

GALO: A Deployable Framework for Providing Better Than Best-Effort Quality of Service

RAHEEM BEYAH

Communications Assurance and Performance Group
Department of Computer Science
Georgia State University
Room 1451, 34 Peachtree St.
Atlanta, Ga. 30303, USA
rbeyah@cs.gsu.edu

RAGHUPATHY SIVAKUMAR^{*} and JOHN COPELAND⁺

School of Electrical and Computer Engineering
Georgia Institute of Technology
Centergy Building
Atlanta, GA 30308
{siva^{*}, jcopeland⁺}@ece.gatech.edu

Abstract-In this paper we propose a deployable approach to improving QoS by using a generic, extendable, overlay architecture; the Generalized Application Layer Overlay (GALO). The goals of this work are to 1) create an overlay architecture which allows us to sample specific path quality metrics among different paths; 2) utilize the proposed overlay architecture in order to implement our proposed QoS-based routing schemes, Application Layer Switching (ALSW) and Application Layer Striping (ALST). Perhaps the most significant contribution of this research is that we are able to achieve better than best-effort QoS without modifying intermediate nodes (i.e., routers), thus encouraging immediate deployment. Additionally, this research is performed on an actual wide area network (WAN) testbed, comprised of universities across the nation. Also, we assemble this architecture as a peer-to-peer framework, encouraging collaborating individuals with average workstations to improve the QoS of their traffic.

Key words: Overlay Networks, Quality of Service, Peer-to-Peer Networks

1. Introduction

The Internet has now evolved to provide a multitude of services. Ever since the Internet proved itself successful at transferring non real-time data, users have pushed for more resource-demanding, real-time applications, such as voice and video, to use the Internet as a transport mechanism. In order to accommodate desired real-time services, or to merely achieve better than best-effort service¹, some level of Quality of Service (QoS) must be established. Many techniques have been proposed to provide such benefits. Though some techniques have proven successful, these solutions are plagued with the inability to be widely deployed due to the massive overhaul of the current infrastructure that they require. Until this ideal massive overhaul of the network occurs, a deployable iterative approach can be taken to provide some level of improved quality of service to best-effort traffic flows. Thus, we have a tradeoff in the solution's ability to be deployed and its ability to quantify and guarantee the level of QoS seen by a flow. This paper takes an iterative approach in that we provide several deployable techniques that do not quantify the flow improvement but provide improved QoS to the flows seen in the test network for the majority of the time, thus moving us closer to the goal of end-to-end QoS. We propose a deployable approach to improving QoS by using a generic, extendable, overlay architecture; the Generalized Application Layer Overlay (GALO).

In this paper we present findings from a multiple week study of Internet traffic dynamics using link throughput as the primary metric. Many traces were conducted through a four node, mesh-connected, "real world" Internet testbed. We generate different types of traffic using a generic, multipurpose, sockets program. Among other things, these findings show improper load balancing and link utilization across Internet paths. We synthetically construct alternate paths showing that 50% of the nodes have a better path quality if alternate links are used, with an increase in throughput seen by a flow, ranging from 30% - 120%.

The aforementioned findings give initial motivation for the establishment of GALO. GALO provides a mechanism to obtain the state of the monitored network via invasive or non-invasive sampling/probing. The sampling/probing is performed by cooperating end nodes. The GALO architecture is designed in a modular fashion and has three primary components: Distributed Client Engine (DCE), QoS Routing Engine (QRE), and the Forwarding Engine (FE).

We extend GALO by introducing the *Application Layer Switching* module (*ALSW*). Application Layer Switching is a concept introduced to provide a better than traditional, shortest path, best-effort, routing service to traffic flows. ALSW is considered a form of QoS-based routing that has the ability to utilize alternate paths to provide an overall better path quality. It can make routing decisions based on more than one metric (i.e., shortest path, loss, delay, delay jitter, and throughput), as well as help to distribute the traffic load among multiple nodes and links in the Internet. This is achieved by constructing non-shortest path routes by appending different link combinations with specific metrics from the source node to end node; using end nodes as switches. We show empirically, that by simply re-routing traffic over underutilized links, we can, in a deployable fashion, immediately improve QoS. The performance analysis of ALSW shows that 70% of the rerouted flows experienced up to a 125% increase in throughput over the average throughput of the default path.

¹ Better than best-effort QoS will not suffice for real-time, delay sensitive, traffic; but can be beneficial for non real-time traffic, e.g., mail transfers between mail servers, or FTP flows.

We further extend GALO with the Application Layer Striping module (ALST). Application Layer Striping has a similar goal of providing better than traditional, shortest path, best-effort, routing service to traffic flows. ALST is also considered a form of QoS-based routing that has the ability to utilize multiple alternate paths to provide an overall better path quality. Additionally, it has the same alternate routing capabilities as ALSW. Using end nodes as switches, ALST has the ability to simultaneously stripe data over multiple alternate paths. We show empirically, that by simply striping data over alternate links, we can immediately improve QoS. The performance analysis of ALST shows that 90% of the striped flows experienced a 10%-350% increase in throughput over the average throughput of the default path.

The rest of this paper is organized as follows: *Section 2* discusses traffic measurement, our experimental methodology, findings from the empirical traffic analysis conducted on our testbed, and the construction of synthetic alternate paths. In *Section 3* we discuss the motivation and description of the GALO architecture. *Section 4* discusses an extension to GALO, the ALSW module. *Section 5* gives the performance analysis using ALSW. In *Section 6* we further extend GALO with the ALST module. *Section 7* gives the performance analysis using ALST, and *Section 8* concludes the paper with future considerations.

2. Traffic Measurement and Alternate Path Evaluation

2.1 Background

Normally, researchers perform empirical studies to understand precisely the behavior of the network of interest. These studies are generally helpful but are most accurate for the particular region of the Internet at the particular time of measurement. Likewise, our study is most accurate for its focus region at the times of measurement. From such focused study, we can only glean and infer principles and characteristics of general Internet traffic behavior. Additionally, empirical studies must be performed regularly due to the inability to predict future types of Internet traffic and their characteristics.

Empirical analysis is not new to the traffic engineering community. It has been around for many years and occasionally is a more approachable method of studying the Internet than simulation [1]. There is a significant amount of literature published on the three methods of analyzing Internet traffic: measurement-based, simulations-based, and analytical models; each has its strengths and weaknesses.

The National Laboratory for Applied Network Research Active Measurement Project (AMP) [2] and National Internet Measurement Infrastructure (NIMI) [3] are popular traffic measurement projects sponsored by NSF. AMP has 130 probes (mostly in the continental US) located at collaborator locations. These nodes actively measure loss, round-trip time, on-demand throughput, and topology information using a series of pings and traceroutes. The NIMI architecture is patterned after the Network Probe Daemon (NPD), which was used to perform one of the early large-scale measurements of end-to-end Internet behavior [4, 5]. It currently uses a number of tools to take measurements, including traceroute, TReno, mtrace, and zing (a generalized "ping" measurement). A more detailed comparison of other popular active Internet measurement projects can be found at [6]. The goal of the above two projects was to develop robust architectures to allow the users to make general present and future measurements. The authors of [7] had a more immediate and specific goal of measuring Internet packet loss. They analyzed a month of packet loss

statistics for voice transmission. They found that packet loss is bursty and usually modeled well as dependent. Also, measuring link asymmetry with packet loss; showed that most losses occur in one direction. They also introduce an analytical technique for measuring loss dependency. In [8] the authors perform a measurement-based study on “path quality.” They observe metrics such as round-trip time, throughput, and loss rate. Their study compares the “path quality” parameters measured on the default path with those of a synthetic alternate path. They found that most of the nodes had potential alternate paths with much better “path quality.” Two other measurement-based studies were done in [9] and [10]. In [9], the authors discuss methods and tools used to detect ACK-compression. The authors of [10] developed a monitoring tool that monitors OC-3 links on backbone networks. They characterize the observed traffic in terms of flow duration, volume, and packet sizes, as well as traffic make up by protocol and application.

In our experiment, we use a generic sockets program to empirically observe Internet traffic dynamics, using throughput as the primary metric in order to observe traffic load imbalance over our testbed.

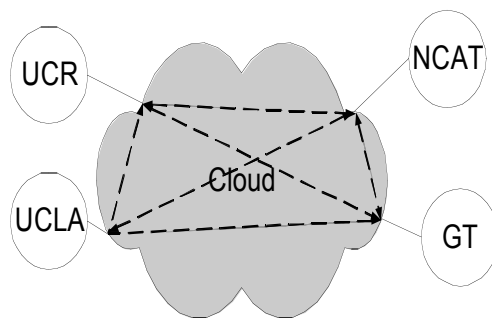


Figure 1. QoS WAN testbed.

2.2 Experimental Setup

We used four end nodes, mesh connected via the Internet, to conduct our experiments. As such, each node takes a different route to every other node. The experiments were bidirectional, so in total, twelve different “paths” were monitored. The nodes were stationed at universities (US) on the east and west coast: Georgia Institute of Technology (GT), North Carolina A&T University (NCAT), University of California Los Angeles (UCLA), and University of California Riverside (UCR) (Figure 1).

We conducted our analysis using a modified version of the *sock* program [11]. TCP data transfers were generated. The transfer duration was varied by modifying the file size. Each node, at a specified time, assumed the role of client and server by executing several instances of *sock*. The *sock* program was modified to collect desired information of a particular flow: IP addresses, port numbers, block size, and arrival time. Once the scenario was completed, the data was copied to a central location for later analysis.

