International Journal of Advances inEngineeringResearch http://www.ijaer.com
(IJAER) 2012, Vol. No. 4, Issue No.II,August **ISSN:2231-5152**

COMPARISON STUDY OF STANDARD TCP (AIMD) AND SCALABLE (MIMD)

SarikaGupta ,Dr. Rajesh Pathak
Department of Computer Science,

ABSTRACT

TCP maintains many traffic control parameters to avoid the congestion .Standard TCP mechanism deploy cubic, Westwood and similar high speed variants. The Multiplicative Increase multiplicative Decrease (MIMD) congestion control algorithm in the form of scalable TCP has been proposed for high speed networks. In this paper do comparison in the TCP Traffic control capabilities. TCP', 'Westwood TCP', 'CUBIC TCP'introduced in Linux Kernel 2.6 with Scalable TCP (MIMD)

INTRODUCTION

TCP provides the mechanisms that provide data to be transferred across networks that are dynamic and have a large variety of resources. For instance congestion control keeps the network resources from being overloaded without the need for specified information about network resources. This allows for the network to be very scalable and autonomous which has most likely been the reason for the success of the web. Without transmission control protocol, network resources like core links could easily get congested or underutilized. TCP purposes to keep the utilization of the link as high as possible by slowing down and speeding up each single connection sharing the link[33].

TCP CONGESTION CONTROL ALGORITHMS

Standard TCP

Standard TCP uses congestion control algorithms described in RFC2581[1]. The algorithms used are:
Slow Start
Congestion Avoidance
Fast Retransmit
Fast Recovery.

A TCP connection is always using one of these four algorithms throughout the life of the connection.

Slow start

TCP maintains a guess at the current reasonable window size, called the slow start threshold (or ssthresh). Whenever TCP starts sending after being idle (or timing out) it would like to send with a window of size ssthresh. It turns out to be a bad idea to send the entire window in a burst, which might force a nearby router to buffer the whole window; far better to spread the window over a round-trip time, so that they are stored in transit on the links. TCP accomplishes this using this algorithm, called “slowstart”.

- Initialize the window size, CWND, to onesegment
- Whenever an ACK that acknowledges new data arrives (a “positive” ACK).Increase CWND by onesegment
- If the resulting CWND is less than ssthresh, stay in slow-start. Otherwise,enter congestion avoidancemode

This doubles CWND every round-trip time, so that TCP opens its window tcpsstresh in time proportional to log sstresh instead of all at once.

A typical initial ssthresh, used when a TCP connection is first created, is 64Kilobytes. ssthreshis adjusted after segment loss as describedbelow.
The second event is receiving the duplicate ACKs for same data. Upon receiving three duplicate ACKs, the connection uses fast retransmit algorithm. The last event that can occur during slow start is a timeout. If a timeout occurs, congestion avoidance algorithm is used to adjust congestion window and slow startthreshold[31]

Congestion Avoidance

The data transfer of TCP starts from a *slow start*,in which TCP tries to increase its sending rate exponentially, until it encounters the first loss. It then switches to another stage, called *congestion avoidance,* in which TCP employs the *Additive Increase ,Multiplicative decrease* mechanism to slowly adapt to the available bandwidth. On further congestion, the TCP goes into the *Fast Recovery &Fast Retransmission* stages .In this scenario, when TCP do not receive an acknowledgment for a packet after some timeout period, it assumes that this packet is lost. & then retransmits that packet and doubles its retransmission timeout value(RTO) detecting packet loss. This process continues until the packet is successfully transmitted & acknowledged. TCP tries to clear congestion by cutting its sending rate inhalf.

Out of the many UNIX like kernels, Linux is a matured product and has a significant share in worldwide server population dealing with TCP traffic under all kinds of traffic scenarios. Hence, the TCP implementation in Linux has been tuned enough to meet the requirements of heavy duty applications depending on it.

Additive Increment

After receiving an ACK for new data, congestion window is increment by
(MSS)2/Cwnd, where MSS is maximum segment size, this formula is known as additive increment. The goal of additive increment is to open congestion window by a maximum of one MSS per RTT. Additive increment can be described by using the equation (1):
Cwnd = Cwnd + a*MSS2/Cwnd (1)
where the value of a is a constant, a = 1.

