Proc. 3rd International Conference on Internet Computing, Las Vegas, NV (June 2002), pp. 201-208.

Fair and Scalable Load Distribution in the Internet

Vasil Hnatyshin and Adarshpal S. Sethi
Department of Computer and Information Sciences,
University of Delaware, Newark, DE 19716

ABSTRACT

The current best effort approach to quality of service (QoS) in the Internet can no longer satisfy a diverse variety of customer service requirements, and that is why there is a need for alternative strategies. We believe that the Internet needs means for providing a fine-grained per-flow QoS that does not cause network congestion and keeps overall link utilization high. In this paper we introduce an efficient, fast and scalable load distribution mechanism, which fairly distributes available resources among the flows based on their resource requirements. The load distribution scheme (LDS) is implemented via a message exchange protocol which maintains high link utilization while incurring low overhead. We study the LDS and compare two fairness mechanisms introduced within the LDS framework using simulations in OPNET.

Keywords: Quality of service, dynamic admission control, load distribution, network feedback, fair rate distribution

1. Introduction.

The "one service for all" model used by the current Internet can no longer satisfy the multitude of customer requirements. We believe that the Internet needs is a scalable mechanism for providing a fine-grained per-flow quality of service that does not cause congestion in the network and keeps overall utilization of the links high. In author's opinion, one of the best ways to achieve this goal is to introduce a load distribution scheme (LDS) at the network boundaries. Such a scheme would require the ingress nodes to limit sending rates of the flows based on the congestion level in the network.

In this paper we introduce a fast, efficient, and scalable load distribution mechanism that fairly distributes available resources among the flows based on their resource requirements. The LDS is based on the idea that during congestion, an overloaded interface notifies the ingress nodes that contribute to the congestion asking them to reduce transmission rates of the flows. Upon receiving congestion notification message, the ingress router computes its new share of the bandwidth on the overloaded interface and distributes it among individual flows. The proposed LDS guarantees that each flow receives at least its minimum requested amount of bandwidth, while any additional resources are shared among all flows in a fair manner. The LDS admits a new flow into the network only if there is enough resources to satisfy the flow's request. The LDS is implemented using a message exchange protocol that retains high link utilization and incurs low overhead.

The LDS is most beneficial to applications that can tolerate variation of their transmission rate while still providing the end-user with an adequate level of service quality. FTP is an example of such an application; it tolerates variation in the downloading speed but can use as much bandwidth as the network can provide. Multimedia applications can also benefit from the LDS. Such applications cannot operate below certain transmission rates because the quality of the picture or sound becomes unrecognizable; however, they can tolerate variation of the allocated bandwidth as long as the quality of arriving data is acceptable to the receiver.

The rest of this paper is organized as follows. Section 2 introduces definitions of fairness, while section 3 describes the message exchange protocol used in the LDS. Section 4 presents simulation results and in Section 5 we discuss scalability issues and examine the problem of providing per-flow QoS within the LDS framework. Section 6 provides an overview of the related. Finally, we conclude in Section 7 with possible future directions.

2. Fairness Definitions

In order to determine a permissible sending rate of a flow, each boundary node maintains a *Requested Load Range*, *RLR* = (b^f, B^f), for the flows that enter the network domain through it. A *flow's RLR* consists of two values: a minimum rate, b^f, below which the flow cannot operate normally and the maximum rate, B^f, that the flow can utilize. The flow's sending rate, R^f, is limited by its RLR and lies within this requested range. Throughout the paper we will often refer to numerous definitions of the RLR aggregates. To avoid potential confusion we define these aggregates as follows.

In addition to the flow RLRs, each ingress node keeps track of the *path RLRs*. The *path RLR*, (b_i^P, B_i^P), or the RLR of the ingress node i on the path P is a load range where b_i^P corresponds to the sum of the minimum requested rates of the flows that originate from the ingress node i and traverse the path P, while B_i^P is the sum of the corresponding requested maximum rates. Using $f \rightarrow P$ to denote that flow f traverses path P, we define the path RLR as follows:

$$b_i^P = \sum_{f \to P} b^f \qquad B_i^P = \sum_{f \to P} B^f \tag{1}$$

Similarly, we define *interface* and *aggregated interface RLRs* for the core router interfaces. The *interface RLR* of interface *k* for the ingress node *i*, (b_i^k, B_i^k), is the sum of the path RLRs of the ingress node *i*, subject to the condition that the paths include interface *k*.

$$b_i^k = \sum_{k \in P} b_i^P \qquad B_i^k = \sum_{k \in P} B_i^P \tag{2}$$

To avoid confusion, we will use an upper-case letter (e.g. *P*) for a path and a lower-case letter (e.g. *k*) for an interface. Finally, the *aggregated interface RLR* of interface *k*, (b^k, B^k), is the sum of its interface RLRs.

$$b^k = \sum_i b_i^k \qquad B^k = \sum_i B_i^k \tag{3}$$

Ingress nodes obtain the flow RLRs from the service level agreements established with the user, and they compute the path RLRs based on these values. Each core interface obtains interface RLRs from the ingress node's advertisements and computes an aggregated interface RLR. Ingress nodes maintain information about individual flows (e.g. flow's RLR) and their corresponding paths (e.g. path RLRs), while the core routers maintain only per-ingress node information (e.g. interface RLRs). A more detailed overview of the data structures maintained in the ingress and core nodes is provided in [4].