Complete datasets were generated using several blocks of “*sock* scenarios” amalgamated by perl scripts. Each block of scenarios was allocated a two-hour time period to execute. These scenarios varied by file size, transport protocol, number of simultaneous flows and time of day. The experiments ran for a total of ten weeks. In order to

reduce interference between types of scenarios, each type of “*sock* scenario” was executed and sampled independently. Table 1 gives the description of the links.

Link #	Link Description
Link 1	UCR->NCAT
Link 2	GT->NCAT
Link 3	UCLA->NCAT
Link 4	UCR->GT
Link 5	NCAT->GT
Link 6	UCLA->GT
Link 7	NCAT->UCR
Link 8	GT->UCR
Link 9	UCLA->UCR
Link 10	UCR->UCLA
Link 11	GT->UCLA
Link 12	NCAT->UCLA

Table 1. Link numbers and descriptions.

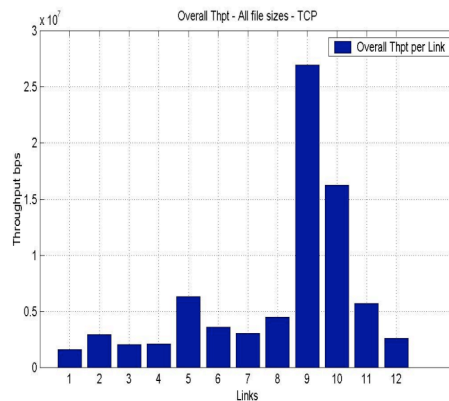


Figure 2. Average throughput of each path.

2.3 Results

Due to space limitations, the following paragraphs merely present a brief summary of the results obtained from this extensive empirical analysis, which is discussed in detail in [12]. Additionally, we discuss in the final paragraph of this section, the specific data used to motivate our efforts in this contribution.

In this experiment we found that there continues to be significant asymmetry between sending and receiving paths. In particular, for TCP we observed that the unidirectional link capacity varied significantly, from 1.27 to 2.15 times greater than their respective return paths. Links 3 and 12 (the path between NCAT and UCLA) show the least amount of asymmetry. The most asymmetry is seen in links 2 and 5, the path between NCAT and GT. For UDP, the link asymmetry was somewhat consistent with TCP, however overall the links appear to be fairly symmetric. We observed that each unidirectional link capacity only varied from 1.03 to 1.14 times greater than its respective return path. The former is represented by links 2 and 5, the path between NCAT and GT. The latter is represented by links 3 and 12, the path between NCAT and UCLA. Since probing with UDP packets estimates the *link capacity* (in contrast to estimating the *available bandwidth* when probing with TCP) we are able to conclude that the asymmetry seen by the TCP traffic was unlikely a result of unbalanced link capacity and more likely a result of congestion.

Also, we showed that TCP throughput is significantly affected by transfer size. Specifically, smaller files are dominated by TCP slow start and therefore do not achieve the throughput that larger files achieve. However, since UDP has no end-to-end congestion control mechanism, its throughput is impervious to transfer size variations.

We further found link throughput when sending multiple simultaneous TCP flows, comparing their total throughput with a single TCP flow’s throughput suggests that TCP flows exhibit an acceptable amount of fairness. Also, we showed that UDP traffic always attains the available bandwidth, which is significantly higher than that attained by TCP traffic.

Finally, we suggested a changing trend of diurnal traffic patterns showing a more congested Internet during evening hours. We were able to achieve higher throughput on most of the links during the morning/afternoon time

period. Since TCP is a “reactive” protocol, we can infer that there was less competing background traffic over the links during this period. From this we suggest a possible shift in traffic patterns. This shift could be attributed to several different factors. We hypothesize that this current trend of changing traffic patterns can be attributed primarily to two factors: the proliferation of home access technologies (i.e., cable/DSL) and different bandwidth-hogging services for home (or dormitory) users such as BearShare.

The results that directly motivate this work are shown in Figure 2. Figure 2 gives the average throughput over the experiment duration seen by TCP flows over each path. Link 9 appears to have the greatest average throughout, while Link1 is plagued with the stigma of the slowest link. This study reaffirms what many before it have concluded, the Internet suffers heavily from load imbalances and even link asymmetry along the same path. This is a result of link capacity imbalance as well as variable demands placed on different links at various times.

2.4 Alternate Paths

Until the work done in [8], most literature in the traffic measurement community stopped at that conclusion drawn above. However, in [8], Savage et al. discussed how, if underutilized links were compounded to create alternate paths to a destination, a significant portion of the flows would have an alternate path with a higher capacity than the default path.

Peripherally considering QoS, the work done in [13] suggests using active networking and generic devices on workstations to perform application-layer routing. Though both [8] and [13] discuss the benefit of using alternate paths, neither implement an architecture or provide results.

Other noted related work include: Detour [14], PlanetLab [15], Emulab/Netbed [16], OverQos [17], QRON [18], and RoN [19].

The Detour project discussed routing and transport inefficiencies as a motivation for re-routing traffic around bottleneck and lossy links. They describe an architecture, but provide only preliminary results that were obtained using a testbed in a lab with traffic generators.

Planetlab developed a generic WAN testbed that serves as a tool for researchers. They focused primarily on creating a robust network that has a suite of management tools. Researchers can use the network to perform short-term experiments.

Emulab is a network experimentation facility supported by the University of Utah. It contains a cluster of nodes that are configured to emulate different network topologies. The Netbed project, which grew out of Emulab has a generic WAN testbed that researchers can use to conduct experiments.

OverQoS from Berkeley proposed an overlay architecture that uses a control loss virtual link value, to provide differential rate allocations and statistical bandwidth and loss assurances. This value characterizes the service received by a flow aggregate and provides a bound on loss and bandwidth of a logical link. They suggest the use of third-party ISPs to provide such services. The results from this architecture are simulation-based.

The authors of QRON propose a general unified framework to support applications that require proprietary functionality offered by specific overlay architectures. Their architecture uses overlay brokers in each autonomous system to form what they define as an overlay service network. The author’s primary focus is QoS-aware routing

protocols for overlay networks. The goal of this work is to find QoS-satisfied paths and to adaptively route traffic meeting the QoS-requirements despite overlay link performance fluctuations. The results from this work are simulation-based.

The MIT RoN project was a large Darpa-funded initiative. The goals of this work were to 1) have fast failure detection and recovery; 2) promote tighter integration with applications; and to 3) enable fine-grained expressive routing policies. This project is similar to the proposed application layer switching technique in that it implements an overlay architecture and observes path quality. However, there primary emphasis is rerouting around network outages. They also reroute traffic during congestion periods and produce some empirical results showing improved throughput, loss, and delay. They have a dedicated infrastructure and thus were able to use raw sockets and other methods to observe the traffic at the packet level, which enabled them to obtain packet loss and delay statistics.

Overall, in addition to 1) implementing an overlay architecture; 2) extending this architecture using the concept of application layer switching that does not require any modification to the current infrastructure; and 3) performing experiments producing encouraging results, our work differs from the related work discussed above in that in our work we 1) produced a measurement-based study of traffic dynamics (*Section 2*); 2) developed a testbed with current traffic patterns to create motivation for rerouting schemes (*Section 2*); 3) defined our architecture as a peer-to-peer framework that operates at the user-level, thus allowing users with average workstations to benefit from the scheme; 4) introduced a notion of the QRE prediction accuracy (a method to determines the accuracy of our the affect of various parameters on our scheme - *Section 5.2.2*); 5) looked at the effect of multiple hops on QRE prediction accuracy and throughput (*Section 5.2.3*); and 6) performed application layer striping (*Section 6*).

Building on the work done in [8], we perform a similar exercise with our testbed. Based on the above results (average throughput of each link), using the existing testbed with four nodes and a total of twelve alternate paths, we synthetically construct alternate paths for each source node to each destination node (every node is a source and destination node). Thus, we have a total of twelve paths (default and alternate paths) from each source to destination. Figure 3 shows a CDF of the throughput of the best alternate path minus the throughput of the default path, for each source to each destination. We observe that 50% of the nodes had a better alternate path with a higher average throughput. Also, Figure 3 shows the percentage increase ranges from approximately 30% to 120%. Though this synthetic construction of alternate paths was done offline, it shows first hand the extent of under utilized links (over time), and motivates the real-time use of ALSW.

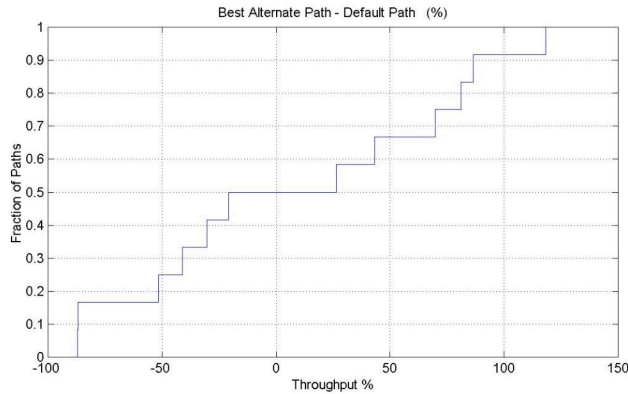


Figure 3. Best alternate path - default path (%).

