LGC: An Active Congestion Control Mechanism[1]

Niraj Prabhavalkar(1), Manish Parashar(1), and Prathima Agrawal(2)
(1) Department of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, Piscataway, NJ 08854. (2) Telecordia Technologies, Morristown, NJ 07960.

Key words: Active congestion control, Active networks, Random early detection.

Abstract: This paper presents the design and evaluation of Limiting Greedy Connections (LGC), an active congestion control mechanism for minimizing the degradation in network performance caused by bandwidth greedy applications. The primary objectives of the LGC mechanism are to limit the impact of greedy connections on a congested node, to keep a loose upper bound on the packet queue occupancy at the intermediate nodes of the network, and to minimize packet loss. The LGC mechanism is evaluated for a variety of network topologies, transmitting sources, and node queue parameters, using a Java-based active network test bed.

1. INTRODUCTION

In spite of large research efforts in industry and academia to eliminate network congestion, the problem continues to persist and grow. Closed-loop congestion control mechanisms have become the norm in the Internet today [2]. In these mechanisms, the network provides negative feedback to the transmitting sources when it is congested or when congestion is building up. They then rely on the transmitting sources to exercise control by cutting back their effect rate of transmission. However, an increasing number of applications such as voice, video, audio and broadcast services require a constant bit rate of transmission while some others tend to "grab" as much network bandwidth as available. These applications

[1] Pubished in "Active Middleware Services," Editors: S. Hariri, C.A. Lee, and C.S. Raghavendra, Kluwer Academic Publishers, pp. 177-187, August 2000.

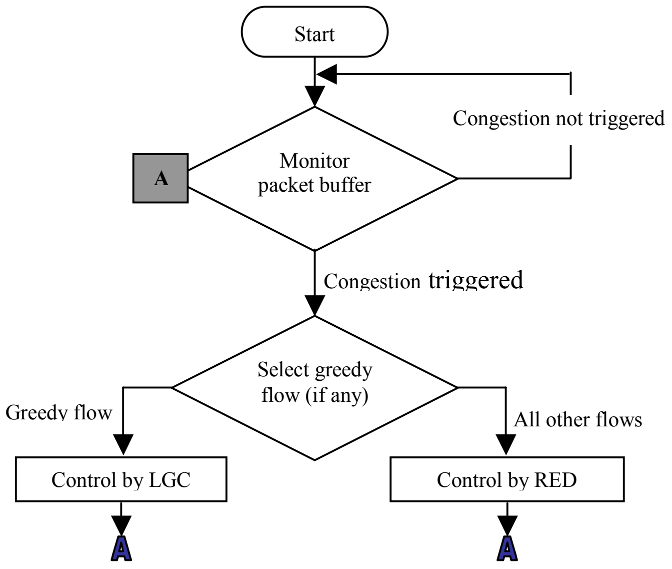
by their very nature tend to ignore or underplay congestion-related feedback from the network.

In recent years, there has been considerable interest in the Random Early Detection (RED) [7] mechanism for congestion avoidance. RED gateways have a FIFO packet queue that is closely monitored to detect the build up of congestion. Based on queue occupancy, the average queue length (*avg*) is computed using a low pass filter with an exponentially weighted moving average. The gateway notifies connections of congestion either by dropping or by marking packets arriving at the gateway. If a packet arrives to a full queue, it is discarded. The RED gateway has two pre-set thresholds called min_{th} (minimum threshold) and max_{th} (maximum threshold). With every arriving packet, the *avg* is computed and compared to these two thresholds. If *avg* is less than min_{th}, arriving packets are not dropped or marked. If the computed *avg* exceeds max_{th}, all arriving packets are marked or dropped. If the computed *avg* lies between min_{th} and max_{th}, the gateway notifies a connection of congestion with a probability that is roughly proportional to that connections share of the bandwidth for the bottleneck link. The primary advantage of RED gateways is that they help in keeping the average queue size low, allow occasional packet bursts and prevent global synchronization of source windows due to the randomness of the RED algorithm in marking or dropping packets at a congested node. However, it has been proven through simulations in [3] that an unresponsive bandwidth greedy connection gets a larger than fair share of the bandwidth at a bottleneck link when competing with responsive connections at a RED gateway. Other congestion avoidance schemes suggested in [3] require multiple queues to be maintained at the intermediate nodes of the network.

