Controlling Unresponsive Connections in an Active Network Architecture

Niraj Prabhavalkar and Manish Parashar

The Applied Software Systems Laboratory
Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey
94 Brett Road, Piscataway, NJ 08854
parashar@caip.rutgers.edu

Abstract: *This paper presents the design, implementation and evaluation of Limiting Greedy Connections (LGC), an active mechanism for controlling unresponsive connections and 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 testbed.*

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

1. INTRODUCTION

Network congestion continues to persist and grow in spite of continued research efforts in industry and academia to eliminate it. Closed-loop congestion control mechanisms have become the norm in the Internet today [1]. In these mechanisms, the network provides negative feedback to the transmitting sources when it is congested or when congestion is building up. The mechanisms then rely on the transmitting sources to exercise control by cutting back their effective rate of transmission. However, an increasing number of applications such as voice, audio and broadcast services require a constant bit rate of transmission, while some others, e.g. streaming and multimedia applications, Internet telephony and fast data exchange services, tend to "grab" as much network bandwidth as available. These applications by their very nature tend to ignore or underplay congestion-related feedback from the network.

The Random Early Detection (RED) [2] mechanism for congestion avoidance helps keep the average queue size low, allows occasional packet bursts, and prevents global synchronization of source windows due to its randomness in marking or dropping packets at a congested node. However, it has been proven through simulations [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 an innovative 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 active networking provides tremendous potential for refining current applications and introducing new ones, it is important to demonstrate the performance benefits accrued from an active networking platform.

In this paper we present the design, implementation and evaluation of the Limiting Greedy Connections (LGC) congestion control mechanism that uses active network capabilities to 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 to 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 optimized for a reservation-less active network.

The rest of this paper is organized as follows. Section 2 presents work related to active network technologies. Section 3 presents the design of the LGC active congestion control mechanism. Section 4 gives an overview of the LGC implementation. Section 5 presents an experimental evaluation of LGC. Section 6 presents our conclusions.

2. BACKGROUND AND RELATED WORK

2.1 Active Networks:

In active networks, the network is no longer viewed as a passive mover of bits, but rather as a more general computational engine: information injected into the network may be modified, stored or redirected while it is being transported. The active network architecture adopted has a direct bearing on its utility and the applications that it can support. Proposed active network architectures include: (1) Smart Packets [5], being developed by BBN Technologies, (2) Active Node Transfer System (ANTS) [6], a continuing research effort of the Software Devices and Systems group at the MIT Laboratory of Computer Science, (3) SwitchWare [7], the active networks research effort at the Department of Computer and Information Science at University of Pennsylvania and Bellcore [8], (4) Composable Active Networks Elements (CANES) [9], the research project at Georgia Institute of Technology, and (5) NetScript [10], a programming language and environment for building networked systems.

There are primarily two ways in which the active network can support processing at intermediate nodes in the network. In the *language-based* approach the active datagrams carry programs that are executed in a suitable environment at the nodes. 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. Hence, we adopt the menu-driven approach in designing our active network.

The current active network architectures are in the developmental phase and a consensus on a standard architecture has still not been reached. Table 1 shows a comparison of the active network architectures described above. Note that the list of contributions and applications is not exhaustive. The Rutgers Active Networks Initiative (RANI) network, which serves as the testbed for our simulation, is presented in Section 4.

Table 1 – Comparison of active network architectures

Architecture	Approach	Key Contributions	Applications
Smart Packets	Language-based	Mobile agents	Network management and diagnostics
ANTS	Language-based	Application specific protocol development	Distributed applications and web caching
SwitchWare	Language-based	Programming language development, network security	Active bridges, bootstrap architectures
CANES	Menu-based	Active components	WAN caches, selective packet treatment
NetScript	Language-based	Designing scripts, mobile agents	Management by delegation
RANI	Menu-based	Design and implementation of an active network testbed	Controlling bandwidth greedy connections, Active Traceroute

2.2 Mechanisms for Managing Network Congestion and Unresponsive Connections:

In this paper we consider the following definitions for network congestion. They are based on the user's perspective and on a demand-supply relationship in the network.

- **A network is said to be congested from the perspective of user *i* if the utility of *i* decreases due to an increase in the network load** [12] (where utility refers to the user's preference for a set of resources). In this definition, congestion is defined as an end-user perception of the state of the network. If the utility of the network remains unaffected for a specific user, even under highly loaded conditions, the network is not congested for that user. However if the utility of the network is adversely affected for other users, they will perceive the network as congested.

- **If, for any interval of time, the total sum of demands on a resource is more than its available capacity, the resource is said to be congested for that interval** [12]. This definition uses a demand-supply relation to identify congested periods in the network. The demand consists of delivering information from end-to-end and satisfying user constraints such as allowable delays and reliability. The supply includes, but is not limited by, network resources such as buffer space, link bandwidth and processor speed. The network is not considered congested if all demands are met. Note that, as shown in [12], congestion is in fact *worsened* by an ad-hoc increase in these network resources. Rather than controlling congestion by increasing supply to match demand, a sound design strategy is needed to minimize the effects of congestion.

In the following subsections we discuss two relevant schemes for congestion avoidance: Random Early Detection (RED) [2] and Explicit Congestion Notification (ECN) [11]. The RED algorithm signals congestion by marking or dropping packets at a gateway or a router. ECN is a specific implementation of RED in which packets are marked to minimize packet loss during congestion at the gateway. RED has been proven to be ineffective in controlling bandwidth greedy connections as explained below.