2.5 Aggregate Throughput over Alternate Paths

Based on the results from *Section 2.3* (average throughput of each link), using the existing testbed with four nodes and a total of 12 paths (every node is a source and destination node), we synthetically construct alternate paths for each source node to each destination node. We now synthetically stripe (split and transmit in parallel) the data along the best paths. This could be a combination of the default and alternate paths or several alternate paths, depending on the path quality. The results here are a best case, and assume the data is intelligently divided at the sender to make the reordering for the application layer trivial. Figure 4 is a CDF of the throughput of the best *two* paths combined, minus the throughput of the default path, for each source to each destination. Similarly, Figure 5 is a CDF of the throughput of the best *three* paths combined, minus the throughput of the default path, for each source to each destination. We observe that each node would significantly benefit from striping. When striping over two paths (Figure 4), we see an increase over the average throughput of the default path ranging from 10% to 280%. Accordingly, we see an increase ranging from 25% to 400% when synthetically striping over three paths (Figure 5). Again, this synthetic construction of alternate paths was done offline, however it shows firsthand the extent of underutilized links (over time), and motivates the real-time use of ALST.

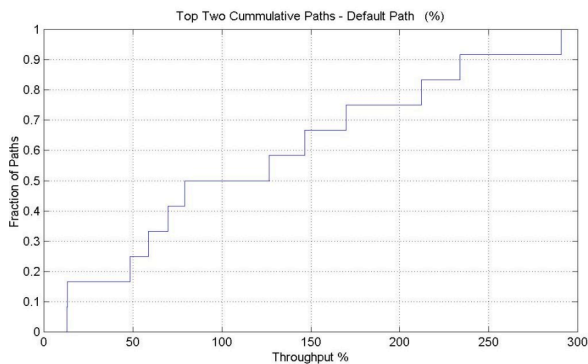


Figure 4. Top two cumulative paths - default path (%).

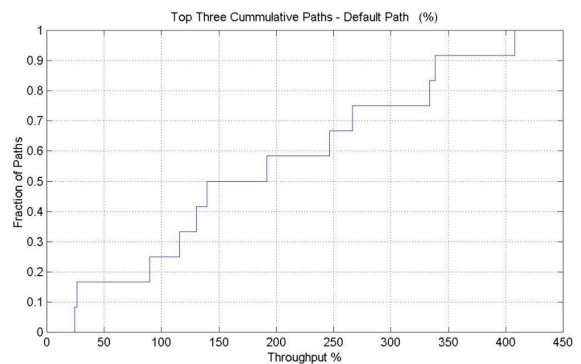


Figure 5. Top three cumulative paths - default path (%).

3. Generalized Application-Layer Overlay

3.1 Background

An overlay architecture is an architecture where the current infrastructure remains in place and a virtual infrastructure or network is run atop it. Overlay networks are often the approach of choice, given that they can provide instant “results” and can span multiple autonomous systems without agreement or cooperation of the ISPs, thus avoiding their logistical conundrum.

Overlay architectures have been used in various instances; ranging from mobile networks [20, 21], virtual private networks [22], computer virus enabling, peer-to-peer file sharing networks [23], and probably the most popular, end system multicast [24, 25]. One of the most widely known overlay architectures and the one most relevant to this research is that of MBone [24].

MBone is a virtual network that originated from an effort to multicast audio and video [25]. An overlay approach was needed since most service providers would not turn the multicast functionality on in their routers. MBone uses a network of routers, known as mrouters, that supports multicast. These mrouters are either upgraded routers or dedicated workstations with special code or modified kernels running cooperatively with standard routers.

MBone tunnels multicast packets inside unicast packets between the different MBone subnets within the Internet. This is done because most IP routers do not support IP multicast (or enable that functionality).

The problems of MBone most relevant to our research are those of manageability and fault tolerance. The overlay architecture was distributed and therefore extremely difficult to manage. Often, MBone sessions would encounter a high packet loss rate, resulting in poor audio and video quality [26]. It was difficult for the end user to determine the reasons for the loss and to find out how to remove it [26]. Packets had even gotten lost on the receiving machine. This phenomenon happened frequently with high-quality video streams as a result of receiving machines not having the processing power to decode and display the video. Programs like mtrace were developed to help track down packet loss [26]. Most often, the poor performance of MBone was a result of old, weak, antiquated workstations used for multicasting. These findings emphasize the importance of appropriate hardware when attempting to employ an overlay architecture.

3.2 GALO Logical Architecture

Thus far, we have given the motivation for our deployable QoS-based routing scheme. We have also presented an empirical study of Internet traffic dynamics. The throughput measured in this empirical study of the WAN links are extracted and used as motivation for additional work. We now propose an overlay architecture that will provide

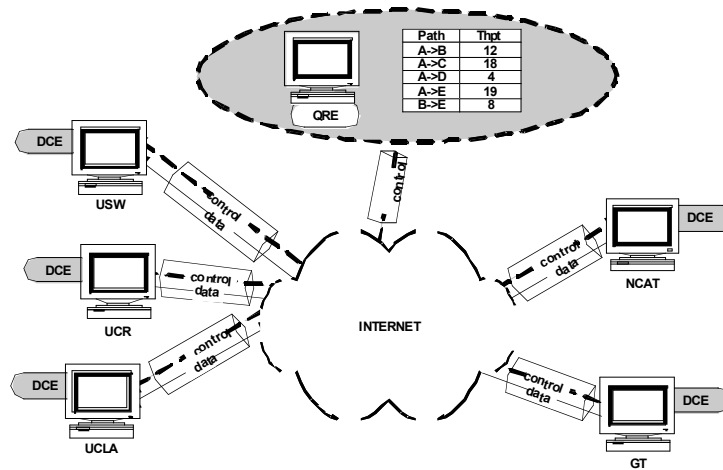


Figure 6. GALO Architecture.

the backbone for an immediately deployable QoS-based routing scheme. In accordance with past overlay approaches [20, 21, 22, 23, 24], this scheme requires no modification to intermediate nodes (routers and switches) and requires only a software upgrade to collaborating end nodes (workstations). The software upgrade is minimal in that a user can install the program without root privileges or any kernel modifications. The Generalized Application-Layer Overlay (GALO) architecture is designed in a modular fashion and has three primary components: Distributed Client Engine (DCE), QoS Routing Engine (QRE), and the Forwarding Engine (FE). The FE is optional depending on the extension modules. We plan to use the data gathered by GALO to perform Application Layer Switching and Application Layer Striping which requires the FE functionality.

- **DCE:** The DCE is a process that resides (possibly in user space) on each collaborating node. The primary purpose of the DCE is to transmit path quality updates back to the QRE. The DCE continuously calculates path quality depending on the specified metric. In our case throughput is the only metric considered. Depending on the desired level of path quality accuracy, the DCE transmits control packets to the QRE. Additionally, depending on the traffic characteristics (i.e., TCP or UDP flows), the DCE may be used to implement packet loss detection as well as perform packet reordering at the application layer.
- **FE:** The FE resides collocated with the DCE on each collaborating node. The FE is invoked only at transit nodes. Its primary responsibility is to act as the switching engine for passing traffic. Once the appropriate signaling² is received from the QRE, the FE dynamically modifies its engine to allow incoming traffic to be received and switched along the outgoing path. The FE's strength lies in its modular design. The current specification requires that the actual switching take place at the application layer. This approach has an obvious inefficiency in that the traffic must traverse the protocol stack to the application layer to be switched, and then traverse the stack again to continue to its destination. As such, this approach initially appears inefficient.

² The Application Layer Communication Protocol (ALCP) is the corresponding signaling/communications protocol that was designed to support this architecture. Due to space limitations, the description of the ALCP is not presented.

Though the success of this technique is a factor of 1) hardware capacity; 2) processing load of the machine; 3) number of hops for a rerouted flow; 4) as well as the inefficiencies of traversing the entire protocol stack twice; we show (*Section 5 and Section 7*) that this delay will be negligible and a significant amount of the rerouted and striped flows benefit from this technique. Also, this approach is desired because it is directly in line with our goals of no modification to the intermediate infrastructure and minimum modification to the end nodes. Another possible approach is to use Source Routing between nodes. This would require that each collaborating node support Source Routing and would be more efficient, as the traffic would only have to go to the network layer for routing decisions, before being sent back out to its next hop. This approach would prove more efficient, but would require more invasive software modification (recompiling the kernel).

- **QRE:** The QRE is centralized in our design and is considered the brains of the architecture. It is the controlling unit for each external process and is responsible for every flow that traverses the network within this domain. The primary responsibility of the QRE is to maintain a current and accurate picture of the path quality between the collaborating nodes. This is achieved by collecting and analyzing the UPDATE messages sent from each QRE and maintaining a table that contains accessibility information to each node. A modified version of Dijkstra's algorithm is used as the table update algorithm. Once the path quality of each link is known, the QRE has the capability to generate control messages and signal to the appropriate nodes specific information, depending on the application running atop GALO.

Available bandwidth estimating techniques can be classified into two categories: passive measurement [27, 28] and active probing [29, 30, 31]. Passive measurement tools use the trace history of existing data transfers. While potentially very efficient and accurate, their scope is limited to network paths that have recently used passive probing and is best initially deployed in an environment where end nodes communicate regularly, allowing passive non-invasive sampling. Therefore, in accordance with our primary design goal of an immediate deployable solution, GALO uses the passive measurement paradigm across a network whose end nodes communicate regularly. One such environment is that of a geographically diverse corporate extranet with mail servers that has recurrent communication. In our model we emulate this environment and create our own data to passively sample to allow the bandwidth measurements.

To obtain the path quality, artificial traffic is generated on each link using a modified version of the sock program [11]. Each node (DCE) has specific ports dedicated to capturing traffic to generate the path quality measurement, namely, in our case the current throughput. In addition to measurement sockets, several control sockets are reserved for messages between the DCE and QRE. The architecture is designed such that these control messages can utilize any reliable underlying transport layer. For our specific implementation, we have chosen TCP. Therefore, a TCP connection is established between each node (DCE) and the QRE. The implementation is transport layer independent, assuming only that it is reliable, not relying on specific characteristics of a particular protocol to maintain the connection. We generate application-layer KEEPALIVE messages to maintain the connections between the QRE and DCEs.

