Modeling TCP NewReno Slow Start and Congestion- Avoidance using Simulation Approach

Saleem-ullah Lar†, Xiaofeng Liao†† and Songtao Guo†††

†College of Computer Science , Chongqing University, Chongqing, China
††Faculty of Computer Science and Engineering, Chongqing University, Chongqing, China
†††Faculty of Computer Science and Engineering, Chongqing University, Chongqing, China

Abstract
TCP steady-state Performance is affected by the congestion in the network and to select an appropriate data for the available capacity (bottle-neck link) is an open issue. This congestion is mainly arises when a large amount of flow (FTP transaction) is to be sent. The aim of this paper is to solve these issues up to the maximum level using simulation approach in already defined algorithms. Our measurement is able to predict more accurately TCP send rate to avoid packet loss and increasing throughput and saw-tooth effect in congestion window while comparing with TCP New Reno. The simulation result shows that the proposed schema can achieve higher throughput and lower delay, full and always link utilization with minor packet loss which will be controlled by using some TCP New Reno already defined mechanism and modifying slow start and congestion avoidance algorithms.

Key words:
Congestion Control, Slow Start, Congestion Avoidance, Multiple Packet loss, Throughput

Introduction

Today Internet traffic is mostly carried out by transmission control protocol (TCP). TCP in co-relation with UDP is the core of current internet transport layer. Recent studies show several efforts in the direction of congestion control and to select the appropriate send rate for TCP bulk transactions. Modern TCP implementations such as [1], [2], [3] and [4] also suggest different congestion control schemas by using congestion window *(cwnd)* and delay techniques. However, the goal is to increase the TCP *cwnd* size as maximum as possible without packet loss and select the appropriate send rate before the congestion to be occur.
Several variant of TCP are deployed [5], it has been recognized that standard TCP New Reno throughput deteriorates in high-speed networks with large bandwidth-delay product (BDP) [6], and new congestion control algorithms have been proposed to address such deterioration. Assuming that the proposed congestion control schemes would be in general used in the Internet, it is imperative to study the interaction among flows of different schemes, in addition to the interaction among flows using the same congestion control. Thus, we propose a new mechanism that can assess interaction among different congestion control proposals. By applying the same experiment setup, including network configuration, flow parameters, and workload of each flow, to multiple experiment runs for the different schemes, we can assess flow-by-flow behavior of different schemes. In addition to such realistic simulation environment that creates networks with multiple bottlenecks, a large number of short-lived and long-lived flows, and a variety of RTTs.
The Internet users' demands for network quality has increased due to services becoming progressively diversified and sophisticated because of the remarkable degree to which the Internet has grown, which is due in part to access and backbone network technologies. Applications involving real-time media delivery services, such as VoIP, video streaming and TV meeting systems, all of which have experienced a dramatic level of development which require large and stable amounts of network resources in order to maintain the Quality of Service (QoS). For example; the quality of real-time streaming delivery applications is highly dependent on propagation delay and delay jitter. The available bandwidth on the end-to-end network path is also an important factor in order to smoothly provide rich contents, including voice and video. There are a number of network-layer technologies, such as *IntServ and DiffServ* [7], that provide such high-quality network services over the Internet. However, implementation of *IntServ* or *DiffServ* architectures would require additional mechanisms to be deployed to all routers through which traffic flows traverse in order to sufficiently benefit from the introduction of *IntServ* or *DiffServ* into the network. Therefore, due to factors such as scalability and cost, we believe that these schemes have almost no chance of being deployed on large-scale networks. On the other hand, a number of video streaming applications use UDP as a transport-layer protocol, and UDP controls the data transmission rate according to the network condition. However, these mechanisms have a large cost when modifying the application program for achieving application-specific QoS requirements, and the parameter settings are very sensitive to various network factors. Furthermore, when

Manuscript received January 5, 2011
Manuscript revised January 20, 2011

such applications co-exist in the network and share the network bottleneck resources, we cannot estimate the performance of the network or that of the applications, because the control mechanisms of such applications are designed and implemented independently, without considering the effect of interactions with other applications. Since TCP controls the data transmission rate according to the network condition

