Volume : 2 | Issue : 4 | Apr 2013 • ISSN No 2277 - 8179

Research Paper

Performance Analysis by Varying Parameters of TCP Variants Using NS-2 Simulator

Engineering

KEYWORDS : TCP variants, multiple packet loss, Congestion control, NS-2,Throughput

Mrs. Heena H Dave M.E. [Digital Communication] Student, Department of E & C Engineering, Technocrats Institute of Technology, Bhopal, Madhya Pradesh

Prof. Parul Dihulia Department of E & C Engineering, Technocrats Institute of Technology, Bhopal, Madhya Pradesh

Prof. Vikas Gupta Asst.Professor and Head, Department of E & C Engineering, Technocrats Institute of Technology, Bhopal, Madhya Pradesh

ABSTRACT

Transmission Control Protocol (TCP) is a reliable, end-to-end transport protocol which is most widely used for data services and is very efficient for wired networks. It also performs well in wireless networks. It is the backbone protocol of most of the internet based applications. In this paper we are comparing different congestion control and avoidance techniques which have been proposed for TCP protocols like Basic TCP Tahoe, Reno, New Reno and SACK. Our analysis is on performance of the variants of TCP. In this paper we carry out performance study of four different variants of TCP to be able to classify which variant of TCP performs better in various possible scenarios. This paper describes an ns-2 based simulation analysis. By varying different parameters and congestion control mechanism we will check its effect on Throughput

Introduction

Now-a-days the world is becoming smaller and various ways of communication are being used to avail the facilities for people. Internet is one of the widely used techniques to serve different purposes like data transfer, for entertainment, for education purpose, for paying bills online, for shopping, to be aware of the new researches and innovations, etc. The Internet is expanding rapidly, major contribution in this expansion is of the global acceptance of the TCP/IP protocol stack and use of wireless links, particularly in case of remote areas.TCP has three control mechanisms: Flow control, Error control, Congestion control. Flow control defines the amount of data source can send before receiving an acknowledgment from the destination. Flow control mechanisms accomplish by sliding window protocol. For error control TCP uses three simple tools: checksum, acknowledgment, and time-out. TCP's

Transmission capacity is governed by congestion window, CWND is reduced to 1 from its current value, which also reduces the flow of packets. Thus it controls congestion in network. Here we will discuss about congestion control mechanisms. There are four algorithms used in TCP: Slow start, Congestion Avoidance, Fast retransmit and Fast Recovery. Slow Start, a requirement for TCP software implementations is a mechanism used by the sender to control the transmission rate. Thus the rate of acknowledgements (ACKs) returned by the receiver determine the rate at which the sender can transmit data. 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. If this happens, Congestion Avoidance is used to slow the transmission rate.

When three or more duplicate ACKs are received, the sender does not even wait for a Retransmission timer to expire before retransmitting the segment (as indicated by the position of the duplicate ACK in the byte stream). This process is called the Fast Retransmit algorithm.

After retransmitting the segment, Sender knows that still data is flowing to the receiver. The reason is because duplicate ACKs can only be generated when a segment is received. So instead of reducing the flow of data abruptly by going all the way into Slow Start, the sender only enters Congestion Avoidance mode. The fast recovery algorithm then governs the transmission of new data until non-duplicate ACKs arrive.

TCP Variants

TCP TAHOE

Tahoe refers to the TCP congestion control algorithm which was suggested by Van Jacobson. TCP is based on a principle of conservation of packets, i.e. if the connection is running at the available bandwidth capacity then a packet is not injected into the network unless a packet is taken out as well. TCP implements this principle by using the acknowledgements to clock outgoing packets because an acknowledgement means that a packet was taken off the wire by the receiver. It also maintains a congestion window CWND to reflect the network capacity. Tahoe suggests that whenever a TCP connection starts or re-starts after a packet loss it should go through a procedure called slow-start. The reason for this procedure is that an initial burst might overwhelm the network and the connection might never get started. The congestion window size is multiplicatively increased that is it becomes double for each transmission until it encounters congestion. Slow start suggests that the sender set the congestion window to 1 and then for each ACK received it increase the CWD by 1. So in the first round trip time (RTT) we send 1 packet, in the second we send 2 and in the third we send 4. Thus we increase exponentially until we lose a packet which is a sign of congestion. When we encounter congestion we decreases our sending rate and we reduce congestion window to one. And start over again. The important thing is that Tahoe detects packet losses by timeouts. Sender is notified that congestion has occurred based on the packet loss.

