different applications and services have varying requirements for goodput, packet loss ratio and end-to-end latency [2]. As surveys reveal, time-critical and mission-critical applications are rapidly growing to be a significant portion of Internet-based applications. In pursuing of greater profit, delay as one of the important performance indices of quality of service is becoming more and more recognized and emphasized by Internet service providers.

The transport layer providers duplex, end-to-end data transport services between applications. Data sent from the application layer ware divided into segment appropriate in size for the network technology. Transmission control protocol (TCP) and user datagram protocol (UDP) are the protocols used at this layer [1].

Transmission control protocol (TCP) provides reliable service by being connection-oriented and including error detection and correction. The connected nature of TCP used only for two end-points to communicate with each other. The connection must be established before a data transfer can occur, and transfers are acknowledged throughout the process. Acknowledgments assure that data received properly [6]. The acknowledgment process provides robustness in the face of network congestion or communication unreliability. It also determines when the transfer ends and closes the connection, thus freeing up resources on the systems. Checksums assure that the data not accidentally modified during transit [3].

Traffic management in TCP examines the reality of two autonomous methods: Delivery control regulated by the recipient using the window specification and Congestion control regulated by the sender for employed the congestion window and slow begin method. The first method oversees the recipient input buffer and the second method registers the channel congestion, hence it helps to decrease the level of traffic. The Congestion Window (CWND) and slow start method gives resolve the full loading of the virtual connection and decreasing the packet loss in case of overloading in the network [4].

2. TCP CONGESTION CONTROL ALGORITHM

The basis of TCP congestion control lies in Additive Increase Multiplicative Decrease (AIMD), halving the congestion window for every window containing a packet loss, and increasing the congestion window by roughly one segment per Round Trip Time (RTT) otherwise.

The second component of TCP congestion control is the Retransmit Timer, including the exponential bakeoffs of the retransmit timer when a retransmitted packet is itself dropped. The third fundamental component is the Slow-Start mechanism for the initial probing for available bandwidth. The fourth TCP congestion control mechanism is ACK-clocking, where the arrival of acknowledgements at the sender is used to clock out the transmission of new data.

There are two windows in TCP: receiver advertised window (rwnd) and congestion window (cwnd). The TCP receiver advertised window (rwnd) is added in each sent ACK packet, which sets the size for the sender's sliding window. The sender's transmission rate is then adjusted by the rwnd value, so that the maximum number of allowed outstanding packets is equal to the size of the receiver advertised window at any given time instant, i.e .(rate = rwnd/RTT). Congestion window, which is used by the sender to estimate the network capability, is the key window in the congestion control mechanism. At any given time, instant, the maximum amount of outstanding bytes is equal to MIN (cwnd, rwnd) [5].

Lost packets in the Internet are generally due to data collision and network congestion. Data collision occurs, when the shared media used such as switch and Hub, and this happen in LAN. Congestion refers to a state of the network, where one or more routers receive packets faster than they can forward them. After the queues of one of those routers fill up, it starts to drop packets.

TCP retransmits lost packets, which introduces an overhead in bandwidth utilization. The purpose of congestion control is to try to minimize congestion and, consequently, the need for retransmission by adjusting the transmission rate of TCP. The main concept of congestion control is the congestion window (cwnd), which controls, with the receiver advertised window (rwnd), the size of sender's sliding window and, thus, the transmission rate. At any given time, instant, the maximum amount of outstanding bytes is equal to min (cwnd, rwnd). Different flavors of TCP have different strategies to react to a loss event, i.e. they resize the cwnd differently.

2.1 Slow Start

Slow start and congestion avoidance are essentially different strategies to grow the cwnd. In the beginning of a connection, TCP sender is in slow start mode. The size of the cwnd is initialized to one Maximum segment size (MSS) and is increased by one each time a new ACK arrives.

The slow start mechanism was a direct attempt to avoid congestion collapse by increasing the packet rate of a connection in a controlled fashion – slowly at first, faster later on - until congestion is detected (i.e. packets are dropped), and ultimately arrive at a steady packet rate, or equilibrium. To achieve this goal, the designers of the slow start mechanism chose an exponential ramp-up function to successively increase the window size. Slow start introduces a new parameter called the congestion window or cwnd, which specifies the number of packets that can be sent without needing a response from the server. TCP starts off slowly (hence the name “slow start”) by sending just one packet, then waits for a response (an ACK) from the receiver. These ACKs confirm that it is safe to send more packets into the network (i.e. double the window size), rather than wait for a response packet by packet. The window size grows exponentially until a router in between the sender and the receiver discards (drops) some packets, effectively telling TCP through a time-out event that its window size has grown too large [10].

2.2 Congestion Avoidance

During the initial data transfer phase of a TCP connection the Slow Start algorithm is used. However, there may be a point during Slow Start that the network is forced to drop one or more packets due to overload or congestion [12].

When a network is congested, queue lengths start to increase exponentially. Congestion avoidance was devised as a technique to signal packet loss via time-outs and make TCP throttle back quickly (more quickly than queue lengths are growing), with the objective of stabilizing the whole system. Near the point of congestion, overly aggressive increases in connection bandwidth can drive the network into saturation, causing the “rush-hour effect”. To avoid the rush-hour phenomenon, the congestion avoidance mechanism increases bandwidth additively rather than exponentially. When congestion is detected via a timeout, bandwidth is scaled back aggressively by setting cwnd to half the current window size. Congestion avoidance is sometimes characterized as being an additive-increase, multiplicative-decrease (AIMD) technique [8].

While slow start and congestion avoidance were devised separately and for different purposes, they are almost always implemented together. The combined algorithm introduces a new variable called ssthresh (slow-start threshold), that effectively determines which mechanism is used.

2.3 Fast Retransmit

The TCP receiver can only acknowledge, the last packet received in sequence. Thus, if packets arrive out of sequence (e.g. one packet was lost but packets sent later arrive correctly), the receiver sends the same ACK more than once [11].

Fast retransmit algorithm makes usage of identical acknowledgement to discover packet loss. In Fast retransmit, during an acknowledgement packet is received a congestion window is fixed to three, TCP sender is adequately assured that the TCP packet is lost and will retransmit the packet beyond waiting for retransmission clock. Fast recovery is approximately connected to retransmit the packet [9].

In Fast recovery algorithm, TCP sender will not arrive in the slow start phase rather it will exactly decrease the congestion window by halve and boost the congestion window by estimating the convenient congestion window. When an acknowledgement of current data is received, it restore to congestion avoidance phase. This appropriate case may cause fast buffer uniform with low use determinant. Suddenly the queues on the maximum loaded lines will build endlessly and, in the end, exceed the width of the buffers at the equivalent nodes. This leads to the known fact that the packets retransmit to the nodes with complete buffers will be reboot and therefore are to be re-entering and that in change effect in wasting of network resources.

