IMPLIMENTATION OF CONGESTION MANAGEMENT SYESTEM USING TCP CONTROLLING TECHNIQUE

MonuRani **Dr. V.K. Pandey**

Deptt.Of Comp. Sci. &Engg.Deptt.Of Comp. Sci. &Engg.

SKITM, (MDU University) *SKITM, (MDU University)*

Bahadurgarh, IndiaBahadurgarh, India

Abstract: -In this research paper we compares 4 existing TCP congestion control mechanisms namely TCP-Reno, New Reno, Vegas and Sack1Four nodes, each generating a particular traffic are considered, and they are connected to a bottle-neck link which in turn is connected to the sink.After comparision of these 4 congestion control mechanisms we provide a TCL script that compare these algorithms performance. Congestion management techniques control congestion after it has occurred. One way that network elements handle an overflow of arriving traffic is to use a queuing algorithm to sort the traffic, then determine some servicing method of prioritizing it onto an output link.

Keywords :-TCP congestion control technique, TCL , EEM(Embedded Event manager)Scripting ,

1. INTRODUCTION

Congestion control is one of the performance metrics of TCP protocol. There are so many TCP Versions tocontrol congestion in the network. NS2 also supports various TCP protocols like TCP Vegas, TCP Reno, TCP, TCP Sack, Full TCP, TCP linux, etc. Each TCP protocols has different mechanism in controlling the congestion. Some are good at Congestion control, some are good at error ands flow control**.**TCP is a reliable connection oriented end-to-end protocol. It contains within itself, mechanisms for ensuring reliability by requiring the receiver the acknowledge the segments that it receives. The network is not perfect and a small percentage of packets are lost en route, either due to network error or due to the fact that there is congestion in the network and the routers are dropping packets. We shall 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 becomes increasingly important for TCP to react to a packet loss and take action to reduce congestion. TCP ensures reliability by starting a timer whenever it sends a segment. If it does not

receive an acknowledgement from the receiver within the 'time-out' interval then it retransmits the segment. We shall start the paper by taking a breif look at each of thecongestion avoidance algorithms and noting how they differ from each other. In the end we shall do a head to head comparison to further bring into ligh

2. CONGESTION CONTROLLING

Congestion is a problem that occurs on shared networks when multiple users contend for access to the same resources (bandwidth, buffers, and queues). Think about freeway congestion. Many vehicles enter the freeway without regard for impending or existing congestion. As more vehicles enter the freeway, congestion gets worse. Eventually, the on-ramps may back up, preventing vehicles from getting on at all.

Congestion typically occurs where multiple links feed into a single link, such as where internal LANs are connected to WAN links. Congestion also occurs at routers in core networks where nodes are subjected to more traffic than they are designed to handle. TCP/IP networks such as the Internet are especially susceptible to congestion because of their basic connection- less nature. There are no virtual circuits with guaranteed bandwidth. Packets are injected by any host at any time, and those packets are variable in size, which make predicting traffic patterns and providing guaranteed service impossible. While connectionless networks have advantages, quality of service is not one of them

MAJOR PERFORMANCE MEASURES & OVERVIEW OF CONGESTION CONTROL SCHEMES

The major performance metrics under consideration are:

- Throughput
- Mean Queue length

The most widely deployed congestion control mechanisms are:

Drop Tail

Drop tail is the simplest and most widely used congestion control scheme in the current Internet routers. It works on first-in-first out (FIFO) based queue of limited size, which simply drops any incoming packets when the queue becomes full. Because of its simple nature, it's easy to implement. Apart from

simplicity other advantages include suitability to heterogeneity and its decentralized nature moreover its FIFO based queue provides better

Active Queue Management

Active queue management is a technique in which routers actively drop packets from queues as a signal to senders that they should slow down. RFC 2309 lists the following advantages of active queue management:

- Burst are inevitable. Keeping queue size small and actively managing queues improves a router's ability to absorb bursts without dropping excessive packets.
- If a source overflows a shared queue, all the devices sharing that queue will slow down (the "global synchronization" problem).
- Recovering from many dropped packets is more difficult than recovering from a single dropped packet.