Related Work

Although RFC 2581[8] and its associated algorithms have been doing an excellent job in ensuring top performance in lieu of congestion on TCP/IP networks, there is still a lot of work going into enhancing TCP performance and responsiveness to congestion. During the 1990's researchers such as Sally Floyd, Van Jacobson, Mark Allman, W. Richard Stevens, Jamshid Mahdavi and a host of others start producing a massive amount of research and experiments with TCP and related congestion control ideas. The wealth of information in this area is really phenomenal and it is hard to pick out some of the best ideas to present in this paper. Nevertheless, this section is an attempt to provide an overview of some of those popular ideas over the last decade. TCP and congestion control on the Internet is an area that is still actively being researched. For more information, consult the references noted in this paper.
The standard TCP congestion control algorithm which we refer to as TCP Reno was developed in 1988. Reference [9], [10], [11], [12], [13], and [14] papers explain several enhancements in TCP Reno. Few modifications addressing the conservative approach of TCP to update its congestion window under congestion condition are:

i. Loss-based TCP congestion control: HSTCP [15], BIC-TCP [16], STCP [17], CUBIC-TCP [18], HTCP [19].
ii. Delay-based congestion control: TCP-Vegas [20], Fast-TCP [21], TCP-LP [22].
iii. Mixed loss-delay based TCP congestion Control: Compound TCP [23], TCP Africa [24].
iv. Explicit congestion Notification: XCP [25].

Most of these protocols deal with modifying the window growth function of TCP in a more scalable fashion. Habibullah Jamal in [26] proposed a TCP-friendly congestion control that realizes efficient data transmission in high-speed networks, fairness with TCP Reno and fair bandwidth allocation among flows with different RTTs. A scheme that determines the size of congestion window each time a new acknowledgment is received instead of employing slow start/congestion avoidance approach is proposed in [12].

NewReno Congestion Control Mechanisms

In TCP Reno, the window size is cyclically changed in a typical situation. The window size continues to be increased until packet loss occurs. TCP Reno has two phases in increasing its window size: slow start phase and congestion avoidance phase. When an ACK (acknowledgment) packet is received by TCP at the sender side at time *t + t'* [sec], the current window size *cwnd*(*t* + *t'*) is updated from *cwnd(t*) as follows

Slow Start Phase

$$\textit{if } cwnd(t) < ssthresh(t)$$
$$cwnd(t+t') = cwnd(t) + 1;$$

Congestion Avoidance Phase

$$\textit{if } cwnd(t) > ssthresh(t)$$
$$cwnd(t+t') = cwnd(t) + 1/cwnd(t)$$

Where, *ssthresh*(*t*) is a threshold value at which TCP changes its phase from slow start phase to congestion avoidance phase. When packet loss is detected by retransmission timeout expiration, *cwnd(t)* and *ssthresh*(*t*) are updated as,

$$cwnd(t) = 1$$
$$ssthresh(t) = cwnd(t)/2;$$

On the other hand, when TCP detects packet loss by a fast retransmit algorithm, it changes *cwnd*(*t*) and *ssthresh*(*t*) as;

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

TCP Reno then enters a fast recovery phase if the packet loss is found by the fast retransmit algorithm. In this phase, the window size is increased by one packet when a duplicate ACK packet is received. On the other hand, *cwnd*(*t*) is restored to *ssth*(*t*) when the non-duplicate ACK packet corresponding to the retransmitted packet is received.
We have seen that TCP connection starts up in slow start mode and exponentially increase the congestion window *(cwnd_)* until it cross the slow start threshold *(ssthresh_)*. Once *cwnd_* is greater than *ssthresh_*, TCP moves to congestion avoidance phase. In this mode the primary objective is to maintain high throughput without causing congestion. If TCP detects any segment loss, this is a strong indication that network is congested, so as a corrective action TCP reduces its data flow rate by

reducing *cwnd_* after that it again goes back to slow start phase.
Having some deeper look on working mechanism, we choose *ssthresh_ = 65535 bytes* and *cwnd_ = 512 bytes (1 segment)*.
Since *cwnd_* < *ssthresh_* TCP state is slow start. TCP *cwnd_* grows from 512 bytes to 64947 bytes also it has been assumed that no segment loss during slow start phase. During slow start *cwnd_* is incremented by 1 segment on every successful acknowledgment from the receiver. When *cwnd_=65459 + 512 > 65535,* at this point TCP changes its state to congestion avoidance phase, and the primary objective in this phase is to get the higher throughput. If during this phase any three duplicate ACKS received the TCP moves to the fast retransmit without waiting for the retransmit timer out and then after it moves to the fast recovery phase.