2.4 Fast Recovery

TCP Reno introduced a new mechanism called fast Recovery that, changes the congestion control behaviour, after retransmit: When three duplicate ACKs received, TCP sets the slow start threshold (ssthresh) to half of cwnd and cwnd to ssthresh plus three packets [15].

Fast recovery works hand in hand with fast retransmit to improve the responsiveness of TCP once congestion has occurred. Fast recovery introduces the concept of partial acknowledgements, which are used to notify that the next in-sequence packet has been lost so it can be re-transmitted immediately. Furthermore, instead of going back into slow-start mode, fast recovery jumps TCP into congestion avoidance mode. The congestion window is also reduced to the slow start threshold (ssthresh) instead of dropping all the way back to a window size of one packet [11].

3. TCP IMPLEMENTATIONS AND EXTENSIONS

In today's Internet, many different versions of TCP implementation coexist and communicate with each other, most common types of these versions are Reno, Vegas, Tahoe and New Reno, In this section, we review two different TCP implementations Reno and Vegas [17].

3.1 TCP Reno

TCP Reno introduced major improvements by changing the way in which, it reacts to detecting a loss through duplicate acknowledgements. The Reno TCP implementation retained the enhancements incorporated into Tahoe TCP but modified the Fast-Retransmit operation to include Fast Recovery [7]. The new algorithm prevents the communication channel from going empty after Fast Retransmit, thereby avoiding the need to Slow-Start to re-fill it after a single packet loss.

The idea is that the only way for a loss to be detected via a timeout and not via the receipt of a duplicate acknowledgements (dup ACK), is when the flow of packets and ACKs has completely stopped this would be an indication of heavy congestion [21].

The response to packet loss events has been modified in order to maintain a high sending rate, in a mildly congested network. The so, called coarse-grained implementation of the TCP, timeout leads to long idle periods, while waiting for the timeout timer to expire. During this waiting period, packet sending is discontinued, which results in low throughput. In TCP-Reno, the lengthy loss recovery phase has been improved upon, via the introduction of the fast-retransmit loss recovery algorithm [13].

Fast retransmit is a mechanism that, sometimes results in a much faster retransmission, of a lost packet than, what would have been possible if only the expire of timeout timers, was used to detect packet loss. The idea behind the fast-retransmit algorithm, is intuitive and easy to grasp. Every time a packet arrives to the receiver, the receiver responds by returning an acknowledgment packet.

Fast recovery replaces slow-start after a packet loss event is discovered by triple duplicates. The effect of fast retransmit / fast recovery is in principle that, if a packet loss is discovered via triple duplicates, the first lost packet will be quickly, resent and the congestion window size halved. If the resulting congestion window size allows it, linear increase during congestion avoidance follows directly. This results in a more aggressive, and more effective utilization of the available Network capacity, resulting in high throughput for TCP-Reno sender, when only a few packets are lost at each congestion event where the fast retransmit / fast recovery instance, will re-send, the first lost packet and quickly resume with congestion avoidance. If multiple packets are lost from a single window, the TCP-Reno fast retransmit / fast recovery, algorithm might, however, lead to multiple consecutive invocations, each invocation halving the congestion window size. in case of multiple packet losses, from a single window, the first re-sent packet will lead to, the receiver acknowledging, that it expects the second lost packet [14]. This ACK for a previously, sent and lost packet could be called a partial ACK. In the TCP-Reno implementation of fast retransmit / fast recovery, the arrival of partial ACKs will initiate a new fast retransmit / fast recovery followed by window halving.

These consecutive window halving, will decrease the congestion window so much that TCP-Reno, will ultimately not be able to send, any new packets, due to the congestion window size restriction, on the number of packets, it is allowed to have, un-acknowledged on the link. Hence, multiple packet losses, might finally lead to the sender having, to wait for a coarse timeout timer to expire, even if the re-sent packets, are being correctly received and acknowledged.

3.2 TCP Vegas

New TCP implementation, called Vegas is presented in 1994 by Brakmo. O'Malley and Petersen. TCP Vegas adopts a more sophisticated, bandwidth estimation scheme. Vegas algorithm estimates the buffering that does arise in reach the system and controls the rate affiliate with appropriate flow. This algorithm is absolutely capable to regulate and decrease the flow rate since the packet loss arise.

It uses the difference between expected and actual flows rates, to estimate the available bandwidth in the network. The idea is that when the network, is not congested, the actual flow rate, will be close to the expected flow rate [6]. Otherwise, the actual flow rate, will be smaller than the expected flow rate. TCP Vegas, using this difference in flow rates, estimates the congestion level, in the network and updates the window size accordingly. This difference in the flow rates can be easily translated into the difference between the window size and the number of acknowledged packets during the roundtrip time, using the equation:

$$\text{Diff} = (\text{Expected} – \text{Actual})\ \text{BaseRTT}$$

Where Expected is the expected rate, Actual is the actual rate, and BaseRTT is the minimum round trip time. The details of the algorithm are as follow:

1. First, the sender computes the expected flow rate:

$$Expected = \frac{CWND}{BaseRTT},$$

   where CWND is the current window size and BaseRTT is the minimum round trip time.

2. Second, the sender estimates the current flow rate by using the actual round trip time.

$$Actual = \frac{CWND}{RTT}$$

   where RTT is the actual round trip RTT time of a packet.

3. The sender, using the expected and actual flow rates, computes the estimated backlog in the queue from diff= (Expected – Actual) BaseRTT.

4. Based on diff, the sender updates its window size as follows:

$$CWND = \begin{cases} CWND+1 & \text{if } diff < \alpha \\ CWND-1 & \text{if } diff > \beta \\ CWND & otherwise \end{cases}$$

TCP Vegas tries to keep at least α packets but no more than β packets in the queues. The reason behind this is that TCP Vegas attempts to detect and utilize the extra bandwidth whenever it becomes available without congesting the network. This mechanism is fundamentally different from that used by TCP Reno. TCP Reno always updates its window size to guarantee full utilization of available bandwidth, leading to constant packet losses, whereas TCP Vegas does not cause any oscillation in window size once it converges to an equilibrium point

4. CONGESTION MANAGEMENT

Congestion can occur at any point in the network where there are points of speed mismatches, aggregation, or confluence Queuing manages congestion to provide bandwidth and delay guarantees.

