International Journal of Information Technology and Knowledge Management
July-December 2009, Volume 2, No. 2, pp. 321-328

A REFINED NEW MODEL FOR EFFICIENT EXPLICIT CONGESTION REDUCTION IN HIGH TRAFFIC HIGH SPEED NETWORKS THROUGH AUTOMATED RATE CONTROLLING

K. Satyanarayan Reddy[*] & Lokanatha C. Reddy[**]

The Transmission Control Protocol (TCP) suffers from poor performance on high bandwidth delay product links meant for supporting data transmission rates of multi Gigabits per seconds (Gbps). During congestion, the TCP's congestion control algorithm reduces the congestion window *cwnd* to ½ and enters additive increase mode, which proves to be slow in taking advantage of large amounts of available bandwidth. In this paper a refined new model, to overcome the drawbacks of the TCP protocol, has been presented. We propose to carry out a study of the refined new model based on various parameters viz., Throughput, Fairness, Stability, Performance and Bandwidth Utilization for supporting high data transmission rates across the High Speed Networks.

***Keywords**:* Congestion Control, High Speed Networks.

1. Introduction

TCP has been the most used transport protocol for the Internet for over two decades. The scale of the Internet and its usage has increased by several orders of magnitudes. The nature of applications has changed significantly. Some of the assumptions made during the early design process of TCP are no longer valid. And yet, TCP remains the main protocol of the TCP/IP protocol stack based on which the Internet runs. The reason for this importance is that it constantly evolves to keep up with the changing network demands [1], [2], [12].

However as the application needs changed, newer rate control schemes were proposed [2], [3], [4], [6], [8], [9], [10] and [12]. As a result we now have an Internet which operates on a variety of congestion control schemes, even though TCP remains the most widely used transport protocol. In [3], [9], [10] the authors have argued that these new congestion control schemes can lead to a new congestion collapse and pose the problem of congestion response conformance (wherein selfish/non-behaving sources get an unfavorable share of bandwidth in comparison to TCP).

The congestion control functionality of TCP is provided by four main algorithms namely slowstart, congestion avoidance, fast retransmit and fast recovery in conjunction with several different timers. Slowstart uses exponential window increase to quickly bring a newly starting flow to speed. In steady state, the flow mostly uses congestion avoidance in conjunction with fast retransmit/recovery.

1.1 Working of TCP

TCP is a self-sufficient and reliable transport protocol, in the sense that the sender uses information provided by the receiver in the form of acknowledgments, to determine the nature of congestion in the network. No explicit feedback is expected from the routers. This self-sufficiency is based on the assumption that anytime packets do not arrive at the receiver in the same order that the sender sent them, then it is due to congestion in the network. While in most conventional networks, this assumption is true, newer network environments challenge it [12].

TCP uses a sliding-window based congestion control algorithm proposed by Van Jacobson and others [1]. The slow-start algorithm is activated (triggered) at the beginning of a transfer or after a Retransmission Timer timeOut (RTO). Slow-start occurs until the congestion window (*cwnd*) reaches the slow-start threshold (*ssthresh*) or if packet loss occurs.

During the slow-start phase, if the receiver buffer size is large enough, the number of segments injected into the network is doubled every Round Trip Time (RTT). When the *cwnd* exceeds the *ssthresh*, the congestion avoidance algorithm is used to lower the sending rate by increasing the cwnd by at most one segment per RTT.

This is the additive increase algorithm of TCP and is used for probing the additional network capacity. Upon the arrival of three duplicated acknowledgements (ACKs) at the sender's end, the fast retransmit algorithm is activated, which retransmits that segment without waiting for the RTO to expire.

[*] Dept. of CS, School of Science & Technology, Dravidian University, Kuppam-517425, A.P., India. *E-mail: ksatyanreddy@yahoo.com*

[**] Dept. of CS, School of Science & Technology, Dravidian University, Kuppam-517425, A.P., India. *E-mail: lokanathar@yahoo.com*

Duplicate acknowledgements may occur when a packet is lost yet three additional packets arrive at the receiver. After the retransmission of the lost segment, the fast recovery method is used to adjust the cwnd. As a result *ssthresh* is set to half the value of *cwnd*, and then the *cwnd* is cut in half plus three segments. At this point, for each duplicate ACK that is received, the *cwnd* is increased by one segment until the ACK of the retransmission arrives. After that, *cwnd* is set to *ssthresh* and the additive increase algorithm is activated until either is equal to the advertised receiver window or until loss is detected, indicating possible congestion.

