

mPath: High-Bandwidth Data Transfers with Massively-Multipath Source Routing

Yin Xu, Ben Leong, Daryl Seah, and Ali Razeen

Abstract—The capacity of access links has increased dramatically in recent times, and bottlenecks are moving deeper into the Internet core. When bottlenecks occur in a core (or AS-AS peering) link, it is possible to use additional detour paths to improve the end-to-end throughput between a pair of source and destination nodes. We propose and evaluate a new *massively-multipath (mPath) source routing* algorithm to improve end-to-end throughput for high-volume data transfers. We demonstrate that our algorithm is practical by implementing a system that employs a set of proxies to establish one-hop detour paths between the source and destination nodes. Our algorithm can fully utilize the available access link bandwidth when good proxied paths are available, without sacrificing TCP-friendliness, and achieves throughput comparable to TCP when such paths cannot be found. For 40% of our test cases on PlanetLab, mPath achieved significant improvements in throughput. Among these, 50% achieved a throughput of more than twice that of TCP.

Index Terms—Multipath TCP, source routing, congestion control.

1 INTRODUCTION

RESEARCH has shown that there are often less-congested paths than the direct one between two end-hosts over the Internet [1, 2]. These alternative paths through the Internet core were initially not exploitable as bandwidth bottlenecks used to be in the “last mile”. Over the past decade, bottlenecks have been observed to be shifting away from the network edges due to the growing capacity of access links [3]. As last mile bandwidth is set to increase dramatically over the next few years [4], we expect that this trend will accelerate and end-to-end data transfers will be increasingly constrained by core link bottlenecks. We now have the opportunity to exploit path diversity and use multiple paths concurrently to fully saturate the available access link bandwidth for high-volume data transfers, e.g. scientific applications [5] or inter-datacenter bulk transfers [6].

While the idea of multipath routing is not new, previously proposed systems either require multi-homing support [7] or the maintenance of an overlay with only a small number of paths [8]. Our approach is to use a large set of geographically-distributed proxies to construct and utilize up to hundreds of detour paths [1] between two arbitrary end-hosts. By adopting one-hop source routing [9] and designing the proxies to be stateless, we also require significantly less coordination and control than previous systems [8, 10] and ensure that our system would be resilient to proxy failures. Our system, which we call *mPath* (or *massively-multipath source routing*), is illustrated in Fig. 1.

There are a number of challenges in designing such a system: (i) good alternative paths may not always exist,

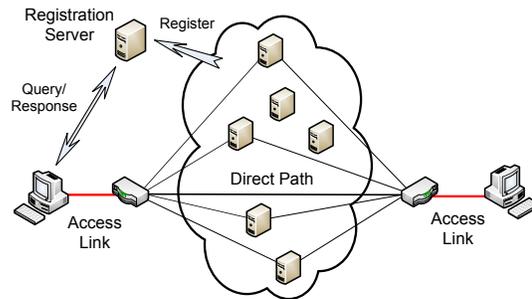


Fig. 1. Massively-multipath source routing.

and in such cases the performance should be no worse than a direct TCP connection; (ii) when good alternative paths do exist, we need to be able to efficiently identify them and to determine the proportion of traffic to send on each path; and (iii) Internet traffic patterns are dynamic and unpredictable, so we need to adapt to changing path conditions rapidly. Our key contribution, which addresses these design challenges, is a combined congestion control and path selection algorithm that can identify bottlenecks, apportion traffic appropriately, and inter-operate with existing TCP flows in a TCP-friendly manner. The algorithm is a variant of the classic additive increase/multiplicative decrease (AIMD) algorithm [11] that infers shared bottlenecks from correlated packet losses and uses an operation called *load aggregation* to maximize the utilization of the direct path.

We model and analyze the performance of mPath to show that our algorithm (i) is TCP-friendly, (ii) will maximize the utilization of the access link without under-utilizing the direct path when there is free core link capacity, and (iii) will rapidly eliminate any redundant proxied paths. We validate our model with experiments on Emulab.

We evaluate our system on PlanetLab with a set of 450 proxies to show that our algorithm is practical and can achieve significant improvements in throughput over

- Yin Xu, Ben Leong and Daryl Seah are with the Department of Computer Science, School of Computing at the National University of Singapore.
- Ali Razeen is with the Department of Computer Science at Duke University.