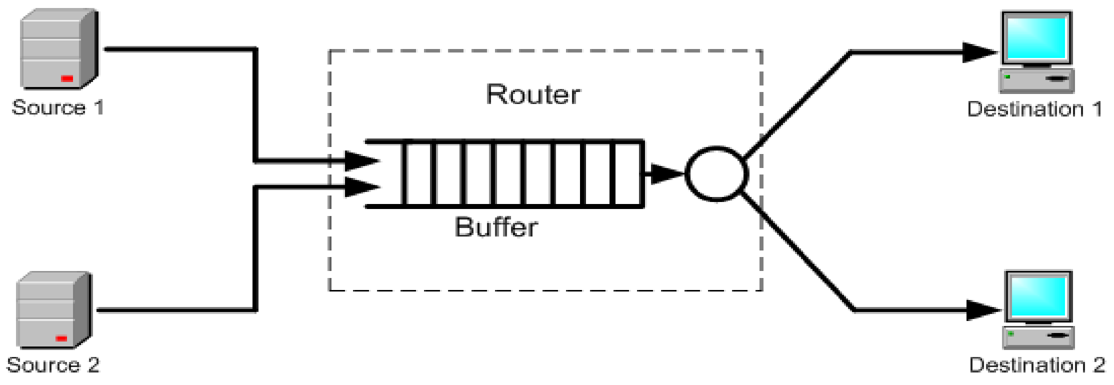

FIGURE 1: Queuing Theory **.**

The queuing theory work showing in Figure 1, when source 1 and source 2, send sum of packets at the same time the router starts to decide, where to direct first packet, and at the same time begin saving, the another packets into, the Buffer as seen in Figure 1. The way which the router (gateway), mange that packets in the buffer is called queuing theory [13].

During congestion, gateway influences, the fairness problem, because its queuing discipline, determines which packet to drop. This has effect on retransmissions used by TCP. Normal or default configuration, of gateways used the FIFO discipline, commonly referred to as Drop Tail (DT) gateway. Other possible gateway's, configurations of discipline include, the Random Drop (RD) Gateway and the Random Early Detection (RED) Gateway [16].

The TCP algorithm, at the sender, and the receiver is only part of the congestion control effort. An equally important part, is the congestion feedback from the network, which is the basis of any congestion control actions [20].

Congestion feedback, can be delivered in different ways. The most primitive feedback, is packet drops. In case of buffer overflow, the network simply drops incoming packets that cannot be accommodated in the router buffer. The source can detect the packet drop, either from a timeout, or from duplicate acknowledgments [22].

5. QUEUING MANAGEMENT ALGORITHMS

In this section we outline the most common queue management algorithms. Queue management is obviously about managing queues in forwarding devices such as routers and switches.

5.1 Drop Tail

The simplest queue management algorithm, where, when a queue becomes full, packets are simply dropped. Since packets can be dropped on all TCP connections simultaneously, many TCP connections on the link can be forced to go into slow-start mode. Figure 2 showing the Droptail queueing [20].

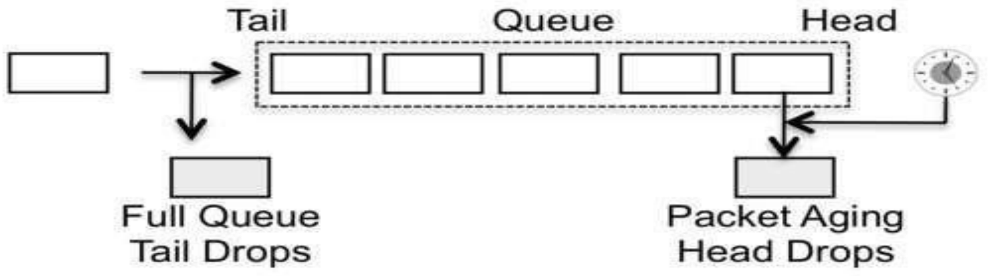

FIGURE 2: Drop Tail Queueing.

5.2 Fair Queuing

It is a queuing mechanism that is used to allow multiple packets flow to comparatively share the link capacity. "Fair Queuing" is an attempt to give the flows equal shares, at least within the limits of actual demand.

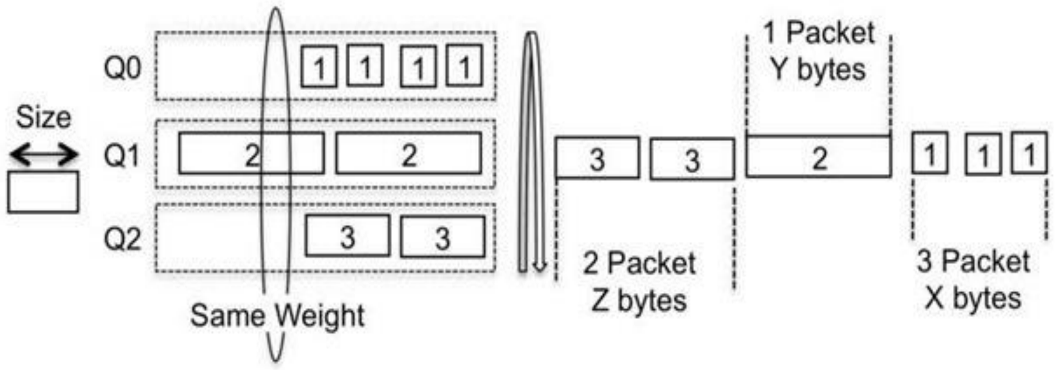

FIGURE 3: Round Robin with Different Size Packets.

The simplest algorithm for fair queuing is round-robin queue service, with all packets of equal size; this is sometimes called Nagle Fair Queuing, each nonempty queue gets to send an equal share of packets, Nagle fair queuing allows other flows to use more than their equal share, if some flows are underutilizing. Shares are divided equally among the active flows. As soon as a flow becomes active (that is, its queue becomes nonempty) it gets to start sharing in the bandwidth allocation; it does not have to wait for other flows to work through their backlogs, Round-robin works as fair queuing as long as all packets have the same size [18]. If packets have different sizes, then flows all get their fair share of packets per second, but this may not relate to bytes per second. FQ also ensure about the maximum throughput of the network. Figure 3 showing the Round Robin [19].