TCP VEGAS

As described in the previous subsection, in TCP Reno (and the older version, TCP Tahoe), the window size continues to be increased until packet loss occurs due to congestion. When the window size is throttled because of packet loss, the throughput of the connection would be degraded. It cannot be avoided because of an essential nature of the congestion control mechanism adopted in TCP Reno; it can detect network congestion *only* by packet loss. However, throttling the window size is not adequate when the TCP connection itself causes the congestion because of its too large window size. If the window size is appropriately controlled such that the packet loss does not occur in the network, the throughput degradation due to the throttled window can be avoided. This is a key idea of TCP Vegas.
TCP Vegas controls its window size by observing RTTs (round-trip times) of packets that the sender host has sent before. If observed RTTs become large, TCP Vegas recognizes that the network begins to be congested, and throttles the window size. If RTTs become small, on the other hand, the sender host of TCP Vegas determines that the network is relieved from the congestion, and increases the window size again. Hence, the window size in an ideal situation is expected to be converged to an appropriate value. More specifically, in congestion avoidance phase, the window size is updated as

$$\textit{if}\ \ (diff < \alpha / base\_rtt) \tag{1}$$

$$cwnd(t+t') = cwnd(t)+1$$

$$\textit{If}\ \ (\alpha / base\_rtt <= diff <= \beta / base\_rtt) \tag{2}$$

$$cwnd(t+t') = cwnd(t)$$

$$\textit{if}\ \ (diff > base\_rtt) \tag{3}$$

$$cwnd(t+t') = cwnd(t)+1$$

Where,

$$diff = (cwnd(t)/base\_rtt) - cwnd(t)/rtt(obs) \tag{4}$$

where *rtt(obs)* is an observed round trip time, *base_rtt* is the smallest value of observed RTTs, and α and β are some constant.
TCP Vegas has another feature in its congestion control algorithm: a slow-start mechanism. The rate of increasing its window size in slow start phase is one half of that in TCP Tahoe and TCP Reno. Namely, the window size is incremented every other time an ACK packet is received. Note that the congestion control mechanism used by TCP Vegas (equation 4) indicates that if observed RTTs of the packets are identical, the window size remains unchanged.
Suppose that *ACK* is received at source at time (T_a) by denoting that R_a bytes received at the receiver, we can measure the following sample bandwidth by using $B_k=R_a/\Delta_a$, where $\Delta_a=(T_a - 1) - 1$ and $T_a - 1$ is the time of previous ACK received. The discrete time filter is used which is obtained by low-pass filter using Tustin-approximation [27].
$B'_k = a_k B'_k\text{-}1 + (1\text{-}a_k)(B_k+B_k\text{-}1)/2$
Where B'_k is the filtered estimation of the available bandwidth at time $T=T_a$, $a_k=(2\tau-\Delta_a)/(2\tau+\Delta_a)$, where $1/\tau$ is the cut-off frequency filter.
The pseudo code of the algorithm for n-duplicate acknowledgment is given as.

Pseudo Code for n-duplicate Ack

```
If (n dup_ack received)
        ssthresh= (BWE*RTT)/MSS;
  if (cwin>ssthresh)
        cwin=ssthresh;
  end if
end if
```

Here MSS denotes the maximum segment size in bits, which is length of TCP Payload, BWE stands for bandwidth estimation.

Similarly the pseudo code for ACK time out expiry is given as.

Pseudo Code for TIME-OUT Expiry

```
if (timeout_expires)
        ssthresh= (BWE*RTT)/MSS;
   if (ssthresh<2)
```

```
        ssthresh=2;
    end if
    cwin=1;
end if
```

TCP Vegas can achieve over 40% higher throughput than TCP Reno. However, it is not clear whether TCP Vegas works well with TCP Reno or not. Our contribution in this paper is that we compare throughput performances of two versions where those share the bottleneck link, in order to discuss the possibility on the deployment of TCP Vegas in the future Internet.