Multiplicative Decrement

Multiplicative decrement occurs after a congestion event, such as a lost packet or a timeout. After a congestion event occurs, the slow start threshold is set to half current congestion window. This update to slow start threshold follows equation(2):

ssthresh = (1 – b)*CWND(2)

CWND is equal to amount of data that has been sent but not yet ACKed and b is a constant, b = 0.5. The congestion window is adjusted accordingly. After a timeout occurs, congestion window is set to one MSS and slow start algorithm is reuse. The fast retransmit and fast

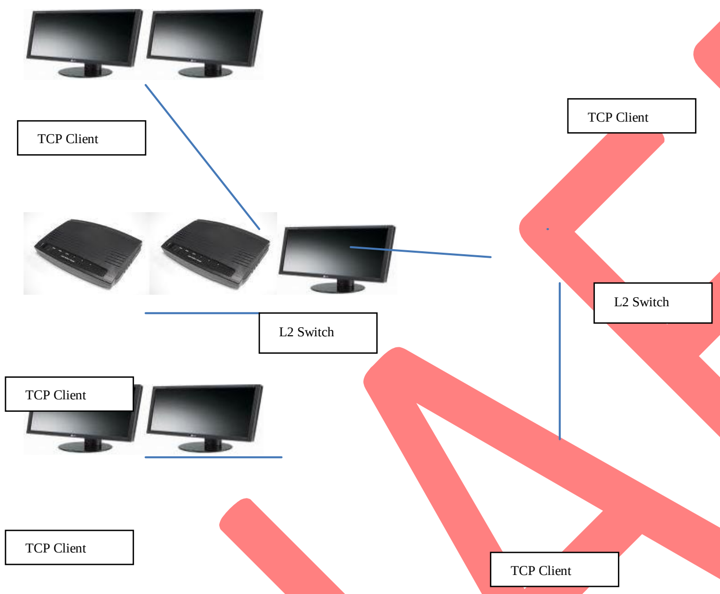
EXPERIMENTAL SETUP

We construct an asymmetric dumbbell sort of topology where two L2 switches are located at the bottleneck between two end points. The end points consist of a set of HP Linux Systems running custom client and server applications dedicated to high-speed TCP variant flows and background traffic. Background traffic is generated by using various web based applications.

We use Linux hosts as communication end points communicating over 100Mbps link with MTU of 1500 bytes. The RTT of each background traffic is random. The socket buffer size of some client machines is fixed to default 64KB while high-speed TCP machines are configured to have a very large buffer so that the transmission rates of high-speed flows are only limited by the congestion control algorithm. Two Layer 2 switches are deployed with four high-speed TCP machines which are tuned to generate or forward high traffic. Each TCP variant has been used individually to analyze the performance aspects.

The custom Java Client and Server programs are used to generate and receive high traffic end to end. The Server TCP suffers from high traffic ingress and has to take corrective and further preventive action evident from the analysis. As the Linux 2.6 TCP has pluggable modules now, we can inject and eject appropriate modules dynamically too.

The analysis has been done for TCP Westwood and TCP CUBIC individually and the results have been compared. The packet sniffer tool „Wireshark" has been used to capture the live TCP traffic and generate logs/reports. A comparative study has also been done for competing TCP modules.



<u>Experimental Linux 2.6 Testbed Layout</u>

This paper is organized as follows. Section II presents different scenarios of TCP Traffic. Section III deals comparative study between these scenarios. Possible future work and concluding remarks are presented in section IV.

DIFFERENT SCENARIOS OF TCP TRAFFIC.

i) Westwood:-TCP Westwood develop two basic concepts: the end to end estimation of the available bandwidth and the use of such estimate to set the slow start threshold and the congestion window.