AIMD: Additive Increase/Multiplicative-Decrease

In traditional TCP, the feed back control algorithm used to avoid congestion is the “additive increase/multiplicative-decrease (AIMD)”. This algorithm is basically used to implement TCP window . When congestion takes place, AIMD linearly expended congestion window with exponential decrease in it. The general rule of additive increase is to increase the congestion window by 1 maximum segment size (MSS) every round trip time (RTT) up to the detection of packet loss.

Random Early Detection (RED):

RED algorithm for RED Gateways was first of all proposed by Sally Floyd and Van Jacobson [5], it calculates the average queue size by using a low pass filter with Exponential Weighted Moving Average (EWMA). RED addresses the shortcomings of traditional Drop Tail algorithm.**Blue:**

Blue is another extension of RED developed by Wu-Chang and Feng et al] which uses packet loss and link utilization (rather than queue size) as a control variables to measure the network congestion.

ECN (Explicit Congestion Notification)

The problem with RED is that it drops packets. A more efficient technique would be for a router to set a congestion notification bit in a packet, and then send the packet to the receiver. The receiver could then

inform the sender to slow down via a message in the ACK. All the while, the receiver gets its packet and we avoid using packet drops to signal congestion.

ECN is an end-to-end congestion avoidance mechanism that adopts this technique. As the name implies, ECN provides direct notification of congestion rather than indirectly signaling congestion via dropped packets.

TCP Rate Control

TCP rate control is a technique in which endpoints can adjust their transmissions based on feedback from network devices that perform rate control. Packeteer is an advocate of rate control and this section describes how the company implements it in its Packet Shaper products. Packeteer's Web site has numerous papers on rate control and other congestion control topics.

TCP Rate Control is also known as ERC (explicit rate control). A form of ERC is implemented in ATM networks. The Lawrence G. Roberts paper mentioned earlier in this section describes ERC in both ATM and TCP networks .

2. TCP CONGESTION CONTROL TECHNIQUE

There are 4 Techniques of tcp congestion control:-

1.TCP Reno

2.TCP New Reno

3. TCP Vages

4. TCP Sack1

TCP RENO:-This Reno retains the basic principle of Tahoe, such as slow starts and the coarse grain re-transmit timer. However it adds some intelligence over it so that lost packets are detected earlier and the pipeline is not emptied every time a packet is lost. Reno requires that we receive immediate acknowledgement whenever a segment is received. The logic behind this is that whenever sequence expected, has been delayed in the network and the segments reached there out of order or else that the packet is lost. If we receive a number of duplicate acknowledgements then that means that sufficient

time has passed and even if the segment had taken a longer path, it should have gotten to the receiver by now. There is a very high probability that it was lost. So Reno suggest an algorithm called 'Fast ReTransmit'. Whenever we receive 3 duplicate ACK's we take it as a sign that the segment was lost, so we re-transmit the segment without waiting for timeout. Thus we manage to re-transmit the segment with the pipe almost full

Problems:

Reno performs very well over TCP when the packet losses are small. But when we have multiple packet losses in one window then RENO doesn't perform too well and it's 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 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 fastretransmit and we would have to wait for a coarse grained timeout. Thus is cannot effectively detect multiple packet losses. 2 and then enter fast recovery. Thus when we exit fast recovery for the second time our window size is set to 2.

