iJournals: International Journal of Software & Hardware Research in Engineering (IJSHRE)
ISSN-2347-4890
Volume 10 Issue 1 January 2022

Simulation and Performance Analysis of Different TCP Variants and Queuing Mechanisms in Different Network Topologies

Authors: MSc. Ing. Veranda Syla[1]; MSc.Ing. Ildi Alla[2]; MSc. Malvina Xhabafti[3]

Mediterranean University of Albania, Department of Informatics and Scientific Education[1,3]; Polytechnic University of Tirana[2]
veranda.syla@umsh.edu.al[1]; ildi.alla@fti.edu.al[2]; malvina.xhabafti@umsh.edu.al[3]

Abstract

The algorithm for controlling TCP Congestion as well as throughput is the main reason we can use the internet successfully today, despite the resource bottleneck or the very large and unforeseen access of users. There are several implementations of this protocol, where we can single out Vegas, Fack and Sack.Through simulations in NS-2, we were able to judge their performance in conditions of "congestion", arguing that Vegas was the most optimal.We have also highlighted the concept of *fairness* and *unfairness* among TCP types.The results of the respective simulations showed how "fair" these variants were in the use of bandwidth, where the Vegas type stands out for the fairer behavior.Finally, we focused on the effects of queuing algorithms, such as RED and the proposed algorithm, as well as their impact on fairness between TCP variants.

Keywords: Congestion, Fairness, Unfairness, Queuing, TCP Fack, Sack, Vegas, CBR

I. Methodology

The environment that came to our aid for the realization of these simulations was the NS2 simulator. The basic network topology was as follows, in which the appropriate maneuvers were then made with the TCP and CBR flows according to the respective requirements of the parts, which will be explained below.

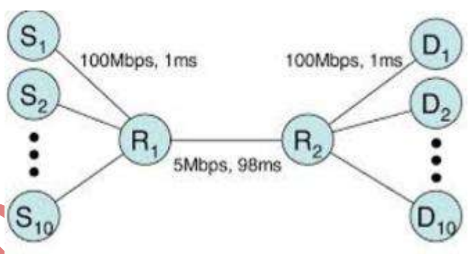


Fig 1. Network topology 1

The network used for the simulations, consists of 22 nodes, positioned as in the figure above, connected with links with bandwidth 100 and 5Mbps.

To find throughput, delay and drop_rate respectively, we referred to awk or perl scripts. The parameters used during the simulations are summarized in the table below. Programs like NAM, XGraph, GNUPlot are used to interpret graphic results.

Parameter	*Value*
Number of Nodes	22
Type of queue	DropTail or RED
Source types	Vegas, Sack, Fack
Simulation time	100 ms
Bandwidth links	100 Mb, 5Mb

Table.1. Simulation parameters

Performance metrics

Packetloss - network failure to enable transmission of one or more packets generated to destination.

Network delay- the time it takes one bit to go from one node to another. Measured in (ms).

Throughput-is the size of data that has been successfully transferred from one node to another for a certain period of time. Usually this is measured in megabits per second (Mbps).

Queue management mechanisms

Drop Tail – in this mechanism each packet is treated identically and when the queue is filled to maximum capacity, it drops each packet that comes up to the moment when the queue has space to receive traffic.

Fair Queuing – uses a queue for each data stream and serves you in turn so that each data stream receives equal resources.

RED-this mechanism drops packets before the buffer is fully loaded. It uses predictive models to decide which packets will be dropped.

II. The results of the simulations of the first part

During the first part we analyzed the performance of TCP variants inconditionsof "congestion". Constantly changing Cbr_rate values from 0.5Mbps to 3Mbps and with bottleneck value of 10. We estimated throughput, delay, and losses of TCP types. In advance, we provide theoretical information about how each of them reacts to the conditions of "congestion", as the results or their interpretation are consequences of the way these protocols work and avoid as much as possible the effects of congestion.