Active networks [4] provide a new networking platform that is flexible and extensible at runtime and supports the rapid evolution and deployment of networking technologies to suit current needs. They allow the network nodes to perform application specific computation on the data flowing through them. Although, with active networking the possibilities for refining current applications and introducing new ones are tremendous, it is important to demonstrate the performance benefits accrued from an active networking platform.

In this paper we presents the design, implementation and evaluation of the Limiting Greedy Connections (LGC) congestion control mechanism that uses active network capabilities address the shortcomings of RED and limit the degradation in network performance caused by bandwidth greedy application flows. LGC limits the impact of greedy connections at a congested node by: (1) maintaining a loose upper bound on the buffer queue occupancy at the intermediate nodes of the network, (2) controlling congestion caused by bandwidth greedy applications, (3) providing a negative incentive for greedy flows, and (4) addressing scalability to handle multiple greedy flows. It requires a single FIFO queue to be maintained at the intermediate active nodes and is optimised for a reservation-less active network.

The rest of this paper is organized as follows. Section 2 presents the design of the LGC active congestion control mechanism. Section 3 gives an overview of the LGC implementation. Section 4 presents an experimental evaluation of LGC. Section 5 presents our conclusions.



2. THE LGC CONGESTION CONTROL MECHANISM

Figure 1. An overview of the LGC Congestion Control Mechanism

Limiting Greedy Connection (LGC) is an active congestion control mechanism for minimizing the degradation in network performance caused by bandwidth greedy applications. The primary objectives of the LGC are to keep a loose upper bound on the packet queue occupancy at the intermediate nodes of the network, and to prevent under/over utilization of network resources. The LGC mechanism extends the RED queue management techniques to detect the onset of congestion at an intermediate node. Once congestion is signalled, competing flows are divided into two distinct categories - greedy and non-greedy flows based on the queue occupancy. We rely on RED mechanisms to control the non-greedy flows. A process of recursive mobile packet filtering controls the “greediest” flow. Specifically, we install a packet filter for the identified connection at the congested node and use active messages to dynamically move the filter towards the source of the connection. In doing so, the congested node is relieved of its additional responsibility of filtering packets and network resources from the source of the connection to the congested node are protected from the aggressive flow. If congestion is not controlled despite filtering the greediest flow, the LGC mechanism continues to successively pick out flows in order of their greediness and subjects them to active filtering. Figure 1 presents an overview of the LGC. The key components of the algorithm are outlined below.

2.1 Detection/Isolation of Bandwidth-greedy Flows

When the demand on the network exhausts available resources, the network nodes are the first to be affected. Specifically, when a node is congested its packet queue gets heavily occupied eventually forcing the node to drop packets that overflow the queue. Hence, packet queues at the intermediate nodes in a network are the ideal location for detecting the build up of congestion. We use queue occupancy metrics to detect bandwidth greedy connections.

In [3] Dong et. al. have proved that bandwidth consumption at a bottleneck link is directly related to the queue occupancy of the connection at the node. A connection with a large share of bandwidth consumption on a link has a correspondingly larger share of packet queue occupancy at the node. Furthermore, in RED gateways it has been observed that maximum disparity between queue occupancy for non-greedy and greedy connections occurs when the average queue size (*avg*) exceeds the maximum threshold (max_{th}). At this time the packet queue is about to overflow and we label the node to be in a '*severely congested'* state. In this state, it becomes easier to correctly identify a bandwidth greedy flow at the node.

To identify the greedy connection at a severely congested node, first we need to determine the fair share (*f*) of a packet queue. The fair share in terms of packet queue occupancy (*f*) is given as follows:

f = Total queue occupancy(p) / number of connections in the queue(n) ---- [a]

Consider an active node having a total packet queue occupancy of 75 packets with 5 connections competing for a share of the bandwidth. In this case $f = 75/5 = 15$ packets. Ideally, to ensure a fair distribution of bandwidth, each connection should not have more than 15 packets buffered at the node. However, a responsive connection may also have more than its fair share of packets buffered at the node due to several reasons including (see [1]) the bursty nature of Internet traffic, high delay-bandwidth links on the receive port of the node, and connections being in different phases of operation. We provision for these discrepancies by a factor 'k' > 1. This factor decides the degree of permissible disparity between greedy and non-greedy sources. Selecting a small value of k may cause the algorithm to wrongly classify a responsive source as greedy, while selecting k to be too large will make it nearly impossible for the algorithm to detect a greedy connection. We have empirically selected k to be $\log_e(3n)$. A similar value is chosen in [6] for identifying flows using disproportionate bandwidth. However, that scheme also relies on the characterization of a conformant TCP source based on an assumed value of round trip time for the connection. Our approach for detecting a greedy connection is purely based on the queue occupancy of the connections when a node is severely congested. We use the observations that if the separation between the min_{th} and max_{th} is sufficiently large, *avg* is unlikely to increase from min_{th} to max_{th} before providing ample time for the responsive connections to back off. In this case, when average queue size exceeds the max_{th}, and a large disparity occurs between queue occupancies of competing connections it is safe to assume that the connection with an exceptionally large number of packets buffered at the severely congested node is bandwidth greedy. Continuing with our example, $k = \log_e (3*5)$ or $k = 2.708$. We calculate the responsive share (r) of the packet queue occupancy as:

$$r = \lceil k*f \rceil \text{ ---- [b]}$$

In our example $r = 2.708 *15 = \lceil 40.62 \rceil$. So, in this example, a connection that has at most 41 packets in the queue (i.e. 54.66% of queue occupancy) during its severely congested state is considered responsive. All connections having more than a responsive share of the packet queue are considered unresponsive. Among the unresponsive connections identified above, the one having the maximum number of packets buffered at the severely congested node is singled out as the ‘greedy’ connection. Combining equations [a] and [b] we have:

$$r = \lceil (\log_e(3n))*(p/n) \rceil \text{ ---- [c]}$$

Finally, the percentage permissible queue occupancy (qo) is given as:

$$qo= 100*r/p = 100* \log_e(3n)/n \text{ ---- [d]}$$

Figure 3 shows the percentage permissible queue occupancy (qo) plotted against the number of connections (n) represented in the queue. The slope of the graph is steep for smaller values of n and becomes a gradual decline as n increases. This implies that a larger variation in queue occupancy is permitted when fewer connections cause severe congestion at a node. One anomaly that appears is that for the special case of n=1, a connection will not be classified as greedy even if it exhausts the entire packet buffer at the node. This is in fact necessary to ensure that a single connection will never be filtered, as there is no competing connection.

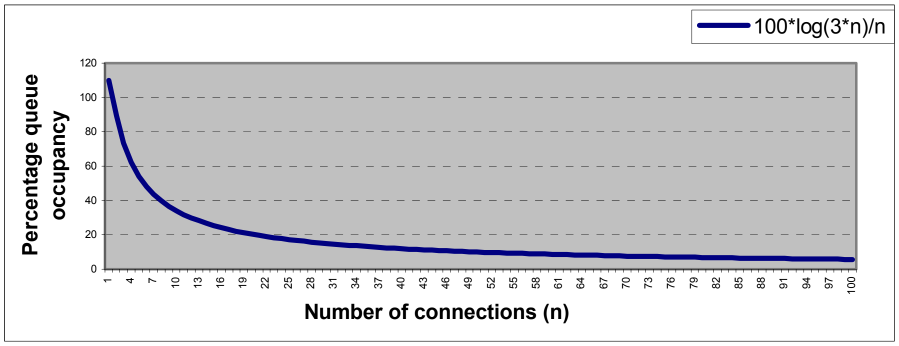


Figure 2. Percentage of permissible queue occupancy (*qo*) v/s number of connections (*n*)

Severe congestion ($avg > max_{th}$) at a node may be caused by greedy application(s) not responding to congestion notification sent by the RED [7] mechanism. If this is the case, the greedy connection(s) is identified by the above. All other flows are assumed non-greedy. Non-greedy, conforming sources either respond to congestion notification or do not make a heavy demand on network bandwidth during congested periods. In the case of non-greedy sources, the control loop is stretched from the congested node to the packet source. We rely on RED [7] mechanisms and the packet source to control the rate at which packets enter the congested node. In the case of greedy connections, stretching the control loop to the packet source is ineffective and hence congestion caused by greedy sources is controlled within the network and not by relying on the greedy sources to cut back their effective rate of packet transmission. We make use of the processing capability of an active network to control these greedy connections as described below.

2.2 Controlling Bandwidth-greedy Flows – Active Filtering

To prevent the severely congested node to degrade into a drop-tail node, it becomes imperative to control the non-conforming bandwidth-greedy flows. We feel that the only effective way to control the inflow of packets from such a greedy connection is by actively filtering packets belonging to the connection. The packet filtering must continue until such a time that the queue occupancy of the packet buffer at the severely congested node is reduced to acceptable levels. Once this happens responsive connections may compete for a fair share of the bandwidth that they were previously denied. A packet filter is first installed at the congested node for the identified greedy connection. The filter is then progressively migrated towards the source of the greedy connection up to the first hop node of the connection. In doing so, the packet drops are made early and reduces the wastage of network resources.