TCP NEW-RENO:-

New RENO is a slight modification over TCP-RENO. It is able to detect multiple packet losses and thus is much more efficient that RENO in the event of multiple packet losses. Like Reno, New-Reno also enters into fast-retransmit when it receives multiple duplicate packets, however it differs from RENO in that it doesn't exit fast-recovery until all the data which was out standing at the time it entered fastrecovery is acknowledged. Thus it overcomes the problem faced by Reno of reducing the CWD multiples times. The fast-transmit phase is the same as in Reno. The difference in the fastrecovery phase which allows for

multiple re-transmissions in new-Reno. Whenever new-Reno enters fastrecovery it notes the maximums segment which is outstanding. The fast-recovery phase proceeds as in Reno, however when a fresh ACK is received then there are two cases: If it ACK's all the segments which were outstanding when we entered fastrecovery then it exits fast recovery and sets CWD to ssthresh and continues congestion avoidance like Tahoe. If the ACK is a partial ACK then it deduces that the next segment in line was lost and it re-transmits that segmentand sets the number of duplicate ACKS received to zero. It exits Fast recovery when all the data in the window is acknowledged.

Problems:

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

TCP SACK:

TCP with 'Selective Acknowledgments' is an extension of TCP Reno and it works around the problems face by TCP RENO and TCP New-Reno, namely detection of multiple lost packets, and re-transmission of more than one lost packet per RTT. SACK retains the slow-start and fast-retransmit parts of RENO. It also has the coarse grained timeout of Tahoe to fall back on, incase a packet loss is not detected by the modified algorithm. SACK TCP requires that segments not be acknowledged cumulatively but should be acknowledged selectively. Thus each ACK has a block which describes which segments are being acknowledged. Thus the sender has a picture of which segments have been acknowledged and which are still outstanding. Whenever the sender enters fast recovery, it initializes a variable pipe which is an estimate of how much data is outstanding in the network, and it also set CWND to half the current size. Every time it receives an ACK it reduces the pipe by 1 and every time it re-transmits a segment it increments it by 1. Whenever the pipe goes smaller than the CWD window it checks which segments are un received and send them. If thereare no such segments outstanding then it sends a new packet [5]. Thus more than one lost segment can be sent in one RTT.

Problems:

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

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 packetloss can still be detected by the coarse grain timeout of the other mechanisms fail.

The two major changes induced by Vegas are: -

New Re-Transmission Mechanism:

Vegas extend on the re-transmissionmechanism 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 re-transmits 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 Ack's. To catch any other segments that may have been lost prior to the re-transmission, 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 multipple 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 Crowcroft;s Tri-S scheme. The details can found in . Thus whenever the calculated rate is too far away from the

4.TCL SCRIPT

```
set ns [new Simulator]
setnam_trace_fd [open tcp_congestion.nam w]
$ns namtrace-all $nam_trace_fd
settrace_fd [open tcp_congestion.tr w]

# open the measurement output files
set throughput_flow_0_trace_fd [open tcp_congestion_0.tr w]
set throughput_flow_1_trace_fd [open tcp_congestion_1.tr w]
set throughput_flow_2_trace_fd [open tcp_congestion_2.tr w]
set throughput_flow_3_trace_fd [open tcp_congestion_3.tr w]

setpacketSize 1000

# define different colors for nam data flows
$ns color 0 Green
$ns color 1 Blue
$ns color 2 Red
$ns color 3 Yellow

#Define a 'finish' procedure
proc finish {} {
global ns nam_trace_fdtrace_fd
globaltrace_fd throughput_flow_0_trace_fd throughput_flow_1_trace_fd throughput_flow_2_trace_fd
throughput_flow_3_trace_fd
```

```
        # close the nam trace file
    $ns flush-trace
close $nam_trace_fd



        # close the measurement files
        close $trace_fd
        close $throughput_flow_0_trace_fd
        close $throughput_flow_1_trace_fd
        close $throughput_flow_2_trace_fd
        close $throughput_flow_3_trace_fd

        execxgraph tcp_congestion_0.tr tcp_congestion_1.tr tcp_congestion_2.tr tcp_congestion_3.tr
-geometry 800x400

        # execute nam on the trace file
  #exec nam tcp_congestion.nam &
exit 0
}

# Records Statistics
procrecord_stat {} {
        globalflow_monitorpacketSize
        global throughput_flow_0_trace_fd throughput_flow_1_trace_fd
throughput_flow_2_trace_fd throughput_flow_3_trace_fd

        # get an instance of the simulator
        set ns [Simulator instance]

        # set the time after which the procedure should be called again
        set time 1
```

```
# get the current time
set now [$ns now]

# how many bytes have been received by the traffic sinks?
setpacket_arrival [$flow_monitor set parrivals_]
setpacket_departure [$flow_monitor set pdepartures_]
setpacket_drop [$flow_monitor set pdrops_]
setflow_classifier [$flow_monitor classifier]


setflow_fd [$flow_classifier lookup auto 0 00]
if { $flow_fd != "" } {
```