TCP Sack

Sack TCP requires that segments not become acknowledged in general, but in particular. So each ACK has a block indicating which segments are being acknowledged. Making it possible for the sender to be clear about who is doing what and who is not. When the sender enters the fast recovery phase he initializes a pipe variable where he calculates the size of the data that has not arrived, he also sets the CWND to half the size of the moment. Each time it receives an ACK it lowers the pipe by 1 and each time it retransmits a segment it raises it by 1. Whenever the pipe becomes smaller than the CWD window, it looks at which segments have not been received and sends them. If he see that there are no segments that have not been made ACK then brings a new package.

TCP Fack

It is an algorithm which works in the upper TCP Sack options. TCP Fack is the use of information provided through Sack to add additional precise data control to the network pipe within the recovery process. The basic concept of the Fack mechanism is from considering the larger number of ACK forward selective sequences, as a sign that the ACK segments were selectively missing. This monitoring allows to significantly improve the process of recovery of packet losses.The Fack algorithm examines lost ACKs between lost packets and Sack blocks. Fack is a good technique to halve the size of the window if a blockage has occurred. If cwnd is halved immediately, the TCP sender interrupts the transfer for a while and then resumes if a sufficient amount of data leaves the network. If blockage occurs, the window size should be halved depending on the correct cwnd reduction. The sender recognizes the blocking state after at least the single RTT has occurred and if through that RTT in the slow start phase then the last value of cwnd will be copied as much as the previous value when the blocking occurred. So in this state, the lock window is first halved to determine the accuracy of the cwnd, which should be further reduced. However, TCP Fack offers blocking avoidance, and fast retransmission mechanisms.

TCP Vegas

Vegas is also considered a modification of Reno. It is based on the logic that proactive measures for congestion management are more effective than reactive ones. It does not depend only on the loss of packages. It detects congestion before packet loss occurs. Vegas determines congestion by reducing the number of packets shipped compared to the expected number. When this gap deepens too much, then the transmission increases, thus making the band usable. In short, Vegas effectively tackles this phenomenon, without losing bandwidth by transmitting large volumes of data and creating congestion, then dropping them, as most other algorithms do.

Such a result is also noticeable from the values in the table below. After the simulations, it is found that the average throughput values of Vegas are higher than the other variants, the delays are almost the same, but with a small difference, lower, as well as the drop rate (calculated as the number of lost packets / number of those sent) is smaller. So the table speaks clearly about the individual performance of each protocol. Note that Sack and Fack exhibit dissimilar behaviors toward changing Cbr_rate values. Package losses start to show upfor the cbr_rate 3mbps value, which for Vegas,though not with any major changes, but are smaller.

VEGAS (DropTail)				
CBR_rate	Sent	Lost	Delay	Throughput
0.5 Mb	99055	200	35.75267 ms	63.8458Mbs
3 Mb	2990	640	17.531206 ms	1.10649Mbs
SACK (DropTail)				
0.5 Mb	65685	180	35.42651 ms	43.7243Mbs
3 Mb	390	120	16.931167 ms	0.0038824Mbs
FACK (DropTail)				
0.5 Mb	53585	145	35.2829 ms	35.6275Mbs
3 Mb	1470	210	21.093362 ms	0.580424Mbs
VEGAS (RED)				
3 Mb	905	250	15.933865 ms	0.26208Mbs
SACK (RED)				
3 Mb	465	45	22.994408 ms	0.0097067Mbs
FACK (RED)				
3 Mb	1250	130	25.051477 ms	0.423595Mbs

Table.2. Performance of TCP types

This increasing throughput trend for the DropTail queue mechanism is related to its feature of dropping all incoming packets in the queue when the queue is full.The graphical dependence of the throughput on time for each of the above TCP cases is shown graphically below.