Filtering packets belonging to a flow is a relatively harsh mechanism of controlling congestion but is deemed necessary, considering the damage that can be done to network resources by the (non-conforming) greedy connection. As multiple flows could be identified as bandwidth greedy, we pick out the greediest flow and dynamically filter packets belonging to it. If congestion is not controlled in spite of actively filtering the greediest flow, the algorithm continues to successively isolate and filter the remaining identified greedy flows in the order of their greediness.

3. LGC IMPLEMENTATION

LGC active congestion control has been implemented on the RANI (Rutgers Active Network Initiative) testbed. There are essentially two ways in which an active network can support application oriented processing at intermediate nodes in the network. In the *language-based* approach, the active datagrams carry programs that are executed in a suitable environment. Users are allowed to inject code into the network making the system highly dynamic and flexible. However, special care must be taken to safeguard the system against malicious users and buggy code. In the *menu-based* approach, the active node supports a fixed set of services. Designated operators may add new services into the node. Active datagrams carry a reference to the type of servicing they require. The implementation details of services are hidden from end user applications. We believe that the menu-based approach gives a strict administrative control over the services that the network can offer and provides a secure infrastructure at the cost of reduced dynamism. Thus, we adopt the menu-driven approach in designing our active network for the evaluation of LGC.

3.1 RANI (Rutgers Active Network Initiative) Testbed

The RANI network consists of a number of active nodes connected to each other via virtual links. For simplicity, we assume that the virtual links are reliable in delivering datagrams. Any node can communicate with other nodes in the network by sending datagrams across the virtual links. Datagrams that do not need active processing are referred to as passive datagrams. Passive datagrams are simply stored and forwarded similar to traditional network forwarding. Datagrams that request additional processing at the intermediate nodes in the network are called active datagrams. Each datagram is considered an atomic element and is processed individually by the active nodes.

The datagram header comprises of a few additional fields as compared to an IPv4 header. These include a previous node visited (*PrevNode*) field, which carries the IPv4 address and port number of the last active node visited by the packet. Active servicing is requested through a type of service (*TOS*) field in the header of the active packet. The time to live (*TTL*) field is modified to represent a time-based upper bound on the life of a packet in the RANI network. Lastly, an active (*Act*) field is set to true if the packet is active and is false otherwise.

The active node is implemented in Java (v1.1) as a user space process on the Windows NT operating system. The node runs at the application layer in the TCP/IP protocol stack. Application oriented processing of active packets may be required at the end nodes as well as intermediate nodes in the network. Thus, we do not distinguish between intermediate nodes and end nodes. Virtual links are implemented as a UDP (User Datagram Protocol) socket pair – one socket is used for receiving datagrams and the other for sending them. Active or passive packets are created and subsequently injected into the active network via the user interface at the node. These packets are propagated as UDP segments in the RANI network.

3.2 Mobile Filtering

The process of mobile filtering begins with the congested node extracting a packet belonging to the greedy connection from its packet buffer. This packet reveals the source of the greedy connection. A *greedy connection identifier* (*GCI*) consisting of the source IP address and port number is formed. Next, the virtual link object connecting the congested node to the greedy source is obtained from the routing table using the *GCI*. The node uses the *GCI* to create a packet filter on the receive port of this virtual link. The packet filter is installed for time *ITime* and drops packets originating from the identified greedy connection. The virtual link object reveals the active node to which it connects. The IP address and port number of this active node is called the *previous hop identifier* (*PHI*). The node then creates and sends an active packet destined for the previous hop requesting the *ActiveFilter* service. This packet contains the *GCI* of the connection to be filtered in its payload. On receiving this message, the previous node invokes the process routine of its *ActiveFilter* service. Execution of this routine at this node involves installation of a packet filter for the greedy connection and propagation of the active filter message to the next node in the path of the connection that is closer to the greedy source. This process continues until the first hop node for the greedy connection is reached. A minor technicality overlooked in the example above was the assumption that a node can automatically learn if it is the first hop node and stop propagating the mobile filter. This is because prior to creating the active filter message each active node performs a previous hop check. The check consists of a comparison of the *GCI* and the PHI fields. If they match it means that the filter has reached the first hop node for the greedy connection. The packet filter is then installed for a longer duration of time *FHTime* and the node does not propagate the active filter message any further. Sending the active filter message to the source of a greedy connection would be futile as source is already found to be un-responsive.