Congestion notification messages carry interface and aggregated interface RLRs. These values allow ingress nodes to *fairly* distribute available resource among individual flows. In this paper we examine two definitions for computing an ingress node's fair share. The first definition, which we call *proportional*, computes the fair share, FS_i^k, of the ingress node *i* proportionally to the minimum value of its interface RLR on the congested interface *k*:

$$FS_i^k = \min\left(b_i^k + \left(C^k - b^k\right)\frac{b_i^k}{b^k}, B_i^k \right) = \min\left(C^k\,\frac{b_i^k}{b^k}, B_i^k \right) \tag{4}$$

where C^k is the capacity of the outgoing link on interface *k*. According to equation (4), each ingress node receives its share of the interface's capacity proportionally to its minimum requested rate on the congested interface. The fair share of each flow that contributes to the congestion is computed in a similar way. Furthermore, the fair share of the flow *f*, FS^f, is also proportional to its minimum requested rate on the congested interface:

$$FS^f = \min\left(C^k\,\frac{b^f}{b^k}, B^f \right) \tag{5}$$

The second definition of fairness computes the fair share of an ingress node proportionally to the difference between the maximum and the minimum requested rates, which is the amount of bandwidth the flow needs to maximize its performance. We call this fairness definition a *maximizing utility fairness*. Thus, the fair share on congested interface *k* of ingress node *i* and of flow *f* is computed as follows:

$$FS_i^k = \min\left(b_i^k + \left(C^k - b^k\right)\frac{B_i^k - b_i^k}{B^k - b^k}, B_i^k \right) \tag{6}$$

$$\begin{aligned} FS^f &= \min\left(b^f + \left(FS_i^k - b_i^k\right)\frac{B^f - b^f}{B_i^k - b_i^k}, B^f \right) \\ &= \min\left(b^f + \left(C^k - b^k\right)\frac{B^f - b^f}{B^k - b^k}, B^f \right) \end{aligned} \tag{7}$$

3. The message exchange protocol

The message exchange protocol consists of three distinct phases. During the first phase, called *path probing*, the ingress node attempts to learn about the current state of a path or to learn the path itself if the route to the flow's destination is unknown. The probe messages collect the current arrival rate of the traffic and the aggregated interface RLR for each traversed link. The probe messages are generated either periodically or when a new flow is activated. Periodic probing is used to determine if the ingress node can increase its sending rate on the path, while the probing caused by the flow activation determines if the new flow can be admitted into the network.

Admission of a newly activated flow into the network or a flow termination initiates the second phase called the *RLR change* phase. The purpose of this phase is to update the interface RLRs along the flow's path. If the admission of the new flow causes congestion anywhere along the path, then the ingress node initiates the third phase, called *the Rate Reduction Phase*. During the third phase congested interfaces notify ingress nodes to slow down.

The message exchange protocol uses the following message types. In the first phase, ingress nodes generate PROBE packets and receive results of the path probing via PROBE_REPLY messages. In the second phase, ingress nodes advertise changes using RLR_CNG packets. CN and CN_CORE messages are used during the rate reduction phase to convey information about congested interfaces to the ingress and core nodes respectively.

3.1 The Path Probing Phase

The purpose of the path probing is to learn the current arrival rate and the aggregated interface RLR at each link on the path of interest. The PROBE messages are generated by an ingress node either periodically or due to the activation of a new flow, and are processed by all core routers on the path to the specified egress node. When a core router receives a PROBE message, it retrieves information about the interface on which the probe message will depart. This information consists of the IP address of the interface, estimated total arrival rate, capacity, and an aggregated interface RLR. The interface information is stored in the body of the PROBE, and the message itself is forwarded further along the path. The PROBE_REPLY message that carries collected path information from the egress node back to the ingress node does not require any additional processing by the core routers.

Information collected by the PROBE is used to update the path state information at the ingress node. Each ingress node maintains three tables: a *flow* table, a *path* table, and an *interface* table. The flow table contains information such as the flow RLR, the current sending rate, and the destination address of each flow. The path table contains information about each path that is being traversed by the traffic originating from this ingress node. The path table entry contains the ordered list of the interfaces that belong to the path and the list of the flows that travel on this path. Information about each interface of the path is maintained in the interface table. An interface table entry contains the IP address of the interface, the total arrival rate, capacity, and an aggregated interface RLR.