2.2.1 RED (Random Early Detection) gateways

RED gateways have a 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 at a full queue, it is discarded. The gateway has two pre-set thresholds – min_{th} (minimum threshold) and max_{th} (maximum threshold). For every arriving packet, *avg* is computed and compared to these two thresholds. If *avg* is less than min_{th}, arriving packets are neither dropped nor marked. If *avg* exceeds max_{th}, all arriving packets are marked or dropped. If *avg* lies between min_{th} and max_{th}, the gateway notifies a connection of congestion with a probability that is approximately proportional to that connection's share of the bandwidth through the gateway. The average packet queue length, i.e. *avg*, is computed as follows:

$$avg = (1-w)*avg + w*q$$

where

$w < 1$ is a queue weight that determines the degree of burstiness permissible by the gateway

q is the number of packets in the queue

The value of *avg* is computed at every packet arrival at the gateway. However in RED, when a packet arrives at an empty queue ($q = 0$), *avg* is calculated differently. The gateway first calculates the idle time for the packet queue as the difference between the time at which the packet arrived and the time at which the queue length became zero. The average packet queue (*avg*) is then computed as if the gateway had transmitted *m* packets during the idle time. The factor *m* is linearly dependent on the time for which the queue was idle. Thus for an empty queue,

$$m = f\,(time - q\,(time))$$

$$avg = ((1 - w)**m) * avg$$

where

q (time) is the time at which q became zero

time is the time at which a packet arrives to the empty queue

time – q(time) is the idle time of the packet queue

f ( ) is a linear function representing the rate at which the packet queue is drained

A detailed explanation of the RED algorithm can be found in [2].

2.2.2 ECN (Explicit Congestion Notification) capable gateways

Explicit congestion notification [11] is a mechanism that notifies transmitting sources of incipient congestion by setting a bit in the IP header of the packet (i.e. packet marking). When the marked packet reaches its destination, congestion notification is echoed back to the sender via the acknowledgement packet. The sender is then expected to cut back the packet transmission rate. The end hosts must be capable of responding to marked packets, i.e. they must be ECN capable, for the scheme to work.

3. The LGC Congestion Control Mechanism

Limiting Greedy Connections (LGC) is an active mechanism for controlling unresponsive connections and minimizing the degradation in network performance caused by bandwidth greedy applications. This section describes the design and operation the LGC algorithm. The primary objectives of LGC 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. Key design requirements include.

- The algorithm must be simple and easily deployable in the Rutgers Active Network Initiative (RANI) testbed. Congestion leads to performance degradation of a network. Deploying a complex algorithm would amount to consuming network resources at a time when resources are scarce.
- The designed algorithm must be efficient and effective. An efficient algorithm would have minimal overheads. The effectiveness of the algorithm must be verified through experimentation.
- The algorithm must accurately detect bandwidth greedy connections at a congested node. It is important to note that unresponsive connections are not necessarily bandwidth greedy. If that were the

case our algorithm would restrict all UDP connections in the active network. Our aim is to limit the degradation in network performance caused by transport mechanisms that tend to either increase or maintain their effective rate of transmission of packets, despite being asked to cut back during periods of congestion.

- The algorithm must provide a negative incentive to greedy connections in order to limit the growth of applications that promote them.
- The algorithm must scale well. It should be capable of handling multiple greedy connections through a congested node.

3.1 A High-level Design of the LGC Algorithm:

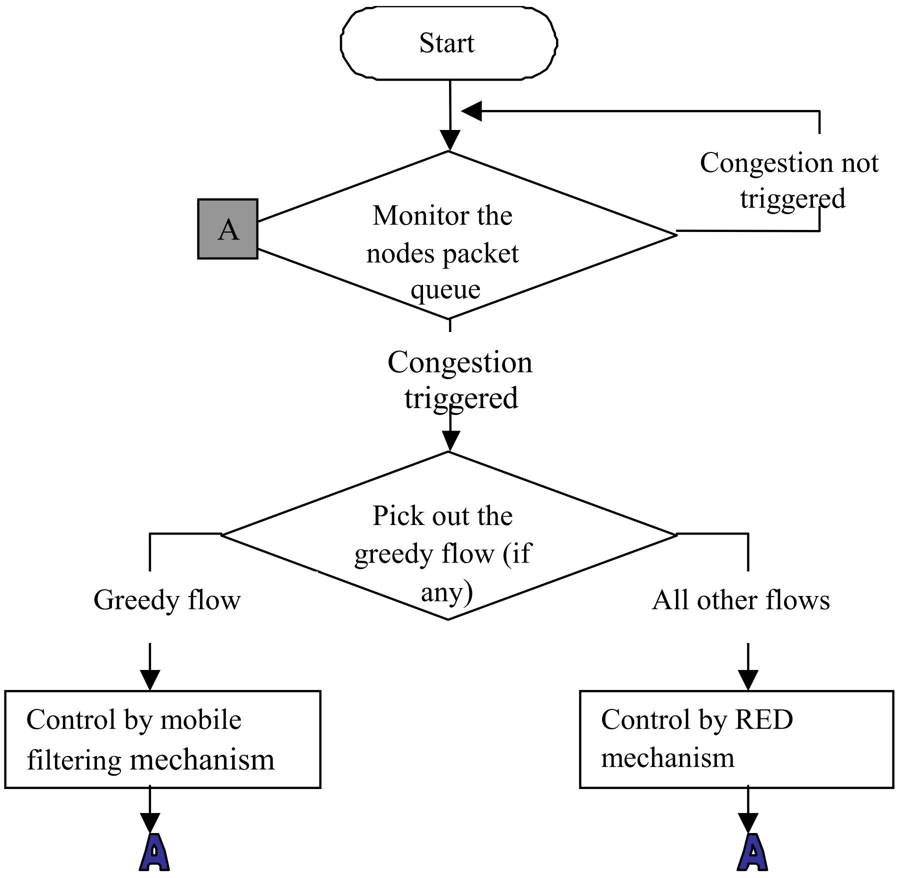

Figure 1 – *An overview of the LGC Congestion Control Mechanism*

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 or 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 detected at a node, the 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 "greedy" flows. Specifically, we install a packet filter for identified "greedy" connections at the congested node and use active messages to dynamically move the filter towards the source of the connection. This relieves the already congested node of the responsibility of filtering packets. Furthermore, it protects the network resources between the source of the connection and the congested node from the aggressive flow. If congestion is not controlled in spite of filtering the greediest flow, the LGC mechanism continues to successively pick out flows in the order of their greediness and subjects them to active filtering. Figure 1 presents an overview of the LGC mechanism. The key components of the algorithm are described below.