TCP for some 40% of the end-hosts. Among these, half of them achieved more than twice the throughput of TCP. In addition, when good proxied paths cannot be found or the bottleneck is at a common access link, mPath achieves throughput that is comparable to TCP and stabilizes in approximately the same time.

The rest of this paper is organized as follows: in Section 2, we describe the mPath algorithm and the design of our system. In Section 3, we present a theoretical model for our multipath congestion control algorithm. In Section 4, we present our evaluation results on Emulab and PlanetLab. Finally, we present an overview of related work in Section 5 and conclude in Section 6.

2 SYSTEM DESIGN

In this section, we describe the design and implementation of the mPath source routing system. As illustrated in Fig. 1, the network is composed of a set of proxies that are tracked by a central registration server (RS). Proxies in mPath are light-weight because they do not maintain connection state. The destination address is embedded in every data packet, so proxies can simply forward the packets received to the destination node. The RS tracks the active proxies in the system and returns a subset of the proxies to a source node when it needs to initiate a new mPath connection. We currently implement the RS as a simple server application, but it can be easily replaced with a distributed system for greater reliability and/or scalability. The application at the end-hosts is provided with a connection-based stream-like interface similar to TCP to perform the data transfer, even though the underlying protocols supporting this interface are UDP-based and therefore connectionless. Depending on the nature of the application supported, the proxies can either be dedicated servers or mPath clients.

We use UDP instead of TCP for various practical reasons. For one, mPath needs direct control over the packet transmissions (and retransmissions) to implement the congestion control algorithm that coordinates between the different mPath flows. Moreover, the use of TCP would limit the scalability of the system since a source node might need to communicate with hundreds of proxies and the overhead of opening and maintaining hundreds of TCP connections is excessive. Given that the majority of hosts on the Internet are behind Network Address Translators (NATs), it is also advantageous to use UDP because the NAT hole punching process for UDP is typically simpler, faster and more likely to succeed than that for TCP [12].

A data transfer begins when the source node establishes a direct connection to the destination. Simultaneously, the source node also queries the RS to obtain a list of available proxies. The data stream from the application is packetized and the packets are initially sent only on the direct path. When congestion is detected on the direct path, packets are forwarded via the proxies in an attempt to increase the throughput.

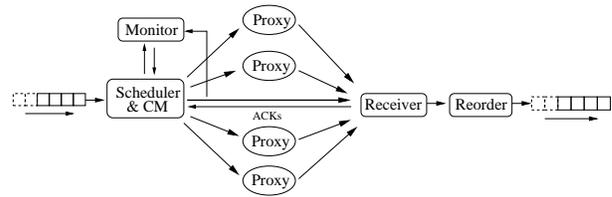


Fig. 2. Overview of mPath.

Acknowledgments for the received data packets are sent from the destination back to the source along the direct path. A congestion manager and scheduler module monitors the acknowledgments to determine the quality of the various paths and controls the transmission and retransmission of packets. Finally, packets are reordered at the destination to produce the original data stream. This process is illustrated in Fig. 2.

In general, the congestion control on the direct path is similar to TCP. Modifications to the standard TCP AIMD algorithm were made to coordinate between the multiple paths and ensure that the combined paths do not behave more aggressively than TCP in increasing the overall congestion window. Also, we implemented a simple algorithm to infer correlated losses between the direct path and proxied paths, and a *load aggregation* mechanism to aggregate traffic onto the direct path when a shared bottleneck is detected. Our algorithm causes the traffic on redundant proxied paths to converge to zero over time. While the overall idea is relatively simple, there are a number of implementation details required to get the system to work in a practical setting. These details are described in the following sub-sections.

2.1 Proxy Probing

Given the large number of proxies, each mPath connection starts with a probing phase that uses data packets to identify proxies that are unreachable, non-operational or exhibit non-transitive connectivity [13]. As mPath is tolerant of packet reordering and losses, it is acceptable to use data packets in the probing process instead of active probe packets.