Once the internal tables are updated, the ingress router examines the cause of the PROBE generation. If it was a periodic probe, then the ingress node examines the path information in order to determine if there is excess bandwidth available on the path. If the probe was generated due to flow activation, then the ingress node calculates the new flow's permissible rate and initiates the RLR change phase. Details regarding computation of the available resources on the path and new flow's rate can be found in [5]. They were omitted due to space limitations.

3.2 The RLR change phase

The ingress node initiates the RLR change phase upon flow activation or termination. The RLR_CNG message generated due to flow activation is always preceded by the path probing phase, during which the ingress node computes the sending rate of the new flow and determines if such a rate increase will cause congestion anywhere on the path. An interface k considered to be congested if the new flow's rate causes the total arrival rate on the interface to be larger then its capacity: $A^f + R^k > C^k$.

If addition of a new flow causes one or more interfaces on the path to become congested, the ingress node identifies an interface that will initiate the rate reduction process. Such interface should satisfy two conditions: it should be congested and it should be located the smallest number of hops away from the destination along the flow's path. The second condition allows aggregation of congestion notifications because the rate reduction process works opposite to the flow of traffic on the path.

Subsequently, the ingress node generates and forwards the RLR_CNG message which contains the RLR and the sending rate of the new flow and the identity of the interface that will initiate the rate reduction process. The interface identity field is empty if there is no congestion on the path or if the RLR_CNG is generated upon flow termination. When a core router on the flow's path receives this message, it updates its interface RLR and the estimated arrival rate on the outgoing interface. The core router initiates the rate reduction process if the identity of its outgoing interface is included in the RLR_CNG message.

The RLR change message generated due to flow deactivation causes a decrease of the interface RLR on each interface of the path. This in turn causes an increase in the fair share for each ingress node that sends traffic through the interface. However, since none of the ingress nodes knows about the RLR decrease, they continue sending traffic at the rate below their new fair share, which may result in temporary underutilization. The LDS relies on the periodic probing instead of explicit advertisement of available bandwidth to deal with such situation. A flow may only increase its sending rate if each interface on the path has excess bandwidth available. This information can be obtained only by probing the path.

3.3 The Rate Reduction Phase

The Rate Reduction Phase begins when the core router interface chosen to initiate the rate reduction process receives the RLR_CNG message. We will use k to denote this interface. The ingress node that requested the RLR change will be called the *initiator*. The Rate Reduction Phase consists of two steps: identifying the ingress nodes to throttle, and generating congestion notifications.

3.3.1 Identifying ingress nodes to throttle

The core router identifies the set U_k that consists of ingress nodes that send traffic through interface k. The set U_k is then divided into two subsets $U_k^{indirect}$ and U_k^{direct}, called *indirect* and *direct notification sets*, respectively. Ingress nodes whose traffic arrives on the same incoming link to interface k as traffic from the initiator belong to $U_k^{indirect}$, while the remaining ingress nodes belong to the set $U_k^{direct} = U_k - U_k^{indirect}$.

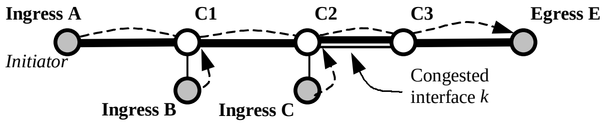

Figure 1. Selection of the direct and indirect notification sets.

Figure 1 shows an example of how these sets are being selected. The initiator A causes congestion on the interface k of the core router C2. The set U_k consists of A, B, and C, because their traffic traverses the interface k. The indirect notification set, $U_k^{indirect}$, contains ingress nodes A and B and the direct notification set, U_k^{direct}, contains only ingress node C.

Since initiator may cause congestion on multiple links, the indirect notification set contains ingress nodes that contribute to congestion not only on the current interface *k* but possibly on other upstream interfaces. Thus, to avoid dealing with multiple congestion notifications and to reduce the overall number of control messages, the core routers aggregate information about the ingress nodes that belong to the indirect notification set. This information is carried in a CN_CORE message to the upstream nodes.

By contrast, traffic from ingress nodes that belong to the direct notification set does not influence the congestion situation upstream but it does contribute to the congestion in the current interface *k*. Thus, these ingress nodes are directly notified through the CN message.

3.3.2 Generating congestion notifications

The CN message carries information about the congested interfaces that requested a rate reduction to a single ingress node in the direct notification set. The CN_CORE message carries such information to multiple (all) ingress nodes in the indirect notification set via other upstream nodes. Both CN and CN_CORE messages carry the following information about the congested interface(s): IP address, interface and aggregated interface RLRs, and the capacity. Ingress nodes that receive CN distribute the load according to the algorithm presented in Appendix A. Upstream nodes that receive CN_CORE may need to update the payload of the forwarded CN_CORE and newly created CN messages.

Upstream nodes that receive the CN_CORE message update notification packets only if the outgoing interface *j* corresponding to the link on which the CN_CORE message arrived is congested and satisfies one of the following rules. The core routers determine congestion status of the interface *j* by comparing the link capacity with the difference between the total arrival rate on the interface *j* and the rate reduction requested downstream. For more details see Appendix B.