In TCP Westwood ,the sender continuously computes the connection Bandwidth Estimate (BWE) Which is defined as the share of bottleneck bandwidth used by the connection BWE is equal to the rate at which data is received to the receiver or rate of acknowledgementreceived

After 3 duplicate packet received(packet loss indication) the sender resets the congestion window and the slow start threshold based on BWE *cwin=BWE*RTT*. RTT is also required to compute the window that support the estimated rateBWE
Initially congestion window increments during slow start and congestion avoidance remain the same as in Reno, that is they are exponential and linear, respectively. A packet loss is indicated by (a) the reception of 3 duplicate acknowledgements or (b) a expiry of Round Trip Time. *TCPWestwood set cwin and ssthresh as follows*

```
If (3 DUPACKs are received)
    ssthresh=(BWE *RTTmin)/seg_size
    if(cwin>ssthresh) /* congestion avoidance*/
        cwin=ssthresh
    Endif
Endif
```

In case a packet loss is initiated by a time out expiration, cwin and ssthresh are set as follows:

```
if(Coarse timeout expires)
    cwin=1
ssthresh=(BWE * RTTmin)/seg_size;
    If (sstresh<2)
        ssthresh=2
    endif
endif
```

Analysis OfWestwood:-

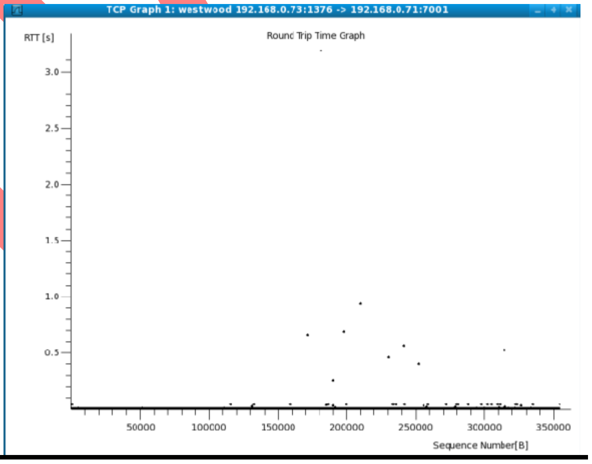


Fig : RTT Graph for TCP Westwood(TCPW)

In the analysis of TCPW, we use Linux hosts as communication end points communicating over 100Mbps link with MTU of 1500 bytes. The RTT of each background traffic is random. The socket buffer size of some client machines is fixed to default 64KB.

As per the graph shown, the minimum RTT was around 0.001 sec and maximum RTT was around 1sec

We can calculate the capacity of the pipe as***capacity (bits) = bandwidth (bits/sec) × round-trip time (sec)***

This is normally called the *bandwidth-delay product*. This value can vary widely, depending on the network speed and the RTT between the two ends.

On similar pattern, TCP Westwood Congestion Control is based on -

(1) congestion window(*cwnd*)

(2) slow start threshold(*ssthresh*)

(3) round trip time of the connection(*RTT*)

(4) minimum round trip time measured by the sender(*RTTmin*).

We compare two high-speed flows with a different RTT i.e. AIMD and Westwood. We observe the RTT of both cases as different and check which one is better RTT fair. As per our analysis around small window sizes, TCPW shows the RTT unfairness. TCPW has window sizes around 200 for 100Mbps. Nonetheless, its RTT unfairness is much better than AIMD where we get a random RTT scenario.

TCPW in Linux can better handle the congestion scenario under excess traffic. The RTT observed is less but the number of segments are also low. This shows the combative state of TCP while the socket receive buffers get continuously overflowed. It shows as if a large number of SYN segments have been dropped either by socket receive buffer or the congestion window sizing.

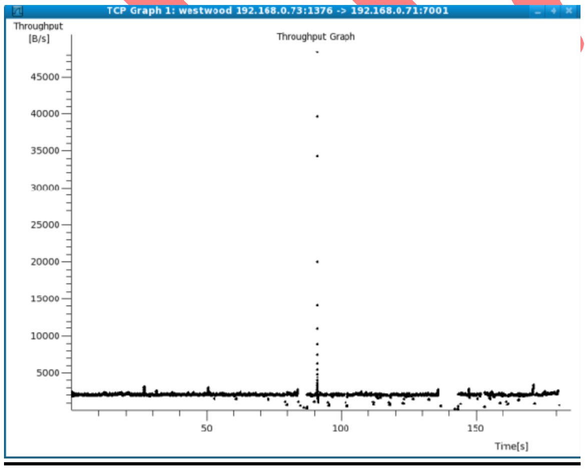


Fig : Throughput Graph for TCP Westwood(TCPW)

In the analysis of TCPW Throughput, we use Linux hosts as communication end points communicating over 100Mbps link with MTU of 1500 bytes. The RTT of each background traffic is random. The socket buffer size of some client machines is fixed to default 64KB.
The random bursts of data attack on the socket receive buffers and TCP enters into congestion avoidance mode.