Probing starts immediately after the source establishes a direct connection to the destination and receives a proxy list from the RS. To prevent path probing from interfering with the data transfer process, we limit the probing rate to one probe every 250 ms, which is approximately the average inter-continental roundtrip time [14]. Sending one probe packet every RTT will not likely interfere with the data transfer because the sender is expected to forward tens or hundreds of packets in one RTT. When the sender decides to probe a proxy, it will randomly select a proxy from the proxy list and attempt to forward a data packet through it to the destination. If the sender receives an ACK for the data packet within τ seconds, the proxy is considered usable and is added to the *available list*, which is the set of proxies that can be used to forward packets. On the other hand, if the sender fails to receive an ACK within τ seconds after two

consecutive probing attempts, the proxy will be marked as *unusable*. Once all the proxies in the list have been probed, mPath will request for more proxies from the RS. Clearly, the threshold τ limits the maximum RTT of the proxied paths in the system and controls the tradeoff between the quality of the paths selected and the size of the buffer required at the destination to handle reordering. We show in Section 4.6 that a value of τ that is two times of the direct path's RTT yields a sufficiently large number of good proxied paths and an acceptable amount of reordering.

2.2 Sequence & Acknowledgment

TCP was designed for a single direct path and only needs one sequence number to handle both ordering and the detection of packet loss. In mPath, packet transmission across multiple paths can result in significant reordering at the receiver. A single sequence number would suffice to preserve ordering. However, having only one sequence number would make it harder to detect packet losses for individual paths, which is needed for proper congestion control. We considered using SACK and the scoreboard data structure proposed in mTCP [8] to record information for all the paths. However, we found this method to be inefficient in handling hundreds of paths. Like MPTCP [7], we use two sequence numbers: a *stream sequence number* and a *path sequence number*. The stream sequence number is used to identify and retransmit lost packets, while the path sequence number is used to detect packet loss and control the congestion window for each path.

Acknowledgments. When the destination successfully receives a number of packets, an ACK packet is sent back to the sender, which contains both the global stream sequence number as well as a set of path entries for the paths on which the receiver had received data packets. Like TCP, the receiver cumulatively acknowledges the receipt of packets by sending back the stream sequence number of the earliest missing packet. In addition, each ACK packet also contains a set of path entries, each recording the largest sequence number seen and the accumulated count of the packet losses observed on the associated path. The path-level acknowledgment is based on the latest packet received rather than the earliest missing packet because we have decoupled stream ordering from path packet losses and we can use new sequence numbers for retransmissions. This also allows mPath to retransmit lost packets on a different path. The accumulated packet loss count is included to ensure that the sender has a more accurate view of the packet losses on each path. Given that there are occasional losses of ACK packets, this allows the congestion control algorithm to recover in the event that it wrongly infers that there are data packet losses.

Negative Acknowledgments. The sender is notified of the holes in the stream sequence with a stream-level NACK (SNACK) packet and of holes in the path sequence with a path-level NACK (PNACK) packet. As

holes in the stream sequence can be the result of reordering across multiple paths, SNACKs are not sent immediately when the holes are detected. Since only paths with RTTs less than τ seconds are used as proxies, we wait up to τ seconds for holes in the stream sequence to be filled before sending a SNACK to avoid false positives. To prevent an overflow of the receiver's buffer arising from the delayed retransmission requests, SNACKs will be sent immediately if the receiver's buffer is more than half full. When the sender receives a SNACK, it retransmits the required packets immediately but does not modify the congestion window. Unlike SNACKs, PNACKs are sent as soon as packet losses are detected. The sender reacts to a PNACK according to the congestion control algorithm and performs a quick retransmission with a newly selected path.

ACK Aggregation. mPath sends ACKs, SNACKs and PNACKs via the direct return path. We found that it is quite common for ACKs to be lost when there is congestion on the return path. If ACKs were sent for every packet, mPath might experience greater ACK losses than TCP because the number of ACK packets for mPath can exceed the number of data packets sent on the direct path if a large number of data packets are sent along proxied paths. To reduce congestion, we reduce the rate at which ACK packets are generated by aggregating up to 10 acknowledgments into a single ACK packet. To ensure that acknowledgments are not delayed excessively, we also limit the delay to no more than 10 ms.

2.3 Path Scheduling & Congestion Control

As the quality of proxied paths can vary significantly and change over time, it is not possible to statically determine the optimal set of paths. Previous work on path selection is mostly based on active probing, i.e. using *ping* [10] or *traceroute* [8], which incurs a large overhead and does not yield accurate results for the entire transmission period. Our approach to path selection is to passively detect changes in path quality and to dynamically react to these changes.