Since the above fast retransmit method can only fix one lost segment per RTT, the subsequent lost segments within that RTT usually have to wait for the RTO to be expired before being resent. For most variants of TCP that are currently being used including TCP Reno and TCP SACK, the sending rate is cut in half, each time a loss occurs. The sending rate is then gradually increased until another loss occurs.

This process is known as Additive Increase, Multiplicative Decrease (AIMD) is repeated until all of the data has been transmitted. This is one of the reasons TCP has difficulty operating efficiently over long delay and error prone networks.

In these cases, the mis-classification is the cause for out-of-order packet delivery or packet losses and congestion forces TCP to use multiplicative decrease of the congestion window and results in degraded performance.

2. CURRENT DRAWBACKS OF TCP

Basic TCP congestion control theory is well-known and in the past couple of years, a number of studies [2], [3], [4], [6] have been carried out to analyze it. Many researchers have worked on improving the TCP congestion control algorithm.

TCP is unable to utilize all the available bandwidth on high-bandwidth and/or high-delay paths due to its conservative congestion avoidance algorithm. In fact TCP can become quite unstable under these conditions. One problem is that the TCP does not have a mechanism to distinguish between a slowest (narrow/bottleneck) link and congested (tight) link. This means that TCP's algorithm will continue to increase the congestion window (assuming tuned large buffers) to increase the sending rate as long as there is no further packet loss.

This leads to problem, since packet drop could be caused by congestion at the narrow link. In either a high-speed and/or long delay path, when a congestion signal comes back to the sender, the outstanding data stream will be the average size of congestion window, which is computed from the acknowledgments during the last round-trip-time (RTT) period.

An examination of the congestion avoidance mechanism shows that the bursts are in two different phases of the TCP congestion control algorithm; slow start and congestion avoidance. In the slow start phase, the algorithm doubles the size of the burst until packet loss occurs, probing for the ceiling of the congestion window.

After seeing packet loss, standard TCP congestion control reduces the congestion window to one half the current window sizes. If TCP sees more packet loss, it will reduce the window further. This is called "multiplicative decrease" which prevents further packets from causing collapse. This slow start algorithm assumes that a possible best congestion window is between the last burst (congestion window) and the previous burst (one half of the congestion window) since the previous burst did not cause packet loss.

However, this does not efficiently avoid packet loss, especially when the bandwidth [5], [7] or path latency is high. Since acknowledgments are asynchronously fed back to the sender, they can cause further fluctuations when the cross traffic is more dynamic. The key issue in the slow start phase is during the last few window adjustments.

In a better TCP design, the last few probes should be used to detect the bottleneck router's queue size and its capacity, and should not use an exponential increase of the burst (window) size to cause loss. Instead, it should use an adaptive algorithm to increase its burst size to avoid losing a large number of packets. This would also allow the detection of the best rate to pace out the packet. Window based congestion control mechanisms also lack the ability to predict congestion on-the-fly and dynamically adjust their sending rate to reflect the new available bandwidth.

3. SUGGESTED NEW REFINED MODEL

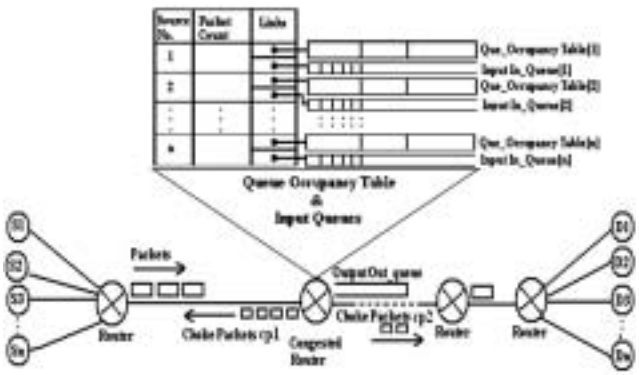

Figure 1: New Refined Model for Congestion Control

Where the model given in Figure 1 above is a modified version of model presented in [17], [18] and

a. *S*1, *S*2… *Sn* are sending sources and *D*1, *D*2… *Dn* are the destination nodes.

b. The choke packet **cp1** is meant for informing the sending source about its new sending rate and choke packet **cp2** is meant for informing the router (in the direction of Destination node) **next** to current Congested router to drop packets of misbehaving source indicated in it.