Rule 1: If the rate reduction on the congested interface *j* is larger than the rate reduction on the downstream interfaces, then the information about these downstream interfaces is replaced with the data of the interface *j*.

Rule 2: Otherwise, if the interface RLR on the congested interface *j* is larger than that on the next-hop downstream interface, then the information about the interface *j* is added to the corresponding ingress node entry in the control messages.

In rule 1, we define the rate reduction on interface *j* to be larger than that on interface *j'* if for the same RLR values, the interface *j* causes larger rate decrease then the downstream interface *j'*. These rules apply to each ingress node entry separately because individual ingress nodes may contribute to congestion on different sets of interfaces.

4. Simulation Results

To test and evaluate the performance of the load distribution scheme, we performed a simulation study using the OPNET network simulator [11]. The goal of our simulation was to show that the load distribution scheme prevents congestion in the network, maintains high link utilization, fairly distributes available bandwidth among individual flows, and incurs a very small load overhead. We also compare the proposed definitions of fairness and show that the maximizing utility fairness achieves higher throughput in the network.

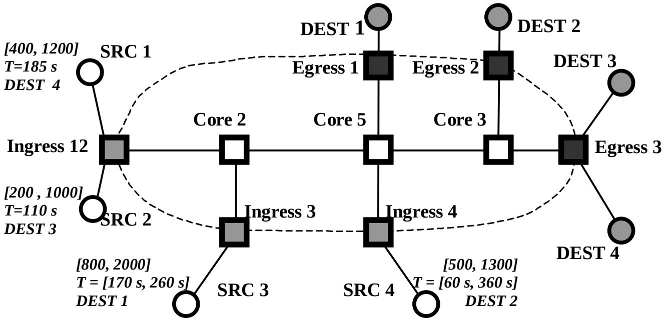

Figure 2. Simulation Topology.

In our simulation we consider four sources sending multimedia traffic through the network of Figure 2, which shows the topology used in the simulation as well as the destination, RLR, and the timeframe of operation for each source. For example, SRC 1 starts sending its traffic to DEST 4 at time 185 seconds and finishes at the end of the simulation. Its RLR is [400 Kbps, 1200 Kbps]. Each link in the network is provisioned with 1544 Kbps. We implemented and tested the bi-directional version of the LDS but in the paper we consider only a unidirectional case in order reduce the complexity of discussion.

4.1 Evaluation of the congestion prevention

If the probe message indicates that additional traffic will not cause congestion on the path, then the new flow can start transmitting before the RLR change message is sent. However, if additional traffic will cause congestion, then the ingress node generates an RLR change message and will allow the new flow to transmit its traffic only when the CN message arrives. Thus, upon a new flow's activation, congestion can still occur only if the ingress node will increase its sending rate before other ingress nodes that contribute to congestion will slow down. However, congestion will only last until all participating ingress nodes receive the congestion notifications and adjust their sending rates.

Ingress node	Flow activation	Probe	Probe Reply RLR change	Congestion Notification
Ingress 12	Source 2	110.02 sec	110.19 sec	110.26 sec
Ingress 3	Source 3	170.01 sec	170.10 sec	170.13 sec

Table 1.Control message exchange timing

Table 1 shows the arrival and departure times of the control packets at the ingress nodes upon the flow's activation. These results were collected using the simulation scenario shown in Figure 2.

One could observe that Ingress 3 needed significantly less time to complete the message exchange as compared to Ingress 12. This phenomenon is explained by the fact that Ingress 12 probes a longer path than Ingress 3. Similarly, both ingress nodes waited longer for the probe reply than for the arrival of the congestion notification after the RLR change. This happens because the probe has to traverse a complete round trip path, while an intermediate core router, and not an egress node, may generate the congestion notification. Thus, in general, a complete control message exchange initiated by a flow activation will last not longer than two round trip times (RTT).

Ingress Node	**Congestion Notification (CN) arrival time**
Ingress12	185.573 seconds
Ingress 3	185.530 seconds
Ingress 4	185.514 seconds

Table 2. Arrival times of CN messages due to the flow activation

To better understand how long the congestion may last, let us examine what happens when SRC 1 starts sending traffic at T=185 seconds. Table 2 provides a list of the congestion notification arrival times caused by the activation of SRC 1. Ingress 12, which requested the RLR change, receives a congestion notification and adjusts its sending rate at time 186.507 seconds, while ingress nodes 3 and 4 receive their congestion notifications at times 186.465 and 186.376 seconds, respectively. Thus, in this situation, congestion was avoided, because ingress nodes 3 and 4 were able to reduce their sending rates before Ingress 12 injected additional traffic.