3.3 GALO Software Architecture

3.3.1 DCE Software Architecture and Implementation

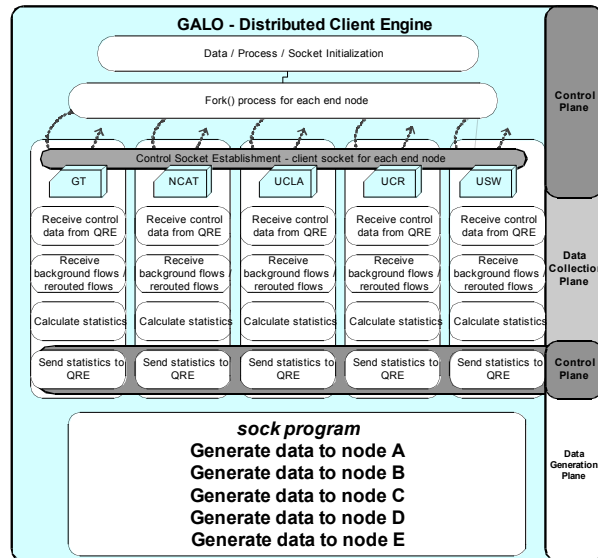


Figure 7. DCE Software Architecture.

Each component of GALO is implemented in the C programming language on the Unix/Linux platforms. The DCE program initializes appropriate parameters then blocks indefinitely at a server TCP socket, created to exchange control messages with the QRE (control plane shown in Figure 7). Once the control connection is established, (between the QRE and DCE) the program moves to the main execution block. In order to afford each communication stream its own resources, we chose to use *fork()* to spawn separate processes for each university; opposed to the multithreading approach. Accordingly, a separate process was generated for each university. The purpose of these sub processes are to absorb probe flows and rerouted flows for every other university and generate statistics to send back to the QRE. Therefore several server sockets are created in each process (data collection and control plane shown in Figure 7).

For example, if our concern is focused on the GT node. The DCE would have sub processes for USW, UCLA, NCAT, and UCR. If we further focus on a single sub process, (e.g., USW sub process) we observe probe flows and rerouted flows (regardless of path taken) sent from USW that are sent to GT. The source address and port are extracted to determine if the current flow is a direct flow (probe flow), rerouted flow that had a single hop, or a rerouted flow that had two hops. Next, an ALCP control message is constructed and sent back to the QRE, carrying the sampled throughput value and additional flow characteristics.

Traffic generation also takes place in the DCE (data generation plane in Figure 7). To accomplish this we again use the *sock* program. We utilize system calls which enable us to execute Unix/Linux from within our C program. Thus generating, in series, probe traffic to every other DCE. Care is given in order to avoid interfering with other DCE generated probe flows. Accordingly, during the sample period each university has a specific time to transmit its probe data, in series, to every other node.

The advantage of choosing the multi-processes approach over multithreading is that each process is allocated its own resources (e.g., memory). Separate memory is not optimal when you have several processes that need to frequently exchange data. For each university process to appreciate the control messages from the QRE, we use System V shared memory to facilitate interprocess communication. The pseudo code for the DCE can be found in Appendix A.

3.3.2 FE Software Architecture and Implementation

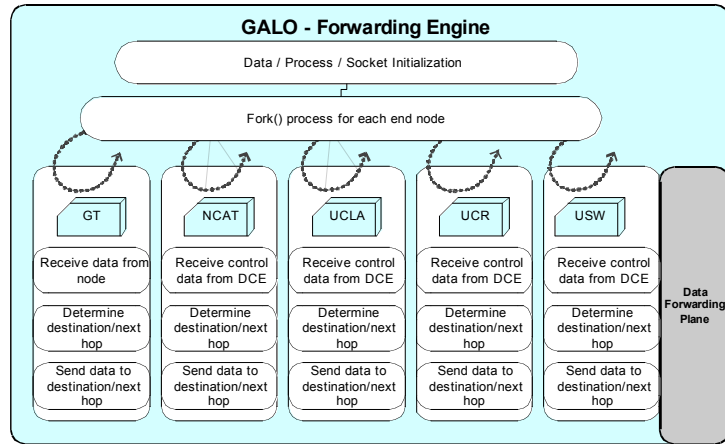


Figure 8. FE Software Architecture.

The FE is conceptually the simplest component of the GALO architecture. In essence, it receives data and forwards data. In order to accomplish this, we again generate several processes, using *fork()*, one for each university. Each process contains a set of client and server ports. Once a flow is received by the forwarding engine, we examine the source address, source port, and destination port (we have predetermined forwarding scheme based on the source address and port combinations) to determine where the flow should be forwarded. If the flow is deemed a multi-hop flow and the current hop is the initial hop, the flow is forwarded to the FE of its second hop. If the flow is deemed a single hop flow, the flow is forwarded to the DCE of its final destination. The forwarding is done by using the same buffer to read and write from/to the respective sockets during a flow rerouting episode. For optimal performance, the buffer size was set to accommodate a single Ethernet packet. This reduced the delay, when using TCP, of waiting for several blocks to be collected before the read buffer (or write buffer) reached capacity, at which point the application/network is finally passed the data. Thus, the TCP socket stream at the FE mirrors that of a UDP stream in that there is minimal delay from the kernel receiving the packet and passing it to the application, as mentioned above; the converse is also true. The pseudo code for the FE can be found in Appendix A.

3.3.3 QRE Software Architecture and Implementation

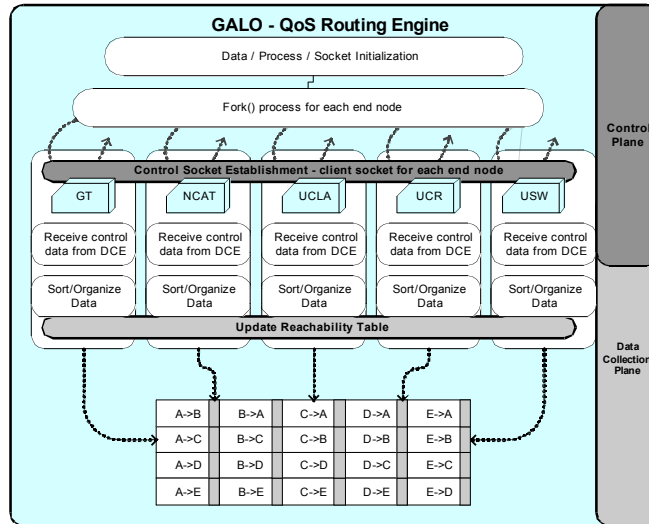


Figure 9. QRE Software Architecture.

The QRE is the controlling unit of the GALO architecture and thus the most complex component. It establishes initial control connections, using TCP sockets, with the DCE of end node. It next spawns separate processes for each university, again using *fork()* (control plane shown in Figure 9). Each process contains several listening sockets to obtain path quality updates from the respective DCE. Using shared memory to host a global path quality table, the DCE's update metric sent to the QRE is consolidated in one centralized location (data collection plane shown in Figure 9). The next section discusses how this table is used to perform ALSW. The pseudo code for the QRE can be found in Appendix A.

4. Application Layer Switching

4.1 Background

Through the years, the Internet has been plagued with inefficiency attributable to the lack of load balancing. Some parts of the Internet are much more loaded (hot spots) than others for various reasons. The truth is, there is no wide-spread load balancing technique within the Internet as we know it. Thus, the path quality is unpredictable, and with many links sometimes unacceptable.

Application Layer Switching was initially implicitly motivated by the work done in [8]. In [8] the authors perform a measurement-based study of many different sites, comparing performance of flows traversing a default path to that of potential alternate paths that were created by synthetically appending links. The primary metrics considered were round-trip-time and loss; throughput was also peripherally discussed.

In this study, the authors used five datasets that were collected in 1995 and 1998. For the loss and round-trip-time measurements they used traceroute. The dataset used for the throughput analysis was collected in 1995. This dataset was generated by the program *tcpanaly*. It initially was generated in [5] and loaned to the authors of [8]. They found that in 30-80% of the cases, the synthetic alternate paths had better quality. This finding reinforced the

fact that the Internet is not equally loaded. It also implied that some sort of mechanism that could be employed to utilize these underutilized links could also improve QoS by taking advantage of better path quality on alternate links. From this, we observe that overutilized links can be avoided, thus improving the QoS of those flows and increasing the load balancing on the Internet. To enable immediate deployment, we suggest that this is done at the Application Layer, by using Application Layer Switching to route traffic on alternate links.

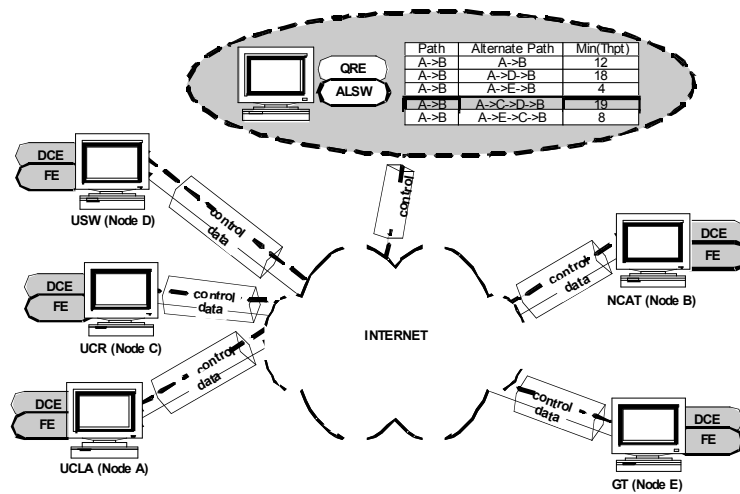


Figure 10. GALO Architecture Extended to Support Application Layer Switching.

