Application Layer Switching: A Deployable Technique for Providing Quality of Service

Raheem Beyah, Raghupathy Sivakumar, and John Copeland
School of Electrical and Computer Engineering
Georgia Institute of Technology

***Abstract* - In this paper, we propose a deployable approach to improving QoS by using the familiar overlay architecture approach. *The goals of this work are to 1) create an overlay architecture which allows us to sample specific path quality metrics among different paths; and 2) utilize the proposed overlay architecture in order to implement our proposed QoS-based routing scheme, Application Layer Switching (ALSW).* We show 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 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.**

I. INTRODUCTION

The Internet has evolved to provide a multitude of services. In order to accommodate desired real-time services, or to merely achieve better than best-effort service[1], some level of Quality of Service (QoS) must be established. Several different techniques have been proposed to resolve this issue, all of which require a moderate to significant overhaul of the current Internet infrastructure. 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 results that 70% of the rerouted flows experienced a 30%-120% increase in throughput over the average throughput of the default path.

The rest of this paper is organized as follows: Section II 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 III we discuss the motivation and description of the GALO architecture. Section IV discusses an extension to GALO, the ALSW module. Section V gives the performance analysis using ALSW, and Section VI concludes the paper.

II. ALTERNATE PATH EVALUATION AND MOTIVATION

A. 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.

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.

[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.

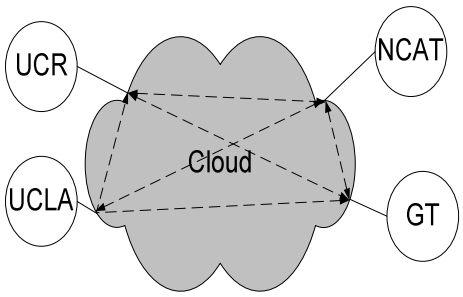

Fig. 1. QoS WAN testbed.

B. *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) (Fig. 1).

We conducted our analysis using a modified version of the ***sock*** program [3]. 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.

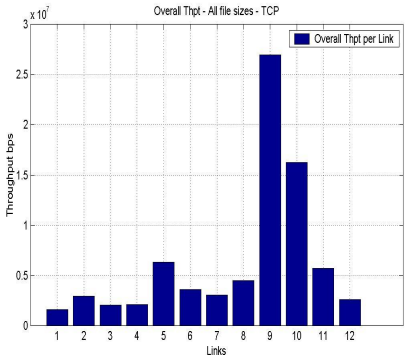

Fig. 2. Average throughput of each path.

Table 1. Link descriptions.

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

C. *Results*

Fig. 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 Link 1 is plagued with the stigma of the slowest link. This study reaffirms what many before it has 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.

D. *Alternate Paths*

Until the work done in [1], most literature in the traffic measurement community stopped at that conclusion drawn above. However, in [1], 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.

Building on the work done in [1], 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. Fig. 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, Fig. 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.

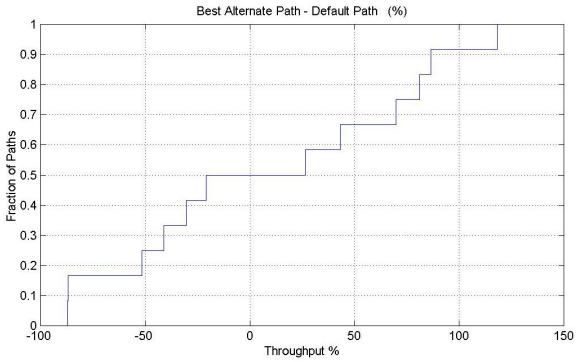

Fig. 3. Best alternate path - default path (%).

III. Generalized Application-Layer Overlay

A. *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 [7, 8], virtual private networks [9], computer virus enabling, peer-to-peer file sharing networks [10], and probably the most popular, end system multicast [4, 5].

B. GALO Logical Architecture

Thus far, we have given an overview of QoS and QoS-based routing. We have also presented an empirical study of Internet traffic dynamics. The throughput measured in this empirical study of the WAN links is extracted and used as motivation for additional work. We now propose an overlay architecture that will provide the backbone for an immediately deployable QoS-based routing scheme. In accordance with past overlay approaches [4, 7, 8, 9, 10], 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 which requires the FE functionality.