3.1.1 Detection and Isolation of Bandwidth-greedy Flows

Network nodes are the first to be affected when the demand on the network exhausts available resources and the network gets congested. 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 the 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 as being 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, we first need to determine the fair share (*f*) of a packet queue. The fair share in terms of total packet queue occupancy (*p*) and the number of connections in the queue (*n*) is given as follows:

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

For example, consider an active node having 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 the bandwidth, each connection should not have more than 15 packets buffered at the node. Note that note all connections with more that this fair share of packets are non-responsive. A responsive connection may have more than its fair share of packets buffered at the node due to several reasons [13] including 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, $k > 1$. This factor determines the degree of permissible disparity between greedy and non-greedy sources. Selecting a small value of k may cause the algorithm to incorrectly classify a responsive source as greedy. On the other hand, 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 [14] for identifying flows using disproportionate bandwidth. However the scheme in [14] 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 observation 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 and unresponsive. 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 *"greediest"* 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 appears for the special case of n=1. In this case 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 are no competing connections.

Severe congestion ($avg > max_{th}$) at a node, caused by a greedy application(s) not responding to congestion notification sent by the RED mechanism, are identified as described 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 these sources, the control loop is stretched from the congested node to the packet source. We rely on RED 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 without 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.

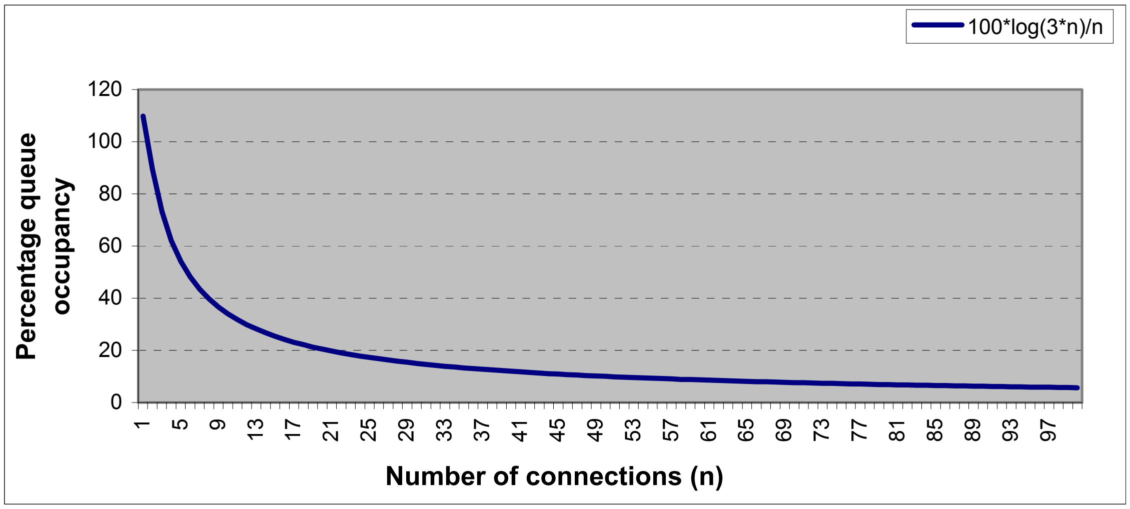

Figure 2 – *Percentage of permissible queue occupancy (qo) v/s number of connections (n)*

3.1.2 Controlling Bandwidth-greedy Flows – Active Filtering

To prevent the severely congested node from degrading 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 thus reducing 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 despite actively filtering the greediest flow, the algorithm continues to successively isolate and filter the remaining identified greedy flows in the order of their greediness.

4. LGC IMPLEMENTAION

LGC active congestion control has been implemented on the RANI (Rutgers Active Network Initiative) testbed described below. We use the menu-based approach in designing our active network for the evaluation of LGC, since it enables strict administrative control over the services that the network offers and provides a secure infrastructure, although at the cost of reduced dynamism.

4.1 RANI (Rutgers Active Network Initiative) Testbed

The RANI network simulator 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. A node can communicate with other nodes in the network by sending datagrams across the virtual links. Datagrams may be marked as either active or passive. Datagrams that do not need active processing are marked 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 marked as active datagrams. Each datagram is considered an atomic element and is processed individually by the active nodes.

The datagram header consists of a few fields in addition to the 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 as opposed to placing a limit on the hop count in traditional networks. Finally, the active (*Act*) field is set to true for an active packet and is false otherwise.

The active node is implemented on the Java (v1.1) platform 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 an 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 datagrams in the RANI network.

4.2 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 IP address and port number of the source 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 a pre-determined interval of time called *Itime* (*I*ntermediate node filter installation *time*) and drops packets originating from the identified greedy connection. The virtual link reveals the active node to which it connects. The IP address and port number of this active node is called the *PHI* (*P*revious *H*op *I*dentifier). The node then creates and sends an active packet destined for the previous hop requesting active filtering (*ActiveFilter)* service.