4.2 Software Architecture of Application Layer Switching

The input to the ALSW module is the path quality table that is populated by the QRE. The ALSW module monitors the shared memory segment containing the path quality table. The module then queries the path quality table and uses the data to generate an optimal alternative path, and thus makes the appropriate decision to reroute a flow. The optimal path algorithm is a variation of the popular Dijkstra's algorithm. The decision is passed back to the QRE, and the QRE messages to the respective DCE to reroute the next flow.

4.3 Overview of Application Layer Switching

Application Layer Switching (ALSW) is a concept introduced to provide a better than traditional, shortest path, best-effort routing service to traffic flows. ALSW can be considered a form of QoS-based routing because it has the ability to utilize alternate paths to provide an overall better path quality. It can make routing decisions based on more than one metric (i.e., shortest path, loss, delay, delay jitter, and throughput), as well as help to distribute the traffic load among multiple nodes and links in the Internet. This is achieved by constructing non-shortest path routes by appending different link combinations with specific metrics from the source node to end node.

ALSW uses an overlay approach working on top of the current best-effort infrastructure. No modification to current routers is necessary. Accordingly, no state is maintained at edge or intermediate routers. This concept follows the same paradigm as DiffServ in that it pushes complexity away from the core network. However, ALSW

pushes the complexity even further to the edges of the network, all the way to the end node itself, making ALSW an immediately deployable approach to providing an improvement in current flow QoS.

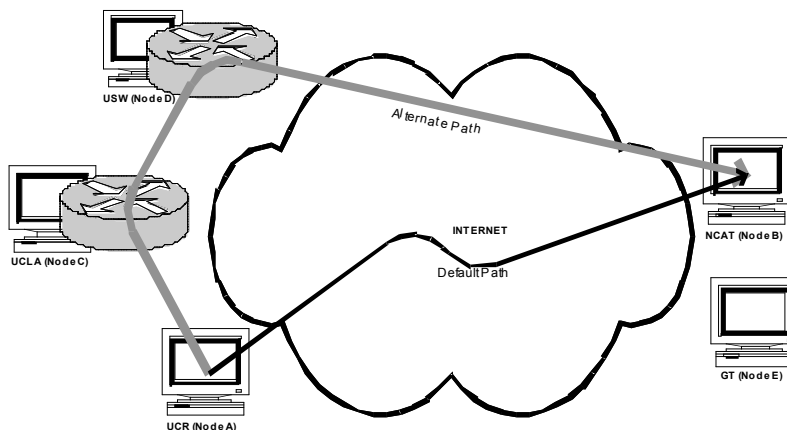


Figure 11. Example of a flow rerouted using *Application Layer Switching*

To accomplish ALSW, we propose an extension to our proposed Generalized Application Layer Overlay, the Application Layer Switching module (Figure 10). For ALSW to be feasible, we use GALO, which assumes that an abundance of trusted, cooperative end nodes (i.e., machines onsite at different locations of a major corporation) are in place. The more end nodes and the more geographically diverse they are, the more paths to choose from, which increases the likelihood of a more desired path. Once the grid of supporting nodes is constructed, path discovery and traffic sampling begins. Periodic updates are broadcast to the QRE using the Application Layer Communication Protocol (ALCP), where it constructs and maintains an accessibility table. This accessibility table is input to the ALSW module where, based on specified metrics (we only consider throughput), alternate paths are created by appending multiple links. Once the path tree is constructed, an optimal path other than the default shortest path can be chosen if it meets the desired improvement in path quality. Once a better path has been defined, the PROVISION message is signaled to the FE on the corresponding path nodes to provision the forwarding engine in each node. These intermediate forwarding nodes act as switches throughout the duration of the rerouted flow (Figure 11). Also, the source node and the destination node are signaled to specify where to send the traffic and where to expect the traffic, respectively. The data is sent shortly thereafter. The data maintains the path until a threshold has passed, a shorter more direct path with acceptable path quality is found, or another path (possibly more hops) is located with a more desirable path quality.

5. Performance Analysis of Application Layer Switching

5.1 Experimental Setup

We expanded the initial network discussed in *Section 2* to have a total of five end nodes and an additional node used to house the QRE. All workstations are shared, general purpose machines. Four of the five end nodes are running Linux, both the fifth end node and the node that houses the QRE have Solaris as their operation systems. The default file size of each flow was 100KB. To perform the experiments in an efficient manner, we designate two

separate blocks. The sampling block is the period of time that a probe flow³ is sent from each source to each destination. Thus, creating traffic to sample, which allows the generation of link throughput data. Within the sampling block, probe flows (a total of 20 – one from each source to each destination) are generated and monitored. The probe flows run in series as to avoid friendly traffic interference. The second period is the reroute block. During this period, no probe flows are transmitted, only a rerouted flow. The QRE keeps a running average of each default path’s throughput, and synthetically creates alternate paths. It then chooses the path with the highest potential increase as the primary candidate for rerouting. During the next reroute cycle, the chosen candidate is rerouted over the alternate path. The alternate paths have been limited to a maximum of two hops. Figure 11 is an illustration of rerouted flow having two hops.

In order to efficiently test this architecture, we repeated the periods throughout the length of the experiments. Thus, the entire duration of the experiments consisted of the repetition of a sample block, followed by a reroute block. The default time for both blocks was 130 seconds. Accordingly, every 260 seconds, a new flow was rerouted and the average throughput of the default links was updated. Additional scenarios were run where the sample block was twice as long as the default case, in order to gauge the effect of sample rate on our results. Moving forward, we label the flows associated with the default rate as SAMP1, and the flows associated with the total block rate of 520 (meaning both the sample and reroute blocks are now 260 seconds) as SAMP2. As a result of limited machine access, the code is executed in user space. Over the duration of the experiments approximately 230 rerouted flows were generated for the default scenario SAMP1. The experiments were allowed to run the same duration of time for SAMP2 and accordingly, approximately half of the amount of flows was generated, due to the sampling period being twice that of the default.

5.2 Results

In this section we present the performance analysis of the ALSW technique. We focus on the overall effectiveness of ALSW at the default sampling rate. Further, we discuss the QRE’s ability to predict the throughput benefit seen by a flow. Additionally, we vary sampling rate and decouple multiple/single hop data and observe their affect on the flows.

5.2.1 Rerouted Flows at Default Sampling Rate

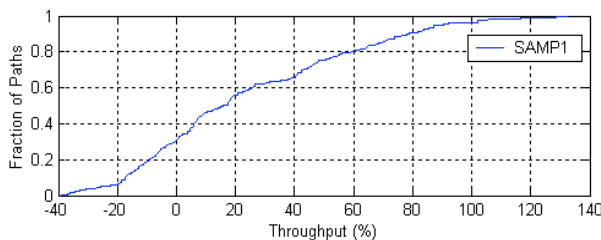


Figure 12. CDF of the total number of rerouted flow’s average throughput compared to the overall average throughput of the corresponding direct path (ALL). *SAMP1*.

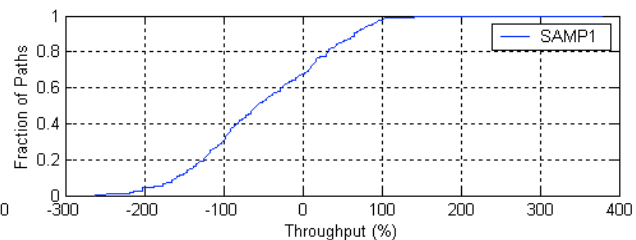


Figure 13. CDF of QRE’s prediction accuracy of throughput for corresponding rerouted flow (ALL). *SAMP1*.

³ Again, the probe flows are only necessary for our experiments. Normally, this architecture would be deployed in an environment where passive sampling would be possible.

Confirming the hypothesis in *Section 2.4*, Figure 12 shows that around 70% of the flows that were rerouted had a higher throughput than that of the average throughput of the default path. Further, these flows had an increase in throughput ranging from 1% to over 125% increase in throughput, with around half of the flows showing at least a 20% increase in average throughput.

5.2.2 QRE Prediction at Default Sampling Rate

As discussed in *Section 5.1*, following a sample block, the QRE chooses the best candidate for rerouting. It also calculates an estimate (based on the synthetic construction of alternate paths) of the percentage increase of throughput the rerouted flow will achieve over the average throughput of its default path. Figure 13 shows the percentage accuracy for the rough estimate of throughput increase made by the QRE. This is achieved by comparing the QRE's estimate of the throughput increase, to the actual throughput increase/decrease realized by the rerouted flow. Though initially, the numbers on the graph look a bit crude, we are still able to achieve a considerable amount of benefit from the technique of ALSW. Figure 13 does NOT show if we will see a benefit from rerouting, rather it reflects: 1) the amount of overhead in routing traffic at the Application layer; 2) the processing load experienced by the end node posing as a router; 3) as well the temporal traffic fluctuations within the links traversed. As further discussion will show, this figure allows us to observe the QRE's precision when considering several different scenarios.