Proposed Mechanism

Modification in Slow Start and congestion avoidance working Mechanism

It is possible that no packet loss even if congestion window is greater than threshold, keeping this point throughput of the system can increase and will be showed in the simulation section. The reason is that it is possible that network is not congested even if the congestion window is greater than slow start threshold, so in this case we have to check the packet loss along with this condition before going to congestion avoidance phase. The following condition is to check the attributes and continue exponential increase congestion window by checking in a more appropriate way to get throughput as maximum as possible. The pseudo code of the algorithm is given as.

Algorithm 1: Appropriate Selection of Congestion Window

```
void TcpAgent::opencwnd()
{
If ((cwnd_ ≤ ssthresh_) || (cwnd_ > ssthresh_ && dupack_ !=3)) {

//Slow start phase (Exponential increase of congestion window
        cwnd_ = cwnd_ + MSS;
} else
        {
        ssthresh_= cwnd_ – MSS;
        cwnd_= ssthresh_ / 2;
        }
//MSS (Maximum segment size=512 bytes)
```

Improving Delay & Throughput Attributes

According to TCP New-Reno fast retransmit process the three duplicate asks has to be required to retransmit the loss packets, which seems to be inconvenient and it has to be observe that 85% of the packets loss after two duplicate asks due to congestion or transmission error. So, why not we use double acknowledgments to denote packet loss instead of three? Therefore, in our simulation we modify the existing behavior from three duplicate acknowledgments to two duplicate acknowledgments and the result shows better performance in terms of delay and it reduced to 27%. As the delay decreases system throughput automatically increases in terms of packets delivered/seconds. The modification of duplicate acknowledgments is done by already defined TCP parameter i.e. ***tcprexmtthresh_ 2.***

Send Bursty Traffic

To Send Bursty data in a more appropriate way to reduce the packet loss as well, the propose mechanism can be also be used for this purpose.

Algorithm 2: Sent Huge Traffic in an Intelligent Way

```
if (valid_ack || aggressive_maxburst_)
    if (dupacks_ == 0)
        send_much(0, 0, maxburst_);
    else if (dupacks_ > numdupacks_ - 1 &&
            newreno_changes_ == 0)
        send_much (0, 0, 2);
if (cwnd_<=ssthresh_ && dupacks_!=3)
        send_much (0,0,maxburst_);
    else if (cwnd_>ssthresh_ && newreno_changes_== 0)
        send_much (0,0,2);
end if
end if
end if
```

Experimental Setup

Our proposed model focuses on two basic algorithms, slow start congestion avoidance and a little bit modification in default behavior of packet loss indication as discussed earlier. We uses following topology in our NS-2 simulation.

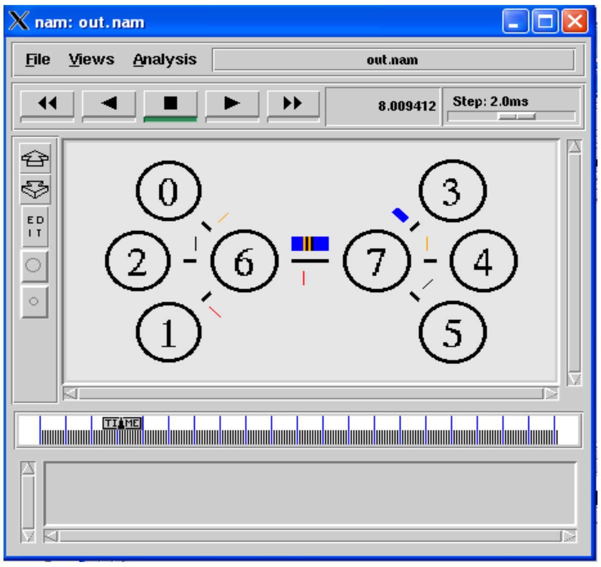

Fig. 1. Simulation Model (Dumb bell)

Other major design parameters defined in the simulation are as follows.