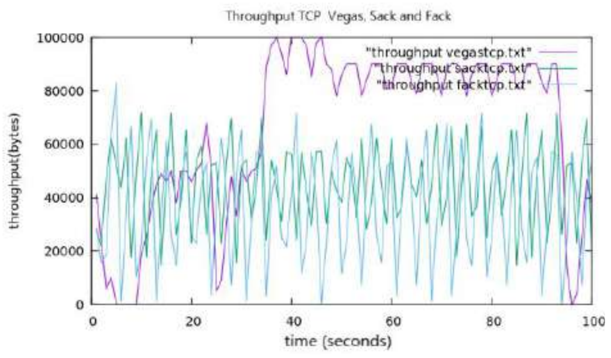


Fig.2. Throughput TCP, CBR_rate 0.5Mb, DropTail

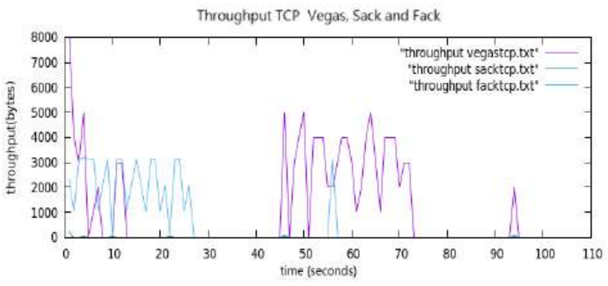


Fig.3. Throughput TCP, CBR_rate 3Mb, DropTail

For all three types of TCP we see that with increasing UDP source rate the number of packets received decreases for TCP and increases significantly for UDP. This comes as a result of increasing the UDP agent transmission rate and keeping the FTP traffic unchanged. Based on the values of Table 2 the number of TCP packages received is higher for Vegas TCP, as expected, as even from the studies conducted Vegas TCP versus TCP Sack / Fack has a throughput of 37-71% higher. This is not achieved by a more aggressive retransmission strategy, but through more efficient utilization of bandwidth. This is because the DropTail algorithm itself relies on the size of the buffer to increase the size of the window. The number of packets received UDP does not make any big difference as the protocol for it is the same in all cases. In any situation the number of UDP packets is greater as the transmission to UDP does not require the receipt of an ACK packet from the receiver and this increases the transmission rate.

Below we will see that if we change the next type in RED, we will not have the same performance. This algorithm is designed for "congestion avoidance". The main parameters are: threshold and maxthreshold.

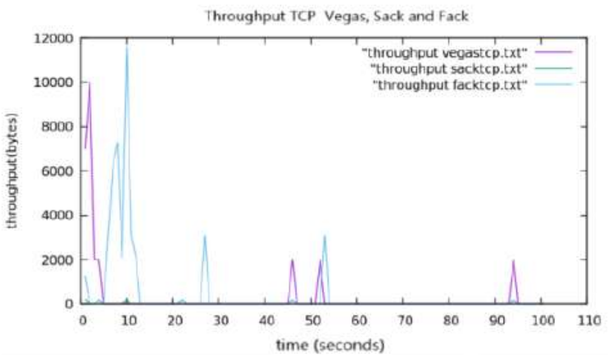


Fig.4. Throughput TCP, CBR_rate 3Mb, RED

The difference is that RED requires a smaller buffer size to maintain connectivity than DropTail. Therefore, TCP Vegas falls short of the performance in this algorithm.We see that TCP Fack has the best performance as this is more in line with the very nature of the RED algorithm.

The graphical representation of the window size from time, for each of the TCP cases, is shown below.