Once a greedy connection is identified and filtered at the congested node, the packet queue occupancy is expected to drop. However, due to the low pass filtering mechanism used in the calculation of *avg*, its value might continue to be greater than max_{th} even after the queue occupancy has decreased. This will again trigger the LGC algorithm. To ensure that LGC is not triggered multiple times in a short interval of time, a minimum idle period, *Tx*, is chosen between two consecutive triggers of LGC.

The selection of the two timing parameters, *ITime* and *FHTime*, is critical to the successful of the active filter mechanism. The *ITime* parameter is based on the time taken for migration of the active filter from an intermediate node to the previous node in the path of the greedy connection. If its value is pre-set to a high value, both these nodes will suffer the overheads of actively filtering a greedy connection. If its value is too low, active filtering at the intermediate node will terminate before it begins at the previous node. From empirical measurements on the RANI network, we set this parameter to approximately five seconds.

The *FHTime* parameter is based on the time it is desired to actively filter a greedy connection at its first hop node. If *FHTime* is too small, the unresponsive connection is not be filtered for a sufficiently long period and may congest the network again. If it is too large, the connection may close but the packet filter will continue to exist adding unnecessary overheads at the node at which it is installed. We set *FHTime* to about 100 seconds in the RANI test bed.

4. EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate the ability of the LGC algorithm to correctly identify and filter a greedy connection. Since the LGC algorithm is triggered only during severe congestion, in our experiment we first force a node into severe congestion. Note that in order to reduce complexity, only necessary components of the RED algorithm were implemented.

The evaluation was conducted using the RANI testbed running on Intel Pentium II 300 MHz processor machines interconnected via a 10BaseT Ethernet LAN at the data link layer. The RANI network was built on the Windows NT operating system substrate. To bring out the effectiveness of the LGC algorithm we simulate responsive and unresponsive connections. The sources are simulated with the help of a packet generator that can be selected to behave as a non-greedy or a greedy source. The transport mechanism for a non-greedy connection is simulated as a rough approximation of a TCP source. A detailed explanation of the implementation of TCP can be found in [5]. The transport mechanism for a greedy connection is simulated by a constant packet-rate source.

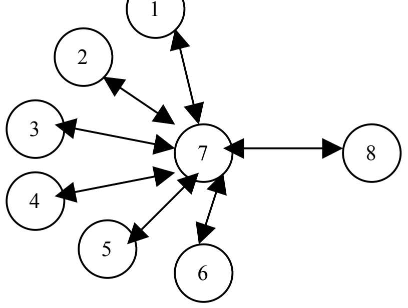


Figure 3. Network topology for experiments

The test active network consisting of six non-greedy sources, one greedy source, one interconnecting node and a sink node is shown in Figure 3. Node 1 is the greedy source and nodes 2,3,4,5,6 and 7 are non-greedy sources. Node 7 is targeted for severe congestion. Node 8 is the common sink for all the sources. Virtual links are shown as double-ended arrows. Node 7 is forced into a severely congested state by having all the sources transmit packets at approximately the same time. To prevent packet drops due to expiration of the TTL field, all packets injected into the network have a large value of initial TTL (10 sec.). The queue parameters for node 7 are set with queue *weight* = 0.02, max_{th} = 25 and buffer size = 50. The non-greedy sources inject 50 packets each with an initial TCP slow-start threshold set to 16. The greedy source injects 200 packets in 5 bursts with an inter-burst duration of 1 second.

The observed results are shown in Figure 4, In the figure the x-axis shows the packets arriving at node 7 and the y-axis shows the queue size measured in packets. The solid line (y = 25) represents the configured value of max_{th} at the node. Notice that the low pass filtering mechanism in calculation of average queue size in RED causes *avg* to change slowly in comparison to the actual queue size. For brevity, the first few packet arrivals have been omitted in Figure 5. Initially as the non-greedy sources open up their windows, the actual queue size remains low (<10). Once the competing sources have sufficiently large windows, the actual queue size increases rapidly. When the average queue size crosses max_{th} (25) at point A, the LGC algorithm is triggered.