(i) Packet Size was defined 1500bytes
(ii) Rate defined for FTP is 8000kb and for UDP is 1Mb.
(iii) Minimum and maximum Simulation time goes from 50 seconds to 1 hour respectively.
(iv) To saturate the bottleneck link, the size of congestion window was large enough to 83000 packets.
(v) Both routers of the topology uses Drop Tail buffers and FIFO queuing.
(vi) All tests are repeated at least 3 times.
(vii) RTT in the range of 10ms ~ 300ms.
(viii) Bandwidth of 2Mbps and 5Mbps.
(ix) Queuing Size of 20% ~ 100% of BDP.

The performance metrics of the topology are effect of congestion window, throughput, and packet loss.

Simulation Results & Performance Analysis

Packet Loss:

Packet loss can be caused by a number of factors, including signal degradation over the network medium due to multi-path fading, packet drop because of channel congestion, corrupted packets rejected in-transit, faulty networking hardware, faulty network drivers or normal routing routines.
In addition to this, packet loss probability is also affected by signal-to-noise ratio and distance between the transmitter and receiver.
We have checked the Packet loss results both for multi (Bursty data) file and mono-file cases.

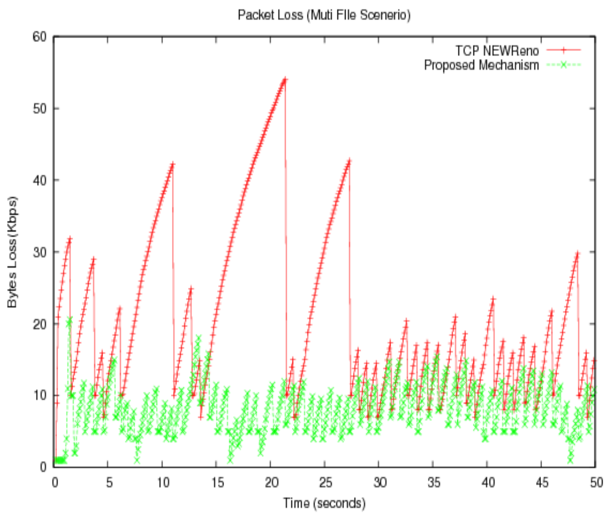

Fig. 2. Packet loss in a Multi files Scenario

Figure 2 shows that, while sending bulk of files at the same time TCP NewReno (red lines) loss the packets in a greater ratio then the proposed mechanism (green lines), in this case the congestion window is bigger than the proposed but a decrease in congestion window is also clearly observed which may also lead to the higher delay and lowering the overall throughput of the system. But the saw-tooth effect of proposed scheme remain same almost 91% as compared to the NewReno haphazard and randomized effect. While in a mono-file scenarios the difference is not too high, but today due to the rapid increase and merging of latest technologies with the internet cause to send huge amount of data simultaneously from different geographical regions. Therefore the proposed mechanism shows the better performance than the NewReno. Packet loss ratio can also be calculated by the following formula

Packet loss %age =NLP / (NLP + NPRS)
Where,
NLP=Number of Lost Packets.
NPRS=Number of packet received successfully.
When packet loss caused by network problems, lost or dropped packets can result in highly noticeable performance issues or jitter with streaming technologies, voice over IP, online gaming and videoconferencing, and will affect all other network applications to a degree. However, it is important to note that packet loss does not always indicate a problem. If the latency and the packet loss at the destination hop are acceptable then the hops prior to that one don't matter.

Throughput:

Our proposed mechanism is 100% suitable while transferring mono file (UDP or FTP), the result in figure 3 clearly shows the difference between the two. If we observe carefully the condition in algorithm 1, which is "*If ((cwnd_ ≤ ssthresh_) || (cwnd_ > ssthresh_ && dupack_ !=3))"*. By tracing and observing it has been found that this statement plays a very significant role by carefully and correctly selecting the congestion window. The parameters *dupack_!=3* is an addition in the check, which also co-relate with the *tcprexmtthresh_ 2* a global parameter which changes the default behavior of TCP from three duplicate acknowledgments to two.