The graph shows that there is an effort to attain a steady state throughput due to Westwood algorithm. There are apparent traces showing the congestion window to become ¾ of the current congestion window.
The throughput touches the peak of 312.5 kbps with an immediate corrective congestion window size afterwards. The nature of the output traffic is almost steady state as there are no sharp increases like Reno and periodic wedges like vegas and it far better matches the objectives of the congestion control algorithm. The nature is less self similar in the trace received by us till the congestion collapse was finally achieved.
TCP Westwood Congestion Control is based on -
(1) congestion window(*cwnd*)
(2) slow start threshold(*ssthresh*)
(3) round trip time of the connection(*RTT*)
(4) minimum round trip time measured by the sender(*RTTmin*).
The stream of returning ACK packets infer an estimate of connection available bandwidth (*BWE*).
At the point of congestion,
When 3 DUPACKs are received by the sender

```
        :ssthresh = (BWE * RTTmin) / MSS;
        if (ssthresh<2) ssthresh=2;
        cwnd = ssthresh;
```

Here,
RTTmin = 0.001 sec (observed)
MSS = 512 B
BWE = 200 B (on an average)

ssthresh = (200*0.001)/512 < 2
Hence ssthresh = 2
cwnd = 2
This way TCPW in Linux handled a congestion scenario.
On the downside, the throughput does not seem to follow sharp 'additive increase' in congestion window like Reno and not even like pure AIMD+Vegas. This can be less useful in case of high

speed networks. The congestion window has become more sensitive to congestion but less aggressive in following 'additive increase'. A peak is observed but a sharp fall follows due to decrease effect.
The graph clearly shows that the congestion happened after 180 sec and the congestion window cautiously controlled it, finally showing a 'multiplicative decrease'. This performance is far better than all the previously analyzed TCP versions.

As the link bandwidth increases, the measurement-based nature of TCPW allows it to track the bandwidth variations of the bottleneck and to linearly build-up its performance. AIMD also improves its performance in same situation but in a less considerable way.

v) TCP CUBIC:-As name suggest it implement cubic function. CUBIC is designed to simplify and enhance the window control of BIC .

$$\boldsymbol{W_{cubic} = C(t-K)^3 + W_{max}}$$

C = scaling factor

t = elapsed time from the last window reduction.

W_{max} = window size just before the last window reduction.

$K=\sqrt{W_{max}ß/C}$ where ß is a constant multiplicative decrease factor applied towindow reduction at the time of loss event (i.e.the window reduces to $ßW_{max}$ at the time of the last reduction).

In this the window grows very fast upon a window reduction but as it gets closer to W_{max} ,it slows down the growth. Around W_{max} , the window increment becomes zero. Above that, CUBIC starts probing for more bandwidth in which the window grows slowly initially, accelerating its growth as it moves away from W_{max} .

$K=\sqrt{W_{max}ß/C}$ where ß is a constant multiplicative decrease factor applied to window reduction at the time of loss event (i.e.the window reduces to $ßW_{max}$ at the time of the last reduction)

In this the window grows very fast upon a window reduction but as it gets closer to W_{max} ,it slows down the growth. Around W_{max} , the window increment becomes zero. Above that, CUBIC starts probing for more bandwidth in which the window grows slowly initially, accelerating its growth as it moves away from W_{max}.

Analysis of TCP CUBIC:-

In this section, we compare the performance of Linux CUBIC TCP w.r.t. AIMD. In the analysis of CUBIC,we use Linux hosts as communication end points communicating over 100Mbps link with MTU of 1500 bytes. The RTT of each background traffic is random. The socket buffer size of some client machines is fixed to default 64KB.We evaluate CUBIC-TCP and AIMD for the bandwidth utilization and RTT.