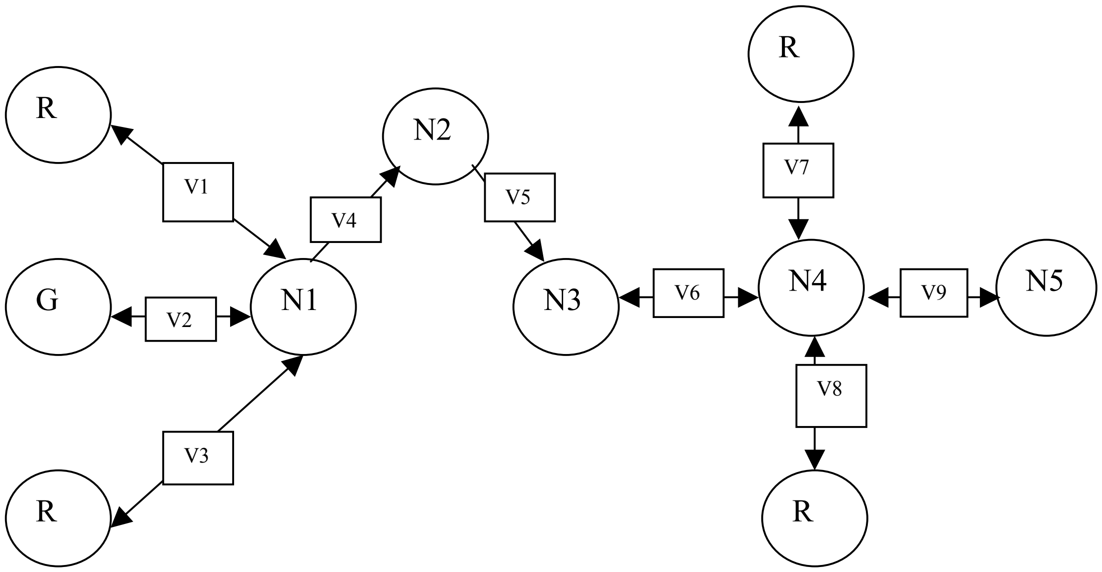

Figure 3 – Mobile filtering mechanisms

For example (see Figure 3) consider a greedy connection G (identified by the procedure described in section 3.1.1) at the severely congested node N4. This connection competes with four other responsive sources (R) for link V9. Node N4 extracts a packet belonging to greedy connection G from its packet queue and forms the *GCI*. Using this *GCI* and looking up its routing table, node N4 learns that the packet was received over link V6 which is connected to node N3. Node N4 first creates a packet filter on the receive side of link V6 to drop packets belonging to greedy connection G. It then sends an active packet carrying the *GCI* and requesting active filter service to node N3, the previous hop node for the identified greedy connection G. Node N3, on receiving the active filter message similarly installs a packet filter for the *GCI* and propagates the active filter message to the next hop closer to G's source, i.e. to node N2.

This process continues till the first hop node N1 for the greedy connection is reached. A node determines whether it is the first hop node and stop propagating the mobile filter as follows. 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 connection G. The packet filter is then installed for a longer duration of time *FHTime* (*F*irst *H*op filter installation *time*) 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. Continuing with the above example, when the active filter message reaches node N1, both *PHI* and the *GCI* are set to G and the filter is not propagated any further.

Once a greedy connection is identified and filtered at the congested node, the packet queue occupancy is expected to drop. However, the value of *avg* changes gradually as compared to the instantaneous queue length, closely following a low pass filter mechanism. As a result, its value may continue to be greater than max_{th} even after the queue occupancy has decreased. This will once again trigger the LGC algorithm. To ensure that LGC is not triggered multiple times within 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 success of the active filter mechanism. The *ITime* parameter is based on the time taken for the active filter to migrate from an intermediate node to the previous node along the path of the greedy connection. If its value is set too high, both these nodes will suffer the overheads of actively filtering a greedy connection. If its value is set 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 about five seconds.

The choice of *FHTime* determines the period for which the greedy connections actively filtered at its first hop node. If *FHTime* is too small, the unresponsive connection will not be filtered for a sufficiently long period and may congest the network once 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.

The LGC algorithm is summarized in Appendix 1.

5. 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 all our experiments 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 running Windows NT and interconnected via a 10BaseT Ethernet LAN. The experiments presented below simulated both responsive and unresponsive connections and demonstrate the effectiveness of the LGC algorithm. Sources are simulated using 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 TCP source. A detailed explanation of the TCP protocol can be found in [15]. The transport mechanism for a greedy connection is simulated as a constant packet-rate source.

5.1 *Experiment 1 – Basic Operation*

In this experiment we test the ability of the LGC algorithm to correctly identify and filter a greedy connection. The test network consisting of six responsive sources, one greedy source, one interconnecting node and a sink node is shown in Figure 4.Node 1 is the greedy source and nodes 2,3,4,5,6 and 7 are responsive sources. Node 7 behaves as a responsive source and 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 an initial TTL of 10 seconds. The queue parameters for node 7 are set with queue weight = 0.02, max_{th} (Upper threshold) = 25 and buffer size = 50. The responsive 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.

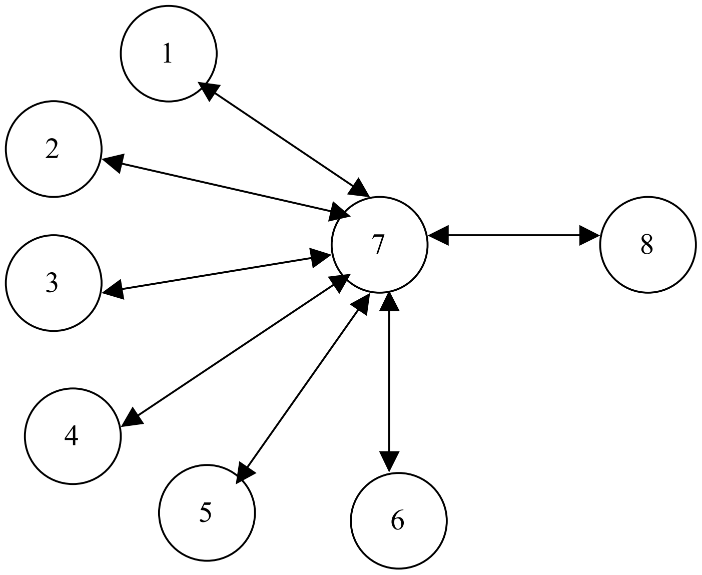

Figure 4 – Network topology for experiments

In Figure 5, 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 of RED causes the average queue size 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 responsive sources open 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} viz. 25 in this case, the LGC algorithm is triggered. From the node's 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. In all, there are seven active connections at node 7. Fair queue occupancy is 39/7 = 5.57. With a permissible factor k of

$\log_e(21)$, 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 an unresponsive 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.