If Ingress nodes 3 and 4 would receive the congestion notification after Ingress 12, then congestion would not be avoided. However, even in the worst case, congestion would last only from the time of the sending rate increase until the time the last ingress node that contributes to congestion receives its notification. This amount should be limited by the longest RTT.

4.2 Fairness schemes and link utilization

Figures 3 – 5 show how the ingress nodes adjust their sending rates for the scenario described in Figure 2. These results were collected using the maximizing utility definition of fairness. Note that throughout the simulation the ingress nodes share available resources fairly. For example, during the time period [185 sec, 260 sec] the link core 2 – core 5 is a bottleneck for ingress nodes 12 and 3, while link core 5 – core 3 is the bottleneck for the ingress nodes 12 and 4. As a result, available resources are distributed as follows: sending rate for the Ingress 12 is 682 Kbps, 862 Kbps for Ingress 3, and 648 Kbps for Ingress 4; their fair shares at the respective bottlenecks. However, since Ingress 12 cannot fully utilize its fair share at the link core 5 – core 3 due to the bottleneck at core 2 – core 5, Ingress 4 is able to use the excess bandwidth and gradually increase its sending rate.

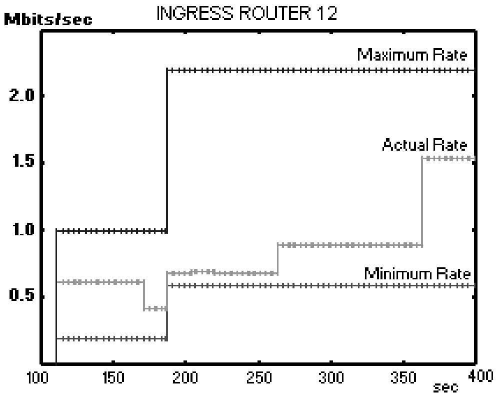

Figure 3. Load distribution by the Ingress Router 12

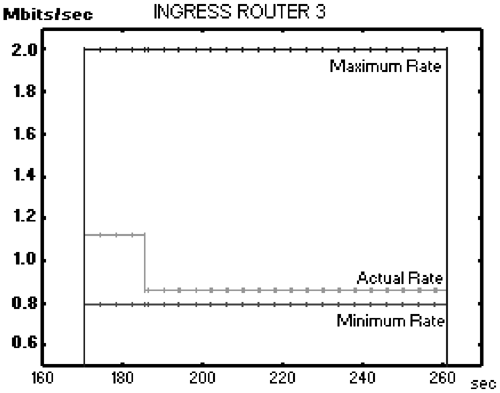

Figure 4. Load distribution by the Ingress Router 3

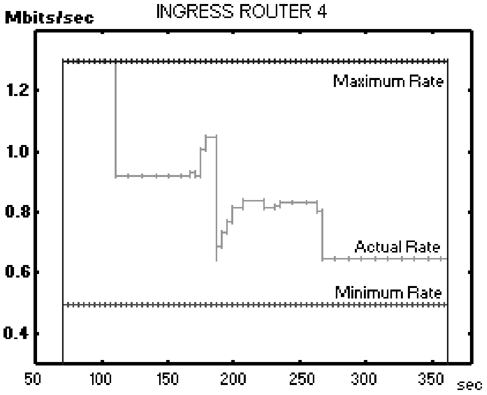

Figure 5. Load distribution by the Ingress Router 4

Figure 6 shows utilization of the links core 2 – core 5 and core 5 – core 3, which indicates that for the duration of the experiment at least one of the links was 100% utilized. Furthermore, even when the Ingress 12 was not able to send traffic at its fair share on the link core 5 – core 3, the link utilization oscillated around 95%.

To compare performance of the proportional and maximizing utility definitions of fairness we slightly modified our scenario. In the new scenario, we change RLR of SRC 4 to [800 Kbps, 900 Kbps]. Figures 7 and 8 show link utilization for the different definitions of the fairness. As Figure 8 shows, proportional fairness does not utilize link resources completely during the time period [110 sec, 170 sec], while the maximizing utility fairness does, as shown in Figure 7.