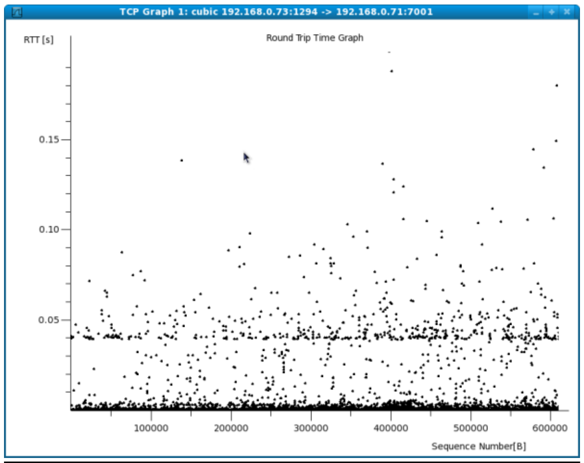


Fig 19: RTT Graph for TCP CUBIC

As per the graph shown ,the minimum RTT was around 0.001 sec and maximum RTT was around 0.18.

The congestion window of CUBIC is determined by

$W_{cubic}=C(t-K)^3+W_{max}$

Where,

C=Scaling Factor

t=elapsed time from the last window reduction.

W_{max} =window size=β/C

K=3

β= Constatnt Multiplication window decrease factor.

t=0.18

C=0.4 and β=0.8[10]

$K=\sqrt[3]{65535*08/04}=50.7965$

Wcubic=0.4(0.18-50.7965)+65535

=13662.601 or 13663 approx.

In this Graph we observe that, CUBIC starts probing for bandwidth in which the window grows slowly initially, accelerating its growth as it moves away from Wmax. This slows growth W_{max} enhances the stability of the protocol,and increases the utilization of the network while the fast growth away from W_{max} ensures the scalability of the protocol

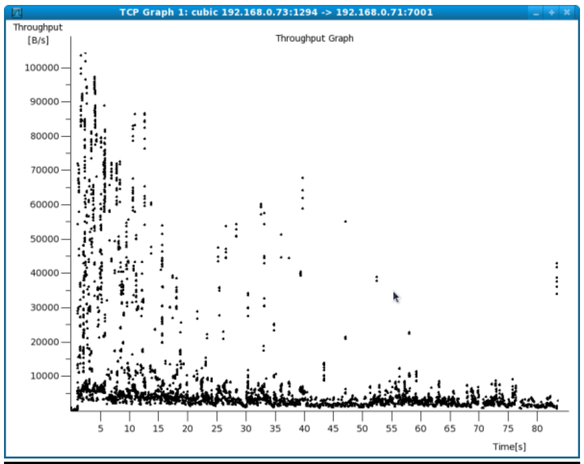


Fig: Throughput Graph for TCP CUBIC

CUBIC TCP achieves greater utilization then standard duTCP . The Graph shows that there is almost steady state due to CUBIC. The highest peak of throughput goes to 105000B/s and with an immediate corrective congestion window size afterwards. Earlier there was some sharp increase and then CUBIC maintains steadystate.

Unlike AIMD,Cubic increases the window W_{max} very quickly and then holds the window for a long time . This keeps the scalibilty of the protocol high,

.

Scalable TCP

In the Internet, data transfer protocols use various congestion control algorithms to achieve ratercontrol. Until now the AIMD algorithm was found to provide satisfactory performance. However, in high speed networks, the additive increase in the sender‟s rate may lead to inefficient link utilization. To overcome this drawback in high speed networks, the MIMD algorithm has been proposed as an alternative to the AIMD algorithm.[1] Standard TCP use AIMD but scalable TCP uses a MIMD(Multiplicative Increase and Multiplicative Decrease.)

.

1 Congestion Avoidance

Scalable TCP uses a different congestion avoidance algorithm than Standard TCP. Scalable TCP uses a multiplicative increment multiplicative decrement (MIMD) rather than the AIMD of Standard TCP

Multiplicative Increment

The multiplicative increment occurs when standard additive increment would normally occurs.
In equation (8) shows the formula used to adjust congestion window after receiving a new ACK.
Cwnd = Cwnd + a*Cwnd(3)
where a is adjustable, the value of a used was 0.02.

Multiplicative Decrement

The multiplicative decrement is same as Standard TCP except that the value of b in equation (2) is adjustable, the value of b used 0.125.

