

HIR: Engineering Multi-Path Background Transfer

Mi Kyung Han Zifei Zhong Mukesh Kumar

University of Texas at Austin

{hanmi2,zfz,mukesh}@cs.utexas.edu

<http://www.cs.utexas.edu/~zfz/hir/>

ABSTRACT

Recent study [1] has reported that 90 percent of Internet traffic is *background* transfers - transfers that humans do not actively wait on - which results in unwanted interference to *foreground* transfers. Harp [20] is a multipath background network architecture which addresses this problem. Harp exploits path diversity, load imbalance, and spare capacity in the Internet, reducing transfer time of both foreground and background transfers. To align deployment incentives, Harp’s architecture suggests background transfer as an edge-service that does not require support from end-hosts or routers. However, the Harp prototype implementation is not publicly available to the Internet research community.

In this paper, we report the implementation of a prototype following the original Harp design. We make our implementation package HIR (Harp in real) publicly available. We have identified the drawbacks of several Harp mechanisms and re-designed them for improving performance. We have validated Harp’s major properties using our implementation. We also provides insights of further improving major functionality the Harp background transfer architecture.

1. INTRODUCTION

Today’s Internet applications have diverse service requirements. Applications such as web browsing, instant messaging [5], and media streaming [8], are highly sensitive to delay. On the other hand, applications such as data backup [7], peer-to-peer swarming [2], massive replication [26], and web prefetching [21] are less sensitive to delay. We refer the former as foreground transfer and the latter as background transfer. However, current best-effort Internet does not distinguish between these two types of transfers. Thus, background transfers result in unwanted delay or data loss to more urgent transfers in the network.

Ideally we want background transfers not to interfere with foreground transfers, yet not to suffer from arbitrary delay. In other words, both foreground and background completion time should be short while fore-

ground transfers has strict priority over background.

This motivates Harp [20], a multipath background network architecture. Harp prioritizes foreground over background transfers to reduce active waiting time. To reduce completion time of both foreground and background transfer, Harp uses multiple paths for background transfer, dissipating background traffic to less-utilized links of the network. As a result, Harp provides better fairness and utilization compared with other unipath end-host protocol. Moreover, authors argue that Harp well aligns the incentive for deployment because it can be deployed an edge service without any modification to application, or support from routers or end-hosts.

The ideas of Harp background transfer worth of a “closer look”. The currently Harp implementation incorporates many thresholds and parameters in several major components, such as reordering delay, delay detection, congestion control, and fairness adaptation. There is no general guideline for setting these parameters. Neither do we know how those parameters would affect performance, nor do we know how environment dependent they are. This is very discouraging for network administrators when they want to deploy Harp, because they may have to hand-tune these parameters, even risking complicating the protocol.

To have deeper understanding of the Harp mechanisms, we decide to implement a Harp prototype following the original Harp design. We intend to validate the major properties of Harp through a suites of realistic experiments. Besides, we believe making the Harp prototype implementation available to the Internet research community would help research to address the above issues.

This paper reports the implementation and evaluation of a prototype of the Harp background transfer architecture. We publish our implementation in a software package named HIR [4] (Harp in real), which includes the source code of the elements we implemented using Click [19], and a set of click configurations that would help researchers to easy test and extend the im-

plementation.

Our contributions can be summarized as follows:

1. *We have implemented a prototype of the Harp background transfer architecture, and built a realistic emulation setting to validate key properties of Harp via a suite of experiments.*

We answer the following question: “How would Harp perform in realistic settings?”. One of the challenges we face achieving this goal was difficulty, if not inability, to reproduce results with Planetlab. We want to evaluate with more sophisticated and realistic settings than simulations yet to be able to control network conditions, such as bandwidth, link latency. Inspired by CBGP [3], a network emulation project that emulates the forwarding of real packets through a WAN topology, we choose to use emulation via Click. For our purpose, we emulate forwarding of real packets on overlay network, where the packet treatment and paths characteristics enforcement can be flexibly handled by Click.

2. *We have provided HIR [4], a publicly-available open-source implementation of Harp based on Click [19].*

Using HIR package, users with one machine can do emulation of multipath routing with overlay network. Also they can do testbed experiments with the same code (eg. on Planetlab). This would certainly contribute to future research on Harp and related projects, which could be either to improve the current functionality of Harp or to build more services based on Harp.

3. *We have identified drawbacks existed in several components of the original Harp design, and proposed fresh design to improve them.*

In our implementation and experiments, we found Harp’s delay indication mechanism is not effective in that it relies on the minimum/maximum delay of each relay path throughout a whole transfer. We proposed a mechanism that takes average within a fixed window size. We also found Harp’s loss detection mechanism is not able to detect loss quickly enough. We propose to use a batch from the entry gateway to help quickly detection loss. Section 7 discusses our findings in detail.

The rest of the paper is organized as follows. We survey related work in Section 2. We overview the architecture of Harp in Section 3. We discuss Harp implementation with Click in Section 4. We describe our emulation setting in Section 5 and present evaluation with our prototype implementation in Section 6, discuss engineering issues and future work in Section 7, finally we conclude in Section 8.

2. RELATED WORK

Harp spans a wide range of related fields : low priority transfer, multipath routing and congestion control, and overlay-based techniques.

Previously, various solutions to low priority transfer has been proposed. Diffserv [11] achieves this service differentiation by router support. While this is well defined in theory, it is not widely deployed in practice due to lack of both consensus and router support. There alternative approaches, TCP Nice [27], TCP-LP [22], 4CP [23], emulates low-priority without router support by modifying the congestion detection and avoidance in TCP. They react both earlier and more aggressively than standard loss-based TCP (Reno). End-to-end delays are used to trigger a congestion signal before loss occurs, a TCP Vegas style approach, while more aggressive back-off mechanisms than those of standard TCP are used to give a lower share of network resources. Although only senderside modification is required, this modification of TCP stack may still be difficult to deploy widely, due to lack of motivations for users to use, risking arbitrary delay.

Also there are application-level approach for background transfers that uses TCP as the underlying transport protocol. Two components are needed: to infer the available capacity and to adjust the sending rate of the background transfer accordingly. [17] tightly couple the available capacity inference and rate adjustment: the rate of the background transfer is controlled by adjusting the receiver-advertised window size, which enforces a limitation on the rate used by the application. In turn, the rate obtained for a given receiver window is used to infer whether that rate is above or below the available capacity, which in turn triggers an adaptation of the receiver window. This is based on the fact that the actual TCP sending window is the minimum of the receiver window and the congestion window. Harp also incorporates this idea in entry gateway’s sender rate control. Alternative approach is coarse-grain scheduling running backup and data prefetching during off-peak hours.

A great amount of work has been done in multipath routing. Multipath routing is used to improve throughput [10] load balancing [13], resilience to path failures or packet losses [18]. Recent work([24] [9]) proposes the use of multipath routing in overlay network. More relevant work to Harp would be [16] and [12], where they propose splitting aggregate traffic flows among multiple paths to achieve load balancing and stability for intradomain traffic engineering.

Other papers proposes using overlay networks for quality of services. Edge-to-edge congestion control [14] supports a limited range of bandwidth services using an overlay framework. Also OverQoS [25] is proposed

Component	Function Descriptions
Sender	Send packets. Measure the throughput
Receiver	Receive packets. Measure the goodput.
Entry gateway	Identify the path and exit gateway associated with the receiver. Harp encapsulation (Section 4.1) Path selection (Section 4.2). Sender rate control: (Section 4.2).
Relay gateway	Identify the exit gateway associated with the receiver. Harp Encapsulation
Exit gateway	Identify the entry gateway associated with the sender. Congestion and loss indication (Section 4.3) Packet reordering (Section 4.4)

Table 1: Key functions of Harp components

to provide limited noninterference to priority traffic.

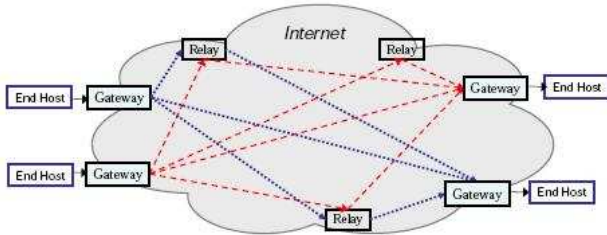


Figure 1: Harp's basic architecture with entry/exit gateway and relays

3. HARP ARCHITECTURE OVERVIEW

First, we introduce a brief overview of Harp architecture. Harp forms an overlay network which consists of the entry gateway node, relay nodes, and the exit gateway node as illustrated in Figure 1 and 2. Packets from the end-host enter the entry gateway and forwarded to exit gateway via one relay node. The entry gateway, interactively with exit gateway, performs multipath routing and forwards packets through less congested path. The exit gateway performs packet reordering and deliver them to the receiver at the end. TCP Acknowledgement generated by the receiver for all packets are sent back directly from the exit to the entry gateway. Table 1 summarizes key functions of each component.

4. IMPLEMENTATION

We implement harp in Click [19]. Our implementation include five elements, *harpencap*, *entrycontroller*, *exitcontroller*, *packetreorder*, which consist of over 3K lines of codes. Figure 3 illustrates click elements for Harp. For emulation purpose, we used *KernelTun* to tunnel connection between end-host to gateways which are physically in one machine. Also we used *Queue* and *BandwithRatedUnqueue* for emulating link for each path. We now discuss each element we newly implement.

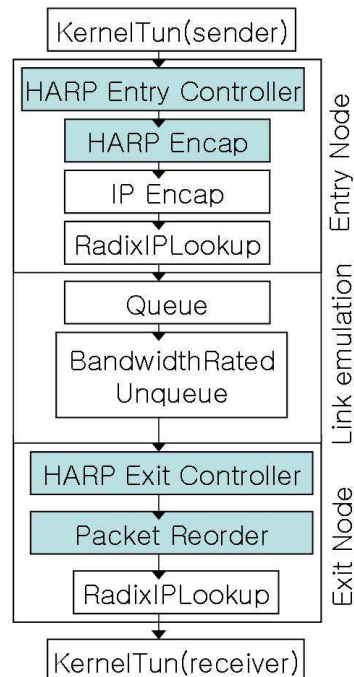


Figure 3: Click elements for Harp: Newly implemented elements are in colored box

4.1 Harp Header and Encapsulation

We design the harp header such that it includes three pieces of information: packet type, timestamp, and path identifier. These information in a packet header are essential for the exit gateway to determine path congestion and then signal to the entry gateway. Packet type indicates a packet is a regular data packet or control packet. Control packet are important because they convey information of path latency/congestion. By checking timestamp on a packet, the exit gateway is able to calculate the latency that packet has experienced. The path identifier tells the exit gateway the path it was going through.

In realization, we use 4 bits for the field of packet type, though currently we only have two types of pack-

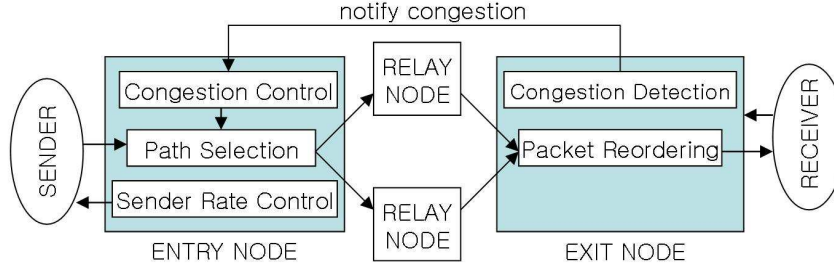


Figure 2: Harp architecture

```

elementclass HARPEnty {
    $pkt_type, $path_id |
    input -> HARPEncap($pkt_type, $path_id)
           -> SetTimestamp
           -> output;
}

elementclass HARPExit{
    input -> StripHARPHeader
           -> output;
}

```

Figure 4: The element classes for HARP header design. A compound element HARPEnty combines HARPEncap and SetTimestamp together. HARPEncap sets packet type and path identifier. SetTimestamp sets timestamp.

ets: data packet and control packet. This ensures extensibility in case that more types of packets get added during future development. In the field of path identifier, we design it to store path ID instead of IP address. A 4-bit path identifier is enough if the number of different relays is less than 16. It is easy to map a path ID to a specific path at the exit gateway. If we use the IP address of a relay node as path identifier, we need to use 32 bits. We do not explicitly allocate a space in the harp header for storing timestamp, because the click has elements that operate on timestamp for packets. We create a compound element class HARPEnty that leverages the SetTimestamp element in click to append our harp header to packets.

Figure 4 presents the definition of compound element class HARPEnty. It combines an element HARPEncap, which set packet type and path identifier, with the element SetTimestamp to set the harp header at the entry gateway. At the exit gateway, element HARPExit is used to strip off the harp header before sending out a packet to the destination.

4.2 Harp Entry Controller

Harp entry controller performs congestion control, path selection, and sender rate control.

Congestion Control

Harp congestion window is maintained per path (not per flow) for congestion control. Harp congestion control uses multiplicative increase and multiplicative decrease scheme. The pseudo code for congestion control is as follows:

1: Definitions

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

The acknowledgement is sent from the receiver and the delay signal and loss signal are sent from the exit controller. By changing parameter ξ , we can instantiate three schemes: *independent* and *joint*.

1. *independent*: If ξ is equal to 1, then each subflow on a path operates as an individual TCP flow. Independent can be unfair to transfers that have fewer paths.
2. *joint*: If ξ is equal to 0, then each multipath connection behaves as a single TCP flow on its best set of paths. Joint can lead to under-utilization.

We conduct experiment comparing between Independent and Joint in Section 6.

Path Selection When a packet arrives from the sender, entry gateway performs load-balancing across the available paths by choosing for each packet with minimum value of $bytes_in_nw_i/cw_i$, where $bytes_in_nw_i$ is the number of unacknowledged bytes sent on path p_i , and cw_i is the congestion window of path p_i .

Sender Rate Control Harp is unaware of congestion control algorithm running at sender and hence it can lead to mismatch between congestion window at entry gateway and sender. This mismatch can lead to packet losses that reduce the window on the sender, hence reducing the overall throughput achieved by background transfer. The way to overcome this problem can be to ensure that the sender doesn't send more bytes than what the entry gateway's congestion window can allow across all paths.

In order to resolve this issue, entry gateway rewrites the TCP Header in the acknowledgements 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. It is implemented by modifying TCPRewriter element.

Issues to investigate In entry gateway, most important thing would be congestion control. Overreaction to delay or loss will lead to under-utilization. Previous Harp paper suggests the congestion window be decremented only once per round-trip time. Would this be sufficient? We believe that setting α and β threshold correctly be even more important for good performance. We tried to set different α and β values in our experiments but we did not find a systematic way to set those values. We end up using the parameters suggested in the Harp paper.

4.3 Harp Exit Controller

Harp exit controller performs congestion notification upon delay and upon losses.

Delay detection Exit gateway maintains packet delay per path. Let $dmin_i$ and $dmax_i$ denote the minimum and maximum delay observed at path p_i , and let dc_i denote current packet's delay on path p_i . If dc_i is greater than $dmin_i + (dmax_i - dmin_i) \times \delta$, where δ is a delay threshold, then exit controller sends delay notification to entry gateway, which will reduce cw_i , congestion window of path p_i .

Loss detection Exit gateway maintains $last_byte_rcvd_i$, the highest byte received on the path p_i and $rcvnext_f$, the next byte expected in sequence for flow f . If $rcvnext_f$ is smaller than minimum of $last_byte_rcvd_i$ for all path p_i , then the loss is detected and send the loss notification to entry gateway. The loss notification packet conveys $rcvnext_f$ and the sequence number of the packet at the head of the reorder queue. Upon receiving the notification, the entry gateway determines

which paths the lost have occurred. For each path that experience loss, the congestion window is reduced.

Issues to investigate There are few engineering issues to investigate for exit controller element. First, how to set δ would manual selection work? Second, should we apply strict $dmin_i$ and $dmax_i$ for entire transfer time, or update $dmin_i$ and $dmax_i$ for recent n packets? Third, is there better way to do loss detection? The loss detection is done via exit gateway without any information from entry gateway. Current detection tend to penalize the same path redundantly. Consider the following scenario. Exit gateway first report loss with sequence number 1 through 4, informing entry gateway that packets with sequence number 2 to 4 are lost. Exit gateway now receives a packet with sequence number 4. Then exit gateway sends loss indication with sequence number 1 through 3. Then paths that the packet with sequence number 2 will get duplicate penalties in this case. One solution to this would be suppressing loss indication if loss indication of beginning of the sequence number is identical. Moreover, to determine paths that experience losses more accurately, we can modify entry controller to give out path selection information to exit controller. In this way, exit gateway knows on which path that the packet is lost and send loss notification for that path.

4.4 Packet Reordering

Packets reach to exit gateway through various relay node and hence they can be out of order. If exit gateway route the packets to receiver in the same order in which it receives, then it will cause receiver to generate duplicate acknowledgements to the sender. Such duplicate acknowledgements falsely indicate packet loss to sender causing significant reduction in congestion window and hence reduction in throughput. In order to avoid this problem exit gateway do reordering of incoming packets.

After receiving a packet, exit gateway checks the sequence number of TCP packet. If it is in order then it forward the packet to receiver immediately otherwise store it into reorder queue. Packet is removed from reorder queue and forwarded to sender when either all missing packets with lower sequence number arrive or timer expires. The value of timer is a factor ρ of minimum delay over longest path.

Key Functions There are two key functions we implement: *push* and *run_timer*. The related data structure is summarized in Table 2. Let us denote *curr_seq_num* as sequence number of currently arrived packet and *exp_seq_num* as the sequence number of expected packet.

Upon a packet arrival, *push* extracts *flowId*, *pathId* and transmission time from the packet. Then it calculates delay of current packet and update the mini-

Data structure	Descriptions
<i>ReorderQueueMap</i>	-Stores per flow reorder queue.
<i>ExpectedSeqNumMap</i>	-Stores next expected sequence number for each flow
<i>DelayMap</i>	-Stores minimum delay for each path

Table 2: Data structure for Packet Reorder

imum delay of that relay path from which packet has arrived. Then sequence number of packet is checked with expected sequence number and depending upon following results. (i) If $curr_seq_num < exp_seq_num$, then it represent that packet is retransmitted packet and hence it is forwarded without any other action. (ii) If $curr_seq_num = exp_seq_num$ then the packet is forwarded and expected sequence number is updated. Then it is checked that if packet with expected sequence number is waiting in reorder queue. In this case, the packet is forwarded and the current expected sequence number is updated. This step is repeated until there is no packet waiting in reorder queue with the current expected sequence number. (iii) If $curr_seq_num > exp_seq_num$, then the packet is stored in reorder queue. A timer object is created, logically attached with packet and scheduled.

If timer expires then *run_timer* function is called. This remove corresponding packet from reorder queue and forward it.

Design decision Packet reordering is a function of exit gateway. However, we decide to make a separate element for packet reordering to simplify code structure, and to achieve higher flexibility in testing. Separating out packet reordering we are able to test the protocol without packet reordering easily.

5. EMULATION SETUP

We use emulation to evaluate the performance of our HIR implementation. Emulation is more realistic than simulation in that emulation uses real packets, while providing flexibility to control network conditions essential for our purpose of validation of the protocol. We build a emulation setting which uses the minimum amount of hardware resource to route real packets. The emulation setting is inspired by the CBGP [3] project and some interesting lab work based on Click [19] at NYU [6].

The CBGP [3] emulation setting involves three computers. It leverages Click [19] to configure AS environments within a single computer, and the other two computers are used as traffic source and destination, respectively. Our emulation setting requires less re-

source, because it only involves a single computer.

Firstly, we setup fake network interfaces to avoid the needs of separate traffic source and destination nodes. This is achieved by using the *KernelTun* elements of Click. As illustrated in Figure 3, the *KernelTun* elements set up two network interfaces, and traffic client and servers are established on the two interfaces.

Secondly, we need to work around the kernel routing table. By default, the kernel establishes routing table for packets transmitting between the two faked network interfaces. To let packets route through the click configuration without going through the kernel, we use the *IPRewriter* elements to achieve this purpose. We rewrite the destination of packets coming from a traffic source, such that the the default kernel routing table will match the destination address to the interfaces it comes from. We play the same trick to ACK packets coming from the destination. This technique is inspired by a lab work of a distributed system course at NYU [6].

After we set up the emulation, we can freely establish real traffic clients and servers at the fake interfaces and transmit real packets.

Comparing with the CBGP [3] emulation setting, we achieve the same goal but with less resource. In addition, we gain manageability by having a more controllable emulation environment. Because of it’s minimum resource requirement and easy manageability, we believe this emulation framework will be widely adopted in the future.

6. EVALUATION

This section presents our emulation results. We conduct three different suites of experiments to investigate HARP’s key properties. As we mentioned above, emulation ensures realistic packet transmission at different network layers and provides flexibility to control network conditions essential for our purpose of validation of the protocol. To emulate the different paths between entry gateway and exit gateway, we use *Queue* and *BandwidthRatedUnqueue*, which are built-in click elements. For traffic generation of both foreground traffic and background traffic, we use the Iperf [15] traffic generator.

Throughout our experiments, we use the parameter values reported in the Harp paper unless stated otherwise. We do not use the ρ value in our implementation, because our packet reorder element implemented a fresh mechanism (see Section 7). We set δ to be 0.3 for delay indication. This number just happens to be “good” in our experiments.

6.1 Exp1: Evaluating the benefit of multi-path

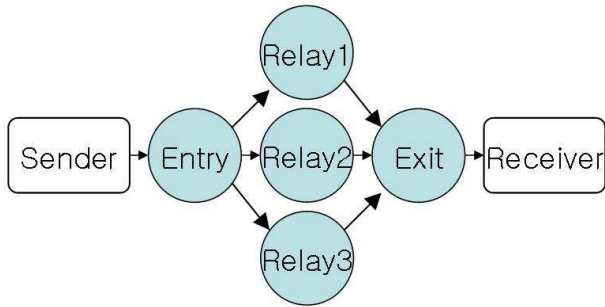


Figure 5: Topology used for Exp1

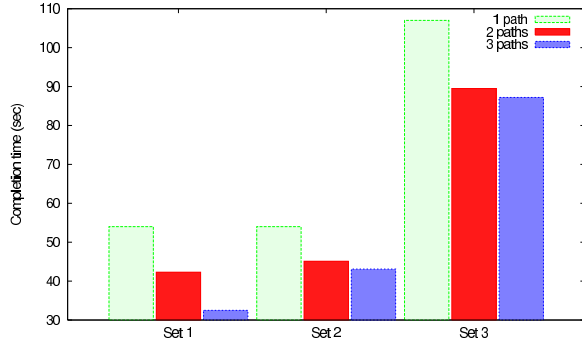


Figure 6: Completion time of background transfers with multiple paths.

In this experiment, we investigate how multiple relay paths would help improve the performance of background transfer. We build an emulation topology where three relay paths exit between the entry gateway and exit gateway. Figure 5 shows the topology.

We have three sets of configurations for this experiment. In each set, we vary the bandwidth and file size. Table 3 shows details of the three sets. Note that for each set, we also vary the number of used relay paths from 1 to 3.

Figure 6 presents the result of this experiment. We run each set of experiments for 3 times and report the average. The performance metric is the time that takes to complete file transfer.

We do observe that our implementation can utilize extra bandwidth when multiple paths repeated. In the first set, where path bandwidth is 200KB/sec, the performance gain increases as we increase the num-

	File size	Path bandwidth
Set 1	10 MB	200KB/sec
Set 2	25 MB	500KB/sec
Set 3	50 MB	500KB/sec

Table 3: Path sets for exp1.

ber of relay paths. This observation corresponds to the results of Figure 8(b) in the original HARP paper. However, as we increase the bandwidth to 500KB/sec, and with files of size increased to 25MB and 50MB, performance gain does not change much from 2-path to 3-path, and both are better than the one-path scenario.

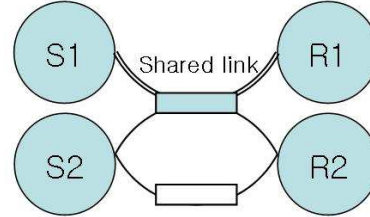


Figure 7: Topology used for Exp2

File size and number of relay paths are the only two reasons that could affect performance. Increase in file size could affect performance, because as file size increases, transmission time grows, and more packets are likely to be out of order. If a packet cannot be reordered during timeout interval, it would be forwarded to the receiver. After seeing an out-of-order packet, the receiver detects loss of packet and sends packet loss notification to the sender, which in turn reduces its congestion window. The performance then would drop, since in TCP the achieved throughput relies upon congestion window. Increase in number of paths could significantly improve throughput, as more packets can be transmitted through more paths. However, as number of paths increases, the load of packet reordering becomes heavy simply due to the increased probability of packet out-of-order. Our-of-order packets could cause reduction in sender window size and hence reduction in throughput. We can not gain unlimited improvement by increasing number of paths infinitely.

We have run the our experiments with file smaller than 10 MB, and we get better performance improvement than that shows in Figure 6. But for larger file size 75MB and 100MB, we don't see obvious improvement by varying number of paths from 2 to 3. To this end, we suspect the our packet reordering elements could not interact gracefully with large number of relay paths.

6.2 Exp2: Evaluating the fairness of resource allocation

As indicated in the previous HARP paper, varying the value of ξ can instantiate different types of multipath congestion control, namely *independent*, and *joint*. (Section 4.2). To validate this claim we con-

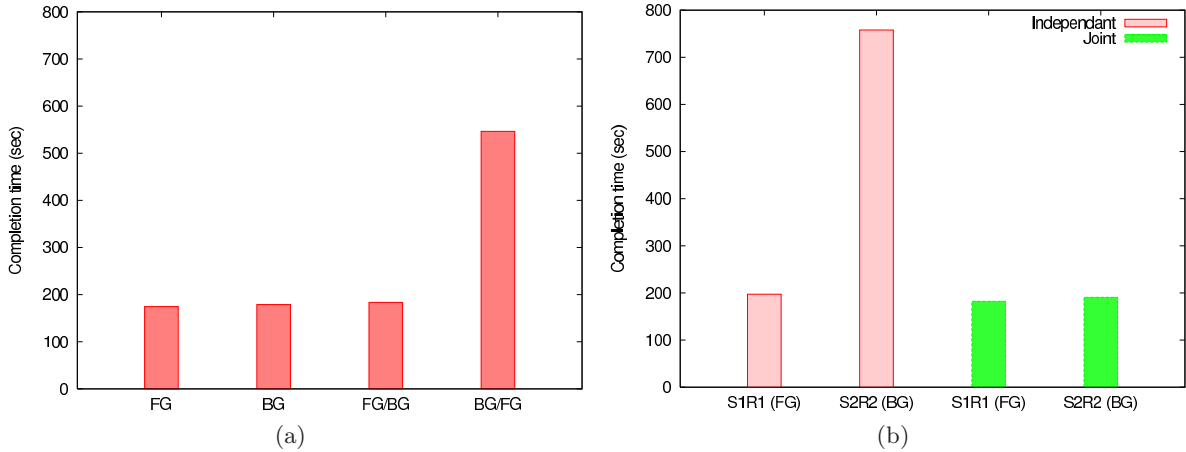


Figure 8: (a) Non-interference: foreground traffic never gets interfered by background traffic; (b) Fairness: joint congestion control achieves better fairness.

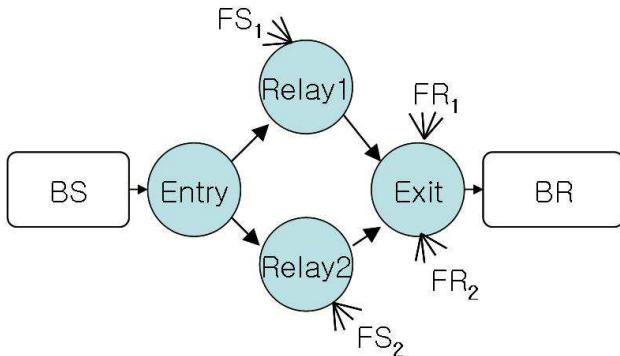


Figure 9: Topology used for Exp3

duct experiments that uses shared link between two background flows. Figure 7 illustrates the emulation topology we build. In the topology, S1R1 and S2R2 share one relay path, and S2R2 has another extra relay path. Both relay paths have bandwidth of 100KB/sec.

Figure 8(a) shows the non-interference property of HARP congestion control. We observe that when foreground and background traffic co-exists in the network, the foreground traffic is not affected by the background traffic. Whenever there is congestion, the background traffic reacts more quickly to reduce its congestion window, and thus the congestion only affects background traffic.

Figure 8(b) shows that using *joint* congestion control achieves better fairness than the *independent* congestion control. This is obviously due to the fairness-friendly property of the joint scheme. Besides, we observed a lot of out-of-order packet loss when using the independent scheme. This indicates that lacking of

fairness consideration in a multiple path setting would significantly worsen TCP performance.

6.3 Exp3: Evaluating with multiple foreground flows

This subsection presents evaluation on HARP’s non-interference property and the ability of utilizing extra network bandwidth, by increasing foreground network traffic.

Figure 9 illustrates the emulation topology we build. We have a background traffic that transmits a 50MB file from a background sender (BS) to a background receiver (BR). There are two relay paths between the entry gateway and exit gateway. Both paths have bandwidth of 1.25MB/sec (10Mbits/sec). We attach a number of foreground traffic senders (FS) to relay nodes, and attach foreground traffic receivers (FR) to the exit gateway.

We first run the experiments to compare our HARP implementation with Iperf [15] TCP implementation under a scenario where only a single relay path is presented. We vary the number of foreground traffic, each of which involves a FR process requesting a 20KB file from a FS. Those foreground FR processes are evenly distributed throughout the whole life time of the background transfer.

Our experimental results are shown in Figure 10. The figure shows that HARP is slightly better than the Iperf TCP when a single relay path is available, and it is about 2x better when 2 relay paths are available. This again shows that HARP can utilize extra bandwidth to improve performance. Though the out-performance over Iperf is modest, we believe by refining our implementation, we can achieve better improvement.

7. DISCUSSIONS AND FUTURE WORK

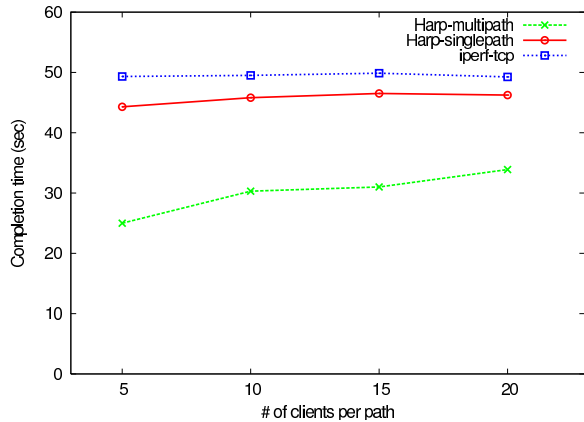


Figure 10: Completion time of a background transfer with multiple foreground traffic.

In this section we discuss the issues we have observed during the HIR implementation and experiments. We propose several fresh design to improve the functionality of Harp’s major components. We plan to complete implementation of our proposals in the near future and test them in a real network.

7.1 Delay Indication Mechanism

The first issue is about characterizing path bandwidth by delay calculated based on timestamp. We record the minimum and maximum delays for each path throughout a whole background transfer. Based on the record, we judge whether an incoming packet gets delayed on a particular path. This turns out not accurate.

We propose to either use exponential weighted average or take average within a fixed window size (For example, take average delay observed over the recent 50 packets).

7.2 Timer for Packet Reordering

The second issue is about packet reordering elements. We note that timer initialization value when a packet comes out of order is extremely important parameter for performance gain. The previous the HARP paper doesn’t address this issue well - that timer is initialized with constant factor of minimum delay on longest path and investigated optimal value of constant factor by simulation.

Using the minimum delay of the most delayed path is not effective. The following problems come up from our experimentation:

- (i) Transmission time on longest path with smaller delay can be lesser than transmission time on smaller path with higher delay.
- (ii) Minimum delay might not reflect the current

load. Average delay of last few transmission might reflect congestion of network more than minimum delay.

We propose to use maximum average delay of all paths (for recent received packets, for example, in a fixed window size) to use instead of minimum delay over the most delayed path. But since maximum average delay might grow continuously in some case, hence a maximum threshold value can be set for that. This might provide better performance than the current delay parameter. We actually implemented this mechanism for our experiments.

7.3 The Loss Detection Mechanism

We found the the packet loss detection mechanism in Harp is not able to identify loss quickly. This mechanism needs to wait till packets received on all paths are greater than the expected one. This might take a long time if the path quality varies significantly.

We propose to use a batch map at the entry gateway. The batch map records down a set of packets that are sent by the entry gateway. Entry gateway include the batch map in packet header, and the exit gateway then can identify loss quickly based on the batch map, instead of waiting on each path.

8. CONCLUSION

In this paper, we report the implementation and evaluation of a prototype following the design a background transfer architecture. We make our implementation package HIR [4] (Harp in the real) publicly available.

We have built a emulation framework which requires minimum of hardware resource and brings manageability to users. We have validated Harp’s major properties using our implementation.

We have identified the drawbacks of several Harp mechanisms and re-designed them for improving performance. We also provides insights of further improving major functionality the Harp background transfer architecture.

9. REFERENCES

- [1] Advanced solutions for p2p networks home page. <http://cachelogic.com/research/p2p2005.php>.
- [2] Bittorrent. <http://www.bittorrent.com/>.
- [3] C-bgp to click converter. <http://cbgp.info.ucl.ac.be/projects/cbcp2click.php/>.
- [4] Harp in the real. <http://www.cs.utexas.edu/~zfv/hir/>.
- [5] Microsoft windows live messenger. <http://im.live.com/messenger/>.
- [6] Network and distributed systems. <http://www.news.cs.nyu.edu/~jinyang/fa06/>.
- [7] Remote backup systems. <http://remote-backup.com/rbackup/>.
- [8] You tube. <http://youtube.com/>.
- [9] A. Akella, J. Pang, B. M. Maggs, S. Seshan, and A. Shaikh. A comparison of overlay routing and

- multihoming route control. In R. Yavatkar, E. W. Zegura, and J. Rexford, editors, *SIGCOMM*, pages 93–106. ACM, 2004.
- [10] D. G. Andersen, A. C. Snoeren, and H. Balakrishnan. Best-path vs. multi-path overlay routing. In *Proceedings of the 2003 ACM SIGCOMM conference on Internet measurement (IMC-03)*, pages 91–100, New York, Oct. 27–29 2003. ACM Press.
- [11] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Service. RFC 2475 (Informational), Dec. 1998. Updated by RFC 3260.
- [12] A. Elwalid, C. Jin, S. Low, and I. Widjaja. MATE: MPLS adaptive traffic engineering. pages 1300–1309.
- [13] Y. Ganjali and A. Keshavarzian. Load balancing in ad hoc networks: Single-path routing vs. multi-path routing. In *INFOCOM*, 2004.
- [14] D. Harrison, S. Kalyanaraman, and S. Ramakrishnan. Overlay bandwidth services: Basic framework and an edge-to-edge closed-loop building block, Feb. 08 2001.
- [15] Iperf, the tcp/udp bandwidth measurement tool. <http://dast.nlanr.net/Projects/Iperf>.
- [16] S. Kandula, D. Katabi, B. S. Davie, and A. Charny. Walking the tightrope: responsive yet stable traffic engineering. In R. Guérin, R. Govindan, and G. Minshall, editors, *SIGCOMM*, pages 253–264. ACM, 2005.
- [17] P. B. Key, L. Massoulié, and B. Wang. Emulating low-priority transport at the application layer: a background transfer service. In E. G. C. Jr., Z. Liu, and A. Merchant, editors, *SIGMETRICS*, pages 118–129. ACM, 2004.
- [18] K.-H. Kim and K. G. Shin. Improving TCP performance over wireless networks with collaborative multi-homed mobile hosts. In K. G. Shin, D. Kotz, and B. D. Noble, editors, *MobiSys*, pages 107–120. ACM, 2005.
- [19] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.
- [20] R. Kokku, A. Bohra, S. Ganguly, and A. Venkataramani. A multipath background network architecture. In *INFOCOM*, pages 1352–1360. IEEE, 2007.
- [21] R. Kokku, P. Yalagandula, A. Venkataramani, and M. Dahlin. NPS: A non-interfering deployable web prefetching system. In *USENIX Symposium on Internet Technologies and Systems*, 2003.
- [22] A. Kuzmanovic and E. W. Knightly. TCP-LP: A distributed algorithm for low priority data transfer. In *INFOCOM*, 2003.
- [23] S. Liu, M. Vojnović, and D. Gunawardena. Competitive and considerate congestion control for bulk-data transfers. Technical Report MSR-TR-2006-24, Microsoft Research (MSR), Feb. 2006.
- [24] A. Sen, B. Hao, B. H. Shen, S. Murthy, and S. Ganguly. On multipath routing with transit hubs. In R. Boutaba, K. C. Almeroth, R. Puigjaner, S. X. Shen, and J. P. Black, editors, *NETWORKING*, volume 3462 of *Lecture Notes in Computer Science*, pages 1043–1055. Springer, 2005.
- [25] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz. OverqoS: offering internet qoS using overlays. *Computer Communication Review*, 33(1):11–16, 2003.
- [26] A. Venkataramani, R. Kokku, and M. Dahlin. Operating system support for massive replication, July 03 2002.
- [27] A. Venkataramani, R. Kokku, and M. Dahlin. TCP nice: A mechanism for background transfers. In *OSDI*, 2002.