5.3 Deficit Round Robin

It is a modified weighted round robin scheduling mechanism. It can handle packets of different size without having knowledge of their mean size. Deficit Round Robin keeps track of credits for each flow. It derives ideas from Fair Queuing. It uses hashing to determine the queue to which a flow has to be assigned and collisions automatically reduce the bandwidth guaranteed to the flow. Each queue is assigned a quantum and can send a packet of size that can fit in the available quantum. If not, the idle quantum gets added to this meticulous queue's deficit and the packet can be sent in the next round. The quantum size is a very vital parameter in the DRR scheme, determining the upper bound on the latency as well as the throughput as shown in Figure 4. [20]

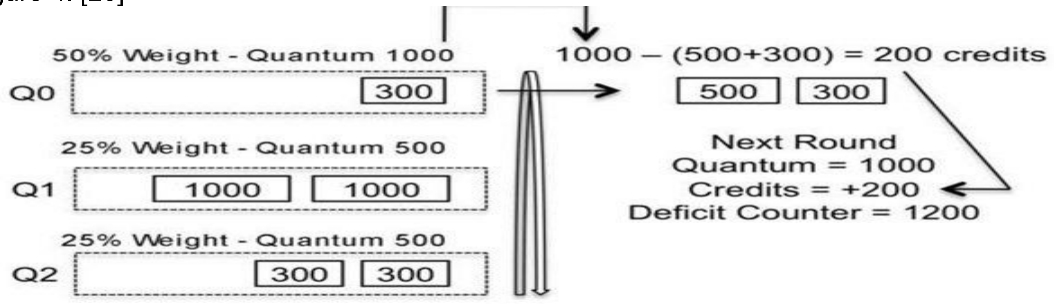

FIGURE 4: Deficit Round Robin.

5.4 RED
Random Early Detection (RED) is a congestion avoidance queuing mechanism. RED is a type of congestion control algorithm/mechanism that takes advantage of TCP's congestion control mechanisms and takes proactive approach to congestion.

It operates on the average queue size and drop packets on the basis of statistics information. If the buffer is empty all incoming packets are acknowledged. As the queue size increase the probability for randomly discarding a packet also increase. When buffer is full probability becomes equal to 1 and all incoming packets are dropped [21].

RED has three modes:

- No drop: When the average queue size is between 0 and the minimum threshold.
- Random drop: When the average queue size is between the minimum and the maximum Threshold.
- Full drop (tail drop): When the average queue size is at maximum threshold or above.

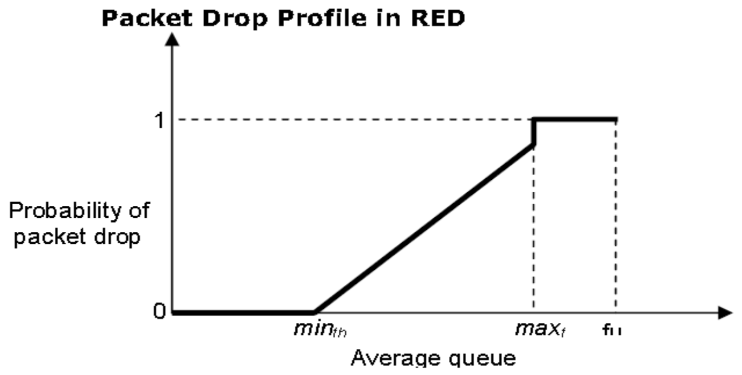

FIGURE 5: Packet Drop Profile In RED.

6. SIMULATION SCENARIO and CONFIGURATION
In this section, we evaluate the performance proposed for the different Queueing Algorithms over TCP Protocols. This network was examined using NS2 Simulator software.

Figure 6 show the proposed scheme of the network, covering the needs of an average networks with a data server that allows to send files and make voice / video calls and FTP Application. Although minor changes could be made to allow for the needs of different Networks. This network was designed to connect two branches; each branch has 4 nodes. In addition, 2 routers were used to connect network.

Here we will be using five different mechanisms which have different behavior for different network configuration and traffic pattern. Most importantly, the task in designing the simulation is to select parameters (bandwidth, queue limit, packet size, etc.) and a typical set of network topology. A simple topology is used in our simulation where different flows share a bottleneck between the two routers. The packets sent from sources queue to the queue of router and wait for transmitting. If the sender keeps sending and the queue overloaded, then congestion occurs.

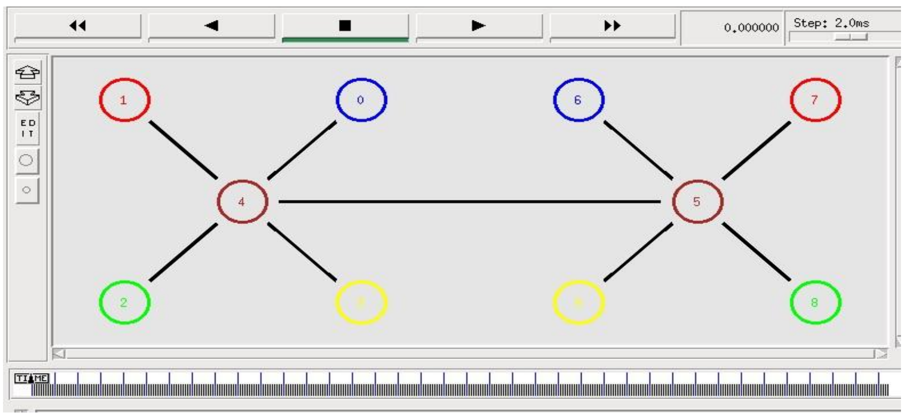

FIGURE 6: Network Topology of The Case Sudy.

There are four nodes at each side of the bottleneck link. Here four nodes are acting as a TCP source (Vegas, Reno) and four nodes are acting as a TCP sink so that both routers are applying the congestion control algorithm. We simulate this network on NS2 for different Queueing algorithms such as Drop tail, Fair Queueing, Random Early Detection and Deficit Round Robin over TCP protocols Vegas and Reno and compare the results for all proposal. This simulation has been observed over the period of 200 seconds.

TABLE 1: Simulation Parameters.

Simulation Parameter	V
Simulator	Ns-allinone-
Type of link	Duplex Link
Routers	2
Nodes	8
Network Applications	FTP, VOIP, Video conferencing
Queue	Drop Tail, FQ, RED, RDD
Speed	2MB
Delay	20ms
Transmission Protocol	TCP
Simulation start time	0s
Simulation finish time	200s

7. SIMULATION RESULTS

The performance of the congestion management system that consists of the queueing algorithm and the link congestion signal, algorithm has many facets, and these variables can impact the QoS, that applications experience from the network. Varieties of metrics that address different aspects of performance used to evaluate a congestion management system. At first, the results will be displayed algorithms, followed by an explanation of how the work of the network and right after, the results will be discussed. A final step and it will explain the main differences between the four algorithms.