The connection starts in the slow start algorithm until channel is filled. The connection uses the multiplicative increment portion of congestion avoidance to adjust congestion window. After a single drop occur around 1.4 seconds, fast retransmit and recovery algorithms are used to cut congestion window by 0.125, the value of b, and congestion avoidance is used again to reopen congestion window

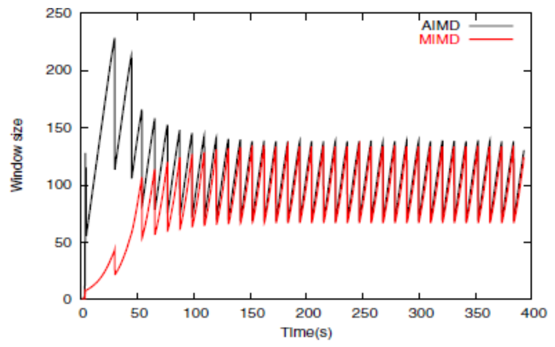
COMPARISON BETWEEN STANDARD TCP AND SCALABLE TCP

Comparison between Standard and Scalable TCP

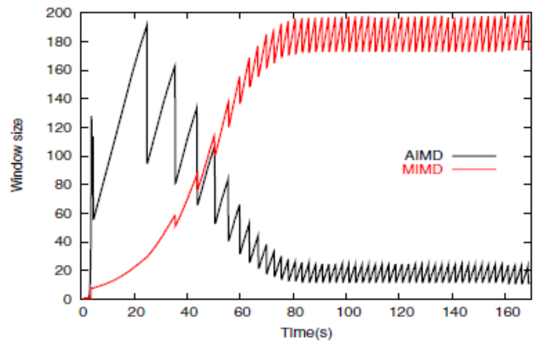
	Standard TCP(AIMD)	**Scalable TCP(MIMD)**
no losses	Wn+1=Wn+1= linear increase dw/dt=1/T	$\mathbf{W_{n+1=}\propto * W_n}$ = multiplicative increase **dW/dt=log[∝]/T*W** =exponentialgrowth
≥1 loss	$\mathbf{W_{n+1}=0.5*W_n}$ **multiplicative decrease**	$\mathbf{W_{n+1}= *W_n}$ **multiplicative decrease**

Throughput Comparison of AIMD and MIMD

We now study the scenario user and AIMD user shares the same link. We note that each user can initiate several sessions of the same algorithm**.** We obtain the condition under which the AIMD user can a obtain better throughput than the MIMD user. First, we consider the case in which each user initiates only one session. In such a scenario, the window size andthe
throughput of each session is obtained from (31)-(33) with l = 2 and k = 1. From (33) and (34), as $\Lambda \rightarrow \infty$ (i.e.,$C \rightarrow \infty$), the ratio of the throughputs, $\eta 2/\eta 1$, goes to0.
This suggests that in high-speed networks, the MIMD userwill get most of the capacity. On the other hand, if the BDP of the network is small, the MIMD user will obtain a lower
throughput compared to the AIMD session.
In this Fig, the window evolution is plotted for the twosessions for C = 13Mbps and βm = 0.5. The BDP, Λ,is less than the Λl. The AIMD algorithm obtains a betterthroughput in this case.



(a) $C = 13\text{Mbps}.\ \beta_m = 0.5.\ \tau = 140\text{ms}.$



(b) $C = 10\text{Mbps}.\ \beta_m = 0.875.\ \tau = 140\text{ms}.$

Fig: Window evolution for one MIMD session and one AIMD session

set*βm*to its recommended value of 0.875. In Fig. 24(b), the corresponding window evolution is plotted. The effect of increasing *βm* is to reduce the share of the AIMD session. a comparison of the values obtained from analysis and simulations is presented. A good match is observed between the analysis and simulations.it was observed that the throughput obtained
by each AIMD session remains constant whereas the total throughput of the MIMD sessions increases with increase in capacity. An AIMD user may want to obtain throughputssimilar to a MIMD user. In this case, the AIMD user may open several sessions in order to improve its observed throughput. Since each AIMD session gets the same throughput independent of the number of AIMD sessions (assuming there is sufficient capacity), an AIMD user can improve its observed throughput by opening multiple sessions.