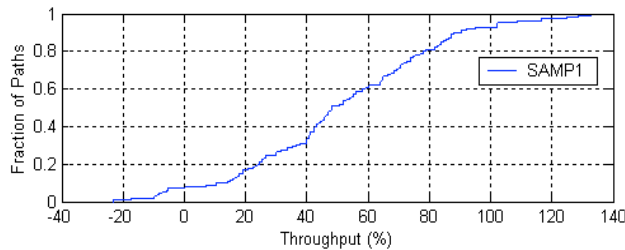


Figure 14. CDF of the total number of rerouted flow's average throughput compared to the overall average throughput of the corresponding direct path (SINGLE HOP). *SAMP1*.

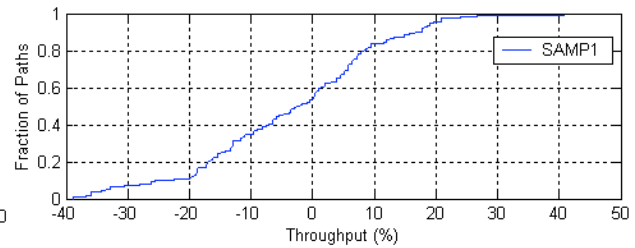


Figure 15. CDF of the total number of rerouted flow's average throughput compared to the overall average throughput of the corresponding direct path (MULTIPLE HOP). *SAMP1*.

5.2.3 The Effect of Single vs. Multiple Hops

As mentioned in the *Experimental Setup* section, flows were allowed to be rerouted over paths containing a single or double hop. Figures 14 and 15 show separate results for single and double hops. As expected, results worsen when considering only multiple hops in that the aforementioned overhead is multiplied. Also, an additional overhead for multiple TCP slow-start (a separate TCP connection is created at each hop) periods as well as a higher loss probability work to degrade the performance of flows rerouted over multiple hops. Though we still see an overall benefit when rerouting over multiple hops, we see that when single hops are considered alone, we have over 90% of the rerouted flows benefiting from rerouting, while only 45% of the flows rerouted over multiple hops

benefit from ALSW. Thus we notice a possible limitation to our current implementation as the number of hops increase.

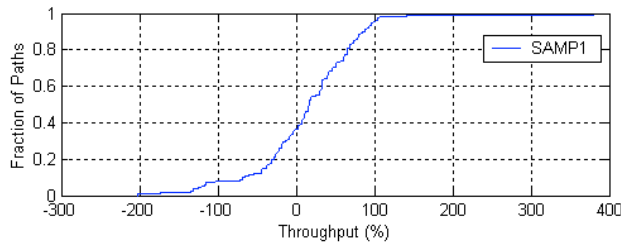


Figure 16. CDF of QRE's prediction accuracy of throughput for corresponding rerouted flow (SINGLE HOP). *SAMP1*.

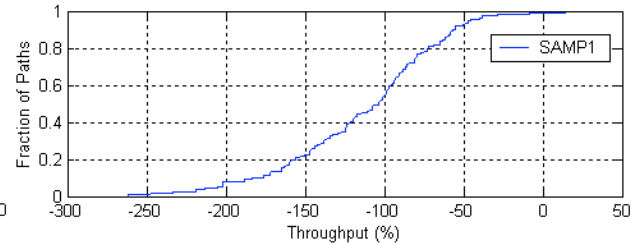


Figure 17. CDF of QRE's prediction accuracy of throughput for corresponding rerouted flow (MULTIPLE HOP). *SAMP1*.

5.2.4 QRE Prediction – Single Hop vs. Multiple Hop

If we again separate the overall results based on the number of hops and look at the QRE prediction, we observe another phenomenon (Figures 16 and 17). As the number of hops increase we tend to always over estimate the throughput expected by the rerouted flow. This occurs because the QRE does not take into account the aforementioned collective extra overhead encountered at the switching nodes, which starts to become significant as the numbers of hops increase. Thus the prediction cycle of the QRE loses its randomness, as seen in Figure 16, which is generated by traffic fluctuation, and results to a rather deterministic flow throughput overestimation (Figure 17).

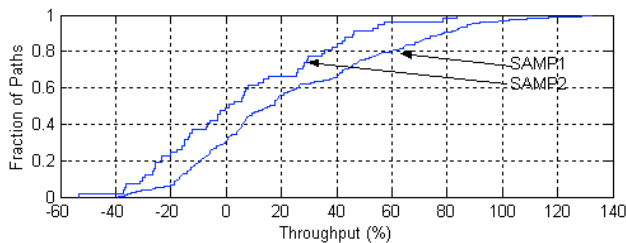


Figure 18. CDF of the total number of rerouted flow's average throughput compared to the overall average throughput of the corresponding direct path. (ALL) *SAMP1* vs. *SAMP2*.

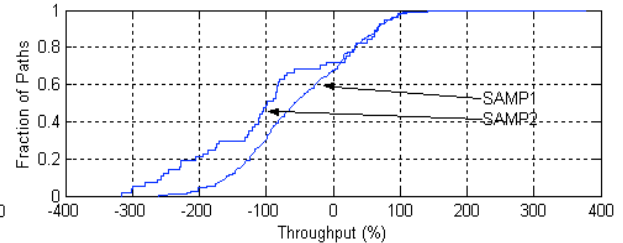


Figure 19. CDF of QRE's prediction accuracy of throughput for corresponding rerouted flow (ALL). *SAMP1* vs. *SAMP2*.

5.2.5 Rerouted Flows QRE Prediction Varying Sampling Rate

Another metric of interest was the sample rate. A higher sampling rate requires more processing and bandwidth overhead, therefore the smallest sampling rate is usually always desired. The default rate (*SAMP1*) had a sample block of 130 seconds and a reroute block of 130 seconds. Thus, the average throughput for a path was updated every 260 seconds. In Figures 18 and 19 we show how doubling this period and updating the average throughput every 530 seconds affected the number of flows benefiting from ALSW (Figure 18). Figure 18 shows the results of both sampling rates, we see that the number of flows that benefit from this technique is directly proportional to the sampling rate. Also, Figure 19 shows that the QRE prediction accuracy is directly proportional to the sampling rate.

Accordingly, as the sampling rate decreases, the QRE has a more coarse depiction of the network, thus producing a more inaccurate prediction of rerouted flow throughput.

6. Application Layer Striping

6.1 Background

Striping is the concept used to combine multiple resources transparently to obtain a higher level of performance [32]. Two primary areas where striping has proven beneficial are network systems [26] and in data archival and retrieval [33].

Striping in network systems at multiple layers has been deployed in the IP, as well as in the ATM worlds. In [34], the authors discuss, among other things, the benefit of ATM cell striping across several SONET frames. In RFC 2950 [35], the Stream Control Transmission Protocol (SCTP) is presented and can provide striping across one or more interfaces. References [23] and [36] present several Application Layer approaches to stripe data (along the same path) to improve throughput and to attain maximum utilization of network resources. Striping has also been used to improve throughput in wireless networks when data is striped over multiple wireless interfaces [37], as well as in satellite systems to alleviate the TCP window size problem in links with a long delay [38].

Disk striping is a concept used to spread data across multiple disks. This concept is not new, but has been used for many years. Disk striping is available in two forms: single-user and multi-user. Single-user striping is of interest to us and will be discussed further. Single-user striping allows data broken into specific units (labeled as the stripe width) to be written in parallel to multiple disk drives [33, 39]. Thus, the average transfer rate, theoretically, can be improved proportional to the number of drives. This is useful when transfer time (not head movement) is the bottleneck.

The striping concept addresses a general question: Can multiple resources be used in parallel to increase a specific performance metric? In the single-user disk striping scenario, the resources are disk drives, and the parameter of interest is transfer time. In this paper, we further extend this notion to the Internet. In such an extension, the resources are specific paths to a single destination and the metric of interest is transfer time or overall (cumulative) throughput. The stripe width could be chosen at the application layer and the data could similarly be striped across different paths to increase the cumulative throughput.

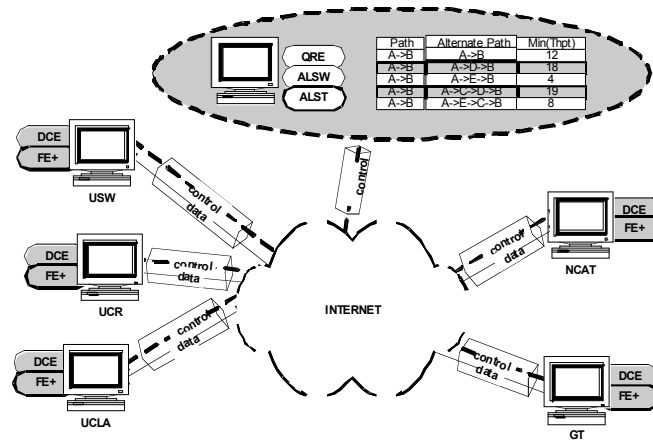


Figure 20. GALO Architecture extended to support Application Layer Striping.

6.2 Overview of Application Layer Striping

Application Layer Striping (ALST) is a concept introduced to improve the traditional, best-effort service provided by the Internet. Several techniques for striping have been proposed [23, 32, 34, 35, 36, 37], however none consider striping in the context of QoS-based routing. Further, most only consider streaming data over the same logical path. ALST alone is not considered QoS-based routing; but, when one considers the QoS of the links across which the data will be potentially striped, ALST falls into the category of QoS-based routing. The basic concept behind ALST is to use multiple paths in parallel along the Internet to carry portions of data destined for a certain node. This is achieved by constructing non-shortest path routes through intermediate nodes by appending different link combinations from the source node to end node. Figure 21 shows an example of a flow being striped over two higher bandwidth alternate paths.