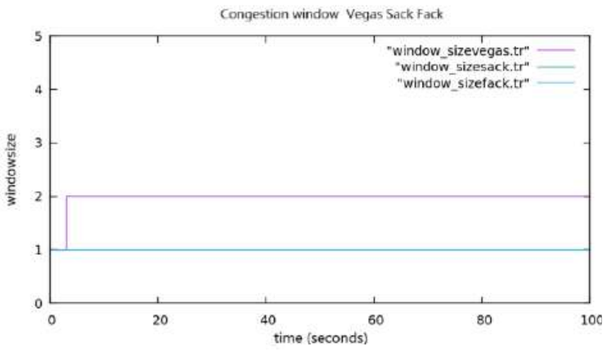


Fig.5. Window sizeCBR_rate 0.5Mb, DropTail

We see that the window size increases by 1 in the case of TCP Vegas and stays constant 1 in the other two types of TCPs. This confirms once again the conclusions reached above. This is because TCP Sack and Fack do not detect congestion before packet loss occurs. So TCP reacts to packet loss by resetting to 1=window size and setting the slow-start threshold to half the current value of the window size congestion.While TCP Vegas determines congestion by reducing the number of packets shipped compared to the expected number and therefore has better performance. We will get the same conclusions for the other CBR_rate DropTail cases and for RED, as we would have the same working principle.

It should be noted that the main actor of these simulations has been the placement of the bottleneck in 10 packages, as this has limited the amount of packages that will be placed in the buffer as well as how to increase the size of the window.

We show the slow start phase, and congestion avoidance according to the following definition:

1. Slow-start phase: if w <wth, TCP increases w by 1 for each ACK packet received.
2. Phase congestion avoidance: if w ≥ wth TCP increase 1/w(t) for each ACK packet received.

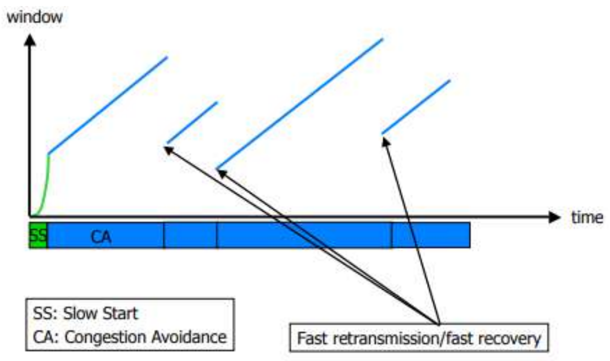


Fig.6. Congestion Control

Through the above presentation we detect SS, CA and FR in the window size graph (Fig.5).

III. The results of the second part

Before moving on to the analysis of the results of the second part, let us briefly dwell on a concise definition of the two concepts examined: fairness and unfairness. The simulated situation is as follows: unlike the first part, we already have five TCP streams, which share a certain bandwidth and the problem is how "fair" they are with each other in sharing bandwidthit. For this, we have simulated the TCP versions to be able to distinguish the most "fair" and the "unfair" pair at the same time. Something like this becomes easily distinguishable from the graphs found below. These two concepts also inevitably depend on link delays as well. When the source does not detect any congestion, it continues to increase the window by "1" over a period of time. Consequently, links with less latency, can change their window faster than those with longer latencies, thus occupying more bandwidth. Relevant scripts with the following commands are called to perform the simulations:

```
- perl throughput_tcp.pl <trfile><destNode><srcNode.port><destNode.port > out
```

```
- awk –f scriptIplote.awk ex2.tr
```

Considering the treated TCP variants: Vegas, Fack and Sack, based on the above-explained explanation of how delay affects fairness, we conclude that the treated cases are fair, which is interpreted as TCP of the same type, since they start at the same time and have almost the same latencies, then they are in equal bandwidth utilization positions. However, even between these there are very small differences as noted below:

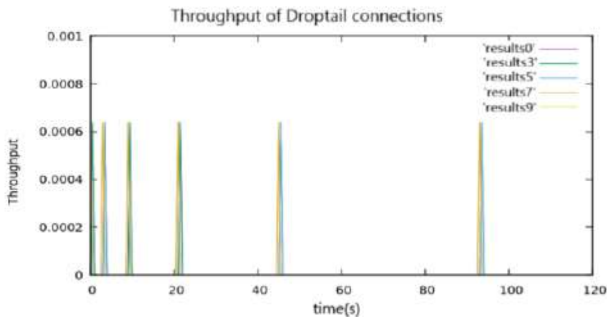


Fig.7. Throughput 5 connectionsTahoe,DropTail

In these, the tendency of the lines that indirectly represent the results for the bandwidth is noticed. We say that to some extent we can consider them fair to each other, as the throughput curves almost follow each other, without any significant difference. We have a large delay compared to the following results.Below are the simulation results in case the next type is already RED.