In networks with sessions using MIMD algorithms, a stream of rate dependent losses, using, for example, some buffer management scheme, would be necessary to ensure fair sharing.
It was also observed that an AIMD user could open multiple sessions in order to improve its observed throughput whereas for the MIMD user the throughput was invariant to the number of sessions it opened.

CONCLUSION

.TCP WESTWOOD

In this work, we analyzed the TCP Westwood (TCPW) protocol in Linux. TCPW is a new TCP scheme, which requires modifications only in the TCP source stack and is thus compatible with TCP Reno and Tahoe destinations. Basically it differs from Reno in that it adjusts the *cwin*(congestion window) after a loss detection by setting it to the *measured rate* currently experienced by the connection, rather than using the conventional multiplicative decrease scheme.

We have analyzed with qualitative arguments and with experimental results that the Linux TCPW converges to “fair share.” At steady state under uniform path conditions. One general concern with is compatibility towards current implementations. Linux TCPW exhibits some “aggressiveness” due to its unique window adjustment. However, if there is adequate buffering at the bottleneck, TCPW and Reno share the channelfairly.

The Linux implementation was developed in order to combat in presence of random errors and under different scenarios. However, unlike previous TCP versions, the TCPW addresses the bandwidth estimation mechanism and its impact on system behavior. The results show that, for a single connection case, Linux TCPW protocol performs better than or, at least, as well as Linux TCP Reno in terms of congestion avoidance. The results also show that TCPW is more robust under varying buffer size, round trip delays and bottleneck bandwidth. The multiple connections case is under investigation and will be considered in the near future.

TCP CUBIC

We analyzed Linux TCP CUBIC which simplifies the BIC-TCP window control and improves its RTT-fairness. CUBIC uses a cubic increase function in terms of the elapsed time since the last loss event. In order to provide fairness to Standard TCP, CUBIC also behaves like Standard TCP when the cubic window growth function is slower than Standard TCP. Furthermore, the real-time nature of the protocol keeps the window growth rate independent of RTT, which keeps the protocol TCP friendly under both short and long RTT paths. Through extensive testing, we confirm that CUBIC tackles the shortcomings of BIC TCP and achieves fairly good congestion avoidance

SCALABLE TCP(MIMD):-

Scalable TCP implements simple changes to the currently used congestion control algorithm. These changes have both a positive and negative effects on the existing network traffic. Each algorithm provides higher channel utilization for high speed and long delay environment. However, the alternative algorithms do not shares channel equally, when mixed with Standard TCP traffic. In a homogenous environment, the overall channel utilization and sharing between streams increments as compared to a mixed environment. Future work is needed to study the effects of more than two competing streams

REFERENCES

1. Allman, M.V. Paxson, and W.Stevens, (1999),‟ TCP congestion Control,‟ Request for Comments, RFC2581.

2. Andrea Zanella, Gregorio Procissi, Mario Gerla,M.Y. "Medy" sanadidi „ TCP Westwood: Analytic Model and PerformanceEvaluation‟
3. D.J.Leith and R.N.Shorten(2006) „ Impact of drop synchronization on TCP fairness in high bandwidth-delay product networks.‟ Workshop on Protocols for Fast Long Distance Networks, Nara,Japan.

4. D.J.Leith(2003) „Linux Implementation issues in high speed networks.‟ Hamilton Institute technical Reportwww.hamilton.ie/net/LinuxHoghSpeed.pdf.

5. D.J.Leith, R.N. Shorten, G.McCullagh, (2005) „Experimental evaluation ofCUBIC-TCP‟ HamiltonInstitute,Ireland.
6. D.-M. Chiu and R. Jain,(1989), „Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks,‟ *Computer Networks and ISDNSystems*, vol. 17, no. 1, pp. 1
7. Evandro de Souza,debAgarwal „ A Highspeed TCP study:Characteristics and Deployment Issues‟ Lawrence Berkeley National Lab, Berkeley, CA,USA.
8. FEDORA 11. [Online]. Available:http://fedora.redhat.com
9. HabibullahJamal,Kiran Sultan (2008) „Performance Analysis of TCP Congestion Control Algorithims‟ International Journal of Computers and Communications. Issue 1, Vol2.