7.1 Congestion Window in TCP Reno

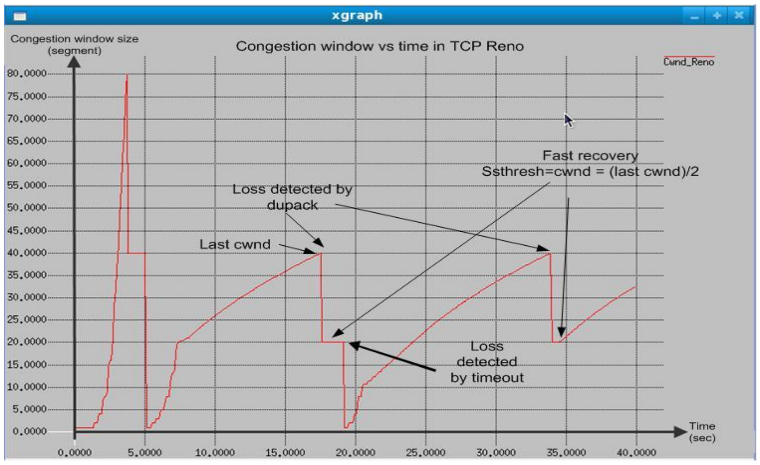

FIGURE 7: Congestion Window In TCP Reno.

7.2 Send, Dropped and ACK Packet In 40 Sec using TCP Reno

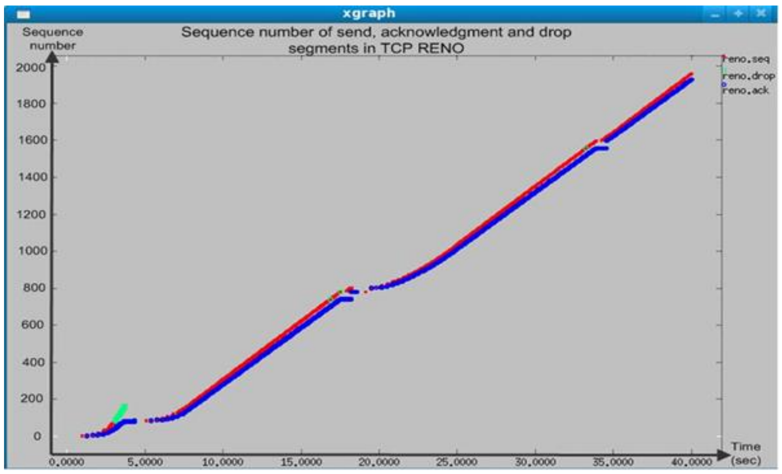

FIGURE 8: Send, Dropped and ACK Packet In 40 Sec using TCP Reno.

7.3 Queueing Algorithms (Droptail, FQ, RDD and RED) over TCP Reno

If all TCP Sources work as TCP Reno and the used queuing type is Drop Tail and Fair Queueing in a bottleneck link, G1-G2 and Queuing limit (Buffer size) is equal to 20. The results shown in Figure 9 Drop Tail and in Figure 10 FQ and tables are below.

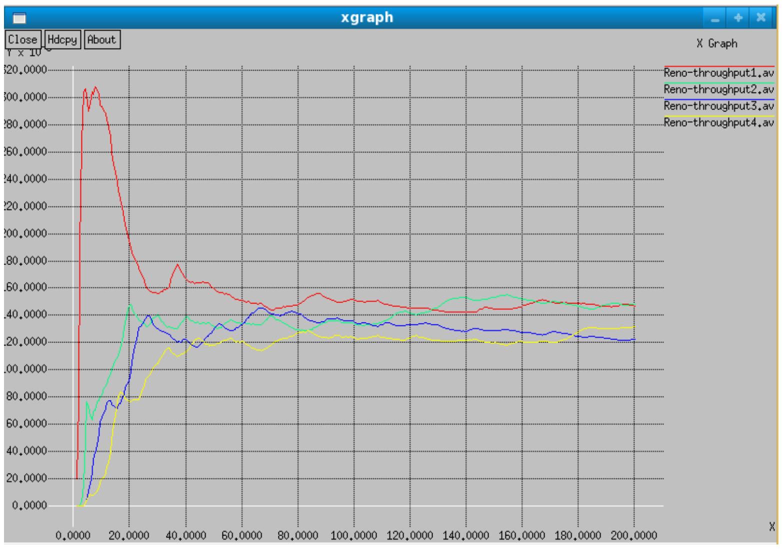

FIGURE 9: Throughput kbps to four sources work as TCP Reno when Queuing type is DropTail.

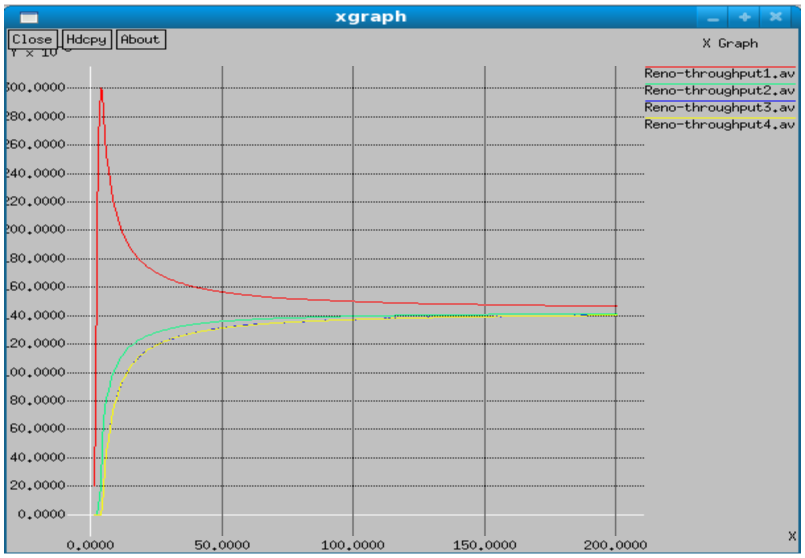

FIGURE 10: Throughput kbps to four sources work as TCP Reno when Queuing type is FQ.

If all TCP Sources work as TCP Reno and the used queuing type is DRR and RED in a bottleneck link, G1-G2 and Queuing limit (Buffer size) are equal to 20. The results shown in Figure 11 Deficit Round Robin and in Figure 12 RED and tables are below.