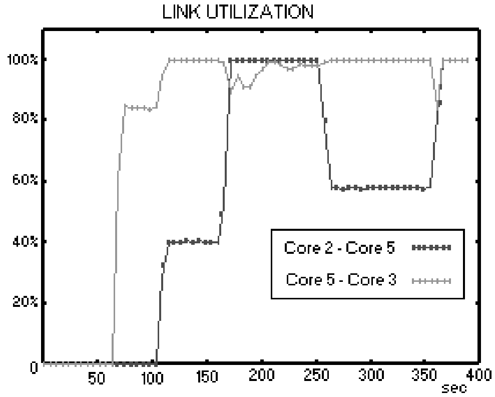

Figure 6. Link Utilization using Maximizing Utility Fairness

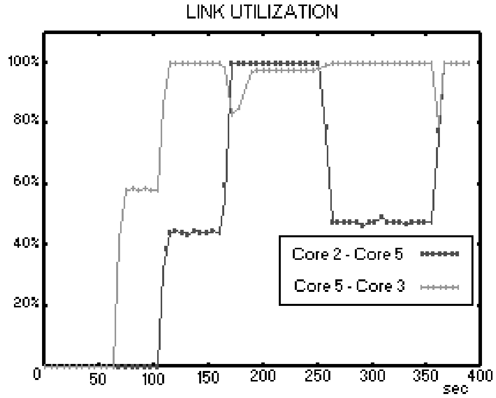

Figure 7. Link Utilization using Maximizing Utility fairness

The proportional fairness suffers from this deficiency because if the fair share of the ingress node is larger than its maximum requested rate, then the ingress node sends traffic at its maximum requested rate and leftover bandwidth is not distributed among the rest of the ingress nodes. If $FS_i^k > B_i^k$ then the link will be underutilized because $\sum_i(FS_i^k) = C^k$. Let us examine when proportional fairness causes link underutilization.

$$C^k \frac{b_i^k}{b^k} > B_i^k \Rightarrow \frac{C^k}{b^k} > \frac{B_i^k}{b_i^k} \qquad (8)$$

Thus, using proportional definition of fairness link i will be underutilized whenever inequality (8) holds. However, maximizing utility fairness always utilizes the link if the maximum value of the interface RLR is larger than the interface's capacity.

$$b_i^k + \left(C^k - b^k\right)\frac{B_i^k - b_i^k}{B^k - b^k} > B_i^k \Rightarrow \left(C^k - b^k\right)\frac{B_i^k - b_i^k}{B^k - b^k} > B_i^k - b_i^k \Rightarrow C^k > B^k \quad (9)$$

Inequality (9) shows that using maximizing utility definition of fairness the link i is underutilized only if the maximum requested rate on the interface is lower than the link's capacity. Otherwise the link bandwidth is completely utilized. Our simulation results support these observations.

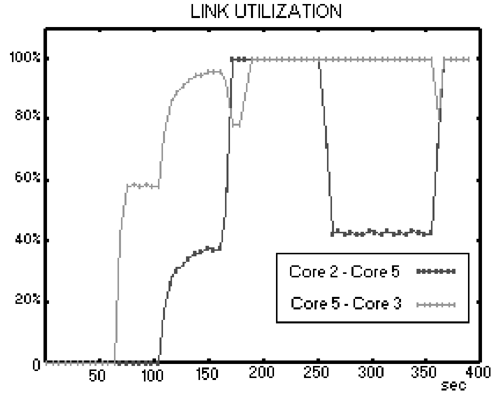

Figure 8. Link Utilization using Proportional fairness

4.3 Examining the overhead

We define the control load overhead caused by the LDS as the ratio between the total number of the data and control packets generated. Figure 9 shows how the frequency of the periodic path probing influences the load overhead. As expected, control packet overhead increases as the probe generation rate grows. Still, even when we raise the probe generation rate to 1 packet every 4 seconds, the overall overhead did not reach 0.8%. However, the probe generation rate is not the only factor that affects the overhead. In a network with a lot of short-lived flows, RLR change and congestion notification messages could significantly increase the overall load. We address this problem in the next section.

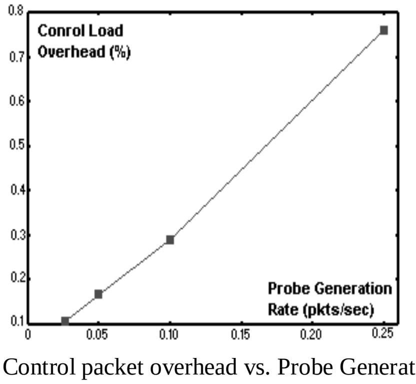

Figure 9. Control packet overhead vs. Probe Generation Rate

5. Scalability and per-flow QoS

In the previous section, we have shown that the LDS incurs insignificant load overhead. However, in our simulation we used a small network with fairly large flows. Large networks usually have a lot of short-lived, small flows, which activate and terminate very frequently. As a result, LDS may incur a noticeable amount of overhead due to control message exchange. Furthermore, frequent advertisements of RLR change and subsequent rate adjustments may cause extreme variation of the congestion level in the network, which would increase the number of control messages being exchanged. Thus, initiating the RLR change phase each time a small flow activates or terminates could be detrimental to the scalability of LDS.

To eliminate this problem, we propose that the ingress nodes request large chunks of the resources to accommodate frequent activation and de-activation of the small flows. For example, for large long-term flows, the ingress nodes would request RLR change each time they are activated or deactivated, but for small short-term flows, ingress nodes would generate RLR change requests only when the aggregated RLR of the small flows goes beyond a certain threshold. This approach would reduce the overall number of control messages and thus would improve scalability of the load distribution scheme.