10. Injong Rhee, and LisongXu „ CUBIC: A New TCP-Friendly High Speed TCP Variant‟ LincolnNE68588-0115USA
11. J. C. Hoe,(1996) Improving the start-up behavior of a congestion control scheme of TCP," *ACMComputer Communication Review*, vol. 26, pp. 270–280, Oct.1996.
12. J. Nagle, (1984), „Congestion Control in IP/TCP Internetworks," *IETF RFC896*
13. JeonghoonMo,RichardJ.La, VenkatAnantharam, and Jean Warland (1998) „Analysis and comparison of TCP Reno and Vegas‟eecs.berleley.edu.
14. JitendraPadhye,victorFiroiu,DonaldF.Towsley, James F.Kurose(2000). „Modeling TCP RenoPerformance : A Simple Model and its Emperical Validation‟ IEEE/ACM Transaction on Networking Vol.8.No.2. April2000
15. Joachim charzinksi,(2000) „HTTP /TCP connection and flow characteristics.Vol.42 issues2-3
16. K.Fall and S.Floyd. (1996),„ Simulation-based Comparisons of Tahoe, Reno, and SACK TCP ACM Computer Communication review26(3)

17. L. S. Brakmo, S. W.O"Malley, and L. L. Peterson, (1994) „TCP Vegas: New techniques for congestion detection and avoidance" *ACM SIGCOMM'94*.
18. LisongXu, khaledHarfoush, and Injong Rhee „ Binary Increase Congestion Control for Fats Long Distance Networks" NSF CAREER ANI-987651,NSFANI-0074012
19. Luigi A. Grieco and SaverioMascolo,(2004)„ Performance Evaluation and Comparisonof Westwood+, New Reno, and Vegas TCP Congestion Control" ACM CCR,vol.34No.2.
20. M.Fisk and W Feng, (2001) „Dynamic right-sizing in TCP, *Los Alamos Computer Science Institute Symposium*,
21. MarioGerla. M.Y.Sanadidi,Ren Wang and Andrea Zanella, Claudio Casetti „ TCP Westwood: Congestion Window Control Using Bandwidth Estimation" Computer Science Dept., University of California, Los Angeles(UCLA)
22. PasiSarolathi, Alexey Kuznetsov(1999) „Congestion Control in Linux TCP" IEEE/ACM Transaction on NetworkingVol.8.No.2.
23. R. Stevens,(1994), „*TCP/IP Illustrated,' Volume 1: The Protocols*.Addison-Wesley,
24. S S.ShenkerL. Zhang, and D. D. Clark, (1990)„Some observations on the dynamics of a congestion control algorithim" *ACM Computer Communication Review*, vol. 20, pp. 30–39,
25. S.Floyd (Dec,2003) „Highspeed TCP fro Large Congestion Windows" RFC3649
26. S.Keshav,(2002), „An engineering approach of ComputerNetworking"
27. SangtaeHa,Injong Rhee, LisongXu (2005)„ CUBIC: A New TCP Friendly High-Speed TCP Variant" nscu.edu
28. SaverioMascolo and Francesco Vacirca (2001), „Issues in Performance Evaluation of New TCP Stacks in HighSpeed".
29. Steven Low, Larry Peterson and Limin Wang Princeton University (February ,2000) „ Understanding TCP Vegas: Theory and Practice" TR616-00
30. T. Iguti, G. Hasegawa, and M. Murata (2005), „A new congestion control mechanism of TCP with inline network measurement" *The International Conference on InformationNetworking (ICOIN) 2005*, pp.109–121,
31. VanJacobson,(1988)„„CongestionAvoidanceandControl",*ProceedingsoftheSigcomm '88 Symposium*, vol.18 (4): pp.314–329. Stanford,CA.
32. Kulvinder Singh„ Experimental Study of TCP Congestion Control Algorithms" IJCEM International Journal of Computational Engineering & Management, Vol. 14, October 2011
33. M. Christiansen, K. Jeray, D. Ott, and F. D. Smith, Tuning RED for web traffic," in Proc. of ACM SIGCOMM'00, (Stockholm, Sweden), September2000