7-1-2005

Slot: Shortened Loop Internet Transport using Overlay Networks

Himabindu Pucha

Y. Charlie Hu

SLOT: SHORTENED LOOP INTERNET TRANSPORT USING OVERLAY NETWORKS

HIMABINDU PUCHA
Y. CHARLIE HU

TR-ECE – 05-12
JULY 2005

PURDUE UNIVERSITY

SCHOOL OF ELECTRICAL
AND COMPUTER ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, IN 47907-2035

Slot: Shortened Loop Internet Transport using Overlay Networks

Himabindu Pucha
Y. Charlie Hu

TR-ECE-05-12
July 1, 2005

School of Electrical and Computer Engineering
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907-1285

Contents

Abstract

Overlay routing has emerged as a promising approach to mitigating many problems with Internet routing, such as improving the reliability of Internet paths and supporting multicast communication. As overlay routing is gaining wider acceptance, we argue that it is time to investigate how overlay networks can benefit Internet transport. This paper presents ***Slot****, a framework that leverages overlay networks to improve the throughput of feedback-based transport protocols. Slot exploits the observation that the throughput of feedback-based transport protocols (e.g., TCP, XCP, VCP, DCCP) is inversely proportional to the length of their end-to-end feedback control loop, and* ***effectively shortens*** *an end-to-end control loop by breaking it up into multiple pipelined shortened sub-loops via intermediaries carefully chosen from an overlay network. As a result, Slot increases the throughput of an end-to-end transport connection to that of the longest sub-loop.*

This paper studies the potential of Slot and addresses key challenges in the design and the deployment of Slot. The contributions of this paper are three-fold. First, we make the case for Slot by measuring and analyzing the control loop lengths of close to 3.7 million node pairs and their potential benefit from Slot using PlanetLab as an example overlay network. Second, we identify key challenges in the design of Slot and show that a simple, low overhead solution can be used to select an overlay path that can achieve close to the maximum throughput improvement possible. Third, we implement a prototype of Slot and deploy it on PlanetLab to fetch a large set of files crawled from popular web servers. Our results show that compared to directly fetching the same documents, Slot improves the throughput of 95% of the large file transfers, and 50% of these transfers achieve more than 30% increase in throughput.

1 Introduction

Overlay networks (Detour [38], RON [8], PlanetLab [33], CDN [1]) have emerged as a promising platform for mitigating many problems with Internet routing. Recent studies have exploited overlay routing to improve reliability, provide QoS, support multicast communication, etc. As overlay routing is gaining wider acceptance, we argue that it is time to investigate how overlay routing can benefit Internet transport. Specifically, in this paper, we present *Slot*[1], a framework

[1] *Slot* stands for Shortened Loop Overlay Transport.

that leverages overlay networks to improve the throughput of Internet transport protocols.

Internet transport protocols typically employ a feedback-based control loop to determine the amount of and the timing of the data bytes to be sent in order to perform congestion and flow control. Thus, the throughput obtained by any feedback-based transport protocol is directly affected by the length of the feedback loop between the sender and the receiver. For example, in TCP, the dominant Internet transport protocol, a *window* of data bytes is sent when feedback (in the form of ACKs) for the previously sent bytes is received from the receiving end host. Also, TCP performs multiplicative decrease followed by additive increase of its window size to cope with congestion between the sender and the receiver. The rate of additive increase in TCP is controlled by the feedback from the receiver. Intuitively, shortening this feedback loop increases the feedback rate allowing the transport protocol to send its window of data bytes more rapidly. Further, a shorter feedback loop also allows the *size* of this window to increase faster. These two effects enabled by a shorter feedback control loop potentially lead to higher transport throughput.

Slot exploits the above observation and aims to improve the throughput achieved by feedback-based transport protocols by *effectively shortening* their feedback control loop using carefully chosen intermediaries from an overlay network. Specifically, Slot breaks up an end-to-end feedback control loop into multiple shortened sub-loops. These sub-loops, i.e., the control loops of the sub TCP connections, together form a multi-overlay-hop path over which data bytes are pipelined to enable end-to-end transport. Due to the pipelining effect, the throughput of an end-to-end Slot path is constrained by the throughput of the worst overlay hop (i.e. the one with the longest sub-loop). Therefore, in order to provide throughput gains, Slot chooses a multi-hop path such that the *maximum length of the sub-loops is shorter than the length of the end-to-end loop.* Slot uses the RTT between two nodes as a measure of the *control loop length* between those two nodes. To maximize the throughput gains, Slot replaces the end-to-end path between two nodes with a multi-hop-overlay path that minimizes the maximum of the RTTs of each overlay hop in the path. Feedback-based transport protocols that will benefit from Slot include reliable protocols such as TCP, XCP [22], VCP [47] and unreliable protocols such as DCCP [25]. Without loss of generality, we use TCP as an example feedback-based transport protocol in the remainder of this paper.

In this work, we first identify the potential of Slot and then address the challenges in the design and the deployment of Slot. An assessment of the potential throughput benefits from Slot

via a large scale measurement study of the RTTs between close to 3.7 million node pairs in the Internet shows that Slot can potentially improve the throughput of more than 95% of the node pairs. Encouraged by this potential gain, we studied the design space of the Slot framework. A fundamental challenge in the design of Slot is to develop a light-weight mechanism that can discover an overlay path that has a low maximum per-hop RTT without incurring high network overhead and long delay. We approach this problem by answering the following two questions: (1) How many hops should the overlay path consist of? (2) How do we find a close-to-optimum overlay path without probing all possible intermediate nodes?

To answer these questions, we comprehensively studied the performance of different strategies using trace-driven simulation and a deployment on 404 PlanetLab nodes. Although it appears that continuously splitting the connections is likely to keep improving the end-to-end throughput performance, surprisingly, we found that reasonably close to maximum throughput improvement can be achieved by shortening the end-to-end loop into only two sub-loops using *one intermediary*. Compared to picking an intermediary node in resilient routing [8, 17] where every overlay node has a high probability of repairing a routing failure, picking the intermediary that shortens the end-to-end loop into two sub-loops is a much harder problem and seemingly requires probing a large number of overlay nodes. Interestingly, we show that a simple, low overhead solution can be used to select an overlay path that can achieve close to the maximum throughput improvement possible, using either an application-specific technique or leveraging an existing network location service.

Finally, to measure the benefit end users experience from running unmodified applications over Slot, we developed a prototype implementation of Slot. Our implementation uses Netfilter [4] to transparently hijack TCP connections on the client machines and route them through the intermediaries chosen to minimize the maximum sub-loop length among possible overlay paths. Using a web-browsing workload consisting of 6,492 files crawled from 1,000 popular web servers, driven repeatedly using an unmodified wget client on a daily basis over a period of one week, Slot was found to improve the TCP throughput for 95% of the large file transfers. In particular, 50% of these transfers had a throughput increase of more than 30%. Further, results from an unmodified scp client running over Slot to upload a Linux rpm to PlanetLab nodes showed that 50% of the transfers had more than 25% throughput gain.

This paper first discusses how Slot compares to other approaches to improving Internet transport (Section 2), explains why the shortened loop improves throughput (Section 3), assesses the potential throughput improvement of Slot over the wide area network (Section 4) and presents major design challenges faced by Slot and our practical solutions (Section 6). The paper then describes our Slot implementation and its evaluation using unmodified client applications (Section 7). Finally, the paper discusses implications of Slot in a wide-spread deployment (Section 8).

2 Related Work

Slot is the first complete framework that leverages overlay networks to improve the throughput of any feedback-based transport protocol by shortening their feedback control loop. It is not tied to any specific application or protocol, and is designed to be transparent to the OS, client and server applications, as well as the specific transport protocol used.

Several variations of TCP have been previously proposed to improve transport throughput. Techniques such as buffer tuning [39] and WAD [15] improve TCP throughput by tuning TCP parameters such as the socket buffer size and the MTU size. Protocols such as HSTCP [16], FAST TCP [21], and Scalable TCP [23] modify TCP protocol features such as the congestion control algorithm. However, all of these variations of TCP still rely on end-to-end feedback, and hence can benefit from Slot which shortens the feedback loop and provides them with faster feedback. Another class of transport protocols (e.g., XCP [22], VCP [47]) improve transport performance by using explicit network feedback from the routers along the path. However, since this feedback information still reaches the receivers and is then sent back to the senders, the performance of these protocols is still limited by the control loop length, and Slot can potentially benefit these protocols as well by shortening the feedback loop. In summary, Slot does not modify transport protocols or tune their parameters, instead only relies on the network properties to improve transport throughput by shortening the feedback loop, and hence is complementary to all the above approaches.

Another class of techniques (parallel TCP) [40, 49, 20] open multiple sockets to the same server over the same physical path or disjoint physical paths to improve TCP throughput. Slot is complementary to such techniques as it can be used to transparently shorten the feedback loop of each of the parallel TCP connections. In contrast to Slot, parallel TCP techniques require

changes to either the application or the OS. Additionally, parallel TCP unfairly grabs a larger share of bandwidth than single-path flows. In contrast, the behavior of a flow under Slot over any given overlay hop is the same as an unmodified TCP flow.

The technique of splitting TCP connections when they span widely varying transmission mediums was proposed in order to decouple and isolate the effect of the different mediums on the same TCP connection. For example, ITCP [11, 48] proposes to split TCP connections into separate wired and wireless sub-connections, STP [19] splits the TCP connection in the presence of satellite links, and the work in [13] splits TCP connections in HFC networks. An improvement in TCP throughput was observed as an auxiliary effect in these scenarios. In contrast to these work, Slot aims to shorten the feedback loops of the TCP connections that traverse the same media to achieve throughput improvement.

Splitting TCP connections has also been put into practice on several occasions, but in an ad hoc fashion, i.e., they are often scenario or application specific. For example, splitting TCP connections has been proposed to improve throughput in overlay multicast [27], media streaming applications [29] and Grid computing applications [42], to improve throughput and fairness in mobile ad hoc networks [26], and to improve average packet delay and jitter for fixed sending rate applications [7]. In contrast to these work, Slot exploits overlay networks to provide a general purpose, scalable, and transparent method for shortening feedback loops that is not tied to any application or protocol. Additionally, the emphasis of this work is on a complete system design that incorporates transparent connection hijacking and efficient intermediary selection and management, as well as answering fundamental questions about the potential and practical benefits of loop shortening for a large set of node pairs in the Internet.

Proxies of any kind inherently split TCP connections [37]. However, this split is ad-hoc in nature depending on the placement of the proxy and does not necessarily provide throughput gains since the maximum sub-loop RTT can still be very close to the direct RTT. For example, a caching proxy in an organization will split a TCP connection but provide almost no end-to-end throughput gains since the feedback loop length from the proxy to the server is likely to be similar to that from the client to the server. In contrast, Slot does not limit itself to splitting a TCP connection using nodes only on the direct path between the client and the server and intelligently chooses intermediaries that minimize the maximum sub-loop length. In addition,

certain high performance proxies may implement TCP splicing [41] which effectively "un-splits" two TCP connections going through a proxy into one end-to-end connection to improve forwarding performance and memory usage and thus cannot benefit from TCP splitting.

Alternate path overlay routing has been previously exploited in [38, 8, 17] to recover from network routing failures. Similar to these approaches, Slot can also route around failures since alternate path discovery is an inherent part of the intermediary selection process. Alternate routing has also been exploited to improve the end-to-end performance [38, 8] by choosing an overlay path that minimizes the end-to-end latency ($\sum RTT_i$). In contrast, Slot improves the end-to-end throughput by splitting the end-to-end control loop into sub-loops and minimizing the maximum of RTTs of the sub-loops along an overlay path ($max(RTT_i)$). In fact, in our experiments, 80% of the paths chosen by Slot have similar or higher end-to-end latency than the direct path, but still provide throughput gains; such paths are not exploited in previous alternate path overlay routing approaches.

3 Slot: Principle

In this section, we explain the principle behind the throughput improvement that Slot provides using TCP as an example. We first describe the dynamics of TCP and the factors that affect its throughput.

3.1 TCP Dynamics

TCP uses a sliding window protocol to control the flow of packets between a sender and a receiver. This sliding window protocol maintains a window size (W) that determines the amount of data (in bytes) that can be transmitted before an acknowledgment is required. TCP relies on feedback (ACKs) for previously sent data bytes to determine when to send the next window of data bytes. Further, TCP performs congestion control by dynamically increasing or decreasing W. TCP uses two algorithms for increasing W. At the beginning of a connection, it uses the Slow Start algorithm that increases W exponentially. Upon reaching steady-state, TCP uses the Congestion Avoidance algorithm to additively increase W. In both cases, the rate at which W increases is controlled by the feedback from the receiver. When a packet loss occurs (indicating congestion), W is multiplicatively decreased. Following this, as ACKs arrive, W is increased either

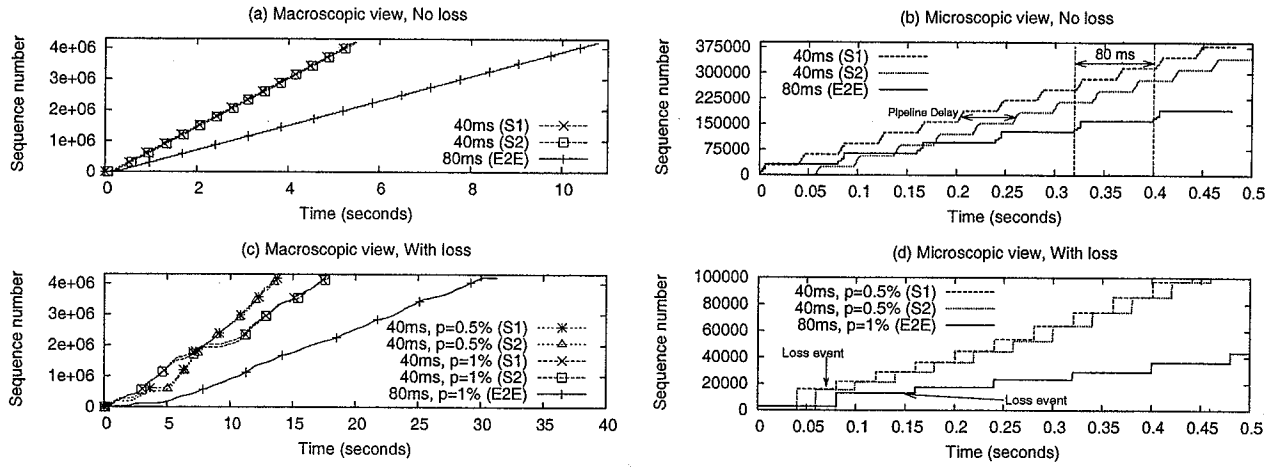

Figure 1: Macroscopic and microscopic investigation of the benefit from Slot loop shortening.

additively or exponentially based on the value of W.

Thus, the throughput of a TCP connection depends on how fast the next window of bytes can be sent and how fast the value of W increases, both of which are controlled by the length of the feedback loop. Further, throughput is also affected by the number of times a window decreases, which depends on the packet loss probability. Analytical results on TCP dynamics [31] also characterize this dependence as TCP throughput$=\frac{\sqrt{3/2}}{RTT\sqrt{p}}$, where RTT is the end-to-end delay between the sender and the receiver and p is the packet loss probability. Note that the feedback loop length between two nodes is determined by the RTT between those two nodes.

3.2 Slot

TCP dynamics indicate that shortening the feedback loop between any two end hosts A and B leads to faster transmission of successive windows of data bytes as well as faster increase in the window size, potentially increasing the throughput. Slot exploits this observation to improve TCP throughput by *effectively* shortening the feedback loop between A and B. Since an end-to-end feedback loop cannot be *physically* shortened, Slot mimics shortening of the feedback loop between A and B by **breaking** the direct path into a multi-overlay-hop path (called multi-hop paths henceforth), where each overlay hop is an independent TCP connection, such that the RTT of each overlay hop is lower than the direct RTT between A and B. We denote the maximum of all the overlay hop RTTs as the *path RTT*, i.e., $path\ RTT = max(RTT_i)$, $\forall i \in [1, n]$, where RTT_i is the RTT of the ith hop in an n-hop path. Further, once a TCP transfer starts, Slot uses **pipelining** to transfer the data bytes through the multi-hop path. In this scenario, each

hop's throughput is governed by the length of that hop's sub-loop and thus will be individually higher than the direct throughput between A and B. Further, because of pipelining, the end-to-end throughput between A and B is constrained by the throughput of the worst overlay hop, i.e., the one with the highest RTT. As a result, the throughput of an n-hop Slot path using $n-1$ intermediaries is given as $T_{multi\_hop} = min(T_i),\ \forall i \in [1, n]$. Thus, Slot effectively shortens the feedback loop between A and B to $pathRTT$ using path breaking and then data pipelining.

To demonstrate the impact of path breaking and data pipelining on the end-to-end throughput, we configured three FreeBSD machines (A, B and S) using *dummynet* [36], such that RTT from A to B is 80ms and RTT from S to both A and B is 40ms each. We then studied the behavior of a direct end-to-end path $A \rightarrow B$ (E2E) by transferring a 4MB file. The same file was then pipelined over a multi-hop path that breaks the direct path into two sub-paths, $A \rightarrow S$ (S1) and $S \rightarrow B$ (S2).

3.2.1 Impact of path breaking

For clarity of discussion, we first study the impact of breaking by comparing E2E and S1 alone under two scenarios.

In the **No Loss** scenario, we set packet loss on E2E, S1 and S2 to be zero. Figure 1(a) shows that the TCP sequence number growth (captured using *tcpdump* and *tcptrace*) on S1 is twice as fast as on E2E, thereby doubling the throughput. A microscopic look at the normalized sequence number growth from 3 to 3.5 secs during the file transfer (Figure 1(b)) reveals that in 80ms, E2E sends only one window of packets while S1 sends two windows of packets as its ACKs come back once every 40ms. Thus, in a lossless scenario, TCP throughput is improved due to faster transmission of successive windows of data bytes from breaking the direct path.

We now study a scenario **With Loss**. First, we consider the case when E2E, S1 and S2 have equal loss rates of 1%. Figure 1(c) shows that the throughput of S1 is approximately double that of E2E. In this case, S1 benefits from faster feedback as well as faster regrowth of W after a loss event, due to the shortened loop. However, since the loss rate for both S1 and E2E is similar, S1 incurs more loss events per unit time than E2E as it transfers more packets per unit time. Thus, the gain from the faster regrowth of W is offset by the higher number of loss events per unit time. Consequently, the throughput gain remains double (similar to the no loss case).

Second, we chose a loss rate of 0.5% for S1 and S2 and 1% for E2E such that the loss events per unit time for E2E and S1 are equivalent. Figure 1(c) shows that in this case, the throughput of S1 is more than double (2.38 times) that of E2E. A microscopic look at the normalized sequence number growth over a small time span (Figure 1(d)) reveals that after a loss event, S1 regrows its window faster than E2E. So before the next loss event occurs, S1 sends more than double the number of packets than E2E. In summary, in a lossy scenario, TCP throughput can be increased due to both faster transmission of successive windows of data bytes and faster increase in window size, and the magnitude of the throughput gain from loop shortening varies depending upon the loss rate of the shortened path versus that of the direct path.

3.2.2 Impact of data pipelining

The throughput on S2 determines the final throughput when using Slot. The microscopic view Figure 1(b) shows that the pipelined segment S2 closely follows the growth of S1 except that it lags behind S1 by a pipeline delay of 60ms[2]. Thus, similar to S1, the throughput on S2 is also almost double that of E2E. Further, the pipeline delay has a negligible impact on the overall end-to-end throughput as it is insignificant compared to the total transfer time. When losses occur, pipeline delay dynamically changes based on the relative losses on S1 and S2. For example, pipeline delay can decrease to a minimum value of $\frac{RTT_{S1}}{2}$ if S2 catches up with S1 due to losses on S1, as shown in Figure 1(d).

Note that the results above are with minimal application-level buffering of 8K per connection at the intermediary, and hence data buffering is largely limited to the 32K send and receive socket buffers. Incorporating larger application-level buffers can potentially help in cases where one side of the connection experiences transient congestion while the other does not followed by a reversal (i.e. the side that was congested becomes uncongested and vice-versa). Studying such effects is part of our future work.

3.3 Practical considerations

We envision that Slot intermediaries will be deployed using an **infrastructure overlay network** set up by a service provider (e.g., ISP, CDN) or using commercial grade platforms in public

[2]S2 is initiated after S executes `accept()` on segment S1 after the TCP 3-way handshake is complete which takes 60ms.

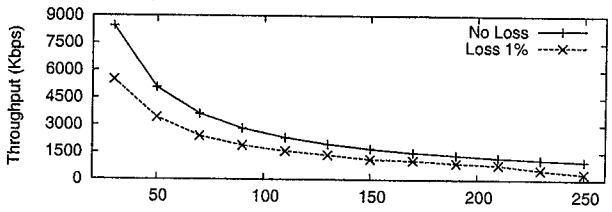

Figure 2: Validation of loop shortening using dummynet.

PlanetLab, set up by PSPs [32]. This model also allows planned geographic placement of intermediaries to improve the opportunities for loop shortening. Such an infrastructure overlay network differs from general p2p overlays in that nodes have low churn, adequate bandwidth, and low loss rates to their peers. In this paper, we use PlanetLab [34], in which most nodes satisfy the above properties, as an example of such infrastructure overlay networks. For example, our loss rate measurement by sending one 41-byte packet every 100ms for 60secs (as in [18]) between each node pair in a set of 52 .edu PlanetLab nodes shows that the average loss rate is 0.56%. Further, 85% of the node pairs have 0% loss rate. Other work [10] also showed that many well connected nodes in the RON testbed had 0% loss rates while the overall loss rates was 0.42%. In such a deployment, when Slot breaks a direct path into a multi-hop path effectively reducing the length of the feedback loop, the loss rates on each overlay hop are expected to be no worse than that of the direct end-to-end path, and hence the length of the feedback loop (RTT) can be used as a good heuristic to obtain multi-hop paths in Slot.

3.4 Validation

We first validated the gain from loop shortening for a wide range of RTTs in the above mentioned controlled setting. The *ttcp* [6] tool is used to transfer a large file (to measure steady state throughput) between A and B while dummynet is used to vary the effective RTT between A and B from 30 msec to 250 msec by varying the RTT from S to A and B. The results of the experiment are depicted in Figure 2. The 'No Loss' curve provides an upper bound on the throughput for a given RTT. Further, it shows the inversely proportional relationship between TCP throughput and RTT. Figure 2 also shows the impact of loss rate. A packet loss rate of 1% results in a consistently lower TCP throughput. However, the inverse relationship between TCP throughput and RTT still holds.

To validate the gains from Slot in a wide area network, we measured the end-to-end throughput

improvement from statically configured 2-hop Slot paths using PlanetLab nodes. Specifically, we measured the TCP throughput while transferring a 4MB file from Purdue U. to three nodes at UCB, Duke U. and HP Labs, Bristol. Each of these connections was broken into 2 hops using an intermediary at U. Colorado, U. Rochester, and MIT, resulting in a throughput increase of 40%, 21%, and 90% respectively compared to the direct path. These improvements were proportional to the corresponding reduction of their path RTTs. The measurements validate that the loop shortening technique in Slot can provide throughput gains in real application scenarios and that RTT is a good heuristic.

In summary, the technique of shortening the feedback loop using Slot is effective in providing TCP throughput gains. The above validation, however, was performed on a few specific TCP connections. In the next section, we study the potential effectiveness of Slot for a large number of node pairs in the Internet.

4 Slot: Potential

In this section, we assess the opportunities to shorten transport loops through Slot on a large scale by analyzing wide area RTT measurements between 3.7 million node pairs.

4.1 Methodology

The example overlay network for our measurement study is PlanetLab which we use to model clients and intermediaries. Consequently, our results reflect its specific topology. However, it does give us insight into the potential benefits possible using Slot.

PlanetLab currently consists of 616 nodes at 290 different geographically distributed sites. For our measurement study, we finally used 404 nodes from 168 sites after removing nodes that were either unresponsive, being debugged, or resulted in login failures nodes as well as nodes that did not route to commercial networks (e.g., Internet2, CAnet nodes). A single client node is considered from each site (for a total of 168 clients) since multiple clients from the same site do not add new information to the results.

The RTT probes consist of TCP ACK/RST packets[3] to high numbered ports. The strong

[3]We used TCP probes since they better reflect the RTT experienced by TCP packets flowing through routers and are not filtered like ICMP probes.

Domain	.com	.edu/.net/.org	.country (.uk, .cn etc)	Other
Servers	1,226	18/187/14	207	27

Table 1: Characteristics of 1,679 web servers.

correlation between RTT and TCP throughput as observed in Section 3 allows us to focus on RTT measurements. Each measurement is an average of 5 probes and was performed by each node 3 times a day from which the minimum recorded for each day were used in order to remove effects of congestion related events, similarly as in [12]. These traces were collected for a week once a month beginning February 2005 to observe the long term trends. Here, we present the results for the latest set of traces collected from September 7-14, 2005. The results for older traces provide largely similar conclusions despite the continued growth and diversity in PlanetLab nodes.

We collected measurement traces modeling two dominant data transfer scenarios: (1) A *peer-to-peer scenario*, in which the 168 client nodes serve as both clients and servers. In this *p2p RTT trace*, 168 client nodes measure the RTT to each other directly as well as through each of the 404 overlay nodes. Additionally, the remaining all-pair RTT measurements between all 404 overlay nodes are collected. (2) A *web scenario*, with 168 client nodes and 1,679 popular web servers from `www.ranking.com` whose characteristics are listed in Table 1. The original list of 2,000 web servers (taken from `www.ranking.com` was filtered to 1,679 web servers which were responsive. The web servers are from all over the world and their exact geographical distribution is not known.

In this *web RTT trace*, 168 client nodes measure the direct RTT to 1,679 web servers as well as through all 404 overlay nodes. Since each client node independently resolves URLs, the list of 1,679 URLs resulted in 8,890 unique IP addresses which are probed from each of the 404 overlay nodes. In addition, all-pair RTT measurements between all 404 overlay nodes are collected. In total, the two traces consist of RTT measurements between 3.7 million node pairs.

To assess the potential throughput improvement from loop shortening, the trace data is then analyzed offline to identify the number of cases in which the direct-path RTT between a pair of nodes is reduced by shortening the end-to-end loop and also the amount of reduction in path RTT in each such case. We modified the Floyd-Warshall algorithm 3 to generate the shortest client-server paths using all the 404 overlay nodes as possible intermediaries. In the algorithm, a path P is defined to be shorter than Q if the path RTT for P is lower than that of Q.

```
FLOYD-WARSHALL-Slot()
(1) for (int k = 0; k < N ; k++)
(2)   for (int i = 0; i < N ; i++)
(3)       for (int j = 0; j < N ; j++)
(4)           if max(D_{i,k},D_{k,j}) < D_{i,j} and max(D_{i,k},D_{k,j}) < INF
(5)               D_{i,j} = max(D_{i,k}, D_{k,j})
```

Figure 3: Modified Floyd-Warshall algorithm to find minimal path RTT multi-hop paths. N is the number of nodes in the trace data, and D is the RTT matrix from the trace.

Trace characteristics	P2P	Web
Client nodes	168	168
Overlay nodes	404	404
Server nodes	168	1,679
Total number of client-server pairs	28,224	282,072
Client-server pairs with DNS failure	252	3,429
Unreachable client-server pairs	1,003	9,923
Valid client-server pairs	26,969	268,720
Client-server pairs without benefit	343	7,865
Client-server pairs with benefit	26,626	260,855
Client-server pairs with >25% RTT reduction	25,003	243,320
Average reduction in *path RTT* (%)	56.12%	54.93%
Client-server pairs repaired	1,003	7,292

Table 2: Potential of Slot loop shortening using traces.

4.2 Results

The analysis of the potential of Slot loop shortening using the p2p and web RTT traces is depicted in Table 2. The results show that Slot can benefit 98.7% and 97.1% of the client-server pairs in the p2p and web RTT traces respectively; with 92.7% and 90.5% of the client-server pairs achieving a 25% or more path RTT reduction. Table 2 also demonstrates that the path RTT between a client-server pair is on average reduced by 56.1% and 54.9% for the p2p and web RTT trace, respectively. This reduction in path RTT can potentially result in almost doubling the throughput. The lower bound of 55-56% reduction can be attributed to the non-uniform geographic distribution of nodes. For example, flows going from the west coast of US to the mid-west cannot be easily shortened to obtain a reduction of over 50% due to limited availability of intermediaries in the central part of US. Slot also improves the reliability of TCP transfers. The data shows that 3.6% of the client-server pairs in the two traces did not have direct connectivity and Slot repaired 100% and 73.5% of such connections in the p2p and web RTT trace respectively. Compared to the p2p RTT trace, the web RTT trace is largely similar except that it provides

slightly lower average reduction in path RTT and has a larger number of unrecoverable failures.

Although trace analysis indicates that Slot can provide gain for a large fraction of client-server pairs, the actual benefit obtained from a running system depends on additional factors: (1) The size of the file transferred by a client-server pair affects the user-perceived benefit as well as the actual throughput gains. We study this effect in detail in Section 7.2. (2) The client access bandwidth can be the limiting factor. However, when this happens, neither Slot nor any other approaches to improving TCP throughput (discussed in Section 2) will be effective.

Summary In summary, the large fraction of client-server pairs in the p2p RTT trace (98.7%) and the web RTT trace (97.1%) that can benefit from loop shortening by leveraging intermediaries provided by overlay networks and the high average path RTT reduction attest that Slot has the potential to bring substantial throughput improvement to standard TCP.

5 Slot: Architecture

In this section, we provide an overview of Slot. Slot consists of two key entities (Figure 4): *client* nodes that use Slot to improve the throughput of their TCP transfers and *overlay* nodes (A-D) that are part of an infrastructure overlay network and act as intermediaries for shortening feedback loops. Slot does not require adding any functionality to the server nodes. Client nodes run the user-level Slot client module which transparently hijacks TCP connections on user-configured ports. The client module then discovers a multi-hop overlay path that minimizes the path RTT for each connection. Subsequently, the Slot client begins a connection establishment process over the chosen path which instantiates the "end-to-end connection" between the client and the server by initializing the overlay sub-connections after which data flow can occur. The user-level Slot intermediary module on the overlay nodes participates in the path selection as well as the connection establishment, termination, and data transfer processes. We envision two possible techniques for intermediary selection in the client module.

5.1 Application-specific Intermediary Selection

The application-specific technique toward intermediary selection in Slot works as follows. Each Slot client obtains the entire list of intermediaries available through a XMLRPC interface (similar

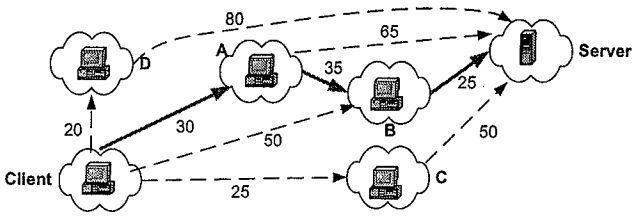

Figure 4: Slot overview. Numeric values on links denote the measured RTT in msec. The direct RTT between client and server is 70 msec. The solid arrows denote the final path chosen by Slot to between the client and the server.

to in PlanetLab) from the Slot website [5]. For each hijacked TCP connection, a Slot client module uses the following *path discovery* process: (1) probing the server to obtain the direct server RTT, (2) requesting the intermediaries to probe the server to obtain *path RTTs* to the server via paths through the intermediaries. The alternate paths probed can range from a minimum of 2-hops (using 1 intermediary) to multiple hops. Paths with more than 2 hops can be obtained by having each intermediary probed by the client to repeat the above process recursively. The path discovery process is depicted in Figure 4 in which the client probes multiple paths toward the server. As seen in Figure 4, the client chooses the path *Client-A-B-Server* which has the minimum path RTT (35 msec) out of all possible paths. Note that the $\sum_{i=0}^{n} RTT_i$ of this chosen path (90 msec) is higher than the direct RTT to the server (70 msec). However, due to pipelining of data over the TCP sub-connections, this path is expected to have a higher TCP throughput which corresponds to its 35 msec path RTT.

5.2 Service-based Intermediary Selection

Another potential technique to select intermediaries is **service-based intermediary selection**. Leveraging an existing system service has its advantages: (1) it offloads the burden of maintaining and sampling the overlay network from the Slot client, and (2) it decouples intermediary selection from the functionality of the Slot client allowing independent optimizations at each layer. service-based intermediary selection in Slot works as follows. For each hijacked TCP connection, the Slot client module provides its own and the server's IP address to a system service as a query. For example, the recently proposed Meridian [46] network location service can be used to support such queries. The client module then receives a query reply containing the intermediaries along the multi-hop overlay path that minimizes the path RTT. Subsequently, the connection is instantiated over the discovered path.

6 Slot: Design

The primary means by which the Slot framework effectively harnesses the potential of loop shortening is *intelligent intermediary selection.* In the following, we embark on a detailed exploration of design issues involved in intelligently selecting such intermediaries. Our study answers two fundamental design questions: (1) How many intermediaries are required in an overlay path to obtain reasonable benefit? (2) How to find a close-to-optimum overlay path without incurring significant overhead or delay in probing intermediaries?

6.1 Methodology

To answer these questions we use two methods: (1) We use the p2p RTT trace and the web RTT trace (Section 4) to simulate how a particular strategy works in a wide area network. The large volume of the trace data allows us to obtain results for a large number of client-server pairs. (2) We evaluate our design choices using a deployment on 404 PlanetLab nodes in order to validate each strategy in a realistic setting and measure throughput gains via actual data downloads. In these experiments, we fix our client at Purdue and perform transfers from five servers that are chosen to have a wide coverage in direct server RTTs[4]. The primary metrics used to evaluate the "goodness" of a strategy are: (1) *Gain ratio* ($\phi_{i,j}$): defined as the ratio of path RTTs using an at most i-hop overlay path to one using an at most j-hop overlay path. For example, $\phi_{1,2}$ is the ratio of the direct RTT along that path to the path RTT of a 2-hop path with 1 intermediary. Note that the minimum value of gain ratios is one. (2) *Throughput*: defined as the ratio of total bytes delivered to the total time taken *including* the delay of selecting the intermediaries.

6.2 How Many Intermediaries?

We first address the question of how many intermediaries are required to shorten the feedback loop of a TCP connection in order to obtain reasonable benefit with reasonable overhead. On one hand, the more intermediaries are used to shorten a connection, the lower the path RTT is likely to be. On the other hand, using more intermediaries to shorten the feedback loop of a TCP connection increases the complexity, overhead, and delay of intermediary selection. To answer the

[4]The servers are located at Berkeley and Duke in US, University of Victoria in Canada, HP labs in UK, and Monash University in Australia.

```
MATRIX-MULTIPLY-Slot(I)
(0) MR = D
(1) for (int n = 0; n < I ; n++) {
(2)  for (int i = 0; i < N ; i++) {
(3)      for (int j = 0; j < N ; j++) {
(4)          Set L_{i,j} to ALN
(5)          for (int k = 0; k < N ; k++) {
(6)              L_{i,j} = min(L_{i,j}, max(MR_{i,k},D_{k,j})
(7)      }
(8)  }
(9)  MR = L
(10)}
```

Figure 5: Modified Matrix-Multiple algorithm to find minimal path RTT multi-hop TCP paths. The algorithm takes parameter I which the the maximum number of intermediaries allowed in a path. N is the number of nodes in the trace data, and D is the RTT matrix from the trace. The result is contained in matrix L. ALN is a A Large Number to initialize the matrix.

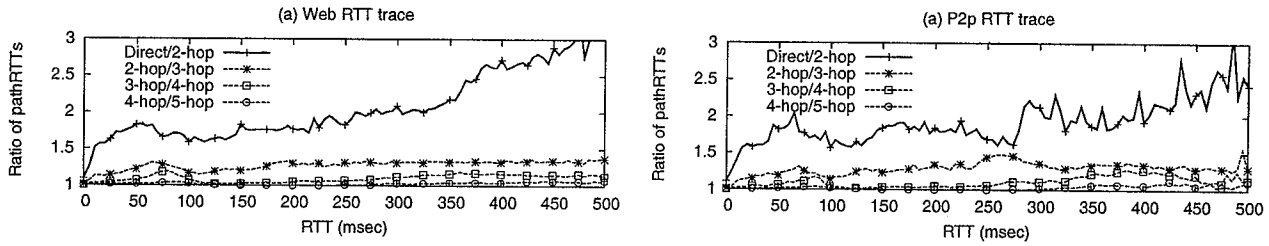

Figure 6: Ratio of path RTTs achievable as path hop count is increased.

above question, we first separate the overhead from the potential benefit and study the potential benefit (increase in gain ratio) as a function of the maximum number of intermediaries in a path.

We analyzed both the p2p and the web RTT traces using a modified Matrix-Multiply algorithm 5 that computes client-server shortest paths using the path RTT metric. A path is "shorter" than any another path if its path RTT is lower. This algorithm provides the gain ratios for each client-server connection for increasing number of intermediaries in a path, i.e. increasing path hop count. For each successive path hop count considered, $\phi_{n,n+1}$ is calculated to measure the additional gain that results when we use (n+1)-hop paths instead of n-hop paths.

The gain ratios are shown in Figure 6 for both RTT traces sorted by the corresponding direct RTT (on the x axis). The following key observations can be made. First, $\phi_{1,2} > 1$ (gain from using a 2-hop path versus a direct path) exists across all values of the direct RTT. For example, even a short direct path between nodes at UC Davis and UC Santa Cruz with a RTT of 5 ms can be shortened using a node at Berkeley to obtain a path RTT of 3.80 ms. Second, we observe that $\phi_{1,2}$ can be up to 3 in both traces, while $\phi_{2,3}$ is at most 1.4 in the p2p RTT trace and at most

1.25 in the web RTT trace. Thus, in both p2p and web scenarios, the gain obtained from using up to 3-hop paths is incremental compared to simply selecting a single intermediary between the client and the server, and this holds across all possible values of direct RTTs.

Third, the additional gain from increasing the path hop count further ($\phi_{3,4}, \phi_{4,5}$) is negligible. This is also valid for connections with very large direct RTTs. This is surprising since connections that have a large direct RTT should potentially benefit more from longer and longer hop counts. However, connections with very large direct RTTs are usually trans-continental and their path RTTs are dominated by long RTT segments across oceans which cannot be broken further. For example, the RTT between Purdue and Monash is 214 ms. This connection is at best shortened using 1 intermediary on the US west coast resulting in a path RTT of 160ms. Thus, the reduction in path RTT is limited by the topology of the overlay network which does not provide an intermediary between the US and Australia in the Pacific ocean. However, future addition of overlay nodes (e.g. in Hawaii) could potentially provide a lower path RTT.

In summary, given the topology of our overlay network, i.e. PlanetLab, most of the benefit can be obtained by shortening the TCP connections using 1 or 2 intermediaries, i.e., 2 or 3 hops. Beyond this, searching for further intermediaries provides diminishing returns. In other words, there is no need to find paths longer than 3 hops. Taking the delay and the potential overhead required to obtain a 3-hop path (as discussed in the next section) into consideration, we conclude that 2-hop paths provide a cost-effective means of shortening feedback loops for reduced path RTT.

6.3 How to Select Intermediaries?

Having demonstrated that one intermediary is enough, the next important question is how to select the one intermediary in Slot. We explore two different design options to choose the intermediary: (1) application-specific solutions integrated with Slot, and (2) a service-based solution.

6.3.1 Application-specific Intermediary Selection

Application-specific intermediary selection in Slot works as follows: (1) probe the server, (2) request the intermediaries to probe the server, and (3) collect the measurements at the client and finally choose the best intermediary to route through to the server. Two key design issues that

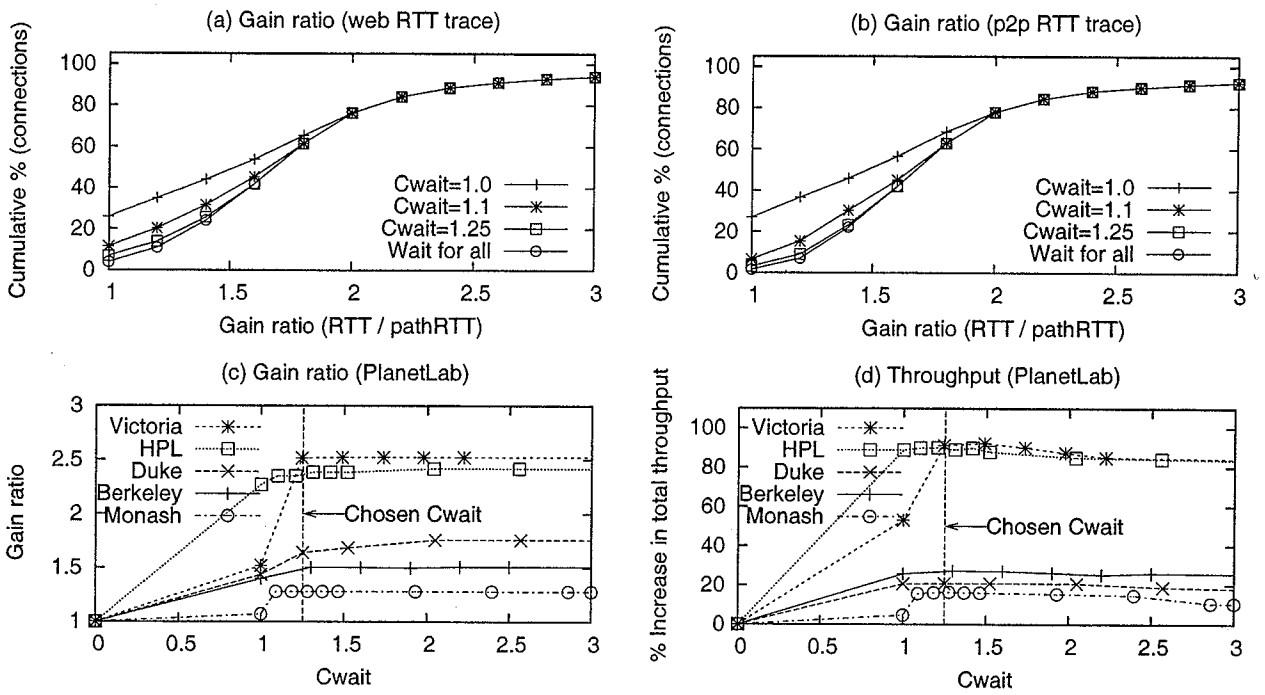

Figure 7: Impact of wait time on quality of selected paths using web and p2p RTT trace and PlanetLab deployment.

arise in this process are determining the set of intermediaries to be probed and the amount of time a Slot client should wait before making its selection. We first discuss the amount of wait time since it is independent of the specific strategy used to pick a set of intermediaries.

How long should a client wait to discover good paths? Tuning the wait time is an important design step as the amount of time the Slot client takes to discover paths affects the potential throughput gain. Waiting a long time to obtain measurements from the intermediaries before choosing paths reduces the effective throughput of the transfer compared to directly downloading the file in which case the wait time is zero. On the other hand, too small a wait time will force the client to choose an intermediary without waiting for many probings to complete, and there is a possibility that a suboptimal intermediary is chosen.

Since the sum of RTTs in any chosen 2-hop overlay path is typically larger than the direct RTT to the server, the minimum wait time for a Slot client is the direct RTT to the server, RTT_{server}. We now evaluate the additional time the client has to wait beyond RTT_{server} to obtain good overlay paths. Since the total wait time, W, is server-specific as probing a faraway server involves a higher absolute wait time, we set W at the Slot client to be $C_{wait} * RTT_{server}$, where $C_{wait} > 1$.

To evaluate a good choice for C_{wait}, we simulated 2-hop overlay path discovery using the web RTT trace while varying C_{wait}. The CDF of $\phi_{1,2}$ for varying C_{wait} is depicted in Figures 7(a)(b). The results indicate that a wait time of $1.25 \cdot RTT_{server}$ is sufficient for a client to obtain benefit similar to that obtained from waiting to hear back from all the intermediaries. Figures 7(a)(b) also show that approximately 20% of client-server pairs have gain ratios larger than 2, indicating that the chosen 2-hop overlay paths have a shorter accumulative RTT than the direct RTT due to violation of the triangle-inequality in Internet routing.

To validate this choice for C_{wait}, we used a client at Purdue to transfer a 4MB file from the five servers using 404 PlanetLab nodes as candidate intermediaries. We varied C_{wait}, measured the quality of the paths returned, and performed a download along the path with the highest $\phi_{1,2}$ obtained during the corresponding wait time. The results, shown in Figure 7(c)(d), confirm that for all servers considered, the highest gain ratio and throughput are obtained when the total wait time (including the server RTT) is less than $1.25 \cdot RTT_{server}$. Thus, we conclude that a wait constant of 1.25 is a suitable choice. In the rest of the paper, a wait time of $1.25 \cdot RTT_{server}$ is used.

How should a client sample intermediaries? A brute-force approach to finding the best intermediary is to sample the entire set of intermediaries, i.e. a *Select All* strategy. However, such a large number of probes (404 probes in our current PlanetLab deployment) sent to a server in a few milliseconds could potentially cause a denial of service attack. Further, as the number of intermediary candidates increases, this problem exacerbates. On the other hand, if only a subset of the intermediaries are sampled, a sub-optimal gain ratio may result. Thus, there is a tradeoff between the overhead of locating a good intermediary versus the gain ratio achieved.

Random-k Strategy A simple way to reduce the overhead of probing is to randomly select a fraction k of the 404 overlay nodes. We call this strategy *Random-k*. We evaluate the *Random-k* intermediary selection strategy using the web and p2p RTT trace by varying k from 4 to 150. The gain obtained for each client-server pair is averaged over 25 runs. The CDF of the gain ratio is depicted in Figure 8.

We make the following observations: First, the gain ratios obtained with a small value of k=4 are significantly lower than *Select All*. This demonstrates that although k=4 is sufficient when

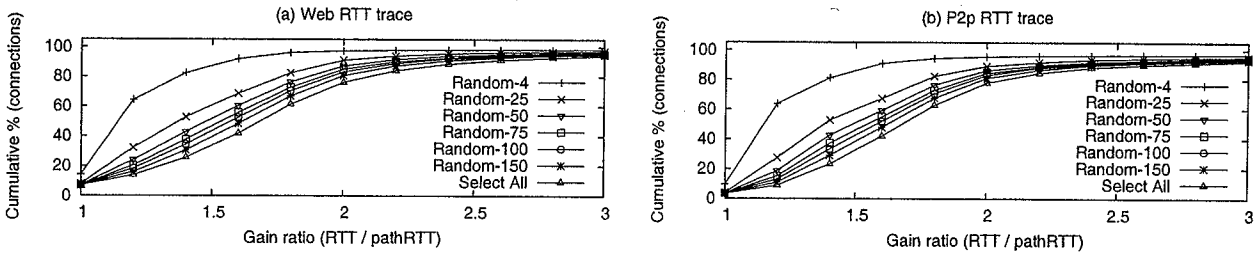

Figure 8: CDF of gain ratio using the *Random-k* strategy.

repairing Internet path failures [17], it is inadequate for Slot since the number of nodes that shorten the feedback loop for a given connection are much smaller than the number of nodes that can potentially repair a failure. Second, as k is increased from 4 to 150, the gain obtained becomes successively closer to *Select All.* However, beyond k=50 the incremental gain is smaller with increasing k. Thus we conclude that k=50 provides a good tradeoff between overhead and gain ratio. Similar results were also observed from a PlanetLab deployment when a client at Purdue was used to transfer a 4 MB file from 5 different servers while varying k. In summary, a simple way for Slot to select an intermediary is to randomly sample a set of 50 intermediaries.

Although simple and effective, the *random-k* strategy has a potential drawback due to the possibility of skew in the population distribution of intermediaries. For example, a large number of intermediaries can exist in close proximity (RTT) to each other and consequently exhibit similar performance. In this case, a fraction of the k samples could be wasted on sampling nodes that provide similar performance, thereby reducing the gain ratio. In order to alleviate this problem, we propose to *group* the 404 overlay nodes into clusters with low intra-group RTT and sample at maximum one representative node from each cluster. Subsequently a *RandGroup-k* strategy entails randomly sampling k clusters picking one representative from each chosen cluster. This provides natural load balance among the nodes in a group. Note that *RandGroup-k* could also potentially reduce the number of samples required by *Random-k* for a given accuracy if such skew already exists in PlanetLab node distribution.

RandGroup-k Strategy The *RandGroup-k* strategy groups nodes into clusters that satisfy a constraint on the intra-cluster RTT. To achieve this we use a Markov clustering scheme proposed in [43]. We provide a graph representation of the infrastructure overlay network with edge weights $w_{ij} = \frac{1}{RTT_{ij}}$. The clustering algorithm forms clusters of nodes that have high edge weights to each

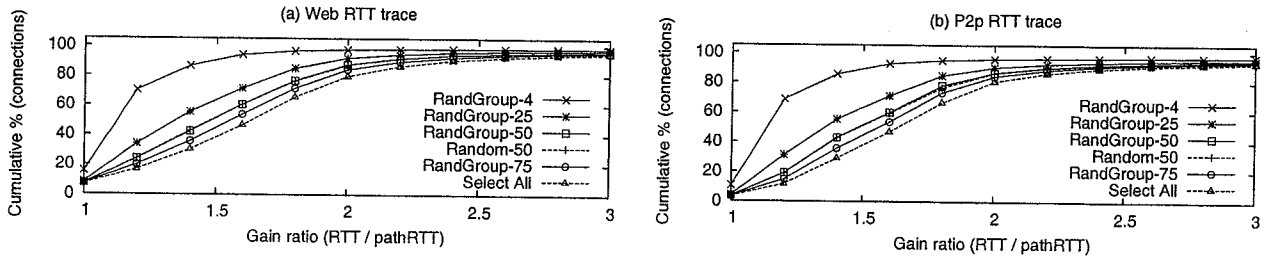

Figure 9: CDF of gain ratio using the RandGroup-k strategy.

other. We obtained a list of 116 groups from 404 overlay nodes using this method. All the nodes in most of the groups belong to the same PlanetLab 'site' while few groups in dense areas (e.g. Boston) consist of nodes from multiple PlanetLab sites.

We evaluate the *RandGroup-k* intermediary selection strategy using the web and p2p RTT trace by varying k from 4 to 50. The CDF of the gain ratio is depicted in Figure 9. The following observations can be made: First, similar to *Random-k*, the gain ratios increase with increasing k. Second, the performance of RandGroup-50 is identical to that of Random-50. This is because given the current PlanetLab topology, the nodes are almost uniformly distributed across all groups. Thus, there is absence of skew in the current intermediary population. The uniform distribution of nodes across groups can be attributed to the fact that a large fraction of PlanetLab sites have similar number of nodes (2 or 3). Only a few sites have more than 3 nodes and only a few sites are within close proximity to each other. However, we propose that Slot use the RandGroup-50 strategy to deal with potential problems with skew that may arise under different deployment topologies. We observed similar results from our PlanetLab deployment experiments.

The overhead in obtaining a 2-hop path is thus 50 pings to the server. Extending this strategy to find a 3-hop path will cause an overhead of 50^2 pings. Thus, given the incremental gain provided by a 3-hop path versus a 2-hop path (Section 6.2), the overhead incurred by 3-hop path selection is not justified.

6.3.2 Service-based Intermediary Selection

To demonstrate service-based intermediary selection, we used Meridian [46] as an example service. Meridian is a p2p framework that resolves queries for location-aware node selection without using virtual coordinates. It has been shown to be more accurate than methods that employ network embedding [30, 14] and does not require servers to be part of the Meridian overlay. Each Merid-

ian overlay node tracks a fixed set of peer nodes, organized into concentric rings of increasing radii (based on RTT). Slot uses the multi-constraint query resolution feature of Meridian which discovers a node within latency bounds of multiple targets. The targets specified in the query by Slot are the client and the server of each TCP connection and the latency bounds are specified as $\frac{1}{2} \cdot RTT_{server}$ for both targets. Note that this requires the client to measure the RTT to the server before issuing the query. This query attempts to discover nodes that are closest to being halfway between the client and the server, thereby attempting to find the best possible 2-hop path. As an optimization, Slot obtains the closest Meridian overlay node once via the "closest node" Meridian query and issues all future queries to this closest node. We refer to Slot using Meridian as *Slot-Meridian.*

6.3.3 Comparing Intermediary Selection Techniques

In this section, we compare the two approaches to intermediary selection in Slot based on the quality of the selected intermediary, the overhead in selecting the intermediary, and the wait time till the selection is resolved. For this experiment, we deployed Meridian[5] and Slot with the RandGroup-50 strategy on 404 PlanetLab nodes. We then used a client program on each PlanetLab node to contact the 1679 popular servers in the web RTT trace and 168 peer nodes in the p2p RTT trace, each time selecting an intermediary for the server using: (1) the RandGroup-50 strategy, and (2) the Meridian service.

Figures 10(a)(b) depict the gain ratios obtained from the chosen intermediary using RandGroup-50 and Meridian. Both RandGroup-50 and Meridian provide gain ratios close to that obtained using *Select All.* However, the accuracy of Meridian is slightly better compared to RandGroup-50. This is expected as the query forwarding in Meridian "zooms in" towards the solution space, at each step handing off the query to a node that has more information to resolve the query.

To compare the overhead of intermediary selection, we measured the average number of pings received by the server and the client. These measurements indicate that Meridian requires on an average 40 pings to the client and 40 pings to the server (pings from all the peers used to resolve a query included) to resolve the single intermediary. In contrast, RandGroup-50 results in 50 pings

[5]The publicly available implementation provided by the authors at http://www.cs.cornell.edu/People/egs/Meridian/ was used.

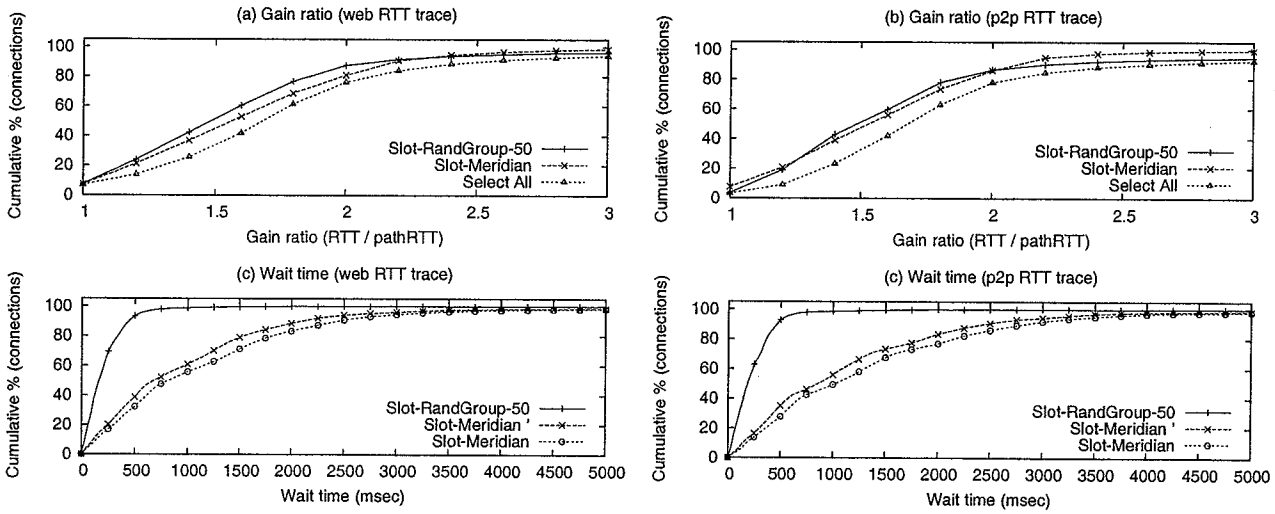

Figure 10: Comparison of Slot-Meridian, Slot-Meridian′ and RandGroup-50 for intermediary selection using real-time measurements.

only at the server. Note that both techniques require much lower pings when measurements are cached at the overlay nodes.

Finally, Figures 10(c)(d) depict the CDF of wait times (time taken to discover the intermediary) experienced across all connections for the two strategies. Meridian has a significantly larger wait time as compared to Slot using RandGroup-50. This can be explained as follows. To resolve a multi-constraint query, the Meridian node that receives the query first measures its distance to the specified targets, and then possibly forwards the query to a set of its own peers based on their proximity to the solution. This process is subsequently repeated by the peers resulting in inter-Meridian-node communication. When a solution is obtained, the query result containing the best intermediary is forwarded back to the Slot client. In contrast, in Slot-RandGroup-50, the client queries the server and the 50 overlays simultaneously and waits for only $1.25 \cdot RTT_{server}$ to choose the intermediary.

To improve the wait time when using Meridian for Slot, we modified Meridian to export a new Slot-specific query API. This new query specifies only the two target nodes without any latency bounds and is issued right away to the Meridian node closest to the client. The Meridian node then measures the direct RTT to the server. Since it is closest to the client, it can use this RTT as an approximation to the RTT from the client to the server. Thus, in one probe, it discovers its latency to the server (its latency to the client is pre-stored) and the two constraints (i.e., half of the RTT) to the two targets. It then proceeds with the multi-constraint resolution scheme as before.

In this way, one server RTT is saved in the total wait time. We refer to this improved version as *Slot-Meridian'* (Slot-Meridian prime). As expected, the wait time incurred by Meridian' is lower than that of Meridian as seen in Figures 10(c)(d). However, the wait time using the RandGroup-50 strategy is still substantially lower than Meridian'. Note that the accuracy and overhead of Meridian' is identical to that of Meridian.

In summary, the usefulness of either intermediary selection technique depends on the application scenario. We propose that service-based selection be used when greater accuracy is desired and the queries are issued less frequently. For example, if Slot is used to shorten the branches of a overlay multicast tree, the wait time will be a one-time cost as intermediaries will be chosen once during tree construction and a greater gain ratio is beneficial since a large amount of data will potentially be delivered along the tree. On the other hand, application-specific selection is more applicable when a small loss of gain ratio can be traded with a much lower wait time. For example, when Slot is used for HTTP transfers, many small requests can occur and low wait time is important.

6.4 Summary of Design Choices

Our studies have shown that (1) a single intermediary exploits most of the throughput gain possible from Slot, and (2) the choice of a specific intermediary selection technique (RandGroup-50 or Slot-Meridian') is scenario-specific. In RandGroup-50, the Slot client module randomly samples 50 groups (one intermediary from each) and waits for $1.25 \cdot RTT_{server}$ to pick the best intermediary. In Slot-Meridian', the Slot client module issues a multi-constraint query to its closest Meridian overlay node and waits for a query response containing the best intermediary. In either case, if no intermediary is found, the Slot client connects directly to the server.

7 Slot: Implementation

In this section, we describe our prototype implementation of Slot. We have implemented Slot in two flavors: a Slot package that is transparent to applications and `libSlot` that provides wrappers for the socket API. In the latter case, applications that use Slot need to modify their socket calls to use the new API. We first focus on the description of the main transparent Slot package.

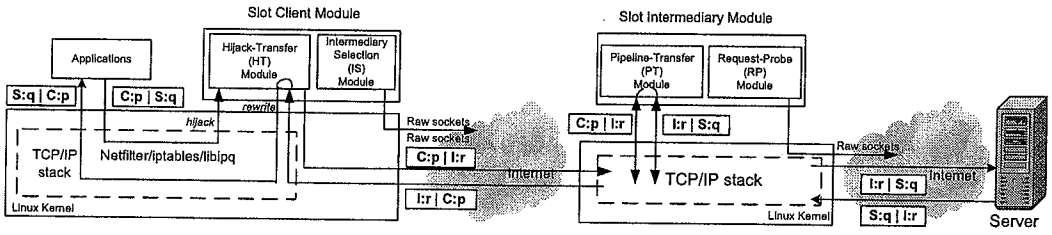

Figure 11: Slot implementation. Application packets are transparently redirected to intermediaries for loop shortening. Client applications, servers are unmodified. Packets are shown with format Source IP:Source Port | Server IP:Server Port.

7.1 Implementation Details

Slot consists of two main components: an application-transparent client module and an intermediary module. No Slot functionality is required on the server. The overall architecture is depicted in Figure 11.

7.1.1 Client module

Figure 11 shows the components of the Slot client module which is implemented with 4588 lines of C code as a user level daemon in Linux. The Slot client handles a TCP connection as follows: (1) The hijack-and-transfer (**HT**) module hijacks all packets from specific applications (e.g., Gnutella, HTTP, ssh) based on their destination ports (e.g., 6346, 80 , 22). This is enabled by specifying the ports to be hijacked via *iptables*, capturing the packets using the standard Linux Netfilter [4], and queuing them up for Slot processing using *libipq*. As an example, assume that the HT module is configured to hijack all packets with destination port q. (2) When the HT module detects the beginning of a transport session from an application (e.g. a SYN packet in TCP), it adds an entry in a *connection table* associating the source port p of the transfer with the destination server S and port number q. The connection table is a hash table with the source port as a key. The SYN packet is then dropped from further kernel processing and buffered in the HT module. (3) At this point the HT module invokes the intermediary selection (**IS**) module. The IS module obtains the intermediary I that provides the best path RTT for S using either the RandGroup-k or the Slot-Meridian′ strategy and returns I to the HT module. For example, in RandGroup-k, the IS module sends a PROBE REQUEST to the sampled intermediaries and waits for replies for $C_{wait} \cdot RTT_{server}$ to select I. RTT_{server} is also measured simultaneously. Note that for RandGroup-k, the IS module obtains a list of groups and their nodes at bootstrap using

the XMLRPC interface from the Slot website. This list is refreshed every 2 hours. (4) The HT module associates I with the source port p in the connection table. It then sends a TCP PIPE ESTABLISH to I on a known control port with a payload containing p, S and q so that the intermediary can set up state required to pipeline this connection as well as connect to the server. (5) The HT module sends out the buffered SYN using raw sockets to I on a known data port r on which the intermediary accepts data connections. (6) The HT module also inserts a filter rule to capture all incoming packets with source port r. It then looks up the connection table using the destination port of each incoming packet. For instance, the incoming packets from I will be looked up using p. The source IP and source port of all such incoming packets are then replaced with the corresponding S and q from the table entry and delivered to the application using Netfilter. For instance, the SYN-ACK received from $I : r$ in response to the SYN sent in step 5 is delivered to the application as if it was sent from $S : q$. Note that to preserve the end-to-end semantics of `connect`, the SYN-ACK is buffered and only delivered to the application if a PIPE OK message is received back from I indicating that it could successfully connect to the server. On the other hand, if a PIPE FAIL message is received, the SYN-ACK is dropped to indicate connection failure to the application and the buffered SYN is retried over the default direct path. (7) If the connection succeeds, the destination address and destination port of every outgoing packet that is part of this transfer (uniquely identified by port p) are modified to I and r respectively and sent using raw sockets. (8) Similarly, for every new transfer a new entry is added to the connection table and a potentially different intermediary chosen.

Processing in the client module is table lookup, packet rewriting and computation of new IP, and transport header checksums for each packet in the forward and reverse direction.

Note that the Slot prototype can be easily extended to operate when the client is behind a NAT. Apart from the stand-alone client scenario, the Slot client module can be also deployed as part of proxies that are used to provide enhanced reliability (e.g. MONET [9]) and web caching or as part of CDN proxy nodes (e.g., CoDeeN [44], CobWeb [2], Coral [3]). The Slot client module on these proxies can then improve the throughput of each of their connections to peer proxies or origin servers. An additional benefit of such an architecture is that the network measurements and intermediary selection results can potentially be cached and reused across multiple user requests reducing network traffic.

7.1.2 Intermediary module

Figure 11 also depicts the components of the Slot intermediary module implemented with 1480 lines of C code as a user-level application. The Slot intermediary performs the following functions: (1) To enable intermediary selection, the overlay node has to be either part of a Meridian deployment or run its own request-and-probe module (**RP**) to facilitate the RandGroup-k strategy. The RP module listens for PROBE REQUEST on a separate port. It timestamps the arrival of this request, measures the latency to the server address specified in the request, calculates the time elapsed from the arrival of the request to its departure to the client, and inserts this in the packet along with the server latency. The client can determine the path RTT from the PROBE REPLY, avoiding measuring the latency to the intermediary separately. (2) The pipeline-and-transfer (**PT**) module listens for PIPE ESTABLISH messages on a known control port. In Figure 11, when the PT module receives a PIPE ESTABLISH message containing p, S and q from C, it sets up a mapping associating packets coming from $C : p$ with $S : q$ and initiates its own TCP connection to S (say on `fd2`). (3) If the connect to S succeeds, the PT module returns a PIPE OK to C to indicate that the end-to-end multi-hop path is connected, else a PIPE FAIL message is sent back to C. (4) When the PT module accepts a new connection on data port r which returns a new socket descriptor `fd1`, it locates the mapping corresponding to the packet's source address C and source port p and the associated socket descriptor `fd2` is mapped to `fd1`. (5) Now the PT module enables the pipeline by writing everything received from the client on `fd1` to the server on `fd2` and vice-versa. Failure or closure events that occur when reading/writing on any connection result in closing the other side of the connection using the *shutdown* system call. The intermediary module maintains the read/write status of both connection segments. If a write failure occurs, the write half of that connection is shut down followed by shutting down the read half of the other side of the connection immediately. If a read failure occurs, the read half of that connection is shut down followed by shutting down the write half of the other side of the connection *after outstanding bytes have been written to the other side.*

7.1.3 Caching

Slot incorporates caching at both the client and the intermediary. The Slot client module caches the results of intermediary selection per server IP address in an *IS cache* to reduce probing overhead. Cache hit occurs if an item is requested from the server IP address for whom the intermediary is cached. The TTL value of an IS cache entry is set to be the same as the TTL value of the DNS cache entry for that server IP address. Because of this, when the Slot client is deployed at a MONET/CoDeeN/web proxy, the IS cache hit rate can be approximated by that of a local DNS cache. The local DNS cache hit rates have been shown to be close to 90% [45]. Thus, caching reduces the client-side probing overhead significantly. The Slot intermediary module caches all RTT measurements per server IP address in a *RTT cache* to reduce probing overhead.

7.1.4 Failure handling

If the Slot client fails to find an intermediary that reduces the path RTT, the connection uses the default path to the server. The buffered SYN is sent out using raw sockets with no modifications and all subsequent packets for the connection are simply re-injected back into the kernel.

If the server is unreachable from the direct path, the Slot client chooses an intermediary, if any, that provides the lowest path RTT from among those that could connect to the server during the intermediary selection process. Note that we do not need to use the random-4 or retry-3 policy as in [17] since 25 intermediaries have already been probed for connectivity to the server.

7.1.5 Support for NAT hosts

Similar to other overlay routing systems such as [17], the architecture above fails when the client is behind a NAT. This is because the state at the intermediary is set up using the client port. However, the intermediary sees a client port assigned by the NAT and fails to set up the connection. To deal with this issue, we implemented a NAT mode in the Slot client. In this scenario, the Slot client select the *localhost* as the intermediary for all connections. We then developed a client-side intermediary module that is basically the original intermediary module with support for probing and choosing intermediaries. The Slot client behind the NAT simple chooses the client-side intermediary module as it's intermediary. This client-side intermediary module then selects

an intermediary in the Internet just like before. The difference is that the PIPE ESTABLISH packet from the client-side intermediary is inserted before the beginning of the data segments from the client application. The intermediary module can then associate the source port after NAT translation in its mapping tables and correctly route the packets back.

7.1.6 libSlot

Apart from the application-transparent Slot package, the libSlot library we have implemented provides wrappers for the socket API (Slot_connect, Slot_write, etc.) which can then be invoked by applications. This is useful for applications that do not connect to well known ports (i.e. the data transfer channel is FTP, ports specified in QUERY_HIT Gnutella messages). In this case packet capture rules cannot be inserted beforehand into Slot. The libSlot library is also useful for users that want to enhance specific applications.

7.2 Evaluation

7.2.1 Methodology

In this section, we evaluate the performance of Slot using the prototype implementation configured with RandGroup-50 by using it to download a set of files. We crawled the first 1000 popular servers and picked files from 4 different file size categories: $<$ 10KB, between 10KB and 100KB, between 100KB and 1MB, and $>$ 1 MB. Since we restrict the number of files picked per server to 2, our trace consists of 2000 files in each of the first 3 file size categories and 492 files $> 1MB$. The Slot intermediary module is deployed on 404 PlanetLab nodes. A client at Purdue enabled with Slot periodically downloaded the files in the file trace each day over a period of 1 week from October 1st to October 7th 2005 using unmodified wget. During the same week, another client at Purdue that did not have Slot installed was also set up to download the files from the file trace using wget. For each day we calculated the percentage throughput increase when using Slot compared to without using Slot. These throughput gains were then averaged out for each file over the 7 day period.

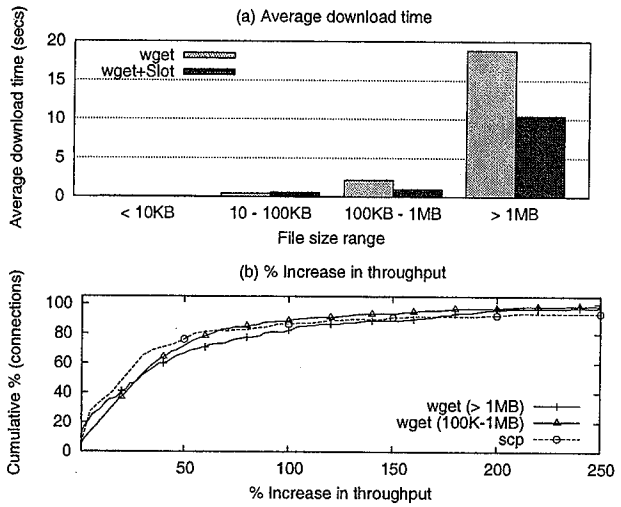

Figure 12: Download latency and CDF of % throughput increase for wget and scp running over the Slot prototype.

7.2.2 Results

During the entire evaluation period, we found that Slot provided better or at least similar throughput compared to the direct connections for all the files in the trace. Figure 12(a) shows the average download time for wget with and without Slot for each file size category. The results show that for file sizes greater than 100K, Slot approximately halves the average download time. For web requests less than 100K, Slot performs slightly worse than vanilla wget since, due to the small file sizes, the time spent on intermediary selection ($1.25 * RTT_{server}$) is a significant fraction of the overall download time. However, the results show that overall download latency for these files with and without Slot is negligible and is not likely to be perceived by a user.

Figure 12(b) shows the cumulative distribution of the percentage throughput increase achieved by Slot. Note that these increases are an average over a week long period and reflect the average performance benefit that Slot provided for each file. The results show that 95% of the files in the $> 1MB$ category and 98% of the files in the 100K-1MB category had throughput gain. Further, in both categories, more than 50% of the files were downloaded with a 30% or higher increase in throughput. As an example, while downloading a shareware program from a web server in Fort Worth, TX, the connection from Purdue was split using a node at Kansas resulting in 49% improvement in throughput. We noticed that for a certain fraction of files (around 20%), using Slot provided consistently low throughput gains of 1-5% or no throughput gains. Using *traceroute*,

we found that the web servers that hosted such files were located close to the client (e.g. in Chicago using the Level 3 ISP). For many such close-by servers, the Slot client at Purdue was unable to find intermediaries to further reduce the RTT from the already low value. In other cases, Slot reduced the RTT by a small amount by breaking the connection with another nearby node which results in small throughput gains. Thus, the low throughput gains observed for the small fraction of files are attributed to the servers themselves being very close to the client.

We also used unmodified scp with and without Slot to upload a 13MB gdb rpm to all 404 PlanetLab nodes. Figure 12(b) shows that Slot was able to provide significant throughput gains for scp uploads as well. 50% of the uploads had more than a 25% increase in throughput.

In summary, we evaluated our Slot prototype implementation on a PlanetLab deployment of over 404 nodes. We found that Slot is able to provide good throughput gains for around 80% of the files that were in our file trace as well as 80% of the scp uploads. For the remaining 20% of files/uploads, we found that the servers were close to the client and Slot was unable to provide high throughput gains.

During the entire evaluation period, we did not encounter any intermediary failure. Note that we do probe the intermediaries using the RandGroup-50 strategy before selecting one to route through. Thus, failures of intermediaries would only be noticeable when they occur during a transfer.

7.3 Application Case Study

7.3.1 Application compatibility

Slot has been running on PlanetLab since March 2005. We evaluated the general usability of Slot using various application software (e.g. different browsers) and found no unexpected application behavior. Applications that use well known ports (e.g. ssh/scp, web browsers) run unmodified on Slot. Applications (e.g., ftp, gnutella) that negotiate port numbers in application packets[6] require extensions to Slot. Slot cannot benefit applications in which the transfers are initiated from outside (e.g. active mode in ftp). In these cases, the server application is required to run Slot. Applications that use end-to-end security above the IP layer (e.g., ssh/scp, https, tls) also

[6]The Gnutella QUERY_HIT specifies the port to connect to the peer. Ftp also negotiates port numbers on the control channel.

work with Slot. Also, our measurement study showed that 80% of Slot paths incurred an average increase of 13.8% in end-to-end latency compared to their direct paths. While this is irrelevant for bulk transfer applications, real-time applications sensitive to overall latency may not benefit from Slot.

7.3.2 Impact on application semantics

Slot blurs the end-to-end nature of TCP since an ACK now signals data reception at an intermediary. However, we argue that Slot does not significantly impact *application* semantics. Already, current socket APIs to TCP do not expose the receipt of an ACK to the application. For example, in the `write` call, successful return of write only indicates copying of data into the kernel buffers. The application is equally unaware of how much data reached the server applications with or without Slot. Typically, well designed applications implement their own mechanisms to ensure that messages are received at the application on the other end. The application semantics for the `read` call remain unchanged. The semantics of `close` are preserved (with a slight delay) by replaying connection terminations that occur at one end of a transfer to the other end. Failures of the client or the server are appropriately replayed to the other end. Further, failures of the intermediary during a connection, although rare in an infrastructure overlay network, would appear as server (client) failure to the client (server) application. As explained in Section 7.1, Slot preserves the semantics of `connect` by ensuring its success *only* when the entire end-to-end connection over the multi-hop path is set up.

8 Discussions

8.1 End-to-end semantics

While we described how Slot does not significantly affect application semantics, it does alter **end-to-end semantics**. For example, connections using Slot cannot use end-to-end IP layer security mechanisms (IPSec) without modifying the intermediary to become a part of the security association. However, running Slot on a client does not preclude the simultaneous use of IP based end-to-end schemes. Slot is completely configurable by the user who can simply insert filter rules into Slot to use it for specific applications or ports only. Additionally, all security mechanisms

above the transport layer (e.g. TLS (RFC2246), SSL) can still be used with Slot to provide end-to-end security.

8.2 Fairness

Using Slot also has implications to TCP **fairness**. Although Slot improves the overall throughput of a flow, each overlay hop of Slot uses unmodified TCP. Thus, in contrast to parallel TCP, Slot's impact is more predictable since each sub-connection behaves like a normal TCP connection sharing resources fairly with other flows *on that sub-path.* Understanding these fairness properties of Slot is part of our future work.

8.3 Interactions with traffic engineering

Like all selfish overlay routing schemes, Slot has **impact on traffic engineering** for ISPs [24, 28, 35]. Large scale deployment of selfish systems [35] such as Slot that utilize overlay networks may result in fundamental changes in the nature of Internet traffic, as such systems make routing decisions a greedy choice optimized by the user rather than dictated by ISPs. However, the impact of Slot is different from previous overlay routing systems in two aspects: (1) Slot could potentially impact a larger number of flows than systems that only invoke overlay routing when failures occur [17, 9]. (2) The nature of selfishness in Slot differs from in systems such as RON that use overall latency as the optimization objective. Slot optimizes the sub-path latency (min of max) rather than the overall (sum) latency. For example, most of the gain comes from paths that have an overall higher latency, and only 20% of the paths that Slot uses actually improve end-to-end latency. While selfish latency optimizing overlay routing has been shown to increase the congestion on certain links [35], it remains interesting to see if similar findings apply to Slot.

8.4 Privacy and Security

We do not address privacy and security issues from the use of intermediaries in this paper. However, using intermediaries at the transport layer poses privacy and security threat since data bytes from a source can potentially be snooped on at these nodes. In addition, malicious compromised intermediaries can corrupt the pipelined data. Thus, mechanisms need to be developed so that a sender can shield its data bytes and a receiver can verify the integrity of the data received.

Experiences from security concerns in other open proxy overlay systems like CoDeeN [44] can be readily used for this purpose.

8.5 Deployment Issues

The wide-spread deployment of Slot will bring about the issue of resource management. As the number of clients using Slot increases, the load on the intermediaries will also increase, imposing constraints on the amount of bandwidth they can provide per client. Incorporating load information into the intermediary selection process is an important part of our future work. The solution space could range from an intermediary-side solution to one on the client-side.

9 Conclusion

This paper presents Slot, the first complete framework that leverages overlay networks to improve the throughput of feedback-based transport protocols by *shortening* their feedback control loop. Slot does not modify or tune transport protocol parameters; it relies only on network path properties to transparently shorten feedback loops to increase throughput. Thus, Slot is complementary to all the standard or improved feedback-based Internet transport protocols. Additionally, Slot is easy to use and readily deployable since it is transparent to the OS, client and server applications.

Our large scale measurement study of 3.7 million node pairs consisting of p2p transfers among overlay nodes as well as web transfers to 1,679 popular web servers shows that Slot has the potential to benefit data transfers between a large fraction of node pairs in the Internet. Efficient intermediary selection is an important building block to harness this potential. We proposed and evaluated two techniques that provide light-weight and scalable intermediary selection and can cater to a wide spectrum of application scenarios. The principle behind Slot is not tied to any specific protocol or application. Slot is designed to be transparent to the OS, client and server applications, as well as the specific transport protocol used. It supports a simple interface to enable/disable hijacking of specific applications and it works with NAT/firewalled hosts. Evaluation of a prototype implementation of Slot using 404 PlanetLab nodes shows that Slot is effective at providing throughput benefits for real applications in the Internet. Specifically,

compared to directly fetching the same files, Slot improves the throughput of more than 85% of the files transfers and 50% of the file transfers achieve more than 30% increase in throughput.

References

[1] Akamai. http://www.akamai.com/.

[2] CobWeb. http://www.cs.cornell.edu/people/egs/beehive/cobweb/.

[3] CoralCDN. http://www.coralcdn.org.

[4] Netfilter. http://www.netfilter.org.

[5] Slot. http://www.slot-net.org/.

[6] ttcp. ftp://ftp.sgi.com/sgi/src/ttcp.

[7] Y. Amir and C. Danilov. Reliable communication in overlay networks. In *Proc. of DSN*, 2003.

[8] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. of ACM SOSP*, 2001.

[9] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Rao. Improving Web availability for clients with MONET. In *Proc. of USENIX NSDI*, 2005.

[10] D. G. Andersen, A. C. Snoeren, and H. Balakrishnan. Best-path vs. multi-path overlay routing. In *Proc. of IMC*, 2003.

[11] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *Proc. of ICDCS*, 1995.

[12] S. Banerjee, T. G. Griffin, and M. Pias. The Interdomain Connectivity of PlanetLab Nodes. In *Proc. of PAM*, 2004.

[13] R. Cohen and S. Ramanathan. Using Proxies to Enhance TCP Performance over Hybrid Fiber Coaxial Networks. *IEEE/ACM Trans. Netw.*, 6(1), February 1998.

[14] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proceedings of ACM SIGCOMM*, 2004.

[15] T. Dunigan, M. Mathis, and B. Tierney. A TCP tuning daemon. In *Proc. of SC*, 2002.

[16] S. Floyd. HighSpeed TCP for Large Congestion Windows, 2003.

[17] K. P. Gummadi, H. Madhyastha, S. D. Gribble, H. M. Levy, and D. J. Wetherall. Improving the Reliability of Internet Paths with One-hop Source Routing. In *Proc. of USENIX OSDI*, 2004.

[18] Q. He, C. Dovrolis, and M. Ammar. On the predictability of large transfer tcp throughput. In *Proc. of ACM SIGCOMM*, 2005.

[19] T. Henderson and R. Katz. Transport Protocols for Internet-Compatible Satellite Networks. *IEEE JSAC*, 1999.

[20] H.-Y. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. In *Proc. of ACM MobiCom*, 2002.

[21] C. Jin, D. Wei, and S. Low. FAST TCP: Motivation, Architecture, Algorithms, Performance. In *Proc. of IEEE INFOCOM*, 2004.

[22] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proc. of ACM SIGCOMM*, 2002.

[23] T. Kelly. Scalable TCP: Improving performance in highspeed wide area networks. *Comput. Commun. Rev.*, 33(2):83–91, 2003.

[24] R. Keralapura, N. Taft, C.-N. Chuah, and G. Iannaccone. Can ISPs take the heat from overlay networks? In *Proc. of HotNets*, 2004.

[25] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP), 2005. Internet Draft draft-ietf-dccp-spec-11.txt.

[26] S. Kopparty, S. Krishnamurthy, M. Faloutsos, and S. Tripathi. Split-tcp for mobile ad hoc networks. In *Proc. of GLOBECOM*, 2002.

[27] G.-I. Kwon and J. W. Byers. ROMA: Reliable Overlay Multicast with Loosely Coupled TCP Connections. In *Proc. of IEEE INFOCOM*, 2004.

[28] Y. Liu, H. Zhang, W. Gong, and D. Towsley. On the interaction between overlay routing and traffic engineering. In *Proc. of IEEE INFOCOM*, 2005.

[29] A. Nakao, L. Wang, and L. Peterson. MSB: Media Streaming Booster. Technical report, Princeton University CS TR-666-02, December 2002.

[30] T. S. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proc. of IEEE INFOCOM*, 2002.

[31] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proc. of ACM SIGCOMM*, 1998.

[32] L. Peterson. Commercialization of PlanetLab: A Whitepaper. Technical Report PDN–05–027, PlanetLab Consortium, June 2005.

[33] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology Into the Internet. In *Proc. of ACM HotNets*, 2002.

[34] PlanetLab. http://www.planet-lab.org.

[35] L. Qiu, Y. R. Yang, Y. Zhang, and S. Shenker. On selfish routing in Internet-like environments. In *Proc. of ACM SIGCOMM*, 2003.

[36] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[37] P. Rodriguez, S. Sibal, and O. Spatscheck. TPOT: translucent proxying of TCP. In *Proc. of WCW*, 2000.

[38] 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*, 19:50–59, January 1999.

[39] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP buffer tuning. In *Proc. of ACM SIGCOMM*, 1998.

[40] H. Sivakumar, S. Bailey, and R. L. Grossman. Psockets: the case for application-level network striping for data intensive applications using high speed wide area networks. In *Proc. of SC*, 2000.

[41] O. Spatscheck, J. S. Hansen, J. H. Hartman, and L. L. Peterson. Optimizing TCP Forwarder Performance. *IEEE/ACM Trans. Netw.*, 8(2):146–157, Apr. 2000.

[42] M. Swany. Improving Throughput for Grid Applications with Network Logistics. In *Proc. of SC*, 2004.

[43] S. van Dongen. Graph clustering by flow simulation. PhD thesis, University of Utrecht, May 2000.

[44] L. Wang, K. Park, R. Pang, V. S. Pai, and L. Peterson. Reliability and security in the codeen content distribution network. In *Proc. of USENIX ATC*, 2004.

[45] C. E. Wills and H. Shang. The contribution of DNS lookup costs to Web object retrieval. Technical Report WPI-CS-TR-00-12, Worcester Polytechnic Institute, July 2000.

[46] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: a lightweight network location service without virtual coordinates. In *Proc. of ACM SIGCOMM*, 2005.

[47] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman. One more bit is enough. In *Proc. of ACM SIGCOMM*, 2005.

[48] F. Xie, J. L. Hammond, and D. L. Noneaker. Steady-state analysis of a split-connection scheme for internet access through a wireless terminal. *IEEE/ACM Trans. Netw.*, 12(3), 2004.

[49] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson, and R. Wang. A transport layer approach for improving end-to-end performance and robustness using redundant paths. In *Proc. of USENIX*, 2004.