PROBLEM WITH TAHOE

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

TCP RENO

This RENO retains the basic principle of Tahoe, such as slow starts and the coarse grain retransmit 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 we receive a duplicate acknowledgment, then his duplicate acknowledgment could have been received if the next segment in 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 have 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.

Volume : 2 | Issue : 4 | Apr 2013 • ISSN No 2277 - 8179

So Reno suggests Fast Re-Transmit. 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. Another modification that RENO makes is in that after a packet loss, it does not reduce the congestion window to 1. Since this empties the pipe. It enters into an algorithm which we call Fast-Recovery Transmit. 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. Another modification that RENO makes is in that after a packet loss, it does not reduce the congestion window to 1. Since this empties the pipe. It enters into an algorithm which we call Fast-Recovery.

PROBLEMS WITH RENO

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 its performance is almost the same as Tahoe under conditions of high packet loss. The reason is that it can only detect a single packet loss. If there is multiple packet drops 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 CWND is reduced twice for packet losses which occurred in one window. Another problem is that if the widow is very small when the loss occurs then we would never receive enough duplicate acknowledgements for a fast retransmit and we would have to wait for a coarse grained timeout. Thus is cannot effectively detect multiple packet losses.

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 fast recovery is acknowledged. Thus it overcomes the problem faced by Reno of reducing the CWND multiples times. The fast-transmit phase is the same as in Reno. The difference in the fast recovery phase which allows for multiple re-transmissions in new-Reno. Whenever new-Reno enters fast recovery 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 fast recovery then it exits fast recovery and sets CWND 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 segment and sets the number of duplicate ACKS received to zero. It exits Fast recovery when all the data in the window is acknowledged.

PROBLEM WITH NEW RENO

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

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 retransmits parts of RENO. It also has the coarse grained timeout of Tahoe to fall back on, in case 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 retransmits a segment it increments it by 1.

Whenever the pipe goes smaller than the CWND it checks which segments are un received and send them. If there are no such segments outstanding then it sends a new packet. Thus more than one lost segment can be sent in one RTT.

PROBLEM WITH SACK

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

RESULT AND ANALYSIS BASED ON THROUGHPUT

We have evaluated the performance of different variants of TCP using NS-2 simulator. We have simulated the performance of these variants for different parameters. Here we have taken three parameters Error rate(ER), Packet size and window size. The topology created by NS-2 is as shown in Figure-1. It contains one sender, router and receiver. Node 1 sends data to node 3 through node 2. There is direct link between node 1 and node 3. We are using duplex link and Drop Tail type queue. With tail drop, when the queue is filled to its maximum capacity, the newly arriving packets are dropped until the queue has enough room to accept incoming traffic. Analysis is done on Throughput, how can we get smooth graph of it using Tool command language created by John Ousterhout.

Here we have kept Bandwidth 1Mb and Delay is 10ms, and simulation time 5sec for this small scale we are getting Throughput at 0.9Mbps. So for further large bandwidth and varying delay we get good output.

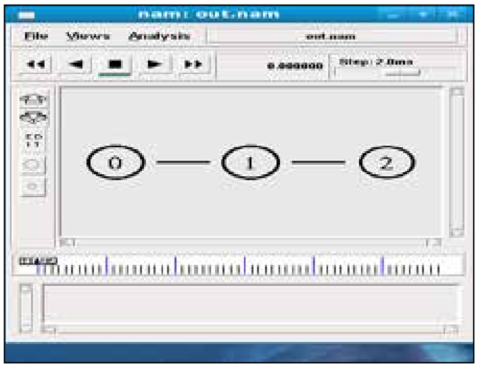

Fig.1 Simulation Topology

Comparison of TCP variants based on Error rate

We can see that from result, at Error rate 0.01 Tahoe drops multiple packets thus throughput degrades at that time. After dropping packets Tahoe enters into slow start so speed reduces.