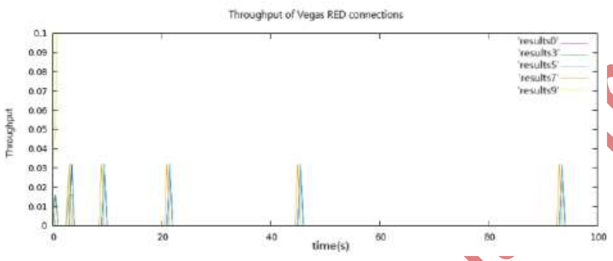


Fig.8. Throughput 5 connections Vegas, RED

Node 9 displays a different result from other network nodes, so it utilizes more bandwidth.

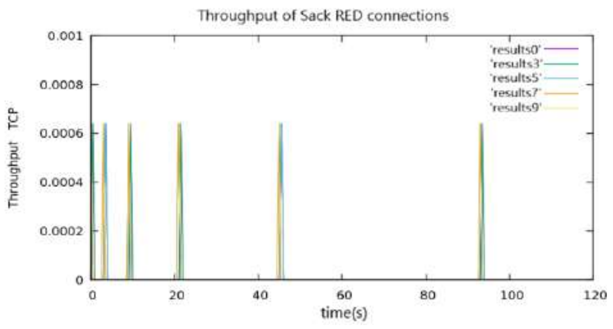


Fig.9. Throughput 5 connections Sack,RED

We see for the figure above, we have graphs that follow each other, so we have the same equal bandwidth utilization.

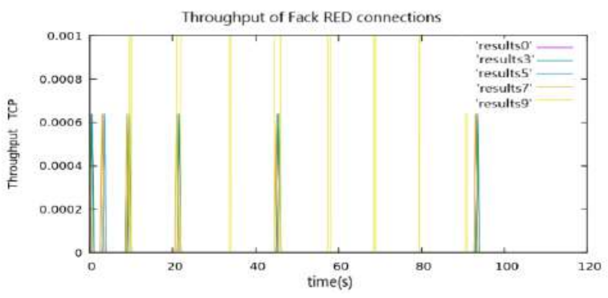


Fig.10. Throughput 5 connections Fack ,RED

In this graph we see that we have unequal bandwidth utilization. This is also shown by node 9 which shows another throughput result.

Almost the same types of graphs are taken for each of the cases, except that the throughput values change. As you can see, TCP Fack is not fair at all, we see that it gets more bandwidth than Sack and Vegas. This is also argued due to the different delays that these two flows have with each other, which are presented as in the following table.

TCP Type	Queue Type	Paketa TCP	ACK from TCP Sink	TCP Drop	ACK Drop	Average Delay
Vegas	RED	62260	12342	0	0	106.0307 ms
Sack	RED	52060	10332	50	0	91.15306 ms
Fack	RED	47780	9480	40	0	87.86996 ms

Table.3. Specifications for each TCP

Regarding the comparison of delays, we say that unfairness is more noticeable, due to the fact that Vegas, RED has the biggest delays, followed by Sack, RED and finally Fack, RED. Since they have obvious differences to some extent in the delay values, then it is understood that more bandwidth will be used by the version with a lower delay, thus not being "fair" with the other version, which has a longer delay. Vegas becomes unstable when network latency is large. When they "coexist" with the Fack and Sack connections, the latter utilize more bandwidth, thus giving the couple an unfair character.