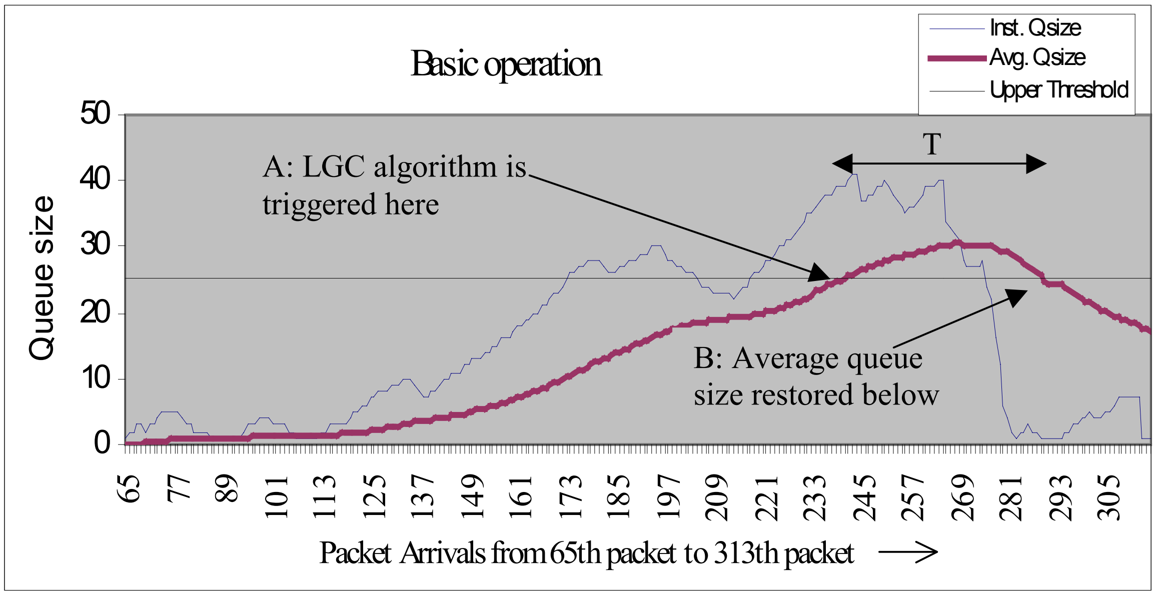

Figure 5 – LCG - Basic Operation

Subsequently all packets arriving from connection 1 were filtered out at node 7. The throughput for responsive connections was 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. Note that if the RED algorithm were implemented in its entirety, the throughput observed for the responsive sources would be less than 100%, since the algorithm would drop all packets arriving at the node when it is severely congested. However, this detail is overlooked in the evaluation of LGC since RED is not implemented in its entirety i.e. arriving packets at the node under severe congestion are not dropped or marked. We only wish to isolate the greedy connections and dynamically filter them to prove that the algorithm is successful.

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 the 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.

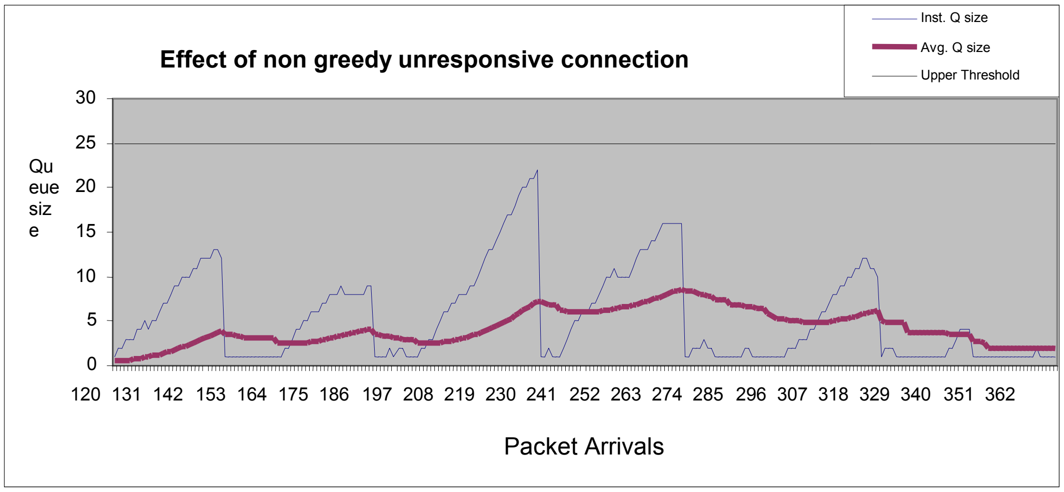

Figure 6 – Plot of queue size v/s packet arrivals for non-greedy connections

The LGC algorithm must also ensure that non-greedy unresponsive connections must not be filtered. To verify this requirement we repeated the above experiment with source node 1 injecting 200 packets in 15 bursts with an inter-burst duration of 3 seconds. Node 1 now simulates a constant packet-rate source making a moderate demand on network bandwidth at the congested node 7. Figure 6 shows the actual and average queue sizes plotted against packet arrivals at node 7. The bursty nature of the connections causes the spikes in the value of instantaneous queue size at the intermediate node 7.

Here, we observe that the average queue size at node 7 remains below 10 at all times implying that demand on resources does not exceed supply. Thus node 7 does not get congested and LGC is not triggered. Since queue occupancy remains low (<25), there is no packet loss and throughput is 100% for all the seven connections.

5.2 Experiment 2 – Mobility of The Active Filter

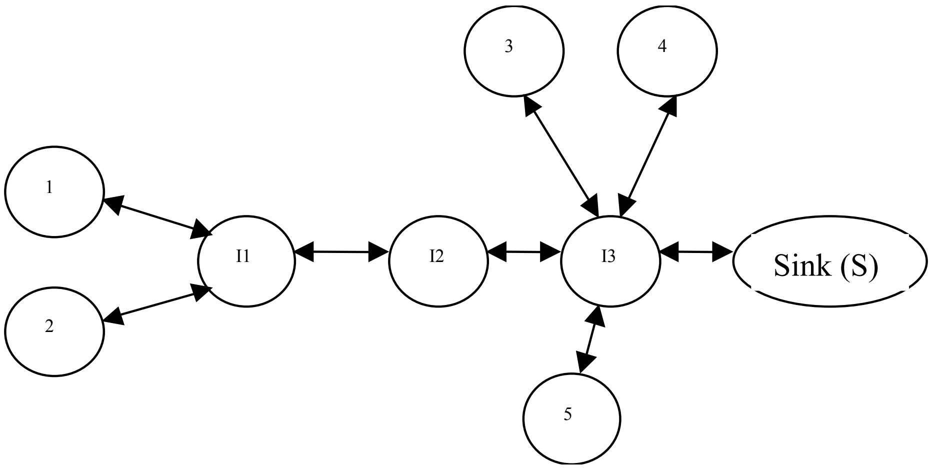

Figure 7 – Network Topology for Experiment 2

After the LGC algorithm identifies and filters a greedy connection at the congested node, it uses active messages to move the filter dynamically towards the source of the identified connection. In doing so, packets belonging to the greedy connection are filtered ‘closer’ to their source, thereby reducing the wastage in network resources. In this experiment we study the movement of the mobile filter towards the source of the greedy connection and the effect of installing the mobile filter at a node. The network topology for the experiment is shown in Figure 7.