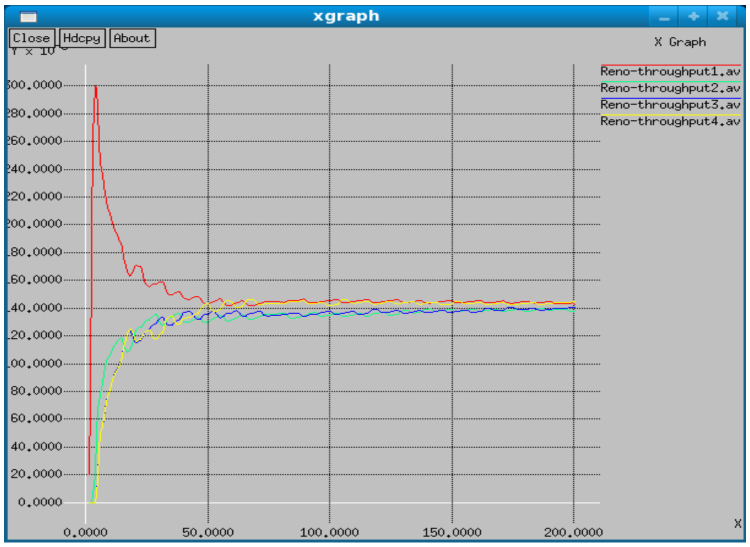

FIGURE 11: Throughput kbps to four sources work as TCP Reno and Queuing type is Deficit Round Robin.

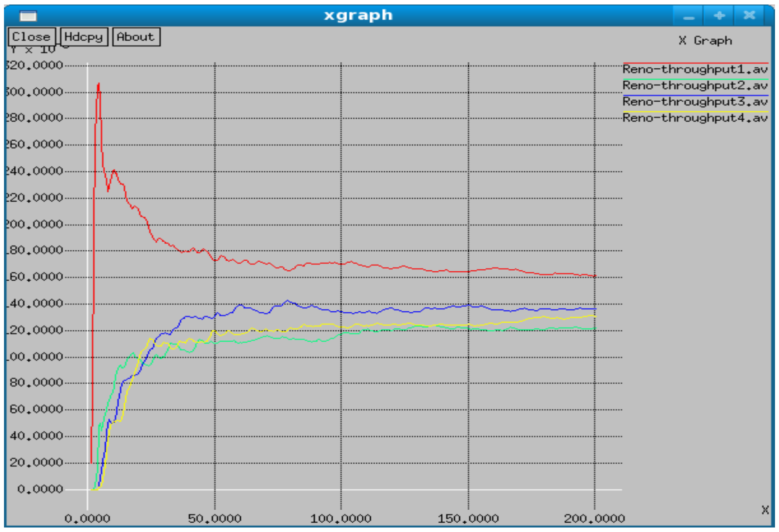

FIGURE 12: Throughput kbps to four sources work as TCP Reno and Queuing type is RED.

TABLE (7-1): Compare between Sent, Received and Dropped packets with different Queuing algorithms in TCP Reno**.**

	Sources	S1	S2	S3	S4
Droptail	**Sent Packet**	3775	3816	3155	3385
	Received Packet	3706	3750	3085	3320
	Dropped Packet	68	66	70	65
	Sent Packet	3719	3588	3559	3559
	Received Packet	3706	3576	3547	3547

Fair Queueing	**Dropped Packet**	0	0	0	0
DRR	**Sent Packet**	3675	3550	3627	3724
	Received Packet	3612	3480	3558	3660
	Dropped Packet	63	70	69	64
RED	**Sent Packet**	4182	3190	3551	3406
	Received Packet	4085	3077	3449	3306
	Dropped Packet	97	113	102	100

7.4 Congestion Window In TCP Vegas

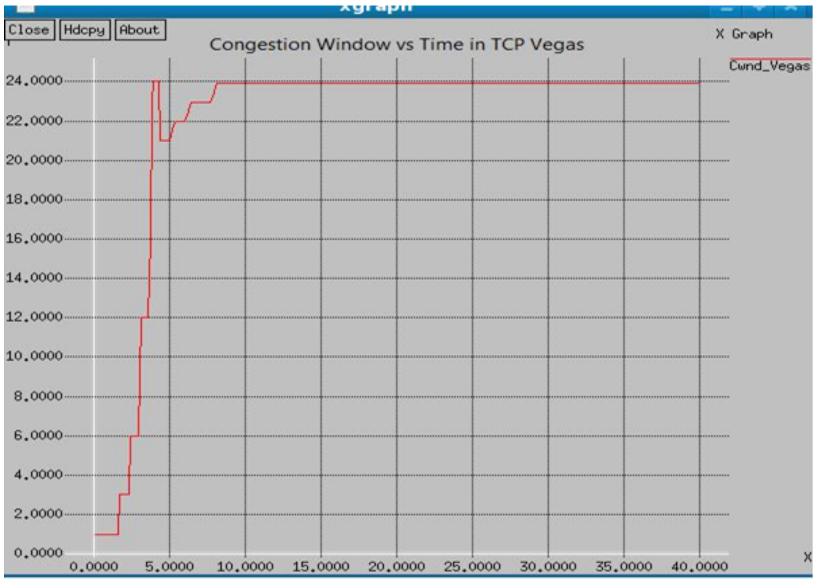

FIGURE 13: Congestion Window in TCP Vegas.

7.5 Send, dropped and ACK packet in 40 sec using TCP Vegas

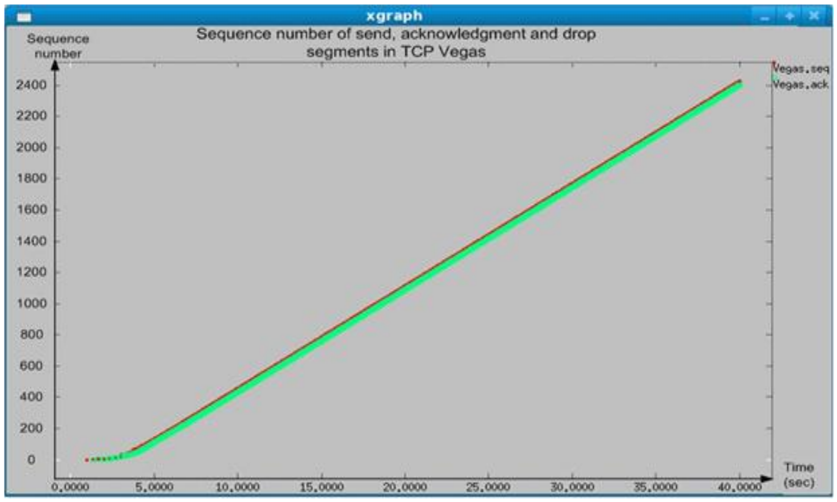

FIGURE 14: Send, Dropped and ACK Packet In 40 Sec using TCP Vegas.