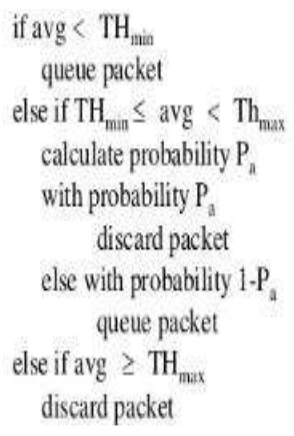
IV. The results of the third part

RED (Random Early Drop) is an algorithm that controls how packets are handled in a queue. In advance, let's make a brief look at how this algorithm works. The main parameter of its performance is: *buffer size*. The reason is that this one needs buffer space to increase or decrease the window. First, we say that this algorithm is designed for "congestion avoidance".

The main parameters are: threshold and maxthreshold. If the queue size exceeds the maxthreshold, then all incoming packets are dropped. It is important that this dimension be positioned between the threshold and maxthreshold values, so that we have good performance.

Even in this case, the same result is obtained regarding fairness between connections with different delays. The larger the buffer size, the better the fairness. The difference is that RED requires a smaller buffer size to maintain fair connections than the algorithm we will propose below.

The RED pseudocode will be as follows:

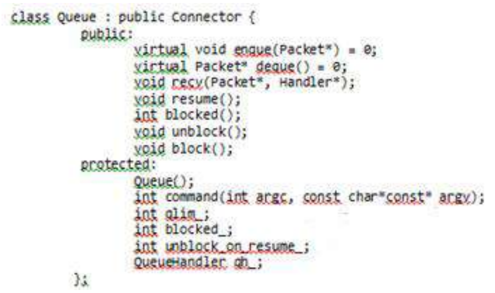


```
if avg < $TH_{min}$
    queue packet
else if $TH_{min} \leq$ avg < $Th_{max}$
    calculate probability $P_a$
    with probability $P_a$
        discard packet
    else with probability 1-$P_a$
        queue packet
else if avg $\geq TH_{max}$
    discard packet
```

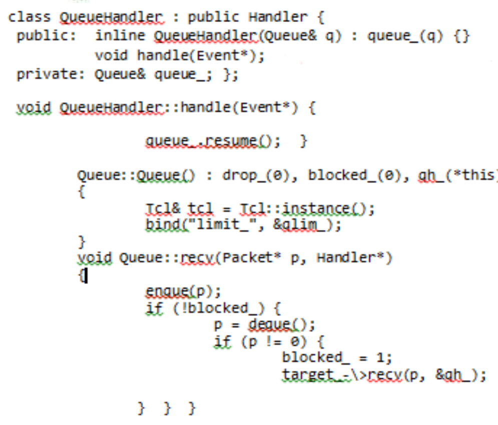
The following example illustrates the implementation of the Queue object. We proposed an idea for queue and buffer management.

The proposal is based on data flow planning according to the FIFO method. The algorithm must control the flow in such a way that the next size does not exceed the size of the buffer. If the next size exceeds the size of the buffer, this will result in many packets being dropped.

The Queue class provides a base class used by specific types of (extracted) queue classes, as well as a call function to implement blocking. The following definitions are given in queue.h:

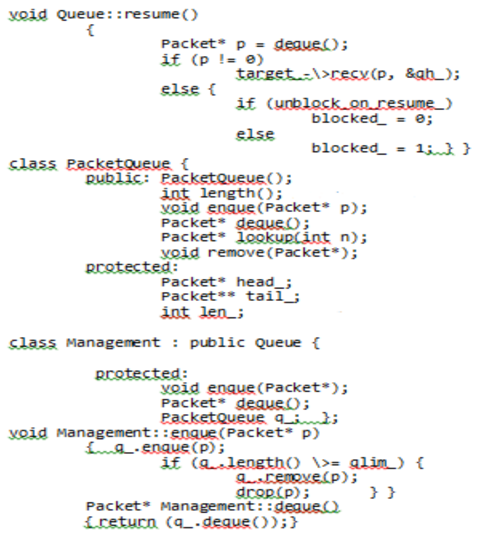


```
class Queue : public Connector {
        public:
                virtual void enque(Packet*) = 0;
                virtual Packet* deque() = 0;
                void recv(Packet*, Handler*);
                void resume();
                int blocked();
                void unblock();
                void block();
        protected:
                Queue();
                int command(int argc, const char*const* argv);
                int qlim_;
                int blocked_;
                int unblock_on_resume_;
                QueueHandler qh_;
       };
```



```
class QueueHandler : public Handler {
 public:  inline QueueHandler(Queue& q) : queue_(q) {}
          void handle(Event*);
 private: Queue& queue_; };

 void QueueHandler::handle(Event*) {

                queue_.resume();  }

        Queue::Queue() : drop_(0), blocked_(0), qh_(*this)
        {
                Tcl& tcl = Tcl::instance();
                bind("limit_", &qlim_);
        }
        void Queue::recv(Packet* p, Handler*)
        {
                enque(p);
                if (!blocked_) {
                        p = deque();
                        if (p != 0) {
                                blocked_ = 1;
                                target_-\>recv(p, &qh_);

              }  }  }
```

Control management here is somewhat subtle. When a Queue receives a packet it calls it a subclass version of the enque function with the packet. If the queue is not blocked, it is allowed to send a packet and calls the specific deque function which determines which packet is being sent, blocks the queue (because a packet is currently in transit), and sends the packet to the next neighbor.



```
void Queue::resume()
        {
                Packet* p = deque();
                if (p != 0)
                        target_-\>recv(p, &qh_);
                else {
                        if (unblock_on_resume_)
                                blocked_ = 0;
                        else
                                blocked_ = 1;_} }
class PacketQueue {
        public: PacketQueue();
                int length();
                void enque(Packet* p);
                Packet* deque();
                Packet* lookup(int n);
                void remove(Packet*);
        protected:
                Packet* head_;
                Packet** tail_;
                int len_;

class Management : public Queue {

         protected:
                void enque(Packet*);
                Packet* deque();
                PacketQueue q_;__};
void Management::enque(Packet* p)
        {__q_.enque(p);
                if (q_.length() \>= qlim_) {
                        q_.remove(p);
                        drop(p);        } }
        Packet* Management::deque()
        {return (q_.deque());}
```

The Queue class, from which Management is derived, provides most of the functionality needed. The Management row holds exactly one FIFO queue, implemented including a PacketQueue class object. Management implements enque and deque. Here, the enque function first stores the packet in the inner queue of packets (which has no size restrictions), and then controls the size of the packet queue versus qlim_. The overtaking space is applied by throwing the recently added packet in the queue, if the limit is reached or exceeded.

Note: in the implementation of the above code, placing qlim_ in n actually means an magnitude of queue n-1. Simple FIFO scheduling is implemented in the deque function always returning the first packet in the packet queue. The values derived from the simulations are as follows:

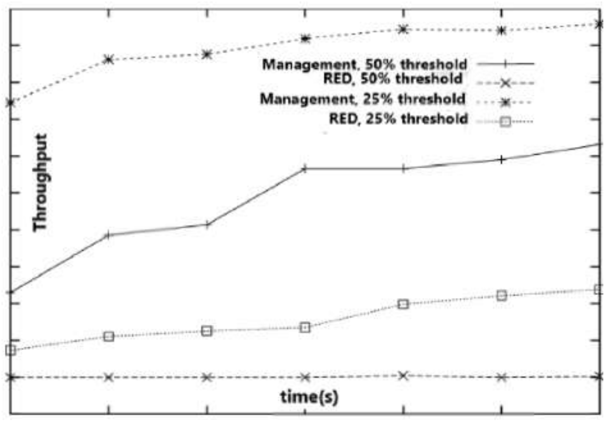


Fig.11. Comparison of algorithm performance

This graph clearly shows the effect of queue algorithms on the throughput of the two variants in question: proposed algorithm and that of RED. The proposed is seen to have the best performance for any threshold value.