c. Total current In_queue capacity = Σ Packet Count[i]; for $i = 1, 2\ldots n$. (1)

d. Actual capacity of '*n*' In_queue's = ΣIn_Queue[i] for $i = 1, 2\ldots n$. (2)

e. from equation (1) and (2) above, the total percentage of In_queue occupancy can be found as follows:

$$\text{Total \% queue\_occupancy} = \frac{\Sigma\,\text{PacketCount[i]}}{\Sigma\,\text{In\_Queue[i]}} \quad (3)$$

f. Individual percentage of i^{th} In_queue occupancy factor 'β' is given as follows:

$$\beta = \text{percent\_occu} = \frac{\text{PacketCount[i]}}{\text{In\_Queue[i]}} \quad (4)$$

g. Packet Count is meant for containing the count of no. of packets present currently in the In-Queue [i] for each $i = 1, 2, \ldots, n$.

The WaitTime for i^{th} source denoted as 'WaitTime^i' is directly proportional to the Round Trip Time (RTT) for i^{th} source denoted as RTT_i i.e.

$WaitTime^i \propto RTT_i$

$$WaitTime^i = k * RTT_i \quad (5)$$

where '*k*' is the constant of proportionality which is is calculated as follows:

h. The Lee-Time factor 'α1' is given by

$$\alpha 1 = 1 - \alpha \quad (6)$$

i. The Extra-Time 'α' is calculated based on the Round Trip Time (RTT) of the i^{th} source RTT_i and is given by

$$\alpha = RTT_i * \alpha 1 \quad (7)$$

j. The WaitTime '*T*' is calculated as follows:

$$WaitTime^i = T = RTT_i + \alpha \quad (8)$$

Putting value of α form equation (7) into equation (8) we get

$$WaitTime^i = T = RTT_i + RTT_i * \alpha 1$$

$$WaitTime^i = T = RTT_i * (1 + \alpha 1) \quad (9)$$

So from equation (5) we can infer

$$k = 1 + \alpha 1 \quad (10)$$

e.g.: Assuming that the RTT_i of the i^{th} source = 36ms then to calculate the $WaitTime^i = T$, we proceed as follows:

Say $\beta = 65\%$ then $\alpha 1 = 1 - \beta = 35\%$ thus

$\alpha = RTT_i * \alpha 1 = 36ms * 35/100 = 12.6ms$ finally the $WaitTime^i = T$ is calculated as

$T = RTT_i + \alpha = 36ms + 12.6ms = 48.6$ ms.

Observation: We see that more, a source misbehaves it gets less Extra-Time 'α'. In other words more the value of In_queue occupancy factor 'β', the chances of a misbehaving source getting penalized increases in terms of reduced Extra-Time 'α'.

The **cp1** and **cp2** are the choke packets towards downlink (sending source) and uplink (receiving node) respectively and cp1 and cp2 have following formats:

cp1 Format:

Table 1

Source IP Address	*Destination IP Address*	*New Sending Rate*	*Extra Time "α"*	*Congested Router's Time Stamp*

cp1: indicates to the sending source (with SourceIP Address) that current rate of data transmission is to be set to New Sending Rate i.e. currentRate ← New Sending Rate.

Also the sending source is informed about the extraTime (besides Sending Source Time–Congested Router's Time Stamp) it has.

cp2 Format:

Table 2

SourceIP Address	*DestinationIP Address*	*DropFlag = 1*

cp2: This is an indication to the router (in the direction of Destination node) **next** to current Congested router that if DropFlag = 1 then all the remaining packets (if any) from the misbehaving source with "SourceIP Address" and meant for "DestinationIP Address" of cp2 should be dropped from its In_Queue and the links for Que_Occupacy Table and In_Queue be freed.

4. THE ALGORITHMS

4.1. Network Traffic Classification

In this new refined model based on [18], we are assuming that at any point of time '**n**' sources *S*1, *S*2, …, *Sn* are communicating with the '**n**' destinations *D*1, *D*2, …, *Dn* (as the model is being developed for Private Network

Service providers supporting High-speed Communications). This new protocol incorporates few changes in the current Transmission Control Protocol (TCP) and it works at the router level.

When the packets are received by the router/switch from the sources, these packets are forwarded for onward transmission based on Store and Forward principle i.e. when the outgoing link is not available for onward transmission of the received packets then such packets are stored in the **In_Queue** before being forwarded to the **Out_queue**.