7.6 Queueing Algorithms (Droptail, FQ, RDD and RED) over TCP Vegas

If all TCP Sources work as TCP Vegas and the used queuing type is Drop Tail and Fair Queueing in a bottleneck link, G1-G2 and Queuing limit (Buffer size) are equal to 20. The results shown in Figure 15 Drop Tail and in Figure 16 FQ and tables are below.

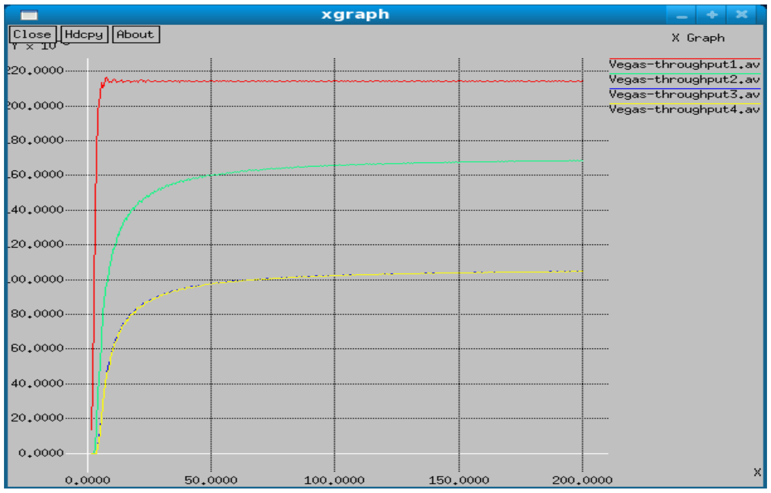

FIGURE 15: Throughput kbps to four-sources work as TCP Vegas when Queuing type is DropTail.

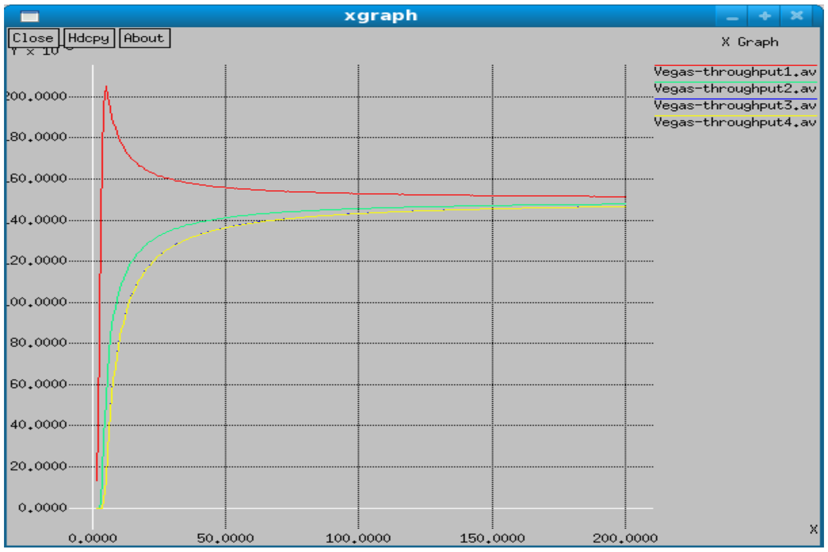

FIGURE 16: Throughput kbps to four-sources work as TCP Vegas when Queuing type is FairQueueing.

If all TCP Sources work as TCP Vegas and the used queuing type is DRR and RED in a bottleneck link, G1-G2 and Queuing limit (Buffer size) are equal to 20. The results shown in Figure 17 Drop Tail and in Figure 18 FQ and tables are below.

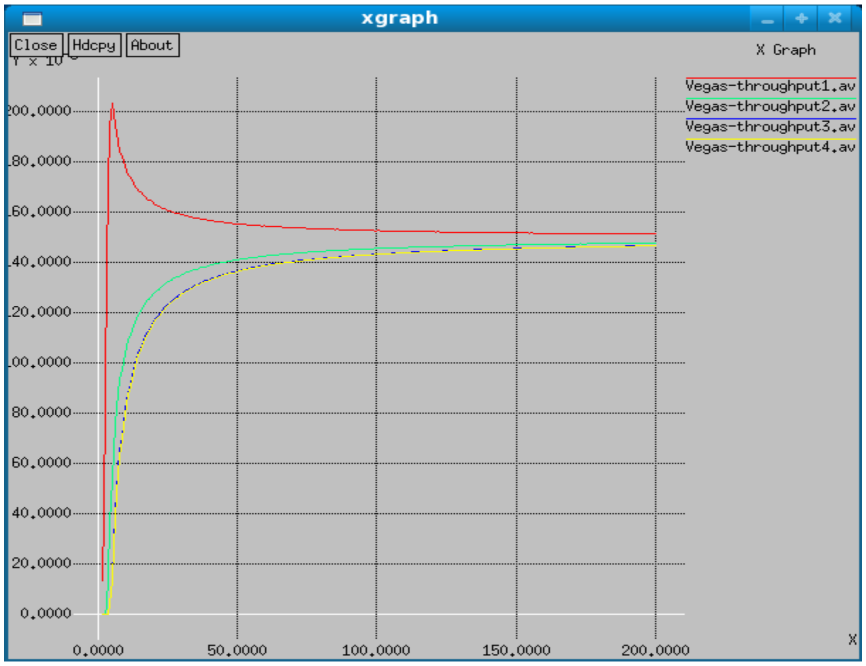

FIGURE 17: Throughput kbps to four sources work as TCP Vegas when Queuing type is DRR.

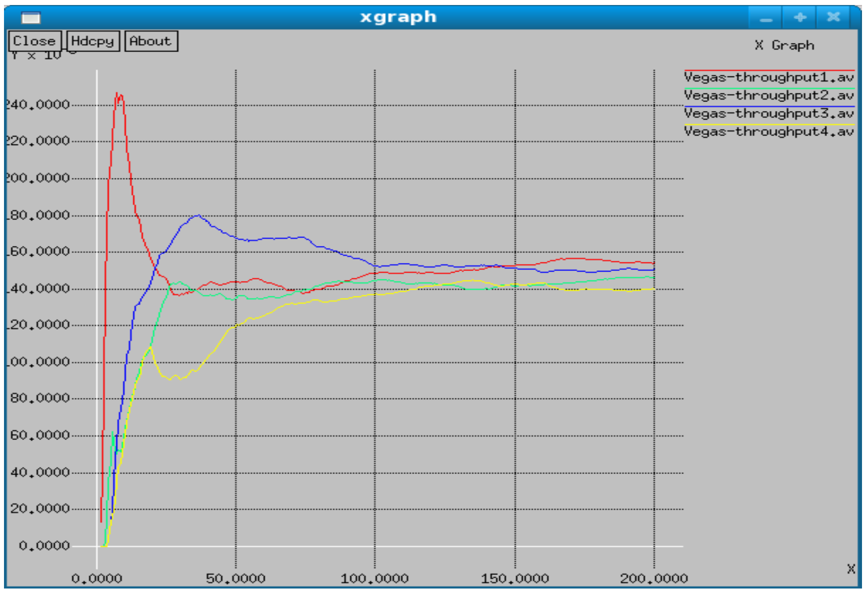

FIGURE 18: Throughput to four sources work as TCP Vegas when Queuing type is RED.