V. Conclusion

In this paper we analyzed the performance of TCP Vegas, Fack and Sack, addressed from several aspects. From the simulations and results of the first part, we pointed out that the most optimal variant was Vegas, due to the higher throughput values, the smallest number of lost packets, as well as the smallest delay until it was the next type of DropTail, when done in RED it was the opposite. Next, in the second part we discussed the concepts of fairness and unfairness, already between the five TCP flows, where Tahoe and Vegas have the best performance. In the last part, we saw the impact of the next RED algorithm on the impact of data flow on the network, proposing an algorithm itself.We can say that for the very importance and applications that TCP has in the world of internet, this paper had an importance special.What we had previously encountered simply in the theoretical framework, we now had the opportunity thanks to the simulations in NS-2, the results become more tangible, through tables or graphs, through which were justified all the theoretical concepts encountered in advance.

One aspect we would like to address in more detail in the future and why not be a part of it is the more detailed performance regarding Routing Overhead, retransmission attempts and the factors that influence them.

References

[1]Zhong Ren, Chen-Khong Tham, Chun-Choong Foo, Chi-Chung Ko, " Integration of Mobile IP and Multi-Protocol Label Switching ", IEEE ICC 2001, vol. 7, pp 2123-2127, Helsinki, Finland, June 2001.

[2]Adam Wierman, Takayuki Osogami, and Jorgen Olsen. "A Unified Framework for ModelingTCP-Vegas, TCP-SACK, and TCP-Reno". Proceedings of MASCOTS, 2003.

[3]M. Asante and R.S. Sherratt (UK). "Mobile IP Convergence in MPLSbased Switching". In Proceeding of Wireless and Optical Communication MultiConfer-ence, July, 2004.

[4] X. Dubois "Performance of Diferent TCP Versions over Common/Dedicated UMTS Channels". Master Thesis, University of Namur, 2005.

[5]Mazleena Salleh and Ahmad Zaki Abu Bakar . Comparison of TCP Variants over Self-Similar Traffic, Proceedings of Computers, Communications, & Signal Processing with Special Track on Biomedical Engineering, Page(s):85 - 90, Nov. 2005.

[6]Muhammad Saeed Akbar, S. Zubair Ahmed, Muhammad Abdul Qadir "Quantitative Analytical Performance of TCP Va-riants in IP and MPLS Networks", IEEE Multitopic Conference, 2006. INMIC'06, 2006.

[7]Madiha Kazmi, Muhammad Younas Javed, Muhammad Khalil Afzal, An experimental study of the performance and effectiveness of TCP Variants in IP and MPLS Networks, Journal of Networking Technology, Volume 2 Number 3 September 2011.

[8]Madiha Kazmia Muhammad Younas Javed, and Muhammad Khalil Afzal , An overview of performance comparison of diferent TCP Variants in IP and MPLS Networks, ndt 2011.

[9]Md. Monzur Morshed* 1, 2, Meftah Ur Rahman2, 3, Md. Rafiqul Islam1, An Empirical Study on variants of TCP over AODV routing protocol in MANET, June 2011.

[10] Hui (Hilary) Zhang, Zhengbing Bian, Evaluation of different TCP Congestion Control Algorithms using NS-2, Spring 2012.

[11]Madiha Kazmi, Azra Shamim, Nasir Wahab, and Fozia Anwar, Comparison of TCP Tahoe, Reno, New Reno, Sack and Vegas in IP and MPLS Networks under Constant Bit Rate Traffic, 2014.