5.CONCLUSTION

This paper presented a TCP algorithm which solves the congestion control problem. Implement a TCP algorithm using TCL Script. The above techniques take so much time and manpower consuming. For avoid this situation network administer use a scripting language,EEM scripting. EEM scripting provide us a simple and automatic way for controlling and analysis the congestion in net work.Harnessing the significant intelligence within Cisco devices, IOS Embedded Event Manager helps enable creative solutions, including automated troubleshooting, fault detection, and device configuration..Congestion analysis and congestion controlling techniques are the way of management of network. By using these techniques network adminisitor can solve the problems occur due to congestion.

6.**REFERENCES**

[1] S. Shalunov, “Low Extra Delay Background Transport (LEDBAT),” IETF Internet Draft, (work-in-progress), Mar. 2010, http://tools.ietf.org/pdf/draft-ietf-ledbat-congestion-00.pdf.

[2] J. Postel, “Transmission control protocol (TCP),” IETF RFC 793, 1981.

[3] F. Kelly, “Mathematical modelling of the Internet,” in Proceedings of the Fourth International Congress on Industrial and Applied Mathematics, Edinburgh, Scotland, 5–9 Jul. 1999, pp. 105–116.

[4] uTorrent, "Micro transport protocol (UTP)," http://www.utorrent.com/documentation/utp, accessed on 24th September 2010.

[5] S. Floyd and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm," IETF RFC 2582, Apr. 1999.

[6] M. Allman, V. Paxson, and W. R. Stevens, "TCP congestion control," IETF RFC 2581, Apr. 1999, http://www.ietf.org/rfc/ rfc2581.txt.

[7] D. Rossi, C. Testa, and S. Valenti, "Yes, we LEDBAT: Playing with the new BitTorrent congestion control algorithm," in Passive and Active Measurement (PAM), Zurich, Switzerland, Apr. 2010, pp. 31–40.

[8] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "LEDBAT: the new BitTorrent congestion control protocol," in Proceedings of the International Conference on Computer Communication Networks, Zurich, Switzerland, Aug. 2010, pp. 1–6.

[9] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti, "The quest for LEDBAT fairness," in Proceedings of IEEE Globecom, Miami, FL, Dec. 2010, p. (to appear).

[10] G. Carofiglio, L. Muscariello, D. Rossi, and C. Testa, "A hands-on assessment of transport protocols with lower than best effort priority," in Proceedings of the 35th IEEE Conference on Local Computer Networks, Denver, CO, Oct. 2010, p. (to appear).

[11] M. I. Andreica, N. Tapus, and P. Johan, "Performance evaluation of a python implementation of the new LEDBAT congestion control algorithm," in Proceedings of IEEE International Conference on Automation, Quality and Testing Robotics (AQTR), Cluj-Napoca,Romania, May 2010, pp. 1– 6.

[12] A. J. Abu and S. Gordon, "A dynamic algorithm for stabilising LEDBAT congestion window," in Proceedings of the International Conference on Computer and Network Technology, Bangkok, Thailand, Apr. 2010, pp. 157–16