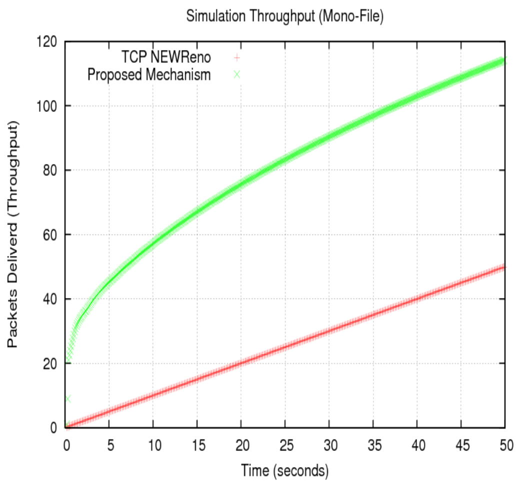

Fig. 3. System Throughput

Effect of Congestion Window:

In TCP, the congestion window is one of the factors that determine the number of bytes that can be outstanding at any time. Maintained on the sender, this is a means of stopping the link between two places from getting overloaded with too much traffic. The size of this window is calculated by estimating how much congestion there is between the two places. The sender maintains the congestion window. When a connection is set up, the congestion window is set to the maximum segment size (MSS) allowed on that connection. Further variance in the collision window is dictated by an Additive Increase/Multiplicative Decrease approach [28]. This means that if all segments are received and the acknowledgments reach the sender on time, some constant is added to the window size. The window keeps growing linearly until a timeout occurs or the receiver reaches its limit. If a timeout occurs, the window size is halved.

Figure 4 describe the effects of *cwnd*, as the propose mechanism in green lines, while red describe NewReno.

.

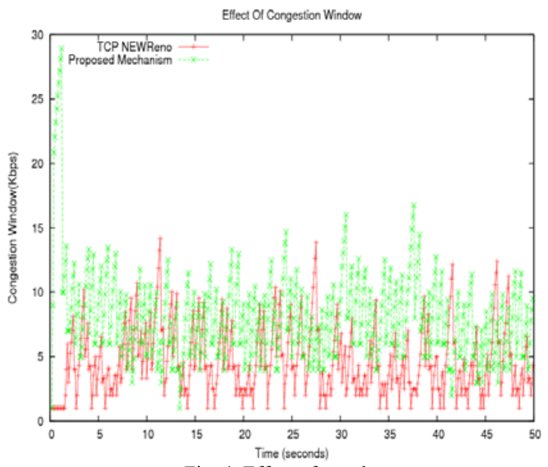

Fig. 4. Effect of cwnd

After separating and expanding the graph, the difference between the two is clearly observed. Our proposed mechanism (figure 5) outperforms the NewReno (figure 6).

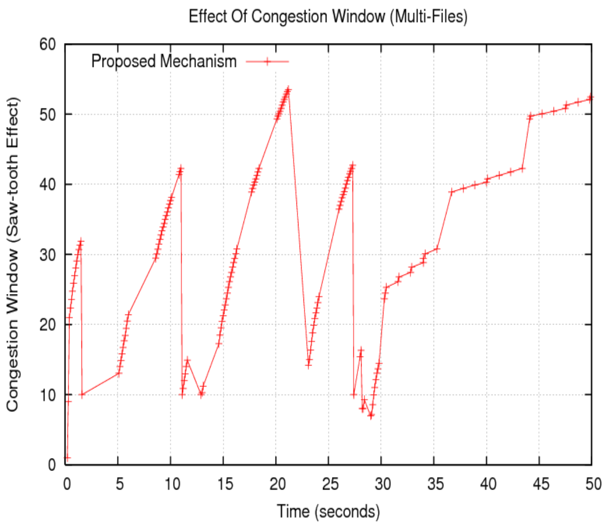

Fig. 5 Effect of cwnd (Proposed Mechanism)

We have tested this idea with the same network to compare the performance of the connections with the modified TCP NewReno. It is easy to see from figure 5 that the saw-tooth effect is more evenly distributed than in the figure 6.