The received packets from the sending sources *S*1, *S*2… *Sn* are accommodated in individual queues (here we have made assumption that the in_queue comprises of '*n*' different queues, one for each transmitting source i.e. the transmission from source *S*1 will be accommodated in the *q*1 of in_queue, the transmission from source *S*2 will be accommodated in the *q*2 of in_queue and so on).

The packets are forwarded to the out_queue on round-robin basis i.e. a packet is chosen from each of the '*n*' in_queue's i.e. a packet from *q*1, a packet from *q*2 and so on a packet from *qn* is chosen. This continues till the time there is no congestion in the network i.e. no packet loss have been observed.

The moment a packet loss is observed, the sending sources are informed to reduce their sending rates through the choke packet (choke packet cp1) as shown in Figure 1 above.

And the router enters in wait mode wherein it performs the above job as usual for a pre-calculated time duration recorded in the Que_occupancy table. Once the wait period is over for a source, and the source fails to comply with the rate reduction then such source is declared to be a misbehaving source and all the packets from such a source are dropped from queue containing packets from the misbehaving sources from the in_queue and a Choke packet (choke packet cp2) as shown in Figure 2 above is sent to the router (in the direction of Destination node) next to current Congested router.

The Bandwidth that was allocated to the misbehaving source is added to the Total Available Bandwidth, so that new sources which are willing to communicate can be allocated requisite bandwidth [5], [14], [18] (subject to the availability of the requested bandwidth).

The Que_occupancy Table has the following format:

Table 3

Source IP Address	*Destination IP Address*	*Current rate*	*New Sending Rate*	*Present Time*	*Wait-Time*

This Table is maintained / updated for each customer who is registered with the High Speed Network connectivity service provider.

Where the terms in the Que_Occupancy Table are defined as follows:

Source No.: This is an integer field corresponding to the source numbers viz. 1, 2, 3,…., *n* for the sources *S*1, *S*2, …, *Sn*.

Source IP Address: It is the IP address of the sending source node.

Destination IP Address: It is the IP address of the Destination node.

CurrentRate: It contains the value of current rate of sending as agreed upon between sending source and the Highspeed Network communication service provider.

NewSendingRate: This rate is initially 0 (zero) in the Que_Occupancy Table till the time the congestion is experienced by the router, but as the intermediate router/ switch experiences the congestion through the packet drops, a new Sending rate is calculated for all the sending sources based on the number of their packets present respectively in the **In_Queue** with respect to overall Que_Occupancy. And it is calculated in terms of overall percentage as depicted in the following Algorithm 4.2.4.

Wait Time: This time is initially 0 (zero) in the Que_Occupancy Table till the time the congestion is experienced by the router, but on packet drops, the algorithm *Congestion Detection* 4.2.2 gets activated and *NewSendingRate* for all the sources are calculated. This *NewSendingRate* (requests the sources for reducing their current sending rate) is conveyed to the sending sources through the choke packets. And the *WaitTime* is calculated & set for all the sending sources by updating the Que_Occupancy Table. During this time none of the packets present in the in_queue are dropped. When WaitTime for a source gets exhausted then all packets from such source in the in_queue are dropped.

4.1.1 Traffic from Behaving Sources

All the Sender nodes that transmit the packets as per the agreed terms of Quality of Service (**QoS**) [13], [15] & [16] and during congestion, the nodes which reduce their current sending rates accordingly after receiving the choke packets from congested node are called the Behaving sources.

4.1.2 Traffic from Non-Behaving Sources

All Sender nodes that do ***NOT*** transmit the packets as per the agreed terms of **QoS** even after receiving the RM or Choke packets from the congested node for reducing their current sending rate are called the non-Behaving sources such UDP traffic. Such non-behaving nodes keep on transmitting more and more packets which may lead to worsening of network congestion due to high percentage of queue occupancy and bandwidth requirements thus not allowing the genuine users to get connected to the Network.

4.2 Algorithm to Work at Intermediate Router Level

4.2.1 Algorithm for "Main Module"

1. Receive the incoming packets from source.
2. Check the source and destination address.
3. if (a packet has been dropped){
4. Call **Congestion-Detection**
5. Go to step1.}
6. Move the packets into the Priority in_queue.
7. if (outgoing link free){
8. Move the packets from in_queue to out_queue}
9. else
10. Call **Wait-Mode**.
11. Go to step 1.

4.2.2 Algorithm for "Congestion Detection":

1. Check for queue occupancy.
2. if (que_occupancy >= 65%){
3. Call **Control Module**
4. Call **Wait Module**
5. Call **Packet-Drop Mode**
6. Call **Scale-up Mode**.}
7. Return to **"Main Module"**.

4.2.3 Algorithm for "Control Module":

1. set i = 1.
2. if ($i > n$)
3. Return to **Congestion Detection** Module.
4. else calculate percent_occu
5. if (percent_occu > 65)
6. newSendingRate = 1/2 * CurrentRate
7. else if (percent_occu > 60)
8. newSendingRate = 1/4 * CurrentRate
9. else if (percent_occu > 55)
10. newSendingRate = 1/8 * CurrentRate
11. else if (percent_occu > 50)
12. newSendingRate = 1/16 * CurrentRate
13. else if (percent_occu > 45)
14. newSendingRate = 1/32 * CurrentRate
15. else if (percent_occu > 40)
16. newSendingRate = 1/64 * CurrentRate
17. else if (percent_occu > 35)
18. newSendingRate = 1/128 * Current Rate
19. else
20. newSendingRate = CurrentRate.
21. send a choke packet **cp1** to i^{th} node with newSendingRate
22. $i = i + 1$.
23. Go to step 2.

4.2.4.0 Algorithm for "Wait Module":

1. set $i = 1$
2. while ($i <= n$) {
3. TimeGiven = CurrentSystemTime–PresentTime
4. if (TimeGiven > $WaitTime_i$)
5. Drop[i] = 1
6. else
7. Drop[i] = 0
8. $i = i + 1$}
9. Return to **"Congestion Detection" Module**.

Where Drop[ ] is an array of flags which is used as an indication for dropping the packets from the **in_queue** and subsequently the details of such misbehaving source are to be removed from the Que_occupancy table. If the flag value of Drop[i] = 1 then the packets of the source 'i' are removed from the *in_queue* otherwise the packets are **not** removed from the *in_queue*.

4.2.4.1 Algorithm for "Wait Mode":

1. Accept incoming packets
2. check In_Queue status
3. while (In_Queue **not** free) {
   drop the packets}
4. Return to **"Main Module"**

4.2.5.0 Algorithm for "Packet-Drop Mode":

1. set i =1
2. while ($i <= n$) {
3. if (Drop[i] = = 1)
4. if (ReceivingRate < = NewSendingRate)

5. go to step 8
6. else
7. Call **DropInQuePackets** (i)
8. $i = i + 1$
9. go to step 2 }
10. Return to **"Congestion Detection Module"**.

4.2.5.1 Algorithm for "DropInQuePackets (i)":

1. delete all the packets from i^{th} source in i^{th} queue of In_Queue[i]; Packet-count [i] = 0
2. Update QueOccupancy Table by deleting all the column values by freeing the links for Que_occupancy Table[i] and the In_Queue[i]
3. send a choke packet **cp2** to next router on uplink
4. add bandwidth "*Bi*" of ith source to Total Available Bandwidth
5. Return to "**Packet-Drop Mode**"

4.2.6 Algorithm for "Scale-up Mode":

1. if (new connection requests pending) // (if any)
2. {receive new connection requests with its IP address, destination IP address and Requested data rate
3. get the amount of BandwidthRequested
4. if (BandwidthRequested <= Total Available Bandwidth) {
5. set CurrentRate = agreed Rate as per QoS
6. set WaitTime = 0
7. set NewSendingRate = 0
8. grant connection to this new source by allocating nodes for Que_OccupacyTable and the In_Queue
9. Update QueOccupancy Table by inserting all the above details in it }
10. else {
11. Reject new Connection request
12. go to step 1 }
13. else {
14. look for behaving sources //the ones which have reduced their rates of sending.
15. increase the Bandwidth of such sources by an amount < = surplus Total Available Bandwidth and not exceeding Maximum data transfer rate of the source (as per QoS)
16. Update QueOccupancy Table by changing the CurrentRate of behaving sources}
17. Return to **"Congestion Detection" Module**.

5. BANDWIDTH MANAGEMENT

We propose to manage the network bandwidth using the Dynamic Programming Algorithm [14] assuming that Network bandwidth is to be allocated amongst 'n' number of hosts, which are willing to connect (to communicate with the other nodes) to the network.

Let *B*1, *B*2, …, *Bn* be the bandwidth requirements of the '*n*' hosts respectively and let the total throughput function "*T*" (the Objective Function) be expressed as *sum* of the product of individual bandwidths and throughputs of the '*n*' hosts as follows:

$$Maximize\ T\ (B_1, B_2, \ldots, B_n) = t_1 \times B_1 + t_2 \times B_2 + \ldots. + t_n \times B; \quad (11)$$

The constraint on bandwidth can be defined as:

$$B_1 + B_2 + \ldots + B_n \le \mathbf{B}\ ; \quad (12)$$

where $\mathbf{B}_i > \mathbf{0}$ for $i = 1, 2, \ldots, n$; and **B** is the total link Bandwidth and

$$\textbf{Case i}.\ t_1 + t_2 + \ldots. + t_n \ge 0.85\ ; \quad (13)$$

"t_i" is the **Throughput** of the link 'i' and $\boldsymbol{t_i > 0}$ for $i = 1, 2, \ldots, n$.

$$\textbf{Case ii}.\ t_1 + t_2 + \ldots. + t_n \ge 0.95\ ; \quad (14)$$

"t_i" is the **Throughput** of the link 'i' and $\boldsymbol{t_i \le B_i}$; $\boldsymbol{t_i > 0}$ for all $i = 1, 2, \ldots, n$.

We define the free or un-allocated bandwidth given by the formula:

$$F(B) = \mathbf{B} - \{B_1 + B_2 + \ldots + B_n\}; \quad (15)$$

We define **TAB: T**otal **A**vailable **B**andwidth is given by the formula

$$\mathbf{TAB} = F(B) + \Sigma\ B_k\ ;\ \text{for all } k = 1, 2, \ldots., m \quad (16)$$

where "B_k" is the bandwidth that became available when k^{th} misbehaving connection was dropped; and

Let **BRI:** Bandwidth requested by j^{th} new node during scale-up is given by:

= B_j bandwidth requirement of new link 'j'.

We now define the **T**otal **A**vailable **B**andwidth ($\mathbf{TAB_s}$) after Scaling (i.e. new connections request for Bandwidth) as follows:

$$\mathbf{TAB_s} = \mathbf{TAB} - \Sigma\ B_j\ ;\ \text{for all } j\ ; \quad (17)$$

where j is the new node requesting Bandwidth "B_j" during the scale-up.

6. THE COMPLEXITY

The overall complexity of the algorithm if of the order of ***O*(*n*)**; as information of the source can be obtained through the source number (the first column of the Que_Occupancy Table) and the same can be used for accessing the contents of a particular in_queue. The best–case complexity is $O(1)$; average-case is $O(n)$ and worst-case complexity is $O(n)$.

7. THE SIMULATION ENVIRONMENT

We are using the Network Simulator NS 2.31 for creating and testing the proposed network Model [11].

8. EXPECTED RESULTS

In this model, the rate of transmission for all the sending sources is not decreased to ½ during the severe congestion (unlike the conventional TCP which reduces the *cwnd* to ½ for all the transmitting sources).

Instead, this model ensures well in advance that congestion is taken care-off i.e. when the *in_queue* is **65%** full, then based on the quantum of total percentage of queue occupancy the new data transmission rate is calculated for individual sending source (based on the QoS parameters as agreed upon) and is conveyed by the router to the respective sending source through the *choke* packets and the congested router waits for the sources to reduce their respective transmission rates.

The model achieves fairness through the fact that the sources which are sending packets indiscriminately are penalized with drastic cut in their transmission rates (maximum to ½ the current rate of transmission, like TCP) and behaving sources may have to reduce their sending rates to a low or moderate levels (but not all the sources are required to reduce their current rate of transmission to ½).

The proposed model based on [8], [9], [10] and [17] is expected to

a. Optimize the Bandwidth (using Dynamic Programming concept of Operation Research) and make the bandwidth available to the Behaving sources under Congestion situation and also when there is No Congestion.

b. Maximize the Throughput for the Behaving sources under Congestion situation and also when there is No Congestion.

c. Meet the QoS demands of the Network Traffic during Congestion situation and also when there is No Congestion.

d. Reject / drop all the packets from the Non-behaving source, during congestion, and packets from the behaving sources are accepted and accommodated in queue for onwards transmission.

e. Allow scaling up i.e. allocating Bandwidth to new host which agrees to behave by sending packets as per QoS agreement.

ACKNOWLEDGEMENT

The authors thank the authorities of the Dravidian University, Kuppam-517425, AP, India, who provided opportunities and resources for carrying out this research and for the liberal grants extended for research activities in the Dept. of CS at the University.

REFERENCES

[1] Jacobson, V., "Congestion Avoidance and Control". In Proceedings of SIGCOMM '88, *Stanford, CA,* (1988).

[2] Sally Floyd, "HighSpeed TCP for Large Congestion Windows and Quick-Start for TCP and IP", *Yokohama IETF,* (2002), *(http://www.icir.org).*

[3] Dina Katabi, Mark Handley, Charlie Rohrs., "Congestion Control for High Bandwidth-Delay Product Networks". SIGCOMM '02, *Pittsburgh,* Pa, (2002).

[4] Tom Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks", ACM SIGCOMM *Computer Communication Review,* (2003). *(http:// citeseerx.ist.psu.edu).*

[5] G. Jin, B. Tierney "Netest: A Tool to Measure Maximum Burst Size, Available Bandwidth and Achievable Throughput", Proceedings of the 2003 ITRE, *Newark, NJ,* Aug.' 10-13, (2003), LBNL-48350.

[6] C. Jin, D. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, S. Singh, "*FAST TCP:* From Theory to Experiments", (2003). *(http:// netlab.caltech.edu).*

[7] G. Jin, "Feedback Adaptive Control and Feedback Asymptotic Convergence Algorithms for Measuring Network Bandwidth", (2002), LBNL-53165.

[8] K. Satyanarayan Reddy, C. Lokanatha Reddy "A Survey on Congestion Control Mechanisms in High Speed Networks"-*International Journal of Computer Science and Network Security (IJCSNS)* **8**(1) (2008) 187–195. *(http:// paper.ijcsns.org).*

[9] K. Satyanarayan Reddy, C. Lokanatha Reddy "A Survey on Congestion Control Protocols for High Speed Networks"-*International Journal of Computer Science and Network Security (IJCSNS)* **8**(7) (2008) 44–53. *(paper.ijcsns.org).*

[10] K. Satyanarayan Reddy, C. Lokanatha Reddy "An Efficient Explicit Congestion Reduction in High Traffic High Speed Networks through Automated Rate Controlling" Appeared in the Proceedings of *International Conference ICSTAORIT–2006–XXVI ISPS CONFERENCE* with Paper Id : IT-45 held at Tirupati, India during the Period 7th January 2007 to 9th January 2007.

[11] The Network Simulator NS2 Documentation, "*The NS Manual*". *(http://www.isi.edu / nsnam / ns / ns-ocumentation.html).*

[12] Sumitha Bhandarkar, "Congestion Control Algorithms of TCP in Emerging Networks", PhD Thesis Submitted to the Office of Graduate Studies of *Texas A. & M. University (TAMU),* (2006).

[13] Weibin Zhao, David Olshefski and Henning Schulzrinne "Internet Quality of Service: An Overview", *Technical Report CUCS-003-00,* Columbia University, (2002).

[14] G. L. Nemhauser " Introduction to Dynamic Programming" *John Wiley, New York,* (1966).

[15] P. Ferguson and G. Huston. "Quality of Service: Delivering QoS in the Internet and the Corporate Network" *Wiley Computer Books, New York,* NY, (1998).

[16] R. Guerin and V. Peris. "Quality-of-Service in Packet Networks: Basic Mechanisms and Directions" *Computer Networks,* **31**(3) (1999) 169–189.

[17] K. Satyanarayan Reddy, C. Lokanatha Reddy "A Model for an Efficient Explicit Congestion Reduction in High Traffic High Speed Networks through Automated Rate Controlling" - International Journal of Computer Science and Network Security (IJCSNS) **8**(11) (2008) 36–43. *(http://paper.ijcsns.org).*

[18] K. Satyanarayan Reddy, C. Lokanatha Reddy "A Modified Model for an Efficient Explicit Congestion Reduction in High Traffic High Speed Networks through Automated Rate Controlling"-IEEE International Advance Computing Conference (IEEE IACC'09) held during 6th and 7th March 2009 at Thapar University, Patiala, Punjab, India, 783–788 (ISBN NO: 978-981-08-2465-5).