ALST uses an overlay approach where no modification of intermediate nodes is necessary. We propose an extension to GALO, the Application Layer Striping Module. The more end nodes, contained in GALO, and the more geographically diverse they are, the greater the probability of alternate paths on which to stripe data. The accessibility table generated in the QRE is also passed as input to the ALST module where, based on the specified metrics (we only consider throughput), alternate paths are created by appending multiple links. Once the quality of the alternate paths is known and the number of paths to stripe has been decided (because of fairness we currently consider a maximum of three), the process begins. The PROVISION message is signaled to the corresponding FE within the path nodes to provision the forwarding engine in each node. Also, the source node and the destination node are signaled to specify where to send the traffic and from where to expect the traffic, respectively. Additionally, the source is given instructions on how to split the data before sending, and the receiver is given instructions on how to reassemble the data. The data is sent shortly thereafter. Alternatively, we can use an intelligent transport mechanism (i.e., pTCP [40]).

6.3 Application Layer Striping Software Architecture

Similar to ALSW, The input to the ALST module is the path quality table that is populated by the QRE. The ALST module monitors the shared memory segment containing the path quality table. The module then queries the path quality table and uses the data to generate the optimal alternative paths (a maximum of three), and thus makes the appropriate decision to stripe a flow. The optimal path algorithm is a variation of the popular Dijkstra's algorithm. The decision is passed back to the QRE, and the QRE messages to the respective DCE to split the data accordingly and to stripe the next flow.

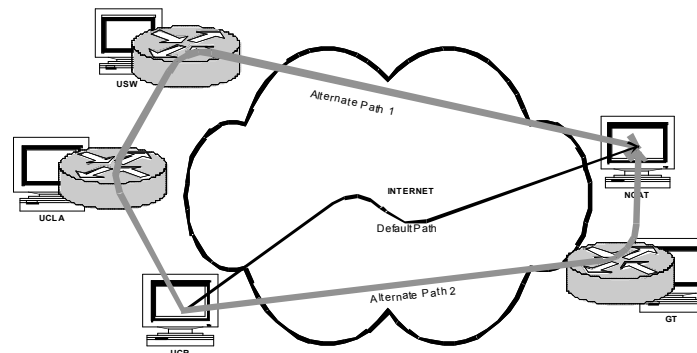


Figure 21. Example of a flow rerouted and striped using Application Layer Striping.

6.4 Application Layer Striping Logical Architecture

Mirroring the ALSW technique, ALST uses an overlay approach where no modification of intermediate nodes is necessary. Again, we extend GALO with the Application Layer Striping Module. The more end nodes and the more geographically diverse they are, contained in GALO, the greater the probability of alternate paths on which to stripe data. The reachability table generated in the QRE is also passed as input to the ALST module where, based on the specified metrics (we only consider throughput), alternate paths are created by appending multiple links. Once the quality of the alternate paths is known and the number of paths to stripe has been decided (because of fairness we currently consider a maximum of three), the process begins. The PROVISION message is signaled to the corresponding FE within the path nodes to provision the forwarding engine in each node. Also, the source node and the destination node are signaled to specify where to send the traffic and where to expect the traffic from, respectively. Additionally, the source is given instructions on how to split the data before sending, and the receiver is given instructions on how to reassemble the data. The data is sent shortly thereafter.

7. Performance Analysis of Application Layer Striping

7.1 Experimental Setup

Both ALSW and ALST are extensions of GALO, accordingly ALST experiments are performed on the same network as ALSW. As with the ALSW case, we designate two separate blocks: the sampling block and the reroute block. The QRE keeps a running average of each default path's throughput, and synthetically creates alternate paths. It then chooses the paths (maximum of three) with the highest potential increase as the primary candidates for

rerouting. During the next reroute cycle, the chosen candidate is rerouted over the alternate path. If data is rerouted over two paths, the flow is statically split in half (half of the default flow 100KB – 50KB per path). Likewise, if the flow is routed over three paths, each flow is approximately 33KB. The alternate paths are again limited to a maximum of two hops. The remainder of the experiment was performed exactly like ALSW (*Section 5.1*) with approximately 100 rerouted flows generated for both the striping over two and three path cases.

7.2 Results

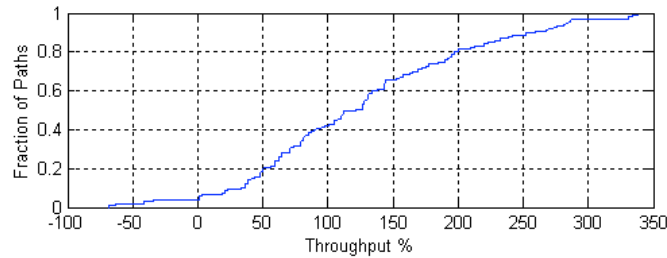


Figure 22. CDF of throughput increase over average when striping data over two paths.

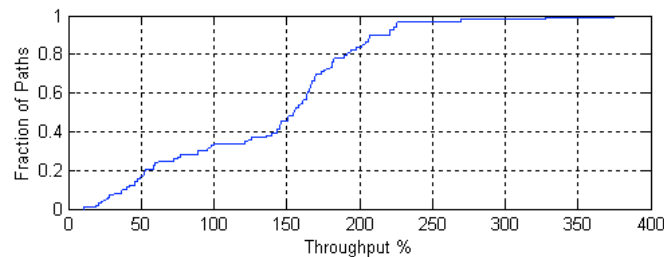


Figure 23. CDF of throughput increase over average when striping data over three paths.

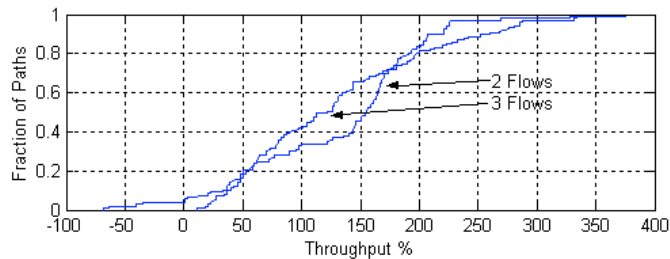


Figure 24. CDF of the comparison of the throughput increase over the default throughput, when striping over two and three paths.

The results for ALST are shown in Figures 22 - 24. Again, this is a best case scenario that assumes that the data is initially segmented at the sender using an intelligent striping protocol so that reassembly at the receiver is trivial. Additionally, these results do not take into account that the flow striped along the slower path will degrade the overall throughput because the aggregate throughput shown in the graphs is only valid when both flows are being received simultaneously. It is believed that this minimal delay will not destroy the integrity of the technique and that the flows, overall, will still benefit significantly from striping.

7.2.1 Two Striped Paths

Figure 22 shows the aggregate throughput of a 100KB flow being striped over the top two logical paths compared to the average throughput seen by the default path. We observe that over 90% of the flows benefit from this technique extending up to approximately 340%.

7.2.2 Three Striped Paths

Figure 23 illustrates the aggregate throughput of a 100KB flow being striped over the top three logical paths compared to the average throughput seen by the default path. We observe that all of the flows benefit from this technique with several flows exceeding 350%.

7.2.3 Two Striped vs. Three Striped Paths

Figure 24 compares the aggregate throughput increase over the default path of the flows striped over two and three paths. Surprisingly, both streams perform about the same with the striping over three paths performing marginally better. This behavior is a result of the overhead encountered with increasing the number of application layer switched paths. Additionally, the flows over the three paths were only 33KB opposed to 50KB, with the smaller file size having a less likely chance of taking advantage of TCP's maximum window size. Thus, it is believed that the data rate is still proportional to the number of paths over which the data is striped. However, we find that the scheme is also significantly affected by overhead encountered along the application layer switched path, as well as the size of the striped flow, thus, virtually offsetting the deficiencies and gains.

8. Conclusion & Future Work

In order to access the traffic load balance in the Internet we constructed a wide area network, Internet testbed comprised of universities across the US. The findings from this performance analysis, as well as larger more extensive analyses, show that the Internet suffers severely from improper traffic balancing. We proposed a deployable approach to improving QoS, by using the GALO overlay architecture along with the ALSW and ALST extensions in order to immediately, in a peer-to-peer approach, improve QoS for traffic flows. We show that by merely using alternate paths, around 70% of the reroute flows achieved a better average throughput than that of the default path. Additionally, we show that by using the striping technique, the flows achieve a maximum throughput increase over that of the average throughput of the single default path, with a maximum increase exceeding 350%.

The encouraging results that were shown are a worst case, in that they are generated from a testbed that used normal, shared machines as routers. It is our conviction that these results can be significantly improved by merely using dedicated machines, or in addition, in an ideal case, the use of Source Routing.

Though the GALO architecture has proven to be robust. We suggest two main areas for improvement: scalability and increased performance.

- **Distributed QRE:** One approach to allowing scalability is having a distributed QRE. The current architecture uses a centralized QRE scheme. This approach proves appropriate in the testbed because it contains only five end nodes. Having a centralized approach prevents the ease of scalability. Thus, in

order to have a scalable architecture we plan to extend this architecture to one which contains a distributed QRE. Once the architecture scales, this approach allows for a QRE to be placed in each autonomous system or domain. In order to prevent coherency problems, the QREs will communicate on a regular basis, updating accessibility information for every other domain. In this setup, the DCE would report to the nearest QRE. This approach requires more control traffic overhead, but provides a scalable solution.