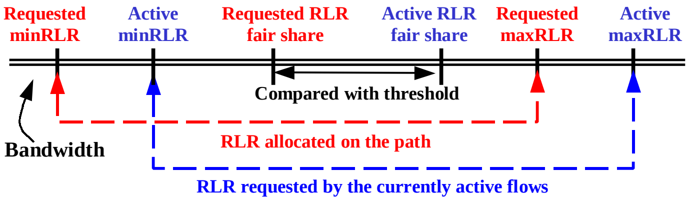

Figure 11. Requested and active RLRs

To implement this approach, we consider when the ingress node should advertise the RLR change of the flow aggregate. Let the RLR that the ingress node requested on the path be the *Requested RLR*, and the RLR that the flows asked for from the ingress node be the *Active RLR*. The ingress node would generate an RLR change message only if the difference between the *Requested RLR fair share* and the *Active RLR fair share* goes beyond some threshold value. Figure 11 shows a relationship between the *Active* and *Requested RLR* values.

Introduced LDS also provides the possibility of implementing a variety of services on a per-flow basis. In order to preserve such LDS properties as fairness among ingress nodes, high link utilization, and congestion prevention, the ingress nodes should use the same definition of fairness throughout the network when computing their bandwidth share. However, the ingress nodes may use different and more complex policies when distributing resources among the flows. Each ingress node may implement its own set of load distribution policies without interfering with the policies of other ingress nodes, which makes LDS extremely flexible in terms of providing per-flow QoS.

6. Related work overview

This paper is a direct extension and improvement of the work done in [4], which addresses the same problem through the approximation of the ingress node fair share based on network feedback. The approach proposed in [4] is less accurate in computing the fair share of ingress nodes, takes longer time to converge, and does not guarantee fair load distribution among the edge nodes under all network conditions.

In [6], Kar et al provided an excellent definition of the dynamic rate control problem and introduced an iterative algorithm that solves it. In [6], individual sources adjust their sending rates based on the utility function and the network feedback which consists of information about the number of congested links on the path. However, the algorithm proposed in [6] converges to the optimal values slowly, operates on a per-flow basis, requires sources to communicate their sending rates to the core routers, and relies on the ACK packets to carry the feedback. In certain situations, the solution proposed in [6] becomes unacceptable because of these features.

Mirhakkak et al introduced a somewhat related idea in [10]. Their goal was to modify the resource reservation protocol RSVP for supporting dynamically changing QoS requirements in mobile ad hoc networks. The proposed dRSVP mechanism also assumes that each flow requests resources in a range. When a new flow enters the network and there are not enough resources to accommodate it, the congested link will adjust the reservations of other flows in order to accept the new flow's reservation. Unfortunately dRSVP also work on a per-flow basis and thus does not scale well. Furthermore, it does not guarantee that the links in the network will be fully utilized and it allows periods of QoS degradation.

The Explicit Congestion Notification (ECN) model [12] requires that the sources will reduce their rates upon reception of the CE marked packets. Both Explicit Congestion Notification approach and simple rate control algorithm [6] assume that the sources are well behaved and would reduce their sending rate upon congestion notification arrival. Unfortunately in the diverse Internet environment, we cannot be sure that all the sources will behave as requested. Thus neither of these approaches provides protection against denial-of-service attacks. On the contrary, the load distribution scheme that we have introduced deals with trustworthy boundary nodes that would adjust sending rates regardless of the user behavior and thus mitigates the possibility of misbehaving sources launching a denial-of-service attack.

The problem of admission control [3, 6-7] and controlled-load services [13] is somewhat related to the issues discussed in this paper. However they address a slightly different problem of determining when a new flow could be accepted into the network, while the LDS examines the problem of how to fairly distribute resources among the sources in order to accommodate the new flow's request.

7. Conclusions and future work

In this paper we introduced a new load distribution scheme that allows fast and fair rate adjustment at the edge nodes. Our scheme requires a core node to maintain information about the edge nodes that send traffic through its interfaces and it uses a message exchange protocol to distribute this information. However, the core nodes do not keep any per-flow information and the message exchange protocol does not cause too much overhead. In the worst case, the proposed load distribution scheme requires two RTTs to adjust the sending rate among the ingress nodes. However, it requires at most the length of the longest RTT to eliminate congestion in the network.

Currently, we are further investigating the characteristics of the introduced load distribution scheme. In particular, since each ingress node probes the path and requests RLR change independently of other nodes, we are examining the possibility of race conditions. In the current version of the LDS implementation, we employ timers to determine the freshness of the collected information; however we believe that this may not be enough to solve the problem of race conditions. In addition, we are examining possibilities of improving the scalability of the LDS, building service policies for providing per-flow QoS, and expanding the LDS to a mobile environment.

8. References

[1] H. Chow, A. Leon-Garcia, "A Feedback Control Extension to Differentiated Services", March 1999. I-D: draft-chow-diffserv-fbctrl.txt.

[2] D. Clark, W. Fang, "Explicit Allocation of Best Effort Packet Delivery Service," *IEEE/ACM Transactions on Networking,* vol. 6, no. 4, pp. 362-373, August 1998.