Node 1 is an unresponsive packet source where as nodes 2, 3, 4 and 5 are responsive packet sources that provide cross traffic to congest I3. Nodes I1, I2 and I3 are interconnecting nodes that forward packets. Node S is the sink for all the packet sources. Node I3 has a buffer size of 60, $\max_{th}$ set to 35 and w set to 0.02. The responsive sources inject 250 packets with an initial TCP slow-start threshold set to 32. The unresponsive source injects 250 packets in 5 bursts spaced 200 milliseconds apart.

In order to monitor the packet flow in the network we label the packets from the various sources as follows. Packets from source 1 are labeled a1 through a250. Packets from source 2 are labeled b1 through b250. Packets belonging to sources 3, 4 and 5 are labeled similarly.

First, we consider the activities at node I3. In Figure 8, the x-axis shows the packets arriving at node I3 and the y-axis shows the queue size measured in packets. For brevity, the first few packet arrivals have been omitted in the chart. When the average queue size crosses 35, the LGC algorithm is triggered. At this time, I3 had received and forwarded 122 packets belonging to source 1, 84 packets belonging to source 2, 91 packets belonging to source 3, 85 packets belonging to source 4 and 66 packets belonging to source 5, making a total of 448 packets. This is shown by the dotted line in Figure 8.

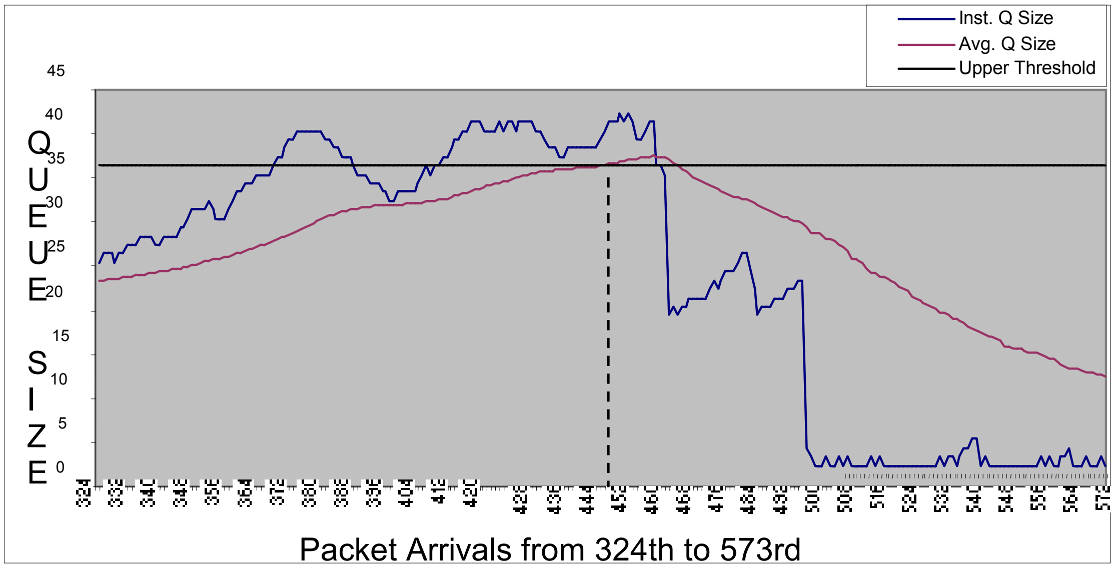

Figure 8 – Plot of queue size v/s packet at Node I3

Based on queue occupancy at the node, source 1 is identified as 'bandwidth greedy'. Consequently a packet filter for source 1 is installed for *ITime* seconds. I3 also sends an active filter message to the previous hop node I2. Now, packets belonging to source 1 are dropped at I3 as long as the packet filter remains in operation. Soon, node I2 installs a similar packet filter and the responsibility of controlling the greedy source 1 shifts one hop closer to the source.

This process continued till the filter migrates to the first hop node. These actions are deduced from the packet drops for source 1 which occur successively at nodes I3 followed by I2 and finally at I1. In Figure 9, the y-axis represents the number of packets and the x-axis marks the nodes I1, I2, I3 and the sink. The bars represent the arrival and departure of packets belonging to source 1 at the nodes I1, I2 and I3.

Let's start with node I3 where packet filtering begins. When the LGC algorithm was triggered, I3 had received and forwarded packets a1 through a122. It then installs the packet filter for source 1 and sends an active filter message to I2. I3 then drops packets a123 through a135 due to active filtering. Now, I2 installs a packet filter for source 1 and propagates the filter message to node I1.Subsequently I2 drops packets a136 through a173 and packets a174 through a250 were filtered at I1. Totally packets a123 through a250 are dropped after LGC is triggered.

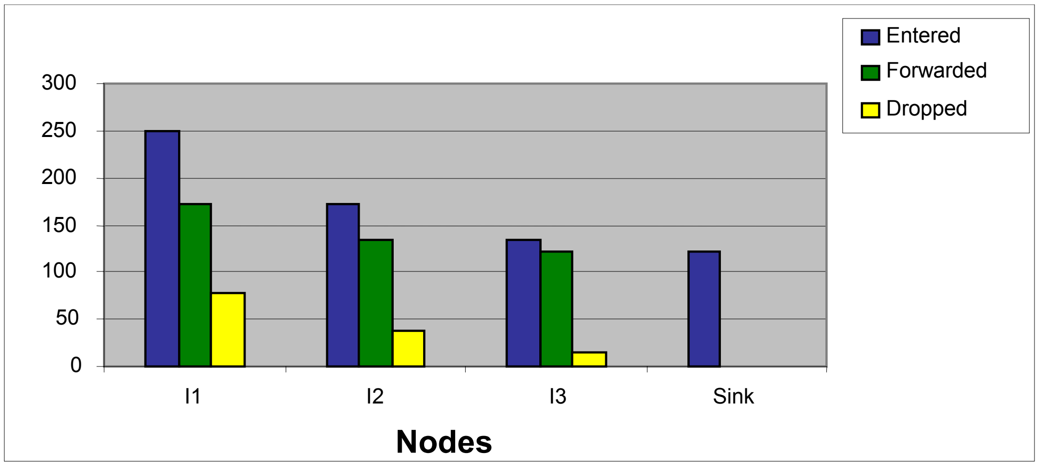

Figure 9 – Packet flow in Experiment 2 after LGC has been triggered

5.3 Experiment 3 - Multiple Bandwidth Greedy Connections

In this experiment we test the ability of the LGC algorithm to handle multiple bandwidth greedy connections. The network topology for this experiment is shown in Figure 10. Nodes 1 and 3 are greedy sources where as nodes 2, 4, 5, 6 and 7 are unresponsive connections making a moderate demand on the network. Node 8 is the common sink for all the sources. Node 1 injects 300 packets in a single burst and node 3 injects 300 packets in 3 bursts with an inter-burst spacing of 3 seconds. The other nodes (2, 4, 5, 6 and 7) inject 120 packets each in 30 bursts with 2 seconds inter-burst period.