From the nodes packet queue we observe that the total queue occupancy is 39 packets. Of these 21 packets belong to connection 1, 4 packets belong to connection 2, 5 packets belong to connection 3, 3 packets belong to connection 4 and 2 packets each to belong connections 5, 6 and 7. Totally there are seven active connections at node 7. Fair queue occupancy is 39/7 = 5.57. With a permissible factor *k* of $\log_e(3*7)$, the permissible queue occupancy is $\lceil 5.57 * 3.0445 \rceil$ = 17 packets. Connection 1 has 21 packets in the node queue and is correctly identified as a greedy connection. Since Node 7 is the first-hop node for this connection, the migration of the packet filter was not necessary and a packet filter for connection 1 was installed at Node 7 for a duration *FHTime* (100) seconds. Subsequently all packets arriving from connection 1 were filtered out at node 7. The throughput for non-greedy connections is observed to be 100% after the LGC algorithm came into effect, but the greedy connection had a throughput of 53.5% due to active filtering at node 7.

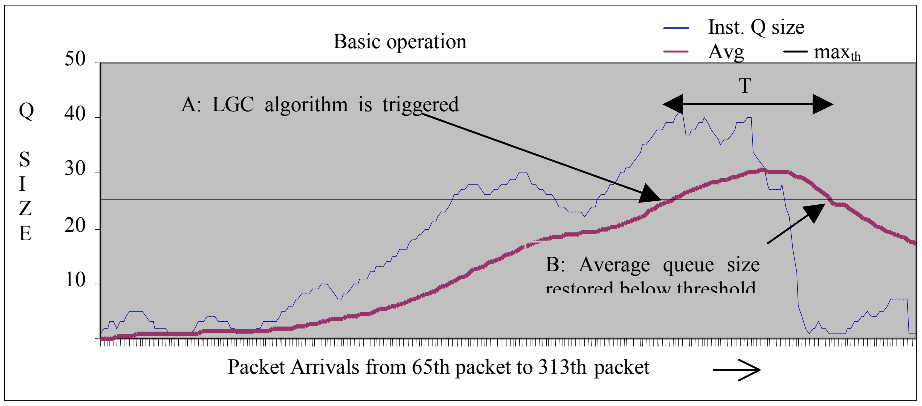


Figure 4. LCG - Basic Operation

Due to the bandwidth greedy nature of connection 1, we observe a sudden drop in the queue occupancy once this connection is filtered. This can be observed in the region of the graph just after the LGC algorithm is triggered. Eventually the queue size is controlled at point B. The time lapse (marked as T in Figure 5) between the LGC algorithm coming into effect (point A) and the reduction in average queue occupancy (point B) occurs due to the low pass filtering mechanism in the calculation of the average queue size. It confirms the requirement for the presence of an idle time (Tx > T) between two successive triggers of the LGC algorithm. If the LGC algorithm were not suspended for time Tx, it would be triggered multiple times since *avg* exceeds max_{th} for duration T, despite active filtering of the greedy connection.

5. CONCLUSIONS

In this paper, we presented the design, implementation and evaluation of the LGC active congestion control mechanism. LGC uses active network capabilities to address the shortcomings of RED and limit the degradation in network performance caused by bandwidth greedy application flows. A process of recursive, active mobile filtering is used to throttle non-conformant bandwidth-greedy flows close to the source. This relieves the congested nodes and minimizes wasted network resources. Experimental results validate the utility of the LGC mechanism in limiting the effects of bandwidth-greedy connections.

6. REFERENCES

1. B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, *Recommendations on Queue Management and Congestion Avoidance in the Internet*, Request for Comments: 2309, April 1998.
2. C. Yang and A. V. S. Reddy, *A Taxonomy for Congestion Control Algorithms in Packet Switched Networks.* IEEE Network Magazine July/August 1995, Volume 9, Number 5.
3. D. Lin and R. Morris, *Dynamics of Early Detection*. Proceedings of ACM SIGCOMM 97 Conference, Cannes, France, September 1997.

4. D. L. Tennenhouse, J. M. Smith, W. David Sincoskie, David J. Wetherall and Gary J. Minden, *A Survey of Active Network Researc*h. IEEE Communications Magazine, January 1997, pp. 80-86.
5. Information Sciences Institute, University of Southern California, *Transmission Control Protocol, Request for Comments: 793*. September 1981.
6. S. Floyd and K. Fall, *Promoting the Use of End-to-End Congestion Control in the Internet*. IEEE/ACM Transactions on Networking, Vol. 7, Issue 4, pp. 458-472, Aug. 1999.
7. S. Floyd and V. Jacobson, *Random Early Detection Gateways for Congestion Avoidance*. IEEE/ACM Transactions on Networking, Vol. 1, No. 4, pp 397-413, Aug. 1993.