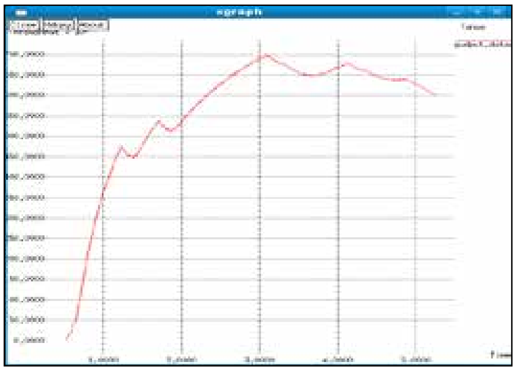

Fig.2 Performance of TCP Tahoe at ER 0.01

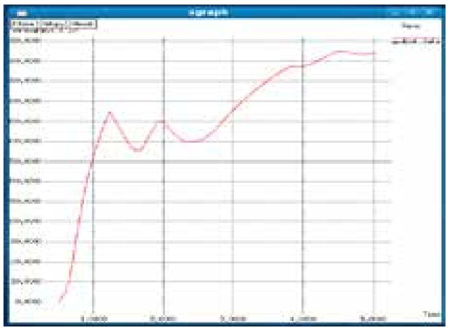

Fig.3 Performance of TCP Reno at ER 0.01

Same way we can see from graph of Reno that in case of drop of multiple packets Reno enters into Fast retransmit and set CWND at half. So throughput becomes smooth as compared to Tahoe.

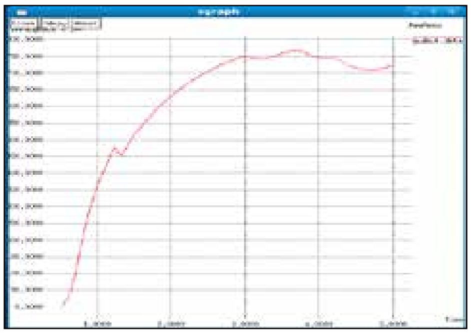

Fig.4 Performance of TCP Newreno with ER 0.01

In case of NewReno same mechanism adopted like Reno but with Fast recovery so after dropped of multiple packets time taken by it less to transmit new packet.

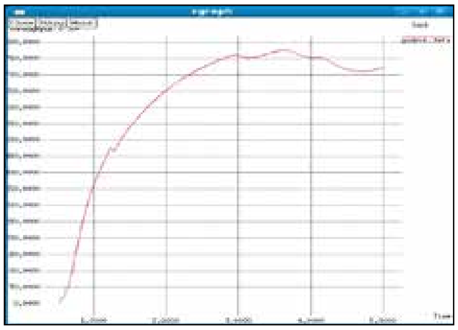

Fig.5 Performance of Sack with ER 0.01

From Figure 5 we can see that Sack only retransmit Selective ACK packet so it saves the time and avoid unnecessary retransmission. TCP Sack emerges as the better option.

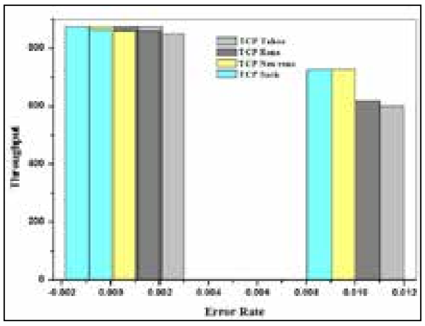

Fig.6 Throughput Vs. Error rate

Comparison of TCP Variants based on Packet Size

In the second case, we have considered the performance of the variants in case of Packet Size. Here Throughput varies according to Size of Packet. At very less packet size throughput of TCP Tahoe is very low but others are at similar level.