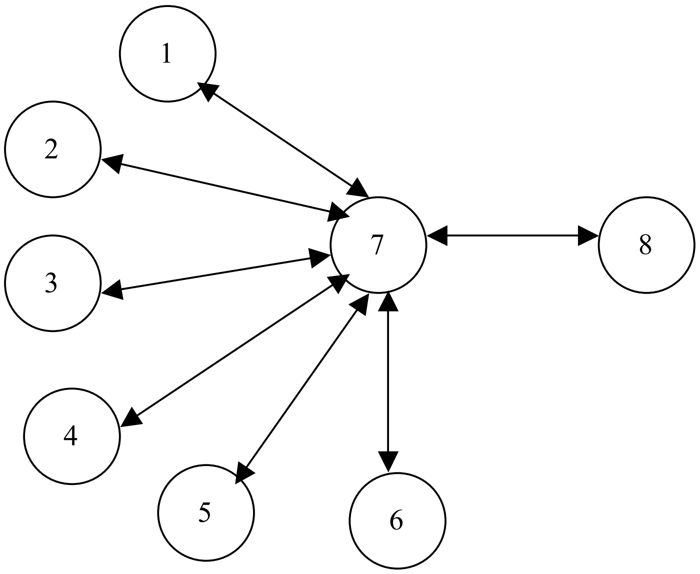

Figure 10 – Network Topology for Experiment 3

The queue parameters for node 7 are set with queue weight = 0.04, $\max_{th}$ = 35 and buffer size = 250. A large buffer size is deliberately chosen to observe the queue occupancy at node 7 and prevent tail dropping of packets.

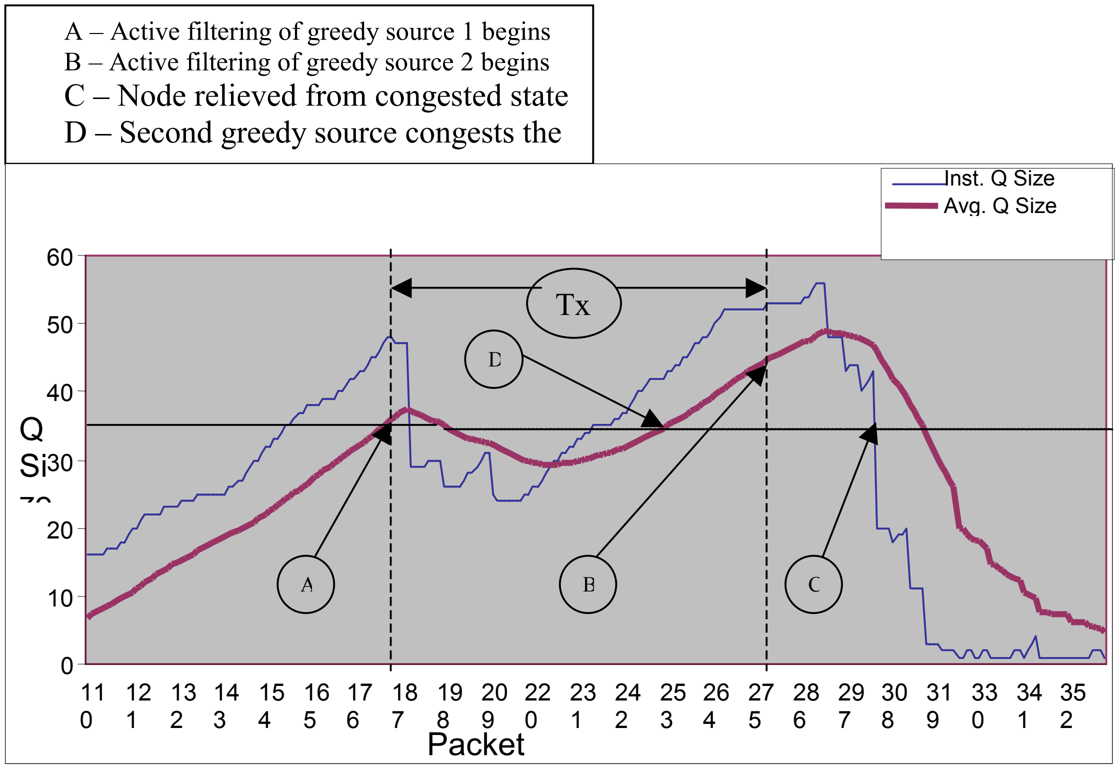

Figure 11 – Queue size v/s packet arrivals for multiple greedy sources

In Figure 11, the x-axis shows the packets arriving at node 7 and the y-axis shows the instantaneous and average queue sizes measured in packets. The line (y = 35) represents max_{th}. For brevity, the first 110 packet arrivals at node 7 have been omitted in the chart. Initially the average queue size remains low (<10). Once the greedy source begins injecting packets, the average queue size increases till max_{th} is crossed (point A). Now, the LGC algorithm is triggered and active filtering of greedy source 1 begins.

At this point the instantaneous queue size drops but the available bandwidth is soon taken up by the second greedy source. This is observed within the *Tx* portion of the graph at point D. The difference here is that although the average queue size crosses the upper threshold (35 in this case), the LGC algorithm is not triggered. The reason being that a minimum time lapse of *Tx* is maintained between successive triggering of the LGC algorithm. After the *Tx* timer expires (point B), the LGC algorithm successfully identifies and filters the second greedy source. Queue occupancy is now controlled and the node emerges from its congested state (point C).

Packets arriving at the common sink node 8 reveal that throughput for the moderate connections were 100% each (mainly due to the large buffer size at node 7). Greedy connection 1 had a throughput of 57.33% and greedy connection 2 had a throughput of 46% due to active filtering at node 7.