- ***DCE:*** The DCE is a process that resides 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.
- ***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 signaling[2] 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. 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 (in Section V) that this delay will be negligible and a significant amount of the rerouted 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 [11, 12] and active probing [13, 14, 15]. 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 have recurrent communication. In our model, to allow bandwidth measurements, we emulate the aforementioned environment and create our own data to passively sample.

To obtain the path quality, artificial traffic is generated on each link using a modified version of the sock program [3]. 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 stream sockets are reserved for messages between the DCE and QRE.

IV. APPLICATION LAYER SWITCHING

A. 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.

[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.

Application Layer Switching was initially implicitly motivated by the work done in [1]. In [1], 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 [2] and loaned to the authors of [1]. 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.

Peripherally considering QoS, the work done in [16], suggests using active networking and generic devices on workstations to perform application-layer routing. Other noted related work includes the Detour [17] and RoN [18] projects. Though both [1] and [16] discuss the benefit of using alternate paths, neither implements an architecture or provides results. We strengthen their work by implementing an overlay architecture, extending this architecture using the concept of Application Layer Switching that does not require any modification to the current infrastructure, and run experiments producing encouraging results.

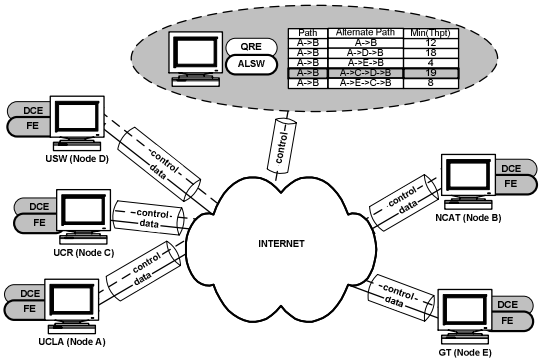

Fig. 4. *GALO* Architecture extended to support *Application Layer Switching.*

B. 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.

C. Application Layer Switching Software Architecture

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 route 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.

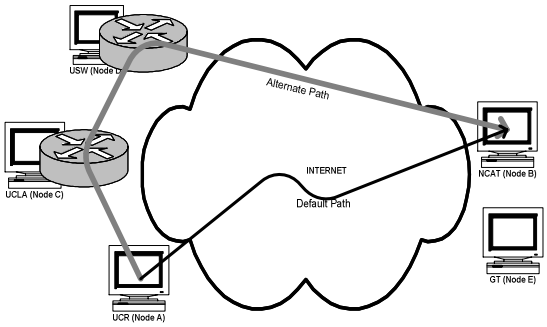

Fig. 5. Example of a flow rerouted using *Application Layer Switching*

To accomplish ALSW, we propose an extension to our proposed Generalized Application-Layer Overlay, the Application Layer Switching module (Fig. 4). 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 (as discussed in Section III). Periodic updates are broadcast to the QRE using the Application Layer Communication

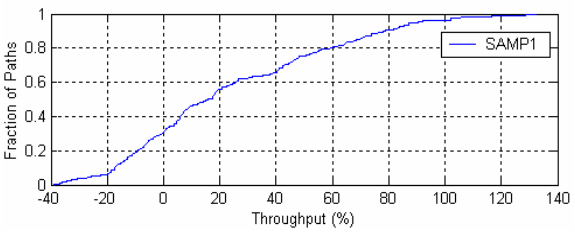

Fig. 6. CDF of the total number of rerouted flow's average throughput compared to the overall average throughput of the corresponding direct path. *SAMP1.*

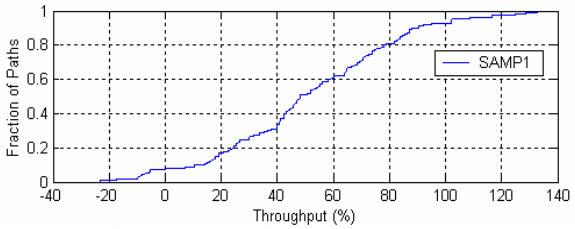

Fig. 7. CDF of the total number of rerouted flow's average throughput compared to the overall average throughput of the corresponding direct path (SINGLE HOP). *SAMP1.*

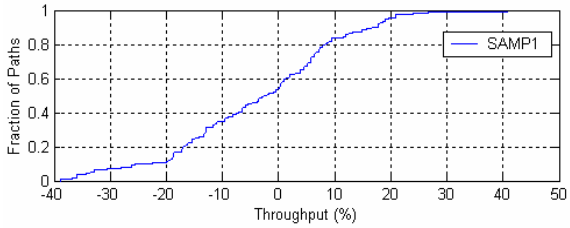

Fig. 8. CDF of the total number of rerouted flow's average throughput compared to the overall average throughput of the corresponding direct path (MULTIPLE HOP). *SAMP1.*

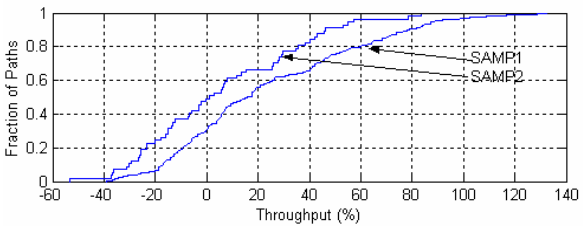

Fig. 9. CDF of the total number of rerouted flow's average throughput compared to the overall average throughput of the corresponding direct path. *SAMP1 vs. SAMP2.*

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 (Fig. 5). 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.

V. PERFORMANCE ANALYSIS

A. Experimental Setup

We expanded the initial network discussed in Section II 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 flow[3] 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. Fig. 9 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

[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.

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.

B. 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. Additionally, we vary sampling rate and decouple multiple/single hop data and observe their effect on the flows.

B.1 Rerouted Flows at Default Sampling Rate

Confirming the hypothesis in *Section II-D.*, Fig. 6 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.

B.2 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. Figs. 7 and 8 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 a 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 increased.

B.3 Rerouted Flows Benefit 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 Fig. 9 we show how doubling this period and updating the average throughput every 530 seconds affected the number of flows benefiting from ALSW. When observing data generated using both sample rates, we see that the number of flows that benefit from this technique is directly proportional to the sampling rate.

VI. 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 extension 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. 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.

REFERENCES

[1] 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.

[2] V. Paxson, “End-to-End Internet packet dynamics”, ACM SIGCOMM '97, September 1997.

[3] W.R. Stevens, TCP/IP Illustrated Volume 1: the protocols. Boston, MA: Addison Wesley, 1994.

[4] H. Erikson. “MBONE: The Multicast Backbone”. Communication of the ACM, pages 54–60, August 1994.

[5] M. Macedonia, D. Brutzman, "MBone provides audio and video across the Internet," IEEE Computer, Vol.27 #4, April 1994, pp. 30-36.

[6] How to Debug an MBone Session. http://www.informatik.unimannheim.de/informatik/pi4/projects/Cost264/HowToDebugMBone.html. April 2002.

[7] M. Stemm, "Vertical Handoffs in Wireless Overlay Networks," Master's thesis, UC Berkeley, May 1996.

[8] R. Katz and E. Brewer. “Wireless Overlay Networks and Adaptive Applications.” In the Proceedings of MobiCom 1996.

[9] VPN Overlay Networks: An Answer To Network-Based IP VPNs? http://networkmagazine.com. February 2002.

[10] Stephanos and Androutsellis-Theotokis. White Paper: A Survey of Peer-to-Peer File Sharing Technologies. ELTRUN, Athens University of Economics and Business, Greece.

[11] S. Seshan, M. Stemm, and R. Katz. Spand. “Shared passive network performance discovery.” In the Proceedings of Usenix Symposium on Internet Technologies and Systems, Monterey, CA, December 1997.

[12] 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.

[13] B. Mah. pchar: A tool for measuring internet path characteristics, 2001. http://www.employees.org/ bmah/Software/pchar/.

[14] V. Jacobson. pathchar - a tool to infer characteristics of internet paths, 1997. presented as April 97 MSRI talk.

[15] N. Hu and P. Steenkiste. Estimating Available Bandwidth Using Packet Pair Probing. Sept. 2002 CMU-CS-02-166. Technical Report.

[16] A. Ghosh, M. Fry, and J. Crowcroft. “An Architecture for Application Layer Routing.” In the Proceedings of IWAN 2000.

[17] 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.

[18] D. Andersen, H. Balakrishnan, M. Kaashoek, R. Morris, "Resilient Overlay Networks," Proc. 18th ACM SOSP, Banff, Canada, October 2001.