- **Hierarchical routing with Domain Speakers:** Hierarchical routing provides another approach to providing scalability. This approach works best when you have several densely populated domains (i.e., all the nodes at several universities are a part of the WAN QoS network). In this case, each university would be classified as a group, and would have a dedicated group speaker. The group speakers of each domain would report their accessibility to the group speaker of every other domain, thus allowing the QRE table size to be directly proportional to the number of groups in the QoS network as opposed to the number of nodes in the QoS network. The drawback to this scheme is that accessibility information passed to the QRE is more coarse since the group speaker reports aggregate state information, as opposed to passing specific accessibility information for a node.
- **Intelligent Striping:** Intelligent striping is one scheme that can allow increased performance in the testbed. In the current architecture, when using Application Layer Striping, we statically divide the flow into halves or thirds depending on whether we are striping over two or three paths. The problem with this approach is that one path usually has significantly more bandwidth available and therefore, should have a larger portion of the flow striped over that path. Accordingly, the current striping scheme, though beneficial, is not the optimal approach to striping. We suggest a method of splitting the data that is based on the current throughput and round-trip-time of the path rather than the static approach implemented in the current architecture. A similar approach that performs intelligent striping, pTCP, is proposed in [40].

References

1. S. Floyd and V. Paxson, "Difficulties in simulating the Internet," IEEE/ACM Transactions in Networking, in press.
2. T. Hansen, J. Otero, T. McGregor, and H-W. Braun, "Active measurements data analysis techniques," unpublished.
3. A. Adams and M. Mathis, "A system for flexible network performance measurement," in Proceedings of INET 2000.
4. V. Paxson, "End-to-End routing behavior in the Internet," IEEE/ACM Transactions on Networking, Vol.5, No.5, pp. 601-615, October 1997.
5. V. Paxson, "End-to-End Internet packet dynamics", ACM SIGCOMM '97, September 1997.
6. Comparison of some Internet Active End-to-end Performance Measurement projects, <http://www.slac.stanford.edu/comp/net/wan-mon/iepm-cf.html>, Jan. 2002.
7. M. Borella, D. Swider, S. Uludag, and G. Brewster, "Internet packet loss: Measurements and implications for End-to-End QoS," in Proceedings of International Conference on Parallel Processing, Aug. 1998.
8. S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, "The End-to-End effects of Internet path selection," in Proceedings of the ACM SIGCOMM, Oct. 1999, vol. 29, pp. 289-299.
9. K. Thompson, G. Miller, and R. Wilder, "Wide-Area Internet traffic patterns and characteristics," IEEE Network, November/December 1997.
10. J. Mogul, "Observing TCP dynamics in real networks," in Proceedings of ACM SIGCOMM '92.
11. W.R. Stevens, TCP/IP Illustrated Volume 1: the protocols. Boston, MA: Addison Wesley, 1994.
12. R. Beyah, R. Sivakumar, and J. Copeland. "A Measurement-based Study of End-to-End Internet Traffic Dynamics." Technical Report GT-CSC-2003.
13. A. Ghosh, M. Fry, and J. Crowcroft. "An Architecture for Application Layer Routing." In the Proceedings of IWAN 2000.
14. S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan, "Detour: a Case for Informed Internet Routing and Transport," IEEE Micro, pp. 50-59, v 19, no 1, January 1999.
15. www.planet-lab.org
16. www.emulab.net
17. L. Subramanian, I. Stoica, H. Balakrishnan, and R.y Katz, "OverQoS: Offering Internet QoS Using Overlays," 1st HotNets Workshop, October 2002.
18. Z. Li and P. Mohapatra, "QRON: QoS-Aware Routing in Overlay Networks," IEEE Journal on Selected Areas in Communications, v 22, Number 1, January 2004.
19. D. Andersen, H. Balakrishnan, M. Kaashoek, R. Morris, "Resilient Overlay Networks," Proc. 18th ACM SOSP, Banff, Canada, October 2001.
20. M. Stemm, "Vertical Handoffs in Wireless Overlay Networks," Master's thesis, UC Berkeley, May 1996.
21. R. Katz and E. Brewer. "Wireless Overlay Networks and Adaptive Applications." In the Proceedings of MobiCom 1996.
22. VPN Overlay Networks: An Answer To Network-Based IP VPNs? <http://networkmagazine.com>, February 2002.
23. Stephanos and Androutsellis-Theotokis. White Paper: A Survey of Peer-to-Peer File Sharing Technologies. ELTRUN, Athens University of Economics and Business, Greece.
24. H. Erikson. "MBONE: The Multicast Backbone". Communication of the ACM, pages 54-60, August 1994.
25. M. R Macedonia, D. P. Brutzman, "MBone provides audio and video across the Internet," IEEE Computer, Vol.27 #4, April 1994, pp. 30-36.
26. How to Debug an MBone Session. <http://www.informatik.unimannheim.de/informatik/pi4/projects/CoSt264/HowToDebugMBone.html>. April 2002.
27. S. Seshan, M. Stemm, and R. Katz. Spand. "Shared passive network performance discovery." In the Proceedings of Usenix Symposium on Internet Technologies and Systems (USITS '97), Monterey, CA, December 1997.
28. M. Stemm, S. Seshan, and Randy H. Katz. "A network measurement architecture for adaptive applications." In the Proceedings of IEEE Infocom 2000, Monterey, CA, March 2000.
29. B. Mah. pchar: A tool for measuring internet path characteristics, 2001. <http://www.employees.org/~bmah/Software/pchar/>.
30. V. Jacobson. pathchar - a tool to infer characteristics of internet paths, 1997. presented as April 97 MSRI talk.
31. N. Hu and P. Steenkiste. Estimating Available Bandwidth Using Packet Pair Probing. September 9, 2002 CMU-CS-02-166. Technical Report.
32. C. Brendon, S. Traw, and J. Smith, "Striping within the network subsystem," IEEE Network, vol. 9, no. 4, pp. 2232, July 1995.
33. Disk Striping http://www.webopedia.com/TERM/D/disk_striping.html. April 2002.
34. J. Duncanson, "Inverse multiplexing," IEEE Communications Magazine, vol. 32, no. 4, pp. 3441, Apr. 1994.
35. R. Stewart et al, "Stream control transmission protocol," IETF RFC 2960, Oct. 2000.
36. H. Sivakumar, S. Bailey, and R. Grossman, "PSockets: The case for application level network striping for data intensive applications using high speed wide area networks," in Proceedings of IEEE Supercomputing (SC), Dallas, TX USA, Nov. 2000, pps. 240-245.
37. H.-Y. Hsieh and R. Sivakumar, "A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts." ACM MOBICOM, Atlanta, GA, September 2002.

38. M. Allman, H. Kruse, and S. Ostermann, "An application-level solution to TCP's satellite inefficiencies," in Proceedings of Workshop on Satellite-Based Information Services (WOSBIS), Rye, NY USA, Nov. 1996.
39. Disk Striping as a Performance Enhancement Tool for Multi-User (UNIX) Systems.
<http://www.1776soft.com/striping.htm>. February 2002.
40. H.-Y. Hsieh and R. Sivakumar, "pTCP: An end-to-end transport layer protocol for striped connections." In the Proceedings of ICNP 2002.

APPENDIX A

Pseudo Code - GALO Distributed Client Engine (DCE)

Provision QRE control channel

Define socket descriptor for the incoming and outgoing control data

Create shared memory segment using *shmget*

Provision incoming outgoing ports for UCLA, UCR, USW, NCAT, and GT

Define socket descriptor for incoming data connections

Define ports used to receive incoming data

Define IP addresses of destination of data

Define socket descriptors for the outgoing data communication

Define ports used to switch and transmit outgoing data

Create process for each destination (fork)

Open, prepare, accept, server socket (listening)

do while (1) //done in each processes

 Read incoming control data from the QRE (*control_data*)

 Populate shared memory segment with *control_data*

end while

if (control_data == collect_stats) //separate process

do while (1)

if data is received on switching socket

 Sample data and collect statistics

 According to sampling rate, send data to QRE using ALCP

end if

end while

else if (control_data == (reroute || stripe)) //separate process

do while (1)

if data is received on switching socket

 During appropriate slot, stripe data or send regularly through ALSW

end if

end while

end if

Send background traffic data to each of the four other nodes during appropriate slot //separate process

APPENDIX A (Continued)

Pseudo Code - GALO Forwarding Engine (FE)

Provision QRE control channel

Define socket descriptor for the incoming and outgoing control data

Provision incoming and outgoing ports for UCLA, UCR, USW, NCAT, and GT

Define socket descriptor for incoming data connections

Define ports used to receive incoming data

Define IP addresses of destination of data

Define socket descriptors for the outgoing data communication

Define ports used to switch and transmit outgoing data

Populate switching table

Populate corresponding ports for incoming data

Populate corresponding ports for outgoing data

Populate corresponding IP addresses for outgoing connections

Create process for each destination (fork)

Open, prepare, accept, server socket (listening)

do while (1) //done in each process

if data is received

Create appropriate client outgoing socket

Read incoming data and write outgoing data

Close socket after flow has terminated

end if

end while

Pseudo Code - GALO QoS Routing Engine (QRE)

Provision incoming and outgoing control ports for UCLA, UCR, USW, NCAT, and GT

Define socket descriptors for incoming data connections

Define ports used to receive incoming data

Define IP addresses of destination of data

Define socket descriptors for the outgoing data communication

Define ports used to switch and transmit outgoing data

Create shared memory segment using *shmget*

Create process for each destination (fork)

do while (1)

Read incoming path quality updates from DCEs

Store path quality updates in table in shared memory segment

Calculate optimal path for each node using a modified Dijkstra algorithm

Receive throughput updates and during striping or switching

Determine flow to reroute (one with most improvement above default route)

Signal to appropriate DCE to reroute the appropriate flow

end while