	Sources	**S1**	**S2**	**S3**	**S4**
Droptail	**Sent Packet**	5382	4234	2630	2630
	Received Packet	5382	4234	2630	2630
	Dropped Packet	0	0	0	0
Fair Queueing	**Sent Packet**	3803	3713	3681	3680
	Received Packet	3803	3713	3681	3680
	Dropped Packet	0	0	0	0
DRR	**Sent Packet**	3801	3712	3682	3681
	Received Packet	3801	3712	3682	3681
	Dropped Packet	0	0	0	0
RED	**Sent Packet**	3960	3761	3865	3625
	Received Packet	3869	3673	3779	3520
	Dropped Packet	91	88	86	105

TABLE (7-2): Compare between Sent, Received and Dropped packets with different Queuing algorithms in TCP Vegas**.**

7.8 Compare between sent and received packets with different Queuing Algorithms in (TCP Reno and Vegas):

Type	TCP Vegas	TCP Reno
Drop Tail	14876	13861
Fair Queueing	14877	14376
DRR	14876	14310
RED	14841	13917

TABLE (7-3): Compare Between Total of Sent Packets.

Type	TCP Vegas	TCP Reno
Drop Tail	1162.1785	276.0121
Fair Queueing	50.1416	65.7381
DRR	48.9540	66.8636
RED	1129.9141	374.0550

TABLE (7-4): Compare Standard Deviation.

8. CONCLUSION

In this paper, we analysed the proposed scheme of ‘A Comparison of Queueing Algorithms over TCP Protocol’.

While looking at the performance metrics (fairness and throughput) for all four queueing algorithms, we find that The DRR algorithm in TCP Vegas is the best in fairness. Then we find FQ algorithm in both TCP (Vegas and Reno) is better than DRR in Reno and the other two algorithms RED and Droptail, so we can conclude that the overall performance of FQ algorithm in both TCP protocols Reno and Vegas is better than DRR. DRR in TCP Vegas is better than FQ only in fairness but in throughput it is the same with FQ and Droptail and better than RED.

Moreover, we find Droptail in Reno is better than RED in both TCP protocols (Reno and Vegas), but when it is in Vegas, it is less fairness than RED and better throughput, so that the overall performance of Droptail in both TCP protocols Reno and Vegas is better than RED except when it is in Vegas, then it is worse in fairness.

From this study we can classify these four Queueing algorithms in accordance to performance metrics as this order:

1-FQ
2-DRR
3-Droptail
4-RED

9. REFERENCES

[1] Lawrence G. Roberts, “Beyond Moore’s Law: Internet Growth Trends,” Computer, vol. 33, no. 1, pp. 117-119, January 2000.

[2] Jeonghoon Mo, Richard J. La, Venkat Anantharam, and Jean Walrand, Analysis and Comparison of TCP Reno and Vegas.

[3] V. Jacobson. “Modified TCP Congestion Avoidance Algorithm”, Technical report, 30 Apr.1990

[4] M. Fomenkov, K. Keys, D. Moore, and K. Claffy, Longitudinal studyofInte rnettrafficin 1998-2003,WISICT’04:Proc.Winter Int. Symp. Info.Commun.Technol, 2004.

[5] K. Fall, and S. Floyd, “Simulation–Based Comparison of Tahoe, Reno and SACK

[6] TCP”, Computer Communications Review ACMSIGCOMM, Vol. 26, No. 3, July 1996 K. Fall, and S. Floyd, “Simulation–Based Comparison of Tahoe, Reno and

[7] SACK TCP”, Computer Communications Review ACMSIGCOMM, Vol. 26, No. 3, July 1996.

[8] M. Chiang, S. Low, A. Calderbank and J. Doyle, Layering as optimization decomposition: Amathematical theory of network architectures, Proc. of the IEEE,95(1): 255–312, 2007.

[9] UDP Protocol,http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/udp.html. 5 NOV 2015.

[10] UDP and TCP Segment Structure http ://searchnetworking.techtarget .com / definition/duplex http://www.ciscopress.com/store/cisco-ip-telephony-cipt -authorized-self-study-guide-9781587054099. , 22 Dec 2015.

[11] Nagle, J. RFC 896: Congestion control in IP/TCP internetworks (1984).

[12] TCP Protocol http://searchnetworking.techtarget.com/definition/TCP. , 5 DEC 2015.

[13] The Advantages and Disadvantages TCP and UDP http://smblog.iiitd .com /2010 /09 /advantages-and-disadvantages-of-tcp-and.html. , 31 DEC 2015.

[14] V. Jacobson, "Congestion Avoidance and Control", In Proceedings of ACM SIGCOMM' 88, PP. 314-329, Stanford, CA, August 1988.

[15] Chunlei Liu, B.Sc., M.S."Wireless Network Enhancements Using Congestion Coherence, Faster Congestion Feedback , Media Access Control and AAL2 Voice Trunking " 2001, The Ohio State University.

[16] B.B. et al."Recommendations on queue management and congestion avoidance in the internet". RFC 2309, April 1998.

[17] Romanow and S.Floyd. "The dynamics of TCP over ATM networks . In Proceedings,1994 SIGCOMM Conference , , London, 1994.

[18] N.Yin and M. G. Hluchyj."Implication of dropping packets from the front of a queue.proc".7th ITC seminar ,1990.

[19] T.Lakshman, A. Neidhardt, and T. Ott. The drop from front strategy in tcp and in tcp over atm,1996.

[20] S. Floyd and V.Jacobson. "Random early detection gateways for congestion avoidance". IEEE/ACM Transactions on Networking,Aug.1993.

[21] J. Postal."Internet control message protocol icmp",1981,ISI.

[22] Michael Welzl "Network Congestion Control Managing Internet Traffic" John Wiley & Sons Ltd,2005.