[3] R.J. Gibbens, F.P. Kelly, "Distributed connection acceptance control for a connectionless network", ITC 16. Elsevier, 1999. 941-952.

[4] V. Hnatyshin and A.S. Sethi, "Avoiding Congestion Through Dynamic Load Control," SPIE's Int'l Symposium on The Convergence of Information Technologies and Communications, Aug. 2001, pp. 309-323.

[5] V. Hnatyshin and A.S. Sethi, "Providing per-flow QoS using load distribution scheme," TR 2002-06, Department of Computer and Information Science, University of Delaware, February 2002.

[6] K. Kar, S. Sarkar, L. Tassiulas, "A Simple Rate Control Algorithm for Maximizing Total User Utility," Proc. of INFOCOM 2001, pp. 133-141.

[7] F.P. Kelly, P.B. Key, S. Zachary, "Distributed Admission Control", *IEEE Journal on Selected Areas in Comm.*, 18 (2000) 2617-2628.

[8] T. Kelly, "An ECN Probe-Based Connection Acceptance Control," *Computer Communication Review* 31(3), July 2001.

[9] A.F. Lobo and A.S. Sethi, "A cooperative congestion management scheme for switched high-speed networks,'' Proc. ICNP-96, International Conference on Network Protocols, Oct.-Nov. 1996, pp. 190-198.

[10] M. Mirhakkak, N. Schult, D. Thomson, "Dynamic Quality-of-Service for Mobile Ad Hoc Networks," MobiHOC 2000. First Annual Workshop on Mobile and Ad Hoc Networking and Computing, 2000, pp. 137 –138.

[11] OPNET Modeler. OPNET Technologies Inc. http://www.mil3.com

[12] K. K. Ramakrishnan, Sally Floyd, D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", March 2001. Internet Draft: draft-ietf-tsvwg-ecn-03.txt.

[13] J. Wroclawski, " Specification of the Controlled-Load Network Element Service," September 1997. IETF RFC 2211.

9. Appendices

Appendix A. The rate reduction at the edges

The congestion notification message that arrives at the ingress node contains the list of the interfaces that experience congestion. Based on the IP address of each congested interface in the list, the ingress node identifies corresponding entries in the router table and updates them with the new information. After that, the ingress node identifies the set of flows that travels through each congested interface. We define the set of flows that visit interface *k* as $\Phi_k = \{ f \mid f \xrightarrow{visit} k \}$.

Subsequently, the ingress node computes a new sending rate for each flow that belongs to the set Φ_k starting from the last congested interface in the list, which is arranged in the order from the closest to the most distant interface from the ingress node. Since each flow may visit multiple interfaces in the list, we adjust the sending rate of the flow only once according to the information of the interface that it visits last, the interface that is located the closest to the end of the list. If there are *n* congested interfaces in the list then the set of flows that reduce their sending rate based on the congestion information of the interface *k* is computed as follows:

$$\Phi^k = \Phi_k - \bigcup_{k<j\le n} \Phi_j \tag{10}$$

Then the new rate for each flow *f* that belongs to the set Φ^k is computed according to equations (5) or (7) presented in Section 2.

Appendix B. Determining the congestion status

The interface *k* is congested if the following inequality holds:

$$R^k - \sum_{i\in U^{down}} \Delta R_i^{down} > C^k \tag{11}$$

We use symbol U^{down} to denote the set of all the ingress nodes that adjust their sending rates because of the congestion downstream and symbol ΔR_i^{down} to denote the amount of bandwidth released by the ingress node $i\in U^{down}$ due to congestion.

Interface *k* computes the amount of bandwidth released by the ingress node *i* due to the congestion downstream, which we call *rate decrease*, as the difference between the fair shares of the ingress node *i* before and after the RLR change phase. Equations (13) and (14) show how the rate decrease by the ingress node *i* is computed for the proportional and maximizing utility definitions of fairness respectively.

$$\Delta R_i^{down} = FS_i^{before} - FS_i^{after} \tag{12}$$

$$\Delta R_i^{down} = \sum_{K^{down}} \left(b_i^k - b_i^{k+1}\right) \frac{C^k}{b^k} \frac{b^f}{b^k - b^f} \tag{13}$$

$$\Delta R_i^{down} = \frac{\sum_{K^{down}} \left(B_i^k - b_i^k\right) - \left(B_i^{k+1} - b_i^{k+1}\right)}{\left(B^k - b^k\right) - \left(B^f - b^f\right)} \left(b^f + \left(C^k - b^k\right) \frac{B^f - b^f}{B^k - b^k} \right) \tag{14}$$

We used superscript *down* to denote information associated with the downstream interfaces. We use notation K^{down} to denote the set of the downstream interfaces that cause ingress nodes to reduce their sending rates. A more detailed description of the rate decrease computation can be found in [5].