Here we are getting variable throughput up to1500 packets after that the Throughput is not changing due to small scale in our project.

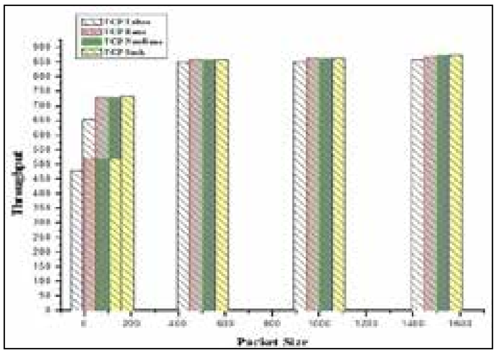

Fig.7 Throughput Vs. Packet Size

If we take graph from NAM editor we can find that the graph of Sack is smooth compared to others. By increasing packet size Throughput is also increase but the response of TCP Sack is very good rather than other variants.

Comparison of TCP Variants based on Window Size

In this case we have considered the effect of Window size on Throughput of TCP variants. Generally window size is kept 20. Here we have taken window size from 5, 10, 15, 50 and 100. After 100 the throughput is not changing.

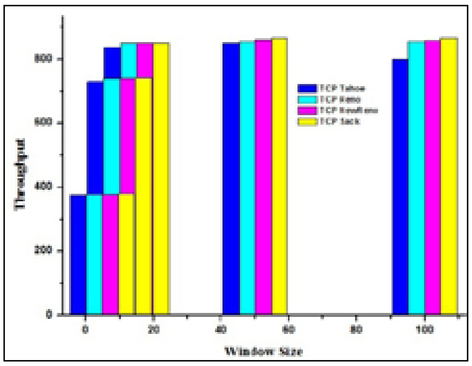

Fig.8 Throughput Vs. Window Size

From figure 8, we can see that at 50 window size the throughput is good especially TCP Sack. Same analysis can be made as we have done in case of Packet Size. The Throughput graph of TCP Sack is very smooth compared to Newreno.

CONCLUSION AND FUTURE PLAN

In this paper, we have analyzed different TCP variants by varying different Parameters. After analyzing the performance from simulated data and graphs obtained, we found that TCP Sack is better than any other TCP variants. The main drawback of TCP Sack is that currently selective acknowledgements are not provided by the receiver to implement SACK otherwise it the better solution. So our Future plan is to study and develop algorithm for receiver side performance.

This project can be expanding by considering other new TCP variants like HS-TCP, TCP Hybla, and TCP Westwood etc.

ACKNOWLEDGMENTThe authors would like to thank Mr. Rahul Mehta, Mr. Sagar Sodhatar and Mr. Sanjay Chhelavda for their contribution to the programming and guidance.

Volume : 2 | Issue : 4 | Apr 2013 • ISSN No 2277 - 8179

REFERENCE

[1] RFC 793- Transmission Control Protocol [2] RFC 2018 TCP Selective Acknowledgment [3] RFC 5681 TCP Congestion Control [4] Kevin Fall and Sally Floyd "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP" Lawrence Berkeley National Laboratory One Cyclotron Road, Berkeley, CA 94720 [5] S. Floyd. "Issues of TCP with SACK," Technical report, Mar. 1996. [6] Yuvaraju B N, Dr. Niranjan N Chiplunkar "Scenario Based Performance Analysis of Variants of TCP Using NS2 – Simulator" IJCA, August-2010. [7] Suhas Waghmare, Aditya Parab, Pankaj Nikose, Prof. S. J. Bhosale "Comparative Analysis of different TCP variants in a wireless Environment" 978-1-4244-8679-3/11/$26.00 ©2011 IEEE [8] Shagufta Henna "A Throughput Analysis of TCP Variants in MobileWireless Networks" 978-0- 7695-3786-3/09 $26.00 © 2009 IEEE. [9] JACOBSON, V. Congestion avoidance and control. In Proceedings of SIGCOMM '88 (Stanford, CA, Aug 1988), ACM. [10] "The ns Manual", The VINT Project, A Collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARz.