A Model for an Efficient Explicit Congestion Reduction in High Traffic High Speed Networks Through Automated Rate Controlling

K. Satyanarayan Reddy†
ksatyanreddy@yahoo.com
Dept. of ISE, EPCET*, Bangalore &
Research Scholar, Dept. of CS,
School of Science & Technology,
Dravidian University, Kuppam-517425, A.P., India.

and

Lokanatha C. Reddy††
lokanathar@yahoo.com
Professor, Dept. of CS,
School of Science & Technology,
Dravidian University, Kuppam-517425, A.P., India

Summary: The conventional TCP suffers from poor performance on high bandwidth delay product links meant for supporting data transmission rates of multi Gigabits per seconds (Gbps). This is mainly due to the fact that during congestion, the TCP's congestion control algorithm reduces the congestion window *cwnd* to ½ and enters additive increase mode, which can be slow in taking advantage of large amounts of available bandwidth. In this paper we have presented a new model and to overcome the drawbacks of the TCP protocol and propose to carry out a study of the same based on various parameters viz., Throughput, Fairness, Stability, Performance and Bandwidth Utilization for supporting data transmission 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 significantly changed. Some of the assumptions made during the early design process of TCP are no longer valid.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 TCP enjoys 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 with a spectrum 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).

TCP resides in layer 4 of the 7-layer OSI network model. It provides a connection-oriented, reliable, byte-stream service that is both flow and congestion controlled to the upper layers (application layer), while assuming or expecting little from the lower layers (IP layer and below). This is accomplished by a complicated set of algorithms.
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.

These algorithms implement the classic Additive Increase/Multiplicative Decrease (AIMD) of the congestion window. When no losses are observed, the congestion window is increased by one for the successful acknowledgment of one window of packets. Upon a packet loss, the window is decreased to half its earlier value, to clear out the bottleneck link buffers. There are several challenges in current networks to this simple AIMD policy.

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].

Manuscript received November 5, 2008
Manuscript revised November 20, 2008

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.

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 *sshthresh* 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 of the cause for out-of-order packet delivery or packet losses as 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 is problematic 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.
Consider a 100ms RTT and 40Gb/s path, TCP needs to send a burst as large as 500 Mbytes of data during one RTT to detect congestion trend. This big burst of traffic plus existing cross traffic will exceed the bottleneck link router queue and cause up to 50% packet loss (more than 160K packets in above example). A self-clocking system could help to reduce the loss probability when cross traffic is less bursty, but this may not be the condition under which the current network is dropping packets.

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 [7] or path latency is high. For example, on a 100ms RTT and 100 Gbps path, the previous burst can be 1 GB, and doubling it can cause the increased 1GB data loss. 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 Model (please see appendix in full-scale):

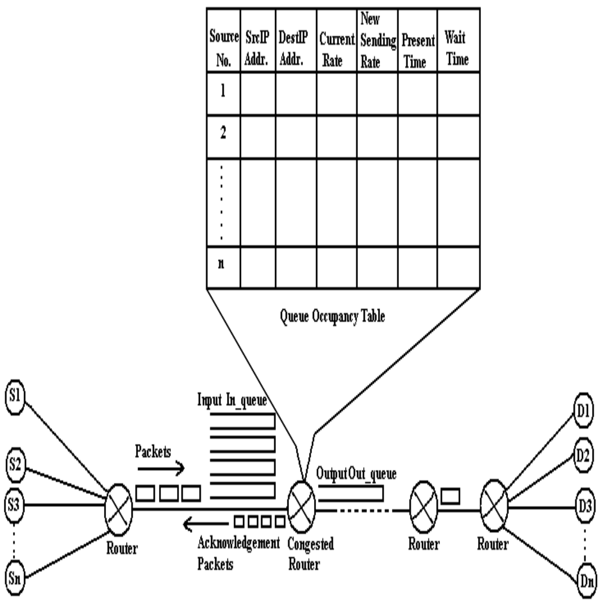


Figure 1: New Model for congestion control

4. The Algorithms:

4.1. Network Traffic Classification:

In this model we are assuming that at any point of time '**n**' sources S1, S2, …, Sn are communicating with the '**n**' destinations D1, D2, …, 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 S1, S2… 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 S1 will be accommodated in the q1 of *in_queue*, the transmission from source S2 will be accommodated in the q2 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 q1, a packet from q2 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 acknowledgement packets (choke packets) 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*.

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 [14] (subject to the availability of the requested bandwidth).

Table1: The Que_occupancy table has the following format:

Source no.	Source IP Address	Destination IP Address	Current Rate	New Sending Rate	Present Time	Wait-Time
1						
2						
:						
n						

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 S1, S2, …, 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.
WaitTime: 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 which will work at intermediate Router/Switch 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.
```

4.2.3 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".
```

4.2.4 Algorithm for "Control Module":

```
1. set i = 1.
2. if ( i > n)
3.         Return to Congestion Detection Module.
4. else
5.percent_occu = $\frac{\text{No. of packets from } i^{th} \text{ source in the in\_queue}}{\text{Total no. of packets in the whole in\_queue}}$.
6. if (percent_occu > 65)
7.         newSendingRate = 1/2 * CurrentRate
8. else if (percent_occu > 60)
9.         newSendingRate = 1/4 * CurrentRate
10. else if (percent_occu > 55)
11.        newSendingRate = 1/8 * CurrentRate
12. else if ( percent_occu > 50)
13.        newSendingRate = 1/16 * CurrentRate
14. else if (percent_occu > 45)
15.        newSendingRate = 1/32 * CurrentRate
16.  else if (percent_occu > 40)
17.        newSendingRate = 1/64 * CurrentRate
18. else if (percent_occu > 35)
19.        newSendingRate = 1/128 * Current Rate
20. else
21.        newSendingRate = CurrentRate.
23. send a choke packet to i^th node with newSendingRate
24. i = i + 1.
25. Go to step 2.
```

4.2.5.0 Algorithm for "Wait Module":

```
1. set i = 1
2. while (i < = n) {
3.   TimeGiven = CurrentSystemTime –
                                   PresentTime
4.         if (TimeGiven > WaitTime)
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*.

4.2.5.1 Algorithm for "Wait Module":

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

4.2.6.0 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".
```

4.2.6.1 Algorithm for "DropInQuePackets":

```
1. delete all the packets from i^th source in i^th queue of
   in_queue
2. Update QueOccupancy Table by deleting all the
   column values except Source IP Address
3. add bandwidth "Bi" of ith source to Total Available
   Bandwidth
4. Return to "Drop Module"
```

4.2.7 Algorithm for "Scale-up Mode":

```
1. if (new connection requests pending) // (if any)
2.  {receive new connection requests with its IP
                                    address
3.      get the amount of BandwidthRequested
4.      if(BandwidthRequested<= Total Available
                Bandwidth) {
5.      set CurrentRate = BandwidthRequested
6.      set WaitTime = 0
7.      set NewSendingRate = 0
8.      grant connection to this new source
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
16      Update QueOccupancy Table by changing the
        CurrentRate of behaving sources}
17. Return to "Congestion Detection" Module.
```

5. The Simulation Environment:

We are using the Network Simulator NS 2.31 for creating and testing the proposed network Model [11]. The structure of the NS2.31 is depicted in the following figure.

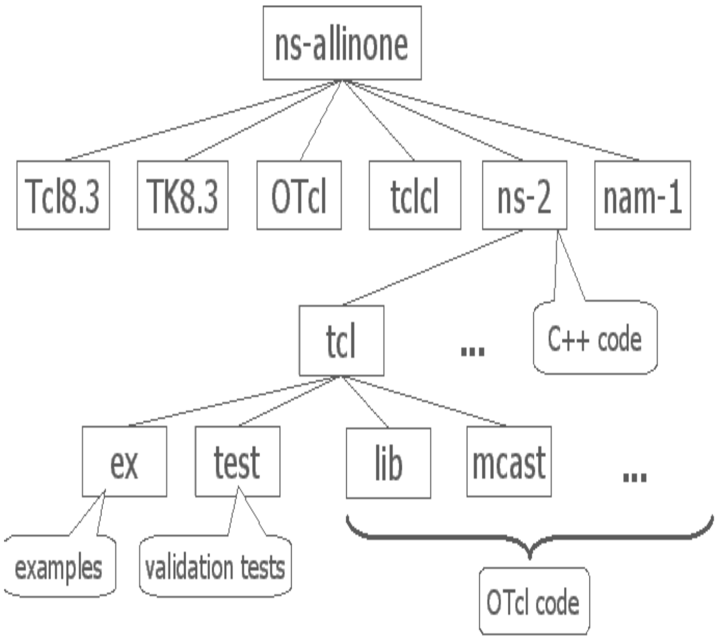


Figure 2: The Structure of NS 2.31

6. 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 (to ½ the current rate of transmission) and behaving sources may have to reduce their sending rates to a low or moderate levels (but not to ½ the current rate of transmission).

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

a. Optimize the Bandwidth 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.

Acknowledgements:

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. The first author further thanks the Management and also the Principal of the East Point College of Engineering & Technology, Virgo Nagar Post, Bangalore – 560 049, India for their support and encouragement extended to him to pursue research in the chosen field of study.

References:

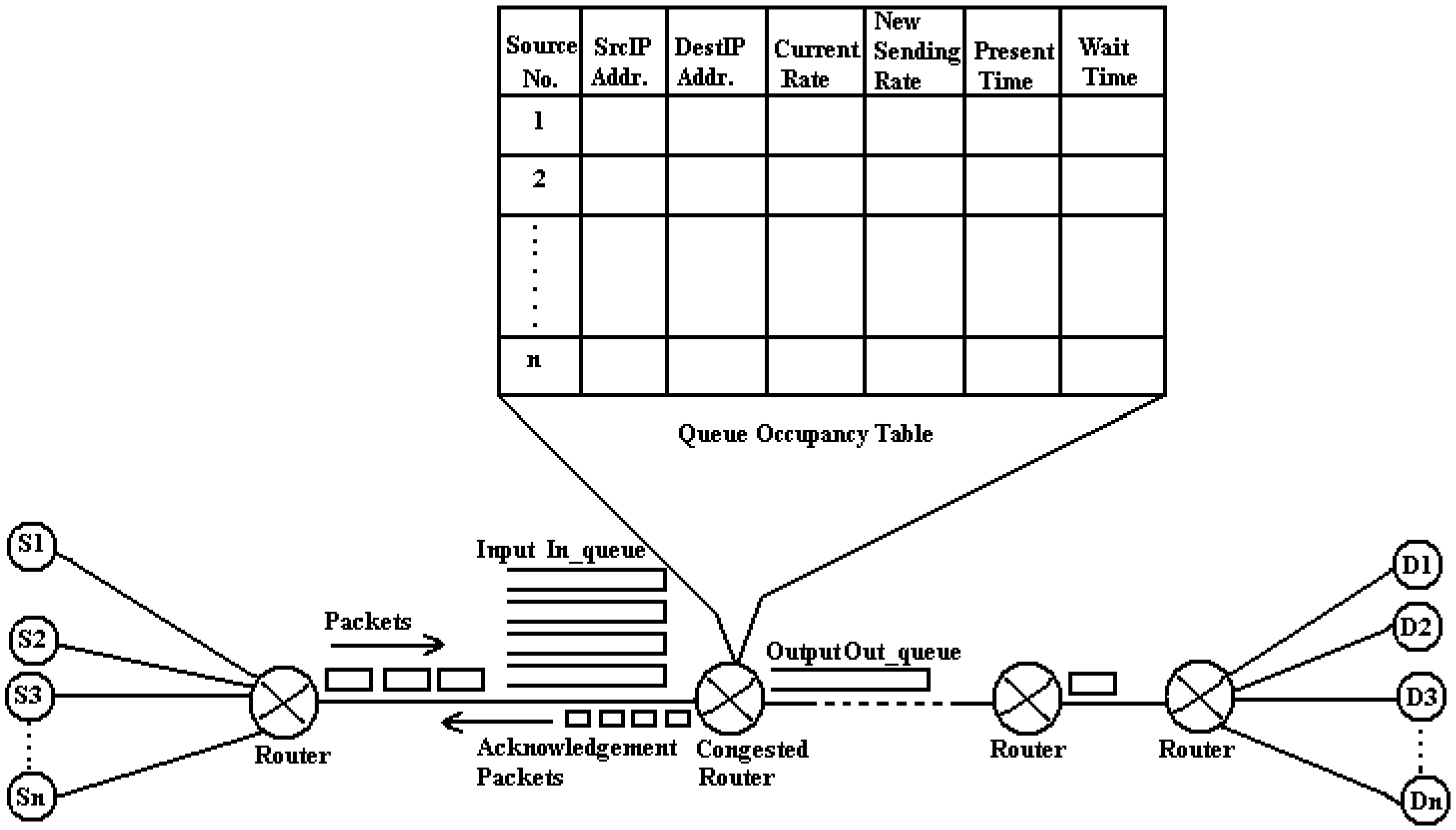
[1] Jacobson, V. Congestion avoidance and control. In Proceedings of SIGCOMM '88, Stanford, CA, Aug. 1988.
[2] Sally Floyd, HighSpeed TCP for Large Congestion Windows and Quick-Start for TCP and IP, Yokohama IETF, July 18, 2002, Available at http://www.icir.org/floyd/hstcp.html
[3] Dina Katabi, Mark Handley, Charlie Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. SIGCOMM '02, Pittsburgh, Pa, Aug. 2002.
[4] Tom Kelly, Scalable TCP: Improving Performance in Highspeed Wide Area Networks, ACM SIGCOMM Computer Communication Review, Feb' 2003. http://.citeseer.ist.psu.edu/kelly03scalable.html
[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, Dec. 6, 2003. http://netlab.caltech.edu/FAST/publications.html
[7] G. Jin, Feedback adaptive control and feedback asymptotic convergence algorithms for measuring network bandwidth. LBNL-53165.
[8] K. Satyanarayan Reddy, C. Lokanatha Reddy "A Survey on Congestion Control Mechanisms in High Speed Networks" has been published in the International Journal of Computer Science and Network Security (IJCSNS) Vol. 8 No. 1, pp. 187 – 195, January 2008. http://paper.ijcsns.org/07_book/200801/20080126.pdf
[9] K. Satyanarayan Reddy, C. Lokanatha Reddy "A Survey on Congestion Control Protocols for High Speed Networks" has been published in the International Journal of Computer Science and Network Security (IJCSNS) Vol. 8 No. 7, pp. 44 – 53, July 2008. http://paper.ijcsns.org/07_book/200807/20080707.pdf
[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] ns the Network Simulator; http://www.isi.edu/nsnam/ns/.
[12] Sumitha Bhandarkar, 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" Columbia University.
[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):169–189, February 1999.

Authors:

[†]**K. Satyanarayan Reddy** received his M.Sc. (Mathematics) & M.Phil. (Mathematics) **D**egrees from Nagpur University, Maharashtra State, India and M. Tech. (Computer Applications) from **I**ndian **S**chool of **M**ines (**ISM**), Dhanbad, Jharkhand, India in 1987, 1988 and 2000 respectively. He is currently associated with Information Science & Engineering department of *East Point College of Engineering and Technology (EPCET), Bangalore, Karnataka State, India. He is a Research Scholar in the Dept. of Computer Science, School of Science & Technology at Dravidian University, Kuppam, AP, India and is pursuing his Ph.D. Degree in Computer Science. His current areas of research are Congestion Control in High Speed Networks and Data Communications.

[††]**Lokanatha C. Reddy** earned M.Sc.(Maths) from Indian Institute of Technology, New Delhi; M.Tech.(CS) with Honours from Indian Statistical Institute, Kolkata; and Ph.D.(CS) from Sri Krishnadevaraya University, Anantapur. Earlier worked at KSRM College of Engineering, Kadapa (1982-87); Indian Space Research Organization (ISAC) at Bangalore (1987-90). He is the Head of the Computer Centre (on leave) at the Sri Krishnadevaraya University, Anantapur (since 1991); and a Professor of Computer Science and Dean of the School of Science & Technology at the Dravidian University, Kuppam (since 2005). His active research interests include Real-time Computation, Distributed Computation, Device Drivers, Geometric Designs and Shapes, Digital Image Processing, Pattern Recognition and Computer Networks.

APPENDIX



Source No.	SrcIP Addr.	DestIP Addr.	Current Rate	New Sending Rate	Present Time	Wait Time
1						
2						
⋮						
n						

Where S1, S2… Sn are sending sources and D1, D2… Dn are the Destination nodes.

The acknowledgement packets are the choke packets informing the source about their new packet sending rates.