5.4 Observations

At the congested node, if there are a large number of connections represented in the node queue, it is observed that the greedy connection tends to shut out the responsive connections and grab a large share of the bandwidth making it easier to identify greedy connections.

There may be some cases in which multiple greedy connections compete for a limited share of the bandwidth in such a way that they restrict other responsive connections, but all of the greedy connections have individual queue occupancies within permissible limits. In this scenario the active node gets congested but the LGC algorithm fails to detect the greedy connections. For example, lets assume that there are ten connections through a node of which five are non-greedy and five are greedy sources. It is possible that when $avg \geq max_{th}$ each of the greedy sources have taken up 15% of the queue leaving the remaining 25% of the queue to be shared amongst the five responsive sources. The permissible value of queue occupancy is obtained from eq. [c] in section 4.8.1. Thus,

$$qo = 100 * \log_e(3*n)/n = 100*\log_e(30)/10 = 34.01\%$$

Since all the connections including the greedy connections have queue occupancy well below this limit, the LGC algorithm does not detect them and the five responsive connections continue to receive a disproportionate share of the bandwidth. Although this example demonstrates a shortcoming of the LGC algorithm we note that such scenarios are an exception rather than the rule. It is a rare occurrence for multiple greedy connections to congest a particular node at the same time and get synchronized in such a way that they make identical demands on network bandwidth.

The overheads of the LGC algorithm include maintaining timers and connection identifiers when the node is severely congested. However, these overheads are minimal in comparison to the benefits accrued in limiting greedy connections and relieving the node from its congested state. If extreme measures such as actively filtering out the greedy connections are not taken there is a high possibility of the node buffer being reduced to a drop-tail queue.

If the packet filter were to be statically positioned at the congested node, the node would suffer the overhead of filtering packets at a time when its resources were scarce. Secondly, network resources such as processing time and bandwidth would be wasted between the source and the congested node at which the packets are being filtered. For the above reasons we considered it beneficial to use active network technologies to migrate the packet filter towards the source of the greedy connection and protect network resources.

6. CONCLUSIONS

In normal closed-loop congestion control mechanisms, the network provides negative feedback to the transmitting sources when it is congested or when congestion is building up. These methods rely on the transmitting sources to exercise control by cutting back their effect rate of transmission. However, as bandwidth greedy applications tend to “grab” as much network bandwidth as available, getting such sources to respond to congestion signals is sometimes not possible. RED mechanism provides an excellent means for congestion avoidance, but it falls short when confronted by a greedy connection.

In this paper, we presented the design, implementation and evaluation of the LGC active congestion control mechanism on the RANI testbed. LGC uses active network capabilities to address the shortcomings of RED and limits 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.

7. ACKNOWLEDGEMENTS

The research presented in this paper was supported in part by the National Science Foundation via grants numbers ACI 9984357 (CAREERS), EIA 0103674 (NGS) and EIA-0120934 (ITR) awarded to Manish Parashar, and by Telecordia Technologies, Morristown, NJ 07960. The authors would like to thank Prathima Agrawal for her contributions to this research.

8. REFERENCES

1. 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.

2. 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.
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. B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell and C. Partridge, *Smart Packets: Applying Active Networks to Network Management*, ACM Transactions on Computer Systems, February 2000, Vol. 18, No. 1, pp. 67-88.
6. D. Wetherall, J. Guttag and D. L. Tennenhouse, *ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols.* Proceedings of IEE OPENARCH'98, San Francisco, CA, April 1998, pp. 117-129.
7. D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles and J. M. Smith, *The SwitchWare Active Network Architecture*. IEEE Network Magazine, Special issue on Active and Controllable Networks, May/June 1998, Vol. 12 no. 3, pp. 29-36.
8. SwitchWare, Department of Computer and Information Sciences, University of Pennsylvania and Bell Core - http://www.cis.upenn.edu/~switchware/ (visited 06/10/2002)
9. Composable Active Network Elements (CANES), College of Computing, Georgia Institute of Technology and Department of Computer Science, University of Kentucky - http://www.cc.gatech.edu/projects/canes/ (visited 06/10/2002)
10. NetScript, Department of Computer Science, Columbia University, http://www.cs.columbia.edu/dcc/netscript/ (visited 06/10/2002).
11. S. Floyd, TCP and Explicit Congestion Notification , ACM Computer Communication Review, V. 24 N. 5, October 1994, p. 10-23
12. S. Keshav, *Congestion Control in Computer Networks*. PhD Thesis published at UC Berkeley, Department of Electrical Engineering and Computer Science, TR-654, September 1991.
13. 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.
14. 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.
15. Information Sciences Institute, University of Southern California, *Transmission Control Protocol, Request for Comments: 793*. September 1981.

APPENDIX I

The LGC algorithm is listed below. The variables used are described in Table 1.

Table 1 – Variables used in the LGC algorithm

avg	Calculated average queue size
max_{th}	Upper threshold for node queue
Suspen_LGC	A Boolean variable used to ensure a minimum idle time between consecutive triggers of LGC
ITime	The time for which the packet filter for the greedy connection is installed at an intermediate node
FHTime	The time for which the packet filter for the greedy connection is installed at the First Hop Node
Tx	The minimum idle time between successive triggers of the LGC algorithm

The LGC Algorithm

1. initialization

```
avg = 0, Suspend_LGC = false
maxth, lTime, FHTime and Tx are pre-set and configurable.
```

2. The LGC algorithm is shown below

```
If (avg ≥  maxth && (! Suspend_LGC)) {
     Set Suspend_LGC to true for Tx
    Find out responsive share of packets for a connection (r)
    Determine the greedy connection and corresponding GCI
    Determine virtual link on which its packets arrive and PHI
    If (GCI != PHI) {
      Install filter for GCI for ITime on virtual link
      Send filter message to previous hop node
    }
    Else {
      Install filter for FHTime for GCI
          }
}
```

3. The process routine of the active filter message

```
Process (active_filter message) {
    Extract GCI from payload of active packet received
    Determine the virtual link on which its packets arrive and PHI
    If (GCI != PHI){
     Install packet filter for GCI for ITime on virtual link
      Send filter message to previous hop node
      }
     Else {
       Install filter for FHTime for GCI
       }
}
```