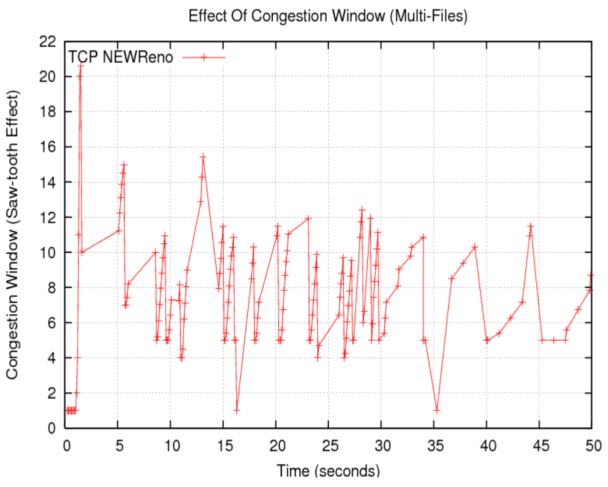

Fig. 6 Effect of cwnd (New Reno)

Conclusion and Future Work

Most new TCP variants showed good link utilization however had still unfairness and aggressiveness problems over standard TCP Reno. When we using multiple flows of a new TCP variant with different RTTs, queuing delay can be an issue as much as a bandwidth aspect. Some new TCP variants seemed to be sensitive to network conditions. Therefore a careful modification in the algorithms leads to better outcomes.

Through results presented in this paper, actual analysis and observations of packet loss and throughput in multi file cases shows a clear difference while comparing results with already defined TCP Reno algorithms. The effect of congestion window mostly remains double then the previous, which shows that values of congestion window are chosen intelligently in the algorithm.

It has also been seen that mostly routers and other network devices are using algorithms whose complexity is very high. So if this approach is selected for the routers as well, a better performance will be achieved in the intranet.

In future both analytical (Engineering) and simulation approaches will be used to enhance this approach. This research is actually an initial stage of my study and in future I will try my maximum efforts to expand the idea and results at a global level.

Acknowledgment

The author would like to thank the anonymous reviewers and the editors for insightful comments and suggestions.

References

[1] T.V. Lakshman and Upamnayu Madhow "The Performance of TOP/IP for Networks with high BDP and Random Loss" IEEE/ACM Transactions. Volume (05)-03, June 1997.

[2] Jitendra Padhye, VictorFiroiu, "Modeling TCP Reno Performance A simple model & its Empirical Validation" IEEE/ACM Transactions Vol.ume(8)-02 April 2000.

[3] Jim Martin, Arne Nilson and Injong Rhee "Delay Based Congestion Avoidance for TOP" IEEE/ACM Transactions. Volume (11)-03, June 2003.

[4] Celio Albuquerque, Brett J Vickers "Preventing Congestion Collapse and promoting fairness in Internet"IEEE/ACM Transactions, Volume (12)-01, Feb 2004.

[5] en.wikipedia.org/wiki/Transmission_Control_Protocol.

[6] Dina Kitabi, Mark Handley and Charlie Rohrs "Congestion Control for High BDP Networks"

[7] Matt Lepinski and Peter Portante "DiffServ and IntServ" Special Notes, December 2001.

[8] www.ietf.org/rfc/rfc2581.txt

[9] Mohamed Tekala and Robert Szabo "Modeling Scalable TCP Friendliness to NewReno TCP"IJCSNS-Volume (07)-03, PP: 89—96.

[10] R. Wang, K. Yamada, M.Y.Sanadidi and M. Gerla, "TCP with sender side intelligence to handle Dynamic, Large, Leaky Pipes", IEEE Journal on Selected Areas in Communications, Volume(23)-02, Feb 2005

[11] Dongmin Kim, Beomjoon Kim, Jechan Han and Jaiyong Lee "Enhancement to the Fast Recovery Algorithm of TCP NewReno,ICOIN 2004, LNCS 3090, pp 332-341."

[12] Andrea De Vendictis, Andrea Baiocchi, and Michela Bonacci, "Analysis and enhancement of TCP Vegas congestion control in a mixed TCP Vegas and TCP Reno network scenario", ACM Volume(53)03-04, Aug 2003.

