A Multipath Background Network Architecture

Ravi Kokku[†]   Aniruddha Bohra[†]   Samrat Ganguly[†]   Arun Venkataramani[‡]

NEC Laboratories America Inc.[†], Princeton, NJ   University of Massachusetts[‡], Amherst, MA

Abstract—

Background transfers, or transfers that humans do not actively wait on, dominate the Internet today. In today's best-effort Internet, background transfers can interfere with foreground transfers causing long wait times that hurt human productivity. In this paper, we present the design, implementation, and analysis of a background network, Harp, that addresses this problem. Harp has three significant advantages over recent end-host based background transport protocols; Harp (*i*) uses multiple paths to exploit path diversity and load imbalance in the Internet to tailor network resource allocation to human needs, (*ii*) has provably better fairness and utilization compared to unipath end-host protocols, and (*iii*) can be deployed at either end-hosts or enterprise gateways, thereby aligning the incentive for deployment with the goals of network customers. Our evaluation using simulations and a deployed prototype suggests that Harp can improve foreground TCP transfer time by a factor of five and background transfer time by a factor of two using just two extra paths per connection.

I. INTRODUCTION

Background transfers, or transfers that humans do not *actively* wait on, dominate Internet traffic today. Examples include peer-to-peer file swarming [1, 2], content distribution [3, 4, 5], Web [6, 7] and media [8] prefetching, remote backup [9, 10], software updates [11] etc. A recent study [12] suggests that up to 90% of Internet traffic may be background. However, today's best-effort Internet does not distinguish between background and foreground, or interactive, transfers. As a result, background transfers interfere with foreground transfers and inconvenience humans, e.g., online purchases or important mails can get delayed when an enterprise network is bogged down by large software updates; the quality of video-conferencing can get hurt by concurrent file swarming or file system backup traffic.

Two approaches exist to address interference: (*i*) priority queueing at routers [13], and (*ii*) end-host based transport protocols [14, 15, 16] that provide lower-than-best-effort service without support from routers. Both approaches suffer from two fundamental problems.

- *Arbitrary delay to background transfers:* Prioritization using either approach can arbitrarily delay background transfers. Furthermore, end-host based protocols are imperfect due to feedback delay and suffer from a marked drop in utilization to achieve near-zero interference. So, users have to plan their daily activities around longer download times for such transfers. For example, even though users do not actively wait on a movie download, a download that takes two hours instead of one will leave users unhappy.
- *Deployability:* While priority queueing at routers appears impractical due to the lack of architectural consensus today [17], modifying the applications and/or operating systems at end-hosts (although easier than modifying routers) is still a significant barrier to widespread adoption, and is worsened by the reluctance of users to install software that makes them "good
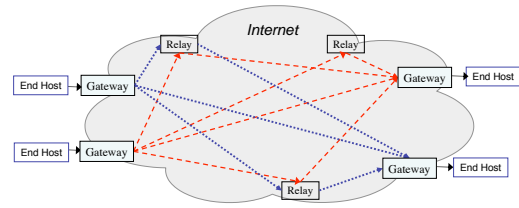

Fig. 1. Harp's basic architecture with gateways and in-network relays.

In this paper, we present the design, implementation, and analysis of a network architecture, Harp, for a background-dominated Internet, which addresses the above problems using two ideas.

- *Wait-time as a first-class metric:* Harp prioritizes foreground over background transfers to reduce active wait-time. More importantly, Harp uses multipath routing to dissipate background transfers to less-utilized regions of the network. Thus, Harp exploits the path diversity, load imbalance, and spare capacity in the Internet to ensure timely completion of both foreground and background transfers.

  Our algorithmic contribution is a background multipath congestion controller that jointly optimizes rate control as well as routing to improve utilization and fairness of network-wide spare resources. The algorithm is based upon recent theoretical works in multipath congestion control for foreground transport [18, 19, 20]. More importantly, our effort to build a practical system revealed a shortcoming of the fluid model assumed in these works, namely, the utilization is significantly below that predicted by theory when the number of flows is small. Our algorithm addresses this problem by smoothly transitioning between joint and independent control of rate and routing depending on the number of other extant flows on the path. Harp's controller is applicable to foreground transport as well with similar utilization benefits.
- *Deployment as an edge service:* Harp supports background transfers as an edge-service. Harp is implemented as an overlay network consisting of two types of transit nodes: *relays* and *gateways*. In the envisioned deployment scenario (Figure 1), an edge-service provider supports relays distributed across the Internet to enable path diversity, and organizations install gateways at their network exits—a popular edge-service business model [21, 22] today. Harp can also be deployed directly at end-hosts and use peer nodes as relays, obviating transit nodes. However, our position is that a practical architecture must align the incentive for deployment with the goals of network customers that are often physically disjoint from end-users as in the case of enterprises, public access points, overlay network services etc. Thus, a gateway is a sweet spot

verse end-hosts of unwilling users.

Our systems contribution is a prototype of Harp built using Click [23] and deployed on Planetlab [24]. We address non-trivial engineering challenges to enable background multipath congestion control as an edge-service without support from end-hosts or routers. Our extensive evaluation using this prototype as well as *ns* simulations demonstrate that Harp can significantly improve transfer completion times. For example, our simulations show that Harp improves foreground TCP transfer completion time by a factor of five. Our prototype experiments show that background completion time can be reduced by a factor of two using just two extra paths per connection.

The rest of the paper is organized as follows. Sections II and III present the design and implementation of Harp. Section IV presents the results of *ns* and prototype-based experiments. Section V presents related work and Section VI concludes.

II. SOLUTION OVERVIEW

Harp uses a multipath network for background transfers deployed as an edge-service and leaves foreground transfers unchanged. Foreground transfers are those that humans actively wait on, while during background transfers, humans typically engage themselves in other activities. We note that categorizing all transfers into two classes is a simplifying assumption, but is appropriate for a background-dominated Internet. Harp does not employ multipath routing for foreground transfers as they are typically too short-lived to get steady-state utilization or fairness benefits.

The design and implementation of Harp is guided by the following principles.

1. *Wait-time first*: Noninterference of background transfers must be treated as a safety property to reduce active wait-time. For example, a loss of a foreground SYN or data packet in slow start can severely hurt its completion time. Nevertheless, a short foreground transfer should minimally disrupt a background flow in steady-state. Harp's multipath background congestion control accomplishes both objectives.

2. *Evolvability*: An evolvable solution must not embed into routers mechanisms to optimize today's background-dominated Internet. Harp is implemented as an edge-service without support from routers.

3. *Incentive compatibility*: Unlike broadband users who might use Harp to reduce wait-time across their own transfers, users at public access points, enterprises, and overlay network services do not have an incentive to slow down their background transfers. Harp's gateway obviates any modification to or cooperation from end-hosts.

4. *Fairness*: An architecture must preserve fairness of resource allocation. Harp provides TCP-fairness using a multipath controller that uses several paths but provides an aggregate throughput equivalent to a TCP flow along the best path.

5. *End-to-end semantics*: An edge-service should be seamless to legacy applications and operating systems that are in charge of end-to-end reliability. Harp's transit nodes unreliably buffer, congestion control, and forward data packets. Acknowledgments and retransmissions are handled by the end-
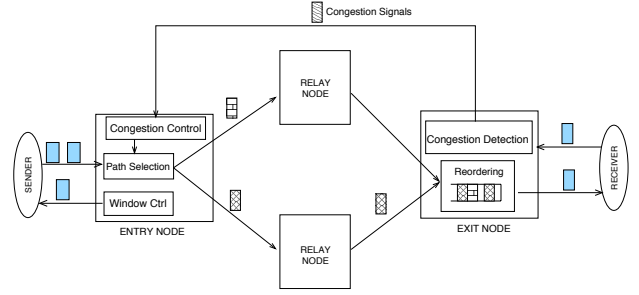

Fig. 2. Harp Architecture

Figure 2 illustrates the basic sequence of steps for packet delivery in Harp. Packets from the end-host enter the entry gateway and go to the exit gateway either directly or through one of the relay nodes. The exit gateway delivers packets to the receiving end-host. The acknowledgments generated by the receiver for all packets are sent back directly from the exit to the entry gateway. The multipath background congestion controller resides at the entry and exit gateways.

III. DESIGN

Harp uses a multipath controller to dissipate background traffic to less-utilized parts of the network, out of the way of foreground transfers. Harp's controller builds upon Kelly and Voice's (KV) multipath routing and congestion controller [19] and our previous work on TCP Nice [14], a unipath background congestion controller. Similar to KV, Harp shares congestion information across multiple paths in a connection to ensure a fair and stable allocation of network-wide resources. Similar to Nice, Harp uses delay-based signals to proactively detect congestion and aggressively backs off in response. The rest of this section presents the congestion control algorithm, and the challenges and design alternatives in realizing it without end-host support.

A. Multipath Congestion Control

The congestion control algorithm executes on the entry gateway. Given a set of paths on which packets can be sent, the algorithm estimates the size of the congestion window on each path that indicates a stable sending rate on the path. Our algorithm uses the following multiplicative increase multiplicative decrease (MIMD) scheme —(1) MI: on each positive acknowledgment on a path, increment the congestion window of the path by an MI parameter α, and (2) MD: on a congestion signal on a path, decrement the window of the path by an MD parameter β times the weighted sum of congestion window on the current path (w_i) and the total congestion window on all paths (W).

Algorithm 1 shows the pseudo code of the above congestion control behavior. Line 6 represents the MI part and Lines 9 and 12 represent the MD part. We use both increased packet delays and packet losses as signals of congestion. To prevent the algorithm from over-reacting to congestion indications and thus ensure stability, we let the congestion window be decremented only once per round-trip time on a path after observing a threshold number of congestion indications. Further, the window is not incremented within one round-trip time of the previ-

Algorithm 1 Multipath Congestion Control Algorithm for Background Transfers

1: **Definitions:** i: path (numbered from 1 to n)
2: w_i: congestion window (in bytes) on path i
3: W: total congestion window $= \sum_1^n w_i$
4:
5: **On ack for path** i
6: $w_i \leftarrow w_i + \alpha$
7:
8: **On delay signal for path** i
9: $w_i \leftarrow max(1,\ w_i - \beta \times (w_i \times \xi + W \times (1-\xi))$
10:
11: **On loss signal for path** i
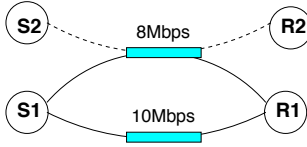12: $w_i \leftarrow max(1,\ w_i - W/2)$

Fig. 3. Example of two senders sharing one path

Setting the value of ξ to different values allows us to instantiate several variants of multipath congestion control.

- **Independent:** $\xi = 1$ makes the multiplicative decrease on a path proportional to the sending rate on that path. This is same as independent congestion control on each path, i.e., each path operates as an individual TCP flow [25]. Hence, a multipath connection using such independent control with n paths operates as n TCP flows.
- **Joint:** $\xi = 0$ makes the multiplicative decrease part same as that of joint routing and congestion control proposed by Han et al. [20] and Kelly et al. [19]. With such control, each multipath connection behaves as a single TCP flow on its best set of paths (one or more).

Following either of the above two approaches for background transfers have both advantages and disadvantages. When multiple background transfers share a set of paths, Independent can be unfair to transfers that have fewer paths at their disposal. For instance, consider Figure 3. The sender-receiver pair S1R1 has access to two paths, whereas S2R2 only has one path. When using independent, S1R1 gets an equal share of bandwidth on the path shared with S2R2 that leads to significantly higher overall bandwidth to S1R1 (14Mbps) than S2R2 (4Mbps). Using Joint instead, would move each transfer to its best set of paths, i.e. S1R1 occupies only the 10Mbps path and S2R2 completely occupies 8Mbps path, thereby leading to a fairer allocation of resources.

Using Joint when S1R1 is the only transfer, however, can lead to under-utilization of the capacity on a path. This is because even if S1R1 is the only transfer on a path, the transfer backs-off proportional to the *cumulative* congestion window on all paths thereby losing more throughput than Independent control. A more detailed explanation of the effect on throughput is in the technical report [26]. This behavior of Joint could be unnecessarily aggressive for background transfers whose goal is to reap as much spare capacity as possible.

variant that combines the behaviors by adapting the value of ξ.

- **Adaptive:** We set $\xi = \frac{w_i}{M_i}$, where w_i is the congestion window during decrement, and M_i is the maximum congestion window size observed by the transfer on path i.

Adaptive control has the following properties. When a multipath connection is the only one active on one or more of its paths, the multiplicative decrement on such paths behaves more like Independent because w_i is close to M_i. As the number of transfers sharing the path increases, the behavior of Adaptive becomes closer to Joint. However, for this technique to work well, each transfer should observe the maximum congestion window M_i at some time during its activity. Due to this requirement, Adaptive could take longer to achieve the same fairness property as Joint. We believe that this tradeoff is acceptable for long running transfers.

We formally summarize the above observations in the theorem below (and present a proof in the technical report [26]).

Theorem 1: *(a) Joint control of congestion and routing can degrade steady-state throughput by an $\Omega(\log(C))$ factor compared to independent control along a path of bottleneck capacity C, (b) Adaptive control achieves the same steady-state throughput as independent when there is exactly one flow along the path.*

B. Path Selection

On the arrival of a packet from the sender, the entry gateway attempts to balance load across the available paths by choosing for each packet a path with minimum $\frac{bytes\_in\_nw_i}{cwnd_i}$, where $bytes\_in\_nw_i$ represents the number of unacknowledged bytes sent on path i. The same expression is used to stripe packets even when $bytes\_in\_nw_i$ exceeds $cwnd_i$ to ensure load balancing on the paths. Once the entry gateway selects a path (i.e., the corresponding relay node) to send the packet through, it uses packet encapsulation to facilitate routing packets through the relay node. Additionally, each encapsulated packet carries a multipath header that contains a packet type (to represent data, probe, loss signal, etc.), a timestamp (representing when the packet left the entry gateway) and a path identifier (which is the IP address of the relay node). This header allows the exit gateway to identify the path taken by the packet, and detect and associate congestion to the path based on the delay observed by the packet.

C. Reordering

Since packets are sent on several paths with different latencies, they can arrive out of order and cause the receiver to send duplicate acknowledgments to the sender. Such duplicate acknowledgments falsely indicate packet loss to the sender that can lead to a substantial reduction of its congestion window thereby reducing its throughput.

To avoid such false indications, our mechanism reorders packets received from multiple paths at the exit gateway before sending them to the receiver. If packets are in sequence, they are sent immediately to the receiver. Otherwise, they are kept in a reorder queue till either the packets before them arrive (and thus complete the sequence) or a timer expires. Since the reorder delay required at the exit gateway is governed by the path with the longest delay, we set the timer to a value that is a factor ρ of the

D. Congestion Indication

To reduce interference, background transfers should *back-off* as early as possible when required. We achieve this behavior by reacting to increased packet delays, in addition to losses.

Detecting Increased Delays Harp uses relative increase in one-way packet delays as early indicators of congestion [15]. To achieve this, each packet is timestamped (in the multipath header) at the entry gateway before being sent on a path. The exit gateway calculates the one-way delay using the timestamp and its current time. Also, the exit gateway keeps track of minimum ($dmin_i$) and maximum ($dmax_i$) delays observed by packets on each path for a connection. If a packet observes a delay greater than $\Delta_i = dmin_i + (dmax_i - dmin_i) \times \delta$, where δ is a threshold parameter, the exit gateway sends a congestion indication to the entry gateway. Observe that in the above condition, we only use the relative difference in packet delays to infer congestion; hence the clocks on the entry and exit gateways need not be perfectly synchronized. Congestion can be indicated by either sending an explicit message to the entry gateway or by piggybacking the indication on a returning acknowledgment. We choose the latter in Harp for efficiency.

Detecting Losses Since the exit gateway maintains a reorder queue, it can detect packet losses earlier than the receiver. The exit gateway maintains a variable $last\_byte\_rcvd_i$ for each path i that indicates the highest byte received on the path. Additionally, it maintains a $rcvnxt$ variable per connection that indicates the next byte expected *in sequence*. When the $last\_byte\_rcvd_i$ on each path exceeds $rcvnxt$, the exit gateway detects a possible packet loss, and sends a loss indication to the entry gateway. The message also contains the range of missing bytes, which can be determined by $rcvnxt$ and the sequence number of the packet at the head of the reorder queue. Observe that the exit gateway can not exactly determine which path the loss occurred. Hence the range is used at the entry gateway to determine the path(s) on which the packets containing the missing bytes were sent. For each path on which any of the missing bytes were sent, the congestion window is reduced as described in Section III-A. This technique of detecting packet losses is simpler and quicker than waiting for and interpreting duplicate acknowledgments from the receiver.

As an optimization, we could cache unacknowledged packets at the entry gateway and respond to loss indications by retransmitting packets and thus avoid exposing the loss to the sender. To avoid exposing the loss, we will also have to interpret and supress duplicate acknowledgments from the receiver. Note that doing so would not violate our first guideline of maintaining end-to-end semantics described in Section II. However, we do not currently incorporate this optimization in our prototype.

Formally, we state the following theorem on interference caused by Harp, and present a proof in the technical report [26].

Theorem 2: The interference, or reduction in throughput of foreground flows, that Harp's multipath background controller inflicts is bounded by a factor $O(e^{(-\frac{B}{m} \cdot (1-\delta)(1-\beta))})$, where B is the bottleneck buffer capacity, δ the delay threshold, β the MD parameter, and m the number of foreground flows along the

E. Congested-path Suppression

Congested paths can increase the packet delays substantially and even cause losses thereby causing packets on other paths to wait in the reorder queue. Such a wait degrades the throughput of the connection because new packets are not sent by the sender unless the receiver acknowledges the old packets.

To reduce the impact of such congested paths on throughput, whenever the congestion window for a path reaches MIN_CWND, the path is temporarily marked as choked. No subsequent packets are sent on this path unless it is unchoked. From then on, a probe (unchoke request) with timestamp is sent on each choked path periodically. If the probe does not perceive delay greater than Δ_i as in Section III-D, the exit gateway returns an unchoke indication to the entry gateway. Otherwise, the exit gateway simply drops the probe. Observe that implementing choking and unchoking as above automatically handles path failures—(1) no data packets are sent on the failed path, and (2) if an unchoke request does not reach the exit gateway, no unchoke indication is sent back and hence the path remains choked.

F. Sender Rate Control

Harp is oblivious to the congestion control algorithm running at the sender. This can lead to a mismatch between the congestion windows on the sender and the entry gateway. The mismatch can lead to packet losses that reduce the congestion window on the sender, thereby reducing the throughput achieved by the background transfer.

One way to overcome the mismatch is to ensure that the sender does not send more bytes than what the entry gateway's congestion window can allow across all paths. To achieve this effect, the entry gateway rewrites the TCP header in the acknowledgments returning to the sender with a receiver window equal to the minimum of the window allowed by the receiver and the congestion window allowed by the entry gateway. To handle receiver window scaling used by most bulk transfer applications [27], we monitor SYN packets to check if end-hosts exchange the scaling option, and scale the receiver window that we rewrite in the acknowledgments accordingly.

G. Implementation

We have implemented a prototype of Harp using the Click router toolkit [23]. Click provides the necessary infrastructure to capture, modify, route, and transmit packets. Using Click's features, we build Harp such that it can be run both at the user- and the kernel-levels. Although for a given end-to-end connection our mechanism is distributed across different transit nodes, each transit node can simultaneously function as entry, relay or exit node for different end-to-end connections.

Transit node initialization In our implementation of Harp, an end-host (sender or receiver) using the mechanism associates itself to a transit node as its entry/exit node. This association can be done explicitly by the end-host, or seamlessly by an administrator. The information about each $< end\_host, associated\_transit\_node >$ pair is exchanged among transit nodes to enable successful routing of a sender's packets through

Param	Value	Remarks
α	0.03	MI parameter, chosen as a small value to ensure stability [19], but large enough to get good throughput.
β	0.125	MD parameter, same as DECbit scheme [29], scalableTCP [30] and VCP [31]
δ	0.15	Threshold on packet delays above which exit gateway indicates congestion. Value borrowed from TCP-LP study [15].
ρ	1.5	reorder delay factor that determines how long exit gateway delays packets.

Table I. Parameters used for simulations and prototype experiments.

a firewall table; the identity of the end-host can be a wild-card to associate a group of IP addresses and (optionally) TCP ports to a transit node.

Initial path selection For each entry and exit node pair, our prototype selects paths through relays that differ in RTTs from the direct path by at most a factor of 1.5.

Background transfer identification Our current prototype assumes that the administrator of the entry node specifies a configuration similar to a firewall table to identify background flows.

IV. EXPERIMENTAL EVALUATION

We validate the effectiveness of Harp using both ns simulations and evaluation of a prototype on the planetlab [24] testbed. Each of these environments allows us to demonstrate different aspects of Harp effectively. Further, simulations helped determine the right parameter settings for our real system prototype. Throughout this section, unless specified, we use the parameter settings as shown in Table I.

A. ns Simulations

In this section, we demonstrate different aspects of Harp using several topologies (shown in Figure 4) and a random topology generated by GT-ITM [28]. Topology1 is designed to (1) demonstrate that Harp can utilize spare capacity on multiple paths without causing much interference to foreground transfers, and (2) perform sensitivity analysis to various system parameters. Topology2 is designed to demonstrate that Harp leads to better allocation of network resources among multiple background transfers. Topology3 is generated by GT-ITM [28], and is used to demonstrate the effectiveness of Harp in complex traffic scenarios. In the first two topologies, T1 to T5 represent the transit nodes, and S# and R# represent source and destination of bulk transfers respectively. Topology3 contains 100 nodes acting as routers, and an additional 20 nodes connected to randomly chosen routers and acting as gateways and relays. The link bandwidths between routers vary from 10Mbps to 100Mbps. Finally, several clients and servers are connected randomly to routers.

The clients and the servers use TCP-Reno as the transport protocol and generate foreground traffic. All routers (represented as shaded circles) perform droptail FIFO queuing. The bottleneck links on each path are represented by dotted lines.

Improvement in completion times We first show that Harp speeds-up background transfers substantially compared to unipath protocols, while causing minimum interference to foreground transfers. We use Topology1 and replay a web proxy trace between server and clients on both paths. The trace has 32

Bytes to 1.7 MB, with an average size of 7.8 KB.

Figure 5(a) shows the completion time for a 100MB background transfer between S1 and R1 with Harp (Harp-multipath), with a unipath protocol (Harp-singlepath), and with TCP-Reno (Reno-singlepath) as we activate different number of clients on each path. The graph shows that, as the number of foreground clients increase, the completion time for the transfer increases in all cases, as expected. However, using multiple paths reduces the background completion time by a factor of two. Observe that Harp-singlepath achieves lower completion time than Reno-singlepath because Reno waits till a loss occurs and halves its congestion window every time the loss occurs. In contrast, Harp-singlepath avoids losses by backing off early (by a smaller proportion defined by β).

To demonstrate that Harp reduces interference significantly compared to a non-background protocol such as Reno, we measure the *stretch* observed by the client requests. We define stretch for each request as the ratio of completion time for client requests when background transfers are active and completion time for the same requests when the transfers are not active. We compare the 90-th percentile stretch observed when the background transfers use one of Harp and TCP-Reno. Figure 5(b) shows that the stretch can be greater than 100% even with a single background transfer using Reno, and it rapidly increases to 500% with 29 transfers. With Harp, the stretch does not increase much with increasing background transfers, and is below 35% even with 29 transfers, thereby demonstrating that Harp transfers compete only for the spare capacity.

Spare capacity utilization Figure 5(c) shows the effectiveness of Harp in quickly harvesting spare capacity left by a long running foreground transfer. The foreground transfer starts at $t = 50s$ and ends at $t = 200s$ and its sending rate is varied during this period as shown in the figure (by controlling its receiver window size). The graph shows that the throughput obtained by Harp closely reflects the spare capacity available with time.

Fairness in resource allocation We first compare the variants (Independent, Joint and Adaptive) of multipath congestion control and show that the Adaptive variant chosen by Harp achieves fairness without compromising utilization. We consider Topology2 where we set up two background transfers: one between S1 and R1 that uses all four paths available (direct and through T2, T4 and T5), and the other between S2 and R2 that uses only one path (S2 to router to T3 to R2). Both transfers share a bottleneck link. The transfer S2R2 starts 100 seconds after S1R1. Figure 6(a) shows the throughput achieved by both the transfers over time. Between 0 and 100s when S1R1 is the only active transfer, the throughput achieved when using Independent and Adaptive is about 20% higher than Joint. When S2R2 becomes active, both Adaptive and Joint yield throughput on path through T2 to ensure that S2R2 gets a better share of resources, whereas Independent does not yield the path thereby being significantly unfair to the transfer S2R2. Furthermore, Adaptive continues to utilize the other three paths more effectively than Joint between 100 and 250s. Note that in this experiment (and the others in this section) line 12 in Algorithm 1 never gets invoked because all transfers backoff based on delays and avoid losses; so the

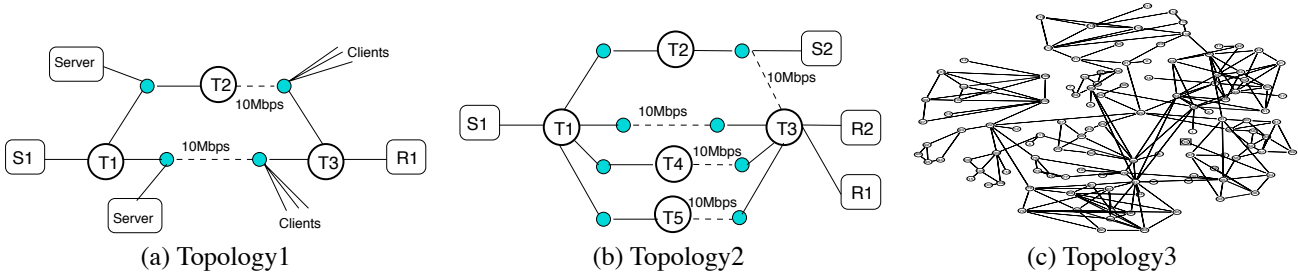(a) Topology1 (b) Topology2 (c) Topology3

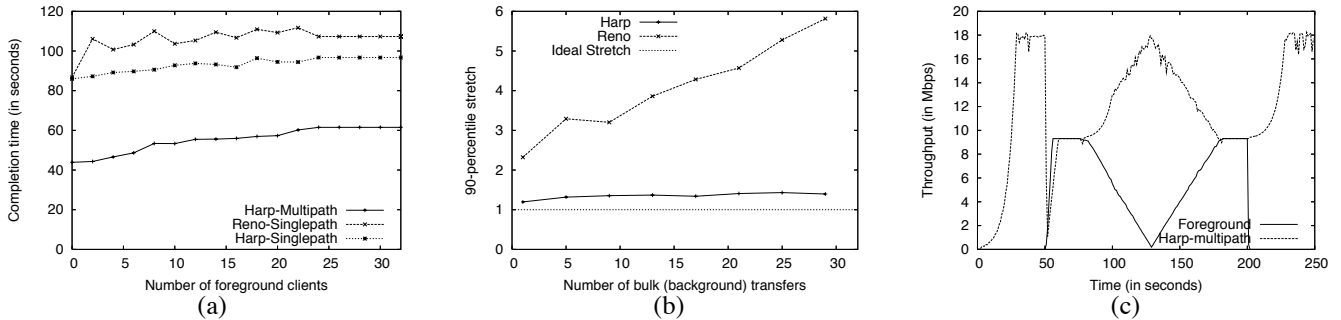Fig. 4. Topologies for ns simulations.

Fig. 5. (a) spare capacity utilization with foreground web transfers, (a) stretch caused to foreground requests, (c) Utilization with a long running foreground transfer

observations show the benefit of Adapative in increasing utilization, while ensuring fair resource allocation when required.

Next, we use Topology2 and start multiple background transfers like S1R1 between T1 and T3, all of them utilizing all paths. Figure 6(b) plots Jain's fairness index [32] for different number of transfers with time. The graph shows that the fairness index stays close to 1.0 with increasing number of transfers when all transfers share the same set of network resources.

Utilization in a large topology We perform several experiments with Topology3 and different foreground and background traffic scenarios. For brevity, we present one result in Figure 6(c). We start 5 background transfers at different entry nodes (each lasting about 50 seconds) and 200 foreground transfers (each lasting about 1 second) at random times in a simulation run of 5 minutes. The graph shows the average throughput achieved by each foreground and background transfer. Harp, when alone (the bar Harp), reaps significant spare capacity in the network. The bars Harp/FG and Harp-S/FG show that (in the presence of foreground transfers) Harp with multiple paths reaps more spare capacity than Harp with single path. The bars FG and FG/Harp represent foreground throughput with and without Harp transfers; Harp has little impact on foreground transfers.

A.1 Sensitivity Analysis

We study the sensitivity of Harp to resource parameters (such as varying bandwidths and link delays on paths), system parameters (such as reorder delay and δ), and design choices (such as choking and handling congestion window mismatch). For brevity, we discuss only a subset of the results. We perform all the experiments using Topology1.

Heterogeneous link delays For this experiment, we vary the delay on the path T1,T2,T3 as a factor of the delay on path T1,T3; e.g., a link delay factor of 1.5 means T1,T2,T3 has 1.5

as the variation in delays in the two paths increases, the completion times achieved by a multipath transfer (when alone and when competing with foreground clients replaying the proxy trace) increases. This graph hints that paths with low variation should be chosen in order to reap maximum benefits of multipath; further, paths with large variation can even be detrimental.

Effect of reordering delay ρ Figure 7(b) shows the effect on completion times of maximum reordering delay ρ (Section III-C) applied to packets at the exit gateway. Small values of ρ result in greater number of packets getting reordered that make the receiver send duplicate acknowledgments and *falsely* indicate congestion to the sender. The sender backs off on such indications and thereby takes longer to complete the transfer. As ρ increases, the completion times decrease and flattens after certain value of ρ. Observe that in presence of foreground transfers, longer reordering delays are required to achieve maximum throughput. Using this graph, we pick ρ=1.5 for our prototype.

Benefit of choking To demonstrate the benefit of choking, we plot in Figure 7(c) the throughput achieved with time by a multipath transfer between S1 and R1 (1) when using and (2) when not using choking. Between t =120s–320s, we start a long-running foreground transfer (not shown in the graph to avoid clutter) from server to a client through T2. The graph shows that when the foreground transfer starts, the bulk transfer backs off from the path T1,T2,T3. However, when choking is not used, the entry gateway sends one data packet per round-trip time to probe for available bandwidth. Since this packet will likely observe increased delay or even get dropped when the foreground transfer is active, packets sent on the direct path get delayed in the reorder queue and thereby reduce the throughput significantly even on the direct path. In contrast, when choking is enabled, the entry gateway sends probe packets that do not contain any data, thereby maintaining high throughput on the direct
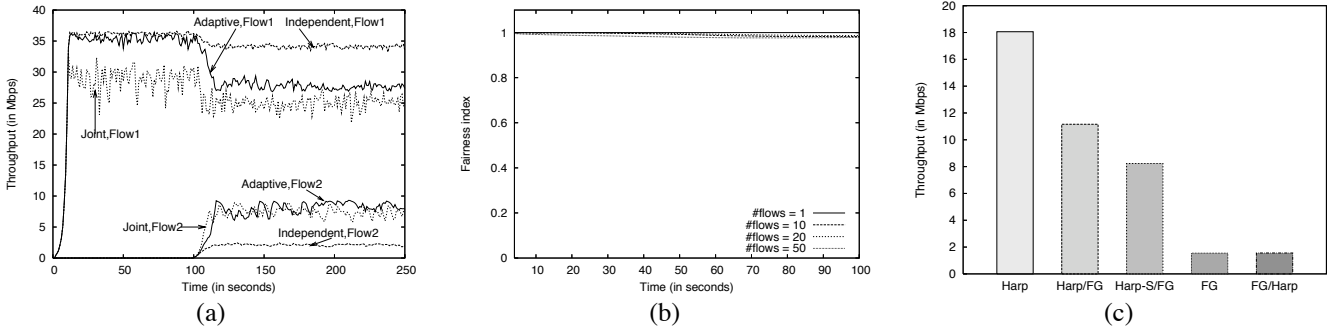
Fig. 6. (a) Resource allocation with different multipath congestion control algorithms, (b) Fairness among multiple multipath transfers, (c) Spare capacity utilization in a large topology
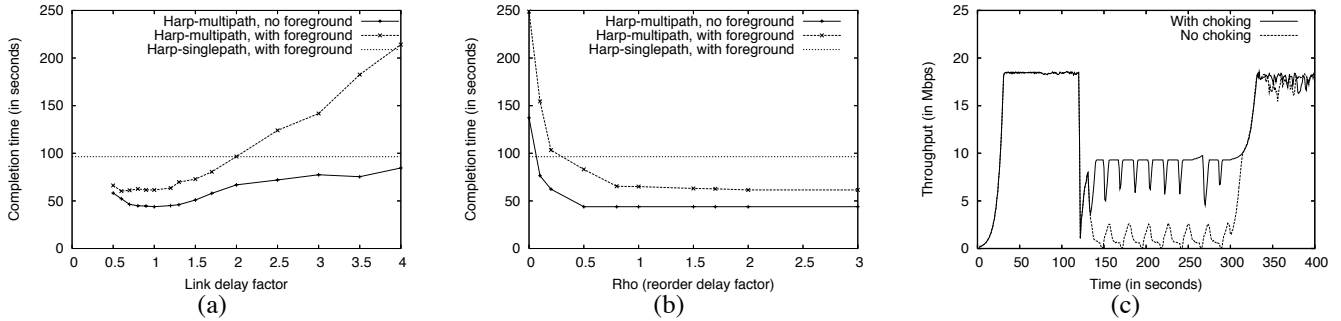

Fig. 7. (a) With increasing indirect path delay, (b) With increasing re-order delay, (c) Benefit of choking
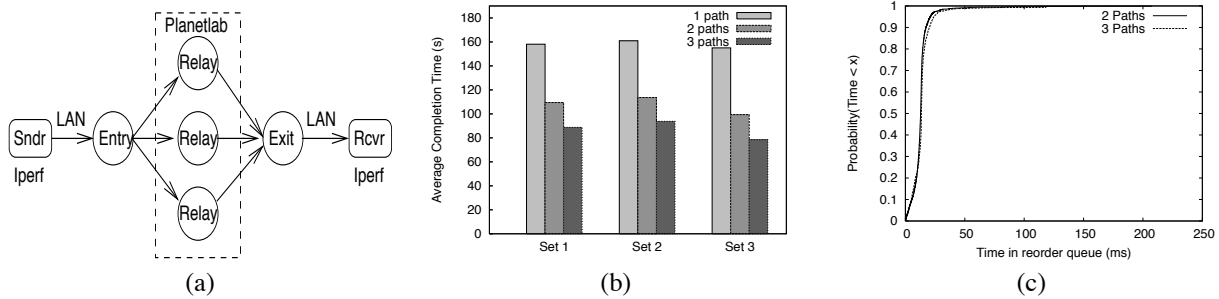

Fig. 8. (a) Wide-area experimental setup, (b) Through achieved by Harp with multiple paths, and (c) reorder delay CDF at the exit gateway.

B. Prototype Evaluation

In this section, we present an evaluation of Harp over Planet-lab [24] to demonstrate its effectiveness in an Internet setting.

Experimental Setup We setup an entry gateway and an exit gateway at Rutgers University and UMASS Amherst, and place a sender and a receiver on the same LAN as the entry and exit gateways respectively to avoid the access link of the sender and the receiver from becoming the bottleneck (see Figure 8(a)). We use several planetlab nodes shown in Table II to act as relay nodes to provide path diversity between the entry and exit gateways. We divide the planetlab nodes into three sets and repeat each experiment on all three sets. The right column in the table shows the range of minimum one-way delays between the entry and exit gateways through the relay nodes. We use Iperf [33] to generate foreground and background transfers of size 100MB.

Results Figure 8(b) shows the completion times observed by the transfer when using one, two and three paths for each of the

	Relay nodes	One-way Delays (ms)
Set 1	Columbia, Dartmouth, MIT	24-29
Set 2	NYU, Purdue UIUC	26-38
Set 3	Cornell, Princeton, RPI	23-32

Table II. Relay nodes, and one-way delays between entry and exit gateways through the relays.

show the average completion time. The figure shows that by using two additional paths, Harp decreases the completion time of a background transfer in all three sets by about a factor of two.

To understand the impact of using multiple paths on packet reordering, we plot in Figure 8(c) the CDF of the time spent by the packets in the reorder queue at the exit gateway when using two and three paths. The graph shows that more than 90% of the packets spend less than 20ms in the queue, demonstrating that a small queue is sufficient to avoid most reordering. Also, less than 1% of the packets suffer a timeout and cause packet reordering.

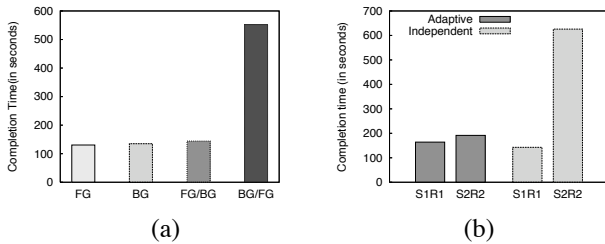To demonstrate the non-interference and fairness properties

Fig. 9. (a) Non-interference, (b) Fairness

ceiver pairs S1R1 and S2R2 sharing links similar to Figure 3; S1R1 has access to two paths (through MIT and Dartmouth) and S2R2 has one path (through MIT) that is common with S1R1. For Figure 13(a), we setup one background transfer between S1R1 that uses the single path through MIT, and one foreground transfer between S2R2. We consider three scenarios— foreground alone, background alone, and foreground with background. We make the following observations: (1) When alone, background completion time (second bar) is close to the foreground completion time (first bar), thereby demonstrating that background reaps spare capacity effectively, and (2) foreground completion time with background (third bar) is close to foreground alone, thereby demonstrating that background effectively backs-off to make way for foreground; background's completion time increases substantially as a result (fourth bar). For Figure 13(b), we setup a background transfer between S1R1 that uses both the available paths and a background transfer between S2R2, and examine two variants of multipath congestion control. The graph shows that S2R2 observes lower completion time with Adaptive than with Independent, thereby demonstrating that Adaptive leads to a fairer allocation of resources.

V. RELATED WORK

This work bridges ideas from background transport protocols, multipath routing and congestion control, and overlay-based network sevices.

The primary goal of background transport is to be noninterfering with foreground transfers while providing practical levels of utilization of spare capacity. Noninterference is conceptually easy to provide using differentiated services, or Diffserv [13]. Once deployed, heterogeneous pricing can incent users to truthfully mark their traffic as foreground or background. However, it is unclear if and when router support to assist transport protocols will become available.

A more pragmatic effort in recent times has led to the development of end-host based protocols such as TCP Nice [14], TCP-LP [15], and Key et al. [16] that emulate low-priority transport without support from routers. The delay-based backoff in Harp's controller is similar in spirit to Nice or LP. Our previous experience with Nice suggests that end-host based protocols are imperfect due to feedback delay and can significantly degrade utilization to maintain noninterference, e.g., in the presence of a competing Web workload, Nice can leave upto 50% of spare capacity on the path unused. Furthermore, even a perfect background protocol will find few takers without a suitable incentive mechanism for users to "nice" transfers.

damentally lack a mechanism for network-wide scheduling of resources and can arbitrarily delay background transfers along a path. Various studies have shown significant levels of spare capacity [14, 34], load imbalance [34], and background traffic in the Internet. Harp's support for multipath background transport exploits this state of affairs to ensure quick completion of both foreground and background transfers.

Application-level approaches for background transport include (*i*) coarse-grain scheduling such as running backup [35] and data prefetching [36] applications during off-peak hours, (*ii*) rate limiting for applications such as Web prefetching [37, 38], (*iii*) adaptive rate control such as in Microsoft XP's Background Intelligent Transfer Service [39] etc. These require tedious manual effort to configure parameters to reduce interference. Overlay quality-of-service approaches such as OverQoS [40] and MPAT [41] can provide limited noninterference among the set of flows they manage. In contrast, Harp provides a network-wide abstraction of low priority service. Harp does not require a large number of flows and can operate equally well with a single user. Harp's relay nodes are simple and provide only forwarding functionality, while its gateway-based congestion control feature exists primarily to ease deployment at access points. Broadband users can directly use Harp without a gateway.

Multipath routing has seen a long history of work since Maxemchuk's seminal work on dispersity routing [42] to improve throughput and resilience to path failures or packet losses. Striping or inverse multiplexing [43, 44, 45, 46, 47] provides link level mechanisms for splitting input flows among multiple links to increase throughput. More recently, multipath techniques have been proposed in the context of overlay routing or multi-homed clients [48, 49, 50, 51]. Katabi et al. [52] and Elwalid et al. [53], propose splitting aggregate traffic flows along multiple paths to achieve load balancing and stability in the context of intradomain traffic engineering. In contrast, Harp uses multiple paths for scheduling transfers with heterogeneous requirements.

Lee et al. [54] and Zhang et al. [25] propose modifications to TCP's fast retransmit or delayed acknowledgment schemes to tolerate packet reordering across heterogeneous paths. Harp interoperates with legacy TCP implementations by interposing a network-layer connection proxy at a gateway or end-host similar in spirit to our previous work on Oasis [55], a refactored network stack for overlay transport services.

Multipath transport mechanisms pTCP [56] and RMTP [57] do not consider fairness when two or more paths for a single flow share the same bottleneck link. mTCP [25] by Zhang et al. explicitly identifies paths with shared bottlenecks and treats them as one flow. Thus, mTCP is TCP-fair on each path. However, it does not provide fairness of network-wide resource allocation, as users having access to more paths obtain an unfair share of network resources.

In contrast, Multipath TCP [20] and KV [19] are based on a utility-theoretic framework for network-wide resource allocation (see Kelly et al. [18] for a seminal paper), and systematically address fairness. These controllers maintain nontrivial sending rates for a multipath connection only along its best set of paths and the aggregate throughout is commensurate to a unipath connection's share of network resources on one of its best

single flow on paths with shared congestion. Harp's congestion controller is based on KV, but uses delay-based congestion signals and provides a background abstraction. Our experimental effort led us to identify and address a critical limitation of the fluid model assumed in the above theoretical works, and is also applicable to foreground transport.

VI. CONCLUSIONS

In this paper, we present the design and implementation of a network architecture, Harp, to allocate network resources more in line with human needs in a background-dominated Internet. Harp is based upon a multipath background congestion control algorithm that moves background traffic out of the way of foreground to ensure quick completion for both kinds of transfers. Compared to end-host protocols, this approach improves utilization and fairness of resource allocation. Our gateway-based implementation obviates modification to or cooperation from end-hosts. Our extensive evaluation using simulations and a prototype deployed on Planetlab demonstrates significant reduction in human wait-time.

REFERENCES

[1] "Kazaa." http://www.kazaa.com.
[2] B. Cohen, "Incentives Build Robustness in BitTorrent," in *http://www.bittorrent.com/bittorrentecon.pdf*.
[3] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: high bandwidth data dissemination using an overlay mesh," in *SOSP*, 2003.
[4] L. Cherkasova and J. Lee, "FastReplica: Efficient Large File Distribution within Content Delivery Networks," in *Proc. of USITS*, 2003.
[5] S. Ganguly, A. Saxena, S. Bhatnagar, S. Banerjee, and R. Izmailov, "Fast replication in content distribution overlays," in *Proc. of Infocom*, 2005.
[6] R. Kokku, P. Yalagandula, A. Venkataramani, and M. Dahlin, "NPS: A Non-interfering Deployable Web Prefetching System," in *USITS*, 2003.
[7] D. Rosenberg, "Prefetching Content for Increased Performance." http://devedge-temp.mozilla.org/viewsource/2003/link-prefetching/index_en.html, Feb 2003.
[8] "Imx series ip set top box." http://www.matrixstream.com/IMX 1000.cfm.
[9] "LiveVault Corporation." http://www.livevault.com/.
[10] "Remote backup systems." http://remote-backup.com/rbackup/.
[11] "Microsoft Windows Update." http://windowsupdate.microsoft.com/.
[12] CacheLogic Home Page, "Advanced Solutions for P2P Networks Home Page." http://cachelogic.com/research/p2p2005.php.
[13] S. Blake *et al.*, "An architecture for differentiated services," in *RFC 2475*, 1998.
[14] A. Venkatramani, R. Kokku, and M. Dahlin, "TCP-Nice: A Mechanism for Background Transfers," in *Proc. of OSDI*, 2002.
[15] A. Kuzmanovic and E. W. Knightly, "TCP-LP: A Distributed Algorithm for Low Priority Data Transfer.," in *Proc. of INFOCOM 2003*, 2003.
[16] P. Key, L. Massoulie;, and B. Wang, "Emulating low-priority transport at the application layer: a background transfer service," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, 2004.
[17] L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet Impasse through Virtualization," in *HOTNETS III*, 2004.
[18] F. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: shadow prices, proportional fairness and stability," in *Journal of the Operational Research Society*, vol. 49, 1998.
[19] F. Kelly and T. Voice, "Stability of end-to-end Algorithms for Joint Routing and Rate Control," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, pp. 5–12, 2005.
[20] H. Han *et al.*, "Multi-path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet," in *IMA Workshop on Measurements and Modeling of the Internet*, 2004.
[21] "Akamai, Inc. Home Page." www.akamai.com.
[22] "Google Enterprise Services." http://www.google.com/enterprise.
[23] E. Kohler *et al.*, "The Click Modular Router," *ACM Transactions on Computer Systems*, vol. 18, August 2000.
[24] "PlanetLab." http://www.planet-lab.org.
[25] M. Zhang *et al.*, "A Transport Layer Approach for Improving End-to-End Performance and Robustness Using Redundant Paths," in *Proc. of the*
[26] R. Kokku, A. Bohra, S. Ganguly, and A. Venkataramani, "A Multipath Background Network Architecture. *http://www.cs.umass.edu/~arun/harp-tech.pdf*," tech. rep., University of Massachusetts, Amherst, 2006.
[27] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison Wesley, 1994.
[28] "Modeling topology of large internetworks." http://www-static.cc.gatech.edu/projects/gtitm/.
[29] K. K. Ramakrishnan and R. Jain, "A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer," in *SIGCOMM '88*, (New York, NY, USA), 1988.
[30] T. Kelly, *Engineering flow controls for the Internet*. PhD thesis, University of Cambridge, 2004.
[31] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, "One more bit is enough," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, 2005.
[32] D. Chiu and R. Jain, "Analysis of the increase/decrease algorithms for congestion avoidance in computer networks," *Computer Networks*, 1989.
[33] "Iperf, The TCP/UDP Bandwidth Measurement Tool." http://dast.nlanr.net/Projects/Iperf/.
[34] A. Akella, S. Seshan, and A. Shaikh, "An empirical evaluation of wide-area internet bottlenecks," in *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, (New York, NY, USA), pp. 101–114, ACM Press, 2003.
[35] C. Maltzahn, K. Richardson, D. Grunwald, and J. Martin, "On bandwidth smoothing," in *4th International Web Caching Workshop*, 1999.
[36] S. Dykes and K. A. Robbins, ""a viability analysis of cooperative proxy caching"," in *INFOCOM 2001*, 2001.
[37] N. Spring *et al.*, "Receiver based management of low bandwidth access links," in *Proc of IEEE Infocom*, 2000.
[38] M. Crovella and P. Barford, "The network effects of prefetching," in *Infocom*, 1998.
[39] "Windows XP Background Intelligent Transfer Service." http://www.microsoft.com/windowsserver2003 /tech-info/overview/bits.mspx.
[40] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz, "Overqos: offering internet qos using overlays," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, 2003.
[41] M. Singh, P. Pradhan, and P. Francis, "MPAT: Aggregate TCP Congestion Management as a Building Block for Internet QoS," in *ICNP*, 2004.
[42] N. F. Maxemchuk, *Dispersity routing in store-and-forward networks*. PhD thesis, Univ. Pennsylvania, Philadelphia, 1975.
[43] H. Adiseshu, G. Parulkar, and G. Varghese, "A reliable and scalable striping protocol," in *ACM SIGCOMM*, 1996.
[44] B. Traw and J. Smith, "Striping within the network subsystem," in *IEEE Network*, vol. 9, 1995.
[45] J. Duncanson, "Inverse multiplexing," in *IEEE Communications Magazine*, vol. 32, 1994.
[46] A. Qureshi and J. Guttag, "Horde: separating network striping policy from mechanism," in *ACM MobiSys*, 2005.
[47] K.-H. Kim and K. G. Shin, "Improving TCP Performance over Wireless Networks with Collaborative Multi-homed Mobile Hosts," in *ACM MobiSys*, 2005.
[48] D. Andersen, A. Snoeren, and H. Balakrishnan, "Best-Path vs. Multi-Path Overlay Routing," in *Proc of IMC*, 2003.
[49] A. Akella and J. Pang and B. Maggs and S. Seshan and A. Shaikh, "A Comparison of Overlay Routing and Multihoming Route Control," in *Proc of ACM SIGCOMM*, 2004.
[50] A. Sen *et al.*, "On Multipath Routing with Transit Hubs," in *Proc. of Networking*, May 2005.
[51] A. Snoeren, K. Conley, and D. Gifford, "Mesh-Based Content Routing using XML," in *SOSP*, 2001.
[52] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the Tightrope: Responsive Yet Stable Traffic Engineering," in *Proc. of SIGCOMM*, (Philadelphia), Aug 2005.
[53] A. Elwalid, C. Jin, S. H. Low, and I. Widjaja, "MATE: MPLS adaptive traffic engineering," in *INFOCOM*, 2001.
[54] Y. Lee, I. Park, and Y. Choi, "Improving TCP Performance in Multipath Packet Forwarding Networks," in *Journal of Communication and Networks*, vol. 4(2), Jun 2005.
[55] H. V. Madhyastha, A. Venkataramani, A. Krishnamurthy, and T. Anderson, "Oasis: an overlay-aware network stack," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 41–48, 2006.
[56] H.-Y. Hsieh and R. Sivakumar, "A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts," in *MobiCom*, 2002.
[57] L. Magalhaes and R. Kravets, "Transport level mechanisms for bandwidth

APPENDIX

I. ANALYSIS OF HARP'S CONGESTION CONTROLLER

Consider a simple droptail FIFO queueing network that induces exactly one bottleneck on each path. Assume a synchronous fluid model, i.e., packets are infinitesimal in size and a loss or delay signal is instantaneously observed by all flows through the corresponding router.

Theorem 1: (a) Joint control of congestion and routing can degrade steady-state throughput by an $\Omega(\log(C))$ factor compared to independent control along a path of bottleneck capacity C, (b) Adaptive control achieves the same steady-state throughput as independent when there is exactly one flow along the path.

Theorem 2: The interference, or reduction in throughput of foreground flows, that Harp's multipath background controller inflicts is bounded by a factor $O(e^{(-\frac{B}{m} \cdot (1-\delta)(1-\beta))})$, where B is the bottleneck buffer capacity, δ the delay threshold, β the MD parameter, and m the number of foreground flows along the path, *independent of the number of background flows.*

II. ANALYSIS OF HARP'S INTERFERENCE

In this section we give a proof of Theorem 2 that derives an upper bound on the intereference inflicted by Harp on TCP flows.

We use a simplified fluid approximation model of the network to help model the interaction of multiple flows using different congestion control algorithms. We assume that each active path has exactly one bottleneck, and that routers perform FIFO droptail queuing. In the following, we use a *flow* to mean a unipath transfer along a point-to-point path and a *connection* to mean a potentially multipath transfer consisting of several individual flows.

Let μ denote the service rate and B the buffer capacity of a router at a bottlenecked link. Assume that all flows through the link have the same round-trip delay; we will relax this assumption subsequently. Let τ denote this round-trip delay excluding all queuing delays.

Consider a fixed number, m, of (unipath) TCP flows and l Harp multipath connections at least one of whose paths goes through the link. Let $q_t = \delta \cdot B$ be the queue size that triggers early delay-based backoff for Harp flows. Recall that δ is the fraction of the queue capacity, B, that triggers this backoff. By assumption, the connections are synchronous, i.e. they experience the same propagation delay τ and, in the case of buffer overflow, all connections simultaneously detect a loss and multiply their window sizes by $1 - \beta$. We model only TCP's congestion avoidance phase to analyze the steady-state behavior. We remark that TCP's slow start is more aggressive than congestion avoidance, so the interference inflicted by Harp is likely to be less in slow start.

We obtain a bound on the reduction in the throughput of TCP flows due to the presence of Harp connections by analyzing the dynamics of the queue at the bottleneck link. We achieve this goal by dividing the duration of the flows into *periods*. In each period we bound the decrease in the number of TCP packets processed by the router due to interfering Harp packets.

Let $W_r(t)$ and $W_n(t)$ denote respectively the total number of

total window size, $W(t)$, is $W_r(t) + W_n(t)$. We trace these window sizes across periods. *The end of a period and the beginning of the next is marked by a packet loss*, at which time each flow reduces its window size by a factor of $1 - \beta$. Just before a loss, we have $W(t) = \mu\tau + B$, and $W(t) = (\mu\tau + B)(1 - \beta)$ just after. Let t_0 be the beginning of one such period just after a loss. Consider the case when $W(t_0) = (\mu\tau + B)(1 - \beta) < \mu\tau$ and $m > l$. The window dynamics in any period can be split into three intervals as described below.

We formulate differential equations to describe the dynamics of the queue size in the following three intervals of a period. In the interval descriptions below, the first term denotes TCP dynamics and the second Harp dynamics.

Additive Increase, Multiplicative Increase $[t_0, t_1]$:

$$\begin{cases} \frac{dW_r(t)}{dt} &= \frac{m}{\tau} \\ \frac{dW_n(t)}{dt} &= \frac{l \cdot W_n(t)}{\tau} \end{cases} \quad (1)$$

Additive Increase, Multiplicative Increase $[t_1, t_2]$:

$$\begin{cases} \frac{dW_r(t)}{dt} &= \frac{m\mu}{W(t)} \\ \frac{dW_n(t)}{dt} &= \frac{l\mu W_n(t)}{W(t)} \end{cases} \quad (2)$$

Additive Increase, Multiplicative Decrease $[t_2, t_3]$:

$$\begin{cases} \frac{dW_r(t)}{dt} &= \frac{m\mu}{W(t)} \\ \frac{dW_n(t)}{dt} &<= -min(\frac{\beta W_n(t)\mu}{W(t)}, \frac{m\mu}{W(t)}) \end{cases} \quad (3)$$

Note that in equation 13, the effective round-trip delay increases from τ to $W(t)/\mu$. Equation 3 is somewhat nontrivial. The right hand side of the second equation in 3 says that the rate of decrease of Harp flows is at most the rate of increase of TCP flows, as otherwise, the net decrease will outweigh increase causing the queue size to drop below the delay threshold halting Harp's backoff. The inequality sign exists as a Harp flow's backoff is proportional to the aggregate rate of the multipath connection to which it belongs. This backoff is greater than or equal to the backoff that would result if each Harp flow in the queue were the only flow with a nonzero sending rate in its corresponding multipath connection.

A. Proofs

Next, we prove three lemmas that make it easier to calculate the reduction in the total amount of TCP flow that gets sent in a period due to interference from Harp flows.

Lemma 1: The values of W_r and W_n at the beginning of periods stabilize after several packet losses, so that each period thereafter is of a fixed duration T.

Proof: Let t_0 mark the beginning of a period just after a packet loss so that $W(t_0) = (\mu\tau + B)(1 - \beta) < \mu\tau$. Let t_1, t_2, t_3 mark respectively the end of the three intervals constituting the current period. The beginning of the interval $[t_2, t_3]$ initiates multiplica-

equation 3. Note that for $t < t_2$, $W_n(t) \geq l$ as each Harp flow maintains a window of at least one packet.

If $W_n(t_2) > 2m$, W_n multiplicatively decreases at a rate just enough to counter the rate of increase of W_r, thereby keeping W fixed at $\mu\tau + q_t$ till time t'_2 such that $W_n(t'_2) = 2m$. For the rest of the period $[t'_2, t_3]$ the system dynamics are given by:

$$\begin{cases} \frac{dW_r(t)}{dt} & = & \frac{m\mu}{W(t)} \\ \\ \frac{dW_n(t)}{dt} & = & -\frac{\beta W_n(t)\mu}{W(t)} \end{cases} \qquad (4)$$

These equations yield a unique solution to $W_r(t_3)$ and $W_n(t_3)$, and therefore $W_r(t_3^+)$ and $W_n(t_3^+)$. The dynamics in every period thereafter exactly emulates the current one.

If $W_n(t_2) < 2m$, it is straightforward to show that W_n at the beginning of the third interval in every period thereafter keeps monotonically increasing or decreasing. However, as stated above above, this value is bounded below by l and above by $2m$. Thus, the length of the period T and the system dynamics of a period converge.

 Corollary 1: After the first packet loss, the residual number of outstanding Harp packets ϵ just before a packet loss is bounded above by $W_n(t_3)$, obtained by solving (4) with the following boundary conditions:

Initial: $W_n(t'_2) = 2m$, $W(t'_2) = \mu\tau + q_t$
Final: $W(t_3) = \mu\tau + B$

Lemma 2: The total amount of flow sent by TCP flows in a period depends only on the initial and final values of W_r in the period.

Proof: The total amount of flow sent by TCP flows is given by integrating the instantaneous sending rate over the duration of the period. The instantaneous sending rate at time t is obtained by dividing the current window size $W_r(t)$ by the current value of the round-trip delay $W(t)/\mu$. Thus, the total amount of TCP flow R sent out in a period beginning at t_0 is given by:

$$R = \frac{1}{m} \int_{t_0}^{t_0+T} \frac{W_r(t)\mu}{W(t)} dt \qquad (5)$$

From (1),(13) and (3) we observe that the rate of increase of W_r throughout a period is given by:

$$\frac{dW_r(t)}{dt} = \frac{m\mu}{W(t)} \qquad (6)$$

From (5) and (6) we get:

$$\begin{aligned} R & = \frac{1}{m} \int_{W_r(t_0)}^{W_r(t_0+T)} W_r(t)dW_r(t) \\ \\ & = \frac{W_r^2(t)}{2m} \Big|_{t_0}^{t_0+T} \end{aligned} \qquad (7)$$

Hence proved.

flows and a nonzero number of Harp flows is shorter than that of a system consisting of only the TCP flows.

Proof: The proof of this lemma follows straighforwardly from the dynamics of $W_r(t)$ alone. Let T' denote the length of a period when only m TCP flows (and no Harp flows) are present, and $W'_r(t)$ denote the total number of outstanding TCP packets at time t. Rewriting (6) for $W'_r(t)$:

$$W'_r(t)dW'_r(t) = m\mu dt \qquad (8)$$

Assume that a period begins at time t_0.

$$\begin{aligned} T' & = \frac{1}{m\mu} \int_{t_0}^{t_0+T'} W'_r(t)dW'_r(t) \\ \\ & = \int_{(\mu\tau+B)(1-\beta)}^{(\mu\tau+B)} W'_r dW'_r \end{aligned} \qquad (9)$$

Similarly, using (6) we obtain for T:

$$\begin{aligned} T & = \int_{t_0}^{t_0+T} W(t)dW_r(t) \\ \\ & = \int_{t_0}^{t_0+T} W(t)dW(t) \cdot \frac{dW_r(t)}{dW(t)} \\ \\ & = \int_{(\mu\tau+B)(1-\beta)}^{(\mu\tau+B)} W dW \cdot \frac{dW_r}{dW} \end{aligned} \qquad (10)$$

Notice that the multiplicative term $\frac{dW_r}{dW} < 1$ throughout the period, as $W = W_r + W_n$. Therefore $T < T'$.

B. Bounding Interference

 Interference is defined as the fractional loss in throughput of TCP flows due to the presence of Harp flows. In order to compute this throughput, we first need the residual number of outstanding Harp packets ϵ just before the end of a period. We use Corollary 1 and solve equatiion 4. Dividing the second differential equation by the first we obtain:

$$\frac{dW_n(t)}{dW_r(t)} = -\frac{\beta W_n(t)}{m}$$

$$\Rightarrow \frac{dW_n(t)}{W_n(t)} = -\frac{\beta dW_r(t)}{m}$$

Integrating both sides we obtain:

$$\int_{m/\beta}^{\epsilon} \frac{dW_n}{W_n} = -\frac{\beta}{m} \int_{\mu\tau+q_t-m/\beta}^{\mu\tau+B-\epsilon} dW_r$$

$$\Rightarrow \log(\frac{\beta\epsilon}{m}) \leq -\frac{\beta}{m}[B - q_t]$$

$$\Rightarrow \epsilon \leq \frac{m}{\beta} \cdot e^{(-(\frac{\beta(B-q_t)}{m}))}$$

$$\Rightarrow \epsilon \leq \frac{m}{\beta} \cdot e^{(-(\frac{\beta(1-\delta)B}{m}))}$$

The fractional loss in throughput of TCP flows may now be easily computed using Lemma 2. The asymptotic throughput obtained by TCP flows is the total flow sent out in a period di-

sent out by the TCP flows is a period depends only on the initial and final values of $W_r(t)$ in a period. Thus, the throughput P obtained by TCP flows is computed using (7) as:

$$P = \frac{\beta}{mT} \cdot [(\mu\tau + B - \epsilon)^2 - ((\mu\tau + B - \epsilon)(1 - \beta))^2]$$

The throughput obtained by the TCP flows in the absence of any Harp flows is given by:

$$Q = \frac{\beta}{mT'} \cdot (\mu\tau + B)^2 (1 - (1 - \beta)^2)$$

The interference I defined as the fractional loss in throughput is given by $\frac{Q-P}{Q}$. By Lemma 3, $T' > T$, which yields:

$$I \leq \frac{[(\mu\tau + B - \epsilon)^2 - (\mu\tau + B)^2]}{(\mu\tau + B)^2}$$

$$\leq \frac{2\epsilon}{(\mu\tau + B)} \qquad (11)$$

$$= \frac{2m}{\beta(\mu\tau + B)} e^{(-(\frac{\beta(1-\delta)B}{m}))}$$

Hence Theorem 1 follows.

C. Case $m < l$

In this case we simply give the differential equations governing the dynamics of the window sizes claim that Lemmas 1 to 3 hold in this case as well. The proof is similar to the previous case and is omitted.

In interval $[t_0, t_1]$ $W(t)$ increases from $W(t_0)$ to $\mu\tau$, at which point the queue starts building up. TCP flows increase linearly and Harp flows increase multiplicatively and their dynamics can be represented as:

$$\begin{cases} \frac{dW_r(t)}{dt} = \frac{m}{\tau} \\ \frac{dW_n(t)}{dt} = \frac{l}{\tau} \end{cases} \qquad (12)$$

The next interval $[t_1, t_2]$ is also marked by additive increase of W_r, and multiplicative increase of W_n. The window dynamics during interval $[t_1, t_2]$ are as follows:

$$\begin{cases} \frac{dW_r(t)}{dt} = \frac{m\mu}{W(t)} \\ \frac{dW_n(t)}{dt} = \frac{l\mu W_n(t)}{W(t)} \end{cases} \qquad (13)$$

The end of this interval is the time t_2 when $W(t_2) = \mu\tau + q_t$, where q_t is the threshold queue size that begins multiplicative backoff for Harp flows. However, again the rate of decrease of $W_n(t)$ is bounded by the rate of increase of increase of $W_r(t)$. Thus, the dynamics of interval $[t_2, t_3]$ are governed by:

$$\begin{cases} \frac{dW_r(t)}{dt} = \frac{m\mu}{W(t)} \\ \frac{dW_n(t)}{dt} < -min(\frac{\beta W_n(t)\mu}{W(t)}, \frac{m\mu}{W(t)}) \end{cases} \qquad (14)$$

The end of the above interval marks the completion of the period. At this point $W(t_3) = \mu\tau + B$, and right after, each flow

the next period.

Using Lemmas 1 to 3, it is easily shown that the interference bound in (11) continues to hold. We note that the magnitude of l does not affect the interference simply because the bigger l is, the faster the multiplicative backoff.

III. ANALYSIS OF INDEPENDENT VS. JOINT CONTROL

In this section, we present an intuition of why Joint control can lead to substantial underutilization of a path in comparison to Independent control. To do so, we first derive the average throughput obtained by a flow using either of the two controls.
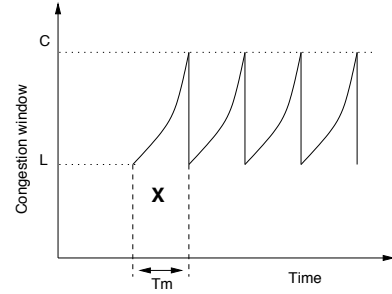

Fig. 10. MIMD behavior

Consider Figure 10 that shows the behavior of a congestion control algorithm using MIMD. C represents the maximum congestion window (in packets) a flow would observe on a particular path before backing-off because of dropped or marked packets. C also represents the maximum capacity of the path. The MD part determines the new congestion window L to which the path of the flow drops. Let α represent the MI parameter, and T_m represent the time it takes for the congestion window to reach C from L. Then,

$$L \times (1 + \alpha)^{T_m} = C$$
$$\Rightarrow T_m = \frac{\log\left(\frac{C}{L}\right)}{\log(1 + \alpha)} \qquad (15)$$

The average throughput B of a single flow on a path under the above behavior can be calculated as $1/T_m \times$ (area of region X), i.e.,

$$B = \frac{1}{T_m} \times \int_0^{T_m} L \times (1 + \alpha)^t \qquad (16)$$

Simplifying Equation 16, and using Equation 15,

$$B = \frac{C - L}{\log\left(\frac{C}{L}\right)} \qquad (17)$$

Now, let β be the MD parameter. On a decrement, the congestion window of Independent is $L = C \times (1 - \beta)$. For joint, this value can be lower than that of Independent because the decrement is based on the congestion windows of other paths that are not in consideration here. Assume that L is lower bounded by 1. In the worst case, Joint can take the value of L down to 1.

Joint can be calculated as

$$B_I = \frac{C \times \beta}{-log\,(1-\beta)} \quad (18)$$

$$B_C = \frac{C-1}{log\,C} \quad (19)$$

Substituting $\beta = 0.125$ in Equation 18 and evaluating the ratio of average throughput obtained by Independent and Joint, we get

$$\frac{B_I}{B_C} \approx \frac{0.94 \times C \times log\,C}{C-1} \quad (20)$$

When C=100 packets, the throughput ratio on a path is 4.3, i.e., a flow using Joint instead of Independent achieves about 75% lower throughput on a path even if it is the only flow using the path.

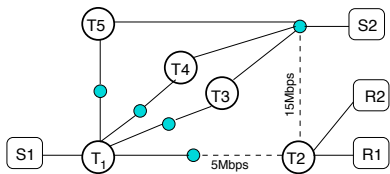## IV. ADDITIONAL SIMULATION AND PROTOTYPE RESULTS

Fig. 11. Topology4.

Fairness on shared bottleneck links We now demonstrate that Adaptive almost inherits from Joint the property of implicitly detecting shared bottleneck links and achieving only a fair share of resources. We use Topology3 for this experiment with S1R1 being the bulk transfer using up to four paths (direct and through T3, T4 and T5), and S2R2 using only the direct path through T2. We consider two scenarios: (1)One-indirect: S1R1 uses only one of indirect path through T3 out of the three, and (2) Three-indirect: S1R1 uses all three indirect paths. The graphs in Figure 12 compare resource allocation between S1R1 and S2R2 using Joint, Independent and Adaptive control. Figures 12(b) and 12(e) (when compared to the other graphs) show that Independent makes S1R1 more aggressive when using all three paths through the bottleneck link shared by S2R2, thereby substantially hurting S2R2. Whereas both Adaptive and Joint eventually backoff from using two of the three paths (except for sending probe packets). Figures 6 and 12 together demonstrate that Adaptive represents the best of Independent and Joint.

Prototype Evaluation To demonstrate the non-interference and fairness properties of Harp in simple realistic settings, we consider two sender receiver pairs S1R1 and S2R2 sharing links similar to Figure 3; S1R1 has access to two paths (through MIT and Dartmouth) and S2R2 has one path (through MIT) that is common with S1R1. For Figure 13(a), we setup one background transfer between S1R1 that uses the single path through MIT, and one foreground transfer between S2R2. We consider three scenarios—foreground alone, background alone, and foreground with background. We make the following observations: (1) When alone, background completion time (sec-

thereby demonstrating that background reaps spare capacity effectively, and (2) foreground completion time with background (third bar) is close to foreground alone, thereby demonstrating that background effectively backs-off to make way for foreground; background's completion time increases substantially as a result (fourth bar). For Figure 13(b),(c), we setup a background transfer between S1R1 that uses both the available paths and a background transfer between S2R2, and examine two variants of multipath congestion control. The graph shows that S2R2 observes higher throughput with Adaptive than with Independent, thereby demonstrating that Adaptive leads to a fairer allocation of resources.

Figure 14 shows the variation of the throughput over time along with the delay between the time the packet is sent out by the entry gateway and the time it is received by the exit gateway. From Figure 14 (a) we see that while most packets are received within 60ms, some observe significant delays which leads to the sender timing out and retransmitting. This leads to the sending rate being drastically reduced and the consequence is a lower average throughput. A similar behavior is seen over two paths (Figure 14 (b)).
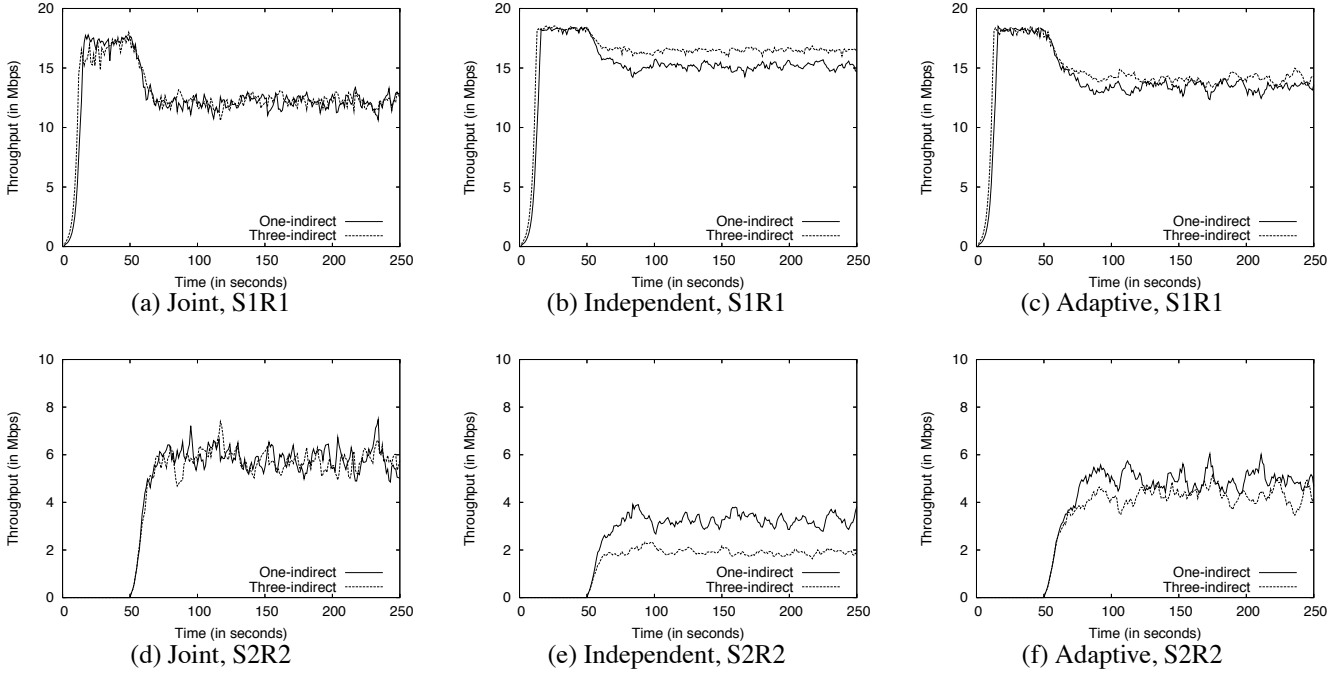
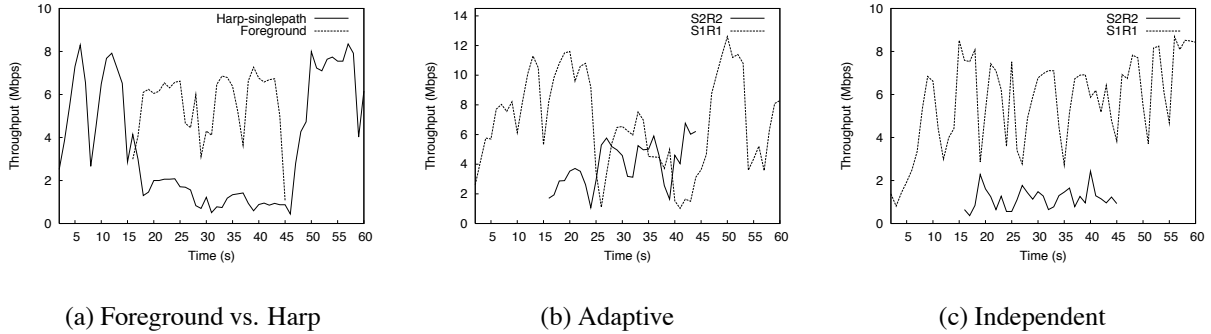Fig. 12. Resource allocation for transfers S1R1 and S2R2 without and with shared bottleneck.

Fig. 13. (a) Non-interference, (b) and (c) Resource allocation among two background transfers with Adaptive and Independent
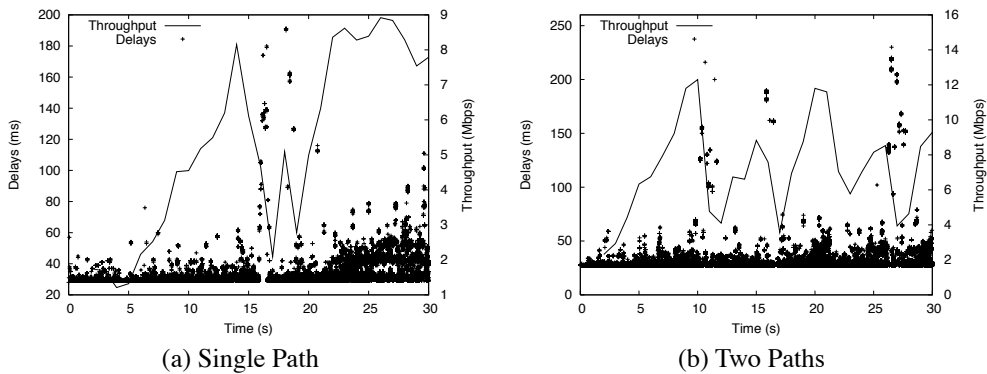

Fig. 14. Throughput in time and the distribution of delays for (a) one path and (b) two paths