[13] Xiao Lu, Ke Zhang, Cheng Peng Fu, and Chuan Heng Foh, "A Sender Side TCP Enhancement for Start-up Performance in High Speed Long Delay Networks", IEEE Communication Society, WCNC 2010 Proceedings.

[14] Sourabh Ladha, Paul D. Amer, Armando Caro Jr., Janardhan R lyengar, "On the prevalence and Evaluation of Recent TCP Enhancements", web1.fandm.edu/ jiyengar/papers/tbit+-globecom2004.pdf

[15] Sally Floyd, Sylvia Ratnassamy and Scott Shenker, "Modified TCP's Congestion Control for High Speeds" May 05, 2002.

[16] L. XU, K. Harfoush and I. Rhee, "Binary Increase Congestion Control for Fast Long Distance Network" In Proceeding of the IEEE INFOCOM March 2004.

[17] Tom Kelly, "Scalable TCP Improving Performance in High Speed WAN" http://www.lce.eng.cam.ac.uk/~ ctk21/scalable/.

[18] Injong Rhee and Lisong Xu, "Cubic: A new Friendly High Speed TCP Variant"

[19] en.wikipedia.org/wiki/**H-TCP.**

[20] Richard J. La, Jean Walrand, and Venkat Anantharam, "Issues in TOP Vegas", Department of Electrical Engineering, University of California-Berkeley.

[21] C. Jin, D. Wei, S. H. Low, "FAST TCP, From Theory to Eexperiments", http://netlab.caltech.edu/FAST/

[22] Aleksandar Kuzmanovic and Edward W. Knightly, "TCP-LP: A Distributed Algorithm for Low Priority Data Transfer",citeseerx.ist.psu.edu/viewdoc/ download? doi= 10.1.1.13.8070&representante

[23] Alberto Blanc, Konstantin Avrachenkov, Denis Collange and Giovanni Neglia, "Compound TCP with Random Losses", LNCS, 2009, Volume 5550/2009, 482-494.

[24] King, R.; Baraniuk, R.; Riedi, R., "TCP-Africa: an adaptive and fair rapid increase rule for scalable TCP", INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies, Proceedings IEEE

[25] http://icir.org/floyd/ecn.html

[26] Habibullah Jamal and Kiran Sultan, "Performance Analysis of TCP Congestion Control Algorithms", International Journal of Computers and Communications, Volume (02)-01, 2008.

[27] en.wikipedia.org/wiki/**Tustin**

[28] Lin cai, Xuemin Shen, Jianping Pan and Jon W. mark, "Performance Analysis of TCP Friendly Algorithm", IEEE Transactions on Multimedia, Volume(07)-02, Apr 2005.

Saleem-ullah Lar was born on Mar 1983 in AhmedPur East (Pakistan). He received his BSc and MIT degrees in Computer Science from Islamia University Bahawalpur and Bahauddin Zakariya University (Multan) in 2003 and 2005 respectively. He has worked for four years as a Network administrator. His research interests include TCP/IP Performance, Quality of Service and Network Congestion Control.

Xiaofeng Liao was born in Sichuan Province (China) in 1964. He received his B.S. and M.S. degrees in Mathematics from Sichuan University, and the Ph.D degree in circuits and systems from University of Electronic Science and Technology of china in 1997. From 1999-2001 he was doing Postdoctoral research at Chongqing University. At present, he is a professor at Chongqing University and university of Electronic Science and Technology of china. From Nov 1997 to Apr 1998, he was a research associate at China University of Hong Kong. From Oct 1999 to Oct 2000, he was a research associate at City University of Hong Kong. From Mar 2001 to Jun 2001 and Mar 2002 to Jun 2002 he remains senior associate at City of University of Hong Kong. He has published over 150 academic journal and conference papers. Currently, his main interests include neural networks. Nonlinear dynamical systems. Bifurcation and chaos, synchronization and control of chaos, and signal processing.

Songtao Guo was born in Henan province (China) in 1976. He received his BS and MS degrees in computer software and theory from Chongqing University (china) in 1999 and 2003 respectively. His research interests include network congestion control, stability analysis, bifurcation control and its synchronization, and controlling the chaos of nonlinear systems with time delays.