Proxied Path Creation. mPath first sends packets on the direct path to the destination. In this state, the system controls congestion, much like standard TCP, by employing a 'slow-start' phase and halving the congestion window if loss is detected. However, in addition to halving the congestion window, packet loss may also trigger the creation of proxied paths. The number of new proxied paths to be created when a loss is detected is a proportion β of the direct path's congestion window (and limited by the number of paths in the available list). We found that $\beta = 0.25$ achieves a good trade-off between the time taken to find good proxies and the utilization of the direct path. Proxies are chosen at random from the unused proxies in the available list. If all the available proxies have been used before, mPath chooses the best proxy that it has observed thus far. For each newly created proxied path i , mPath also maintains a congestion window w_i that is initially set to one.

To prevent rapid and uncontrolled creation of proxied paths, the system will only create new paths after all existing paths have encountered loss.

Multipath AIMD. mPath eliminates bad paths and exploits good paths by scaling congestion windows with an additive increase/multiplicative decrease (AIMD) algorithm [11]. Like TCP, a packet loss causes the congestion window of the affected path to be halved. The main difference between mPath and TCP is in how the congestion windows are increased when ACKs are received. During slow-start, the congestion window of the direct path will increase by one for every ACK received. In congestion control mode, cumulative ACKs received will increase the congestion window of either the direct path or the proxied paths. The congestion window of the proxied paths is increased with probability P and the congestion window of the direct path is increased with probability $1 - P$. The probability P is obtained with the following formula:

$$P = \frac{w_0}{W} \rho + \frac{W - w_0}{W} \quad (1)$$

where ρ is the proportion of proxied paths that have not encountered loss, w_0 is the congestion window of the direct path, and $W = \sum w_i$ is the total congestion window (over all paths inclusive of the direct path). The intuition is to use P to apportion the load between the direct path and the proxied paths according to their states. If some proxied paths have never encountered loss, we would like to increase their congestion windows rapidly; if all the proxied paths have encountered loss, then any increase in the overall congestion window should be divided between the direct and proxied paths according to their estimated relative available bandwidths.

When we decide to increase the congestion window of an existing proxied path, mPath selects an active path that has recently received an ACK and increases its congestion window by one. Paths that have never encountered loss are given higher priority. If all the paths have encountered loss, the addition goes to path i with probability $\frac{w_i}{\sum w_j}$, where w_j is the congestion window of proxied paths that have recently received ACKs.

Shared Bottleneck Detection. We need to identify the shared bottlenecks between proxied paths and the direct path so that we can shift traffic to the direct path. We use a simple but effective scheme to infer the existence of such bottlenecks: mPath records the transmission time for every packet sent and the loss detection time if an ACK arrives indicating that packets were not received. Each path then maintains a time range (“loss interval”) indicating when the last lost packet was detected and the time that the packet was sent on the path. If the loss interval on the direct path overlaps with the loss interval on a proxied path, we deduce that the packet losses are correlated and the paths share a common bottleneck. This is illustrated in Fig. 3.

Load Aggregation. Upon detecting the shared bottleneck, mPath will then move a proportion $\min(\alpha, \frac{w_0}{\sum w_j})$

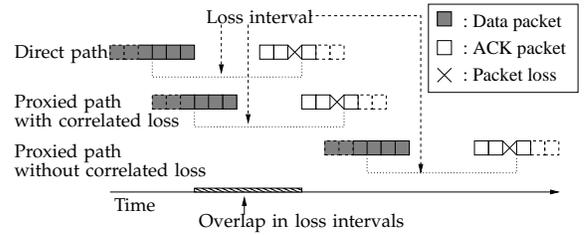


Fig. 3. Inference of correlated packet losses.

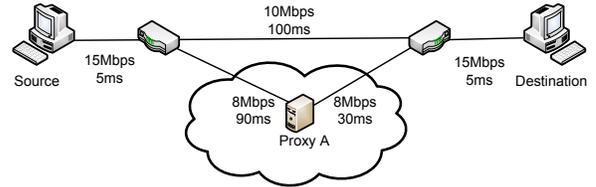


Fig. 4. An example of bottleneck oscillation.

of the proxied path’s remaining congestion window to the direct path, where w_j is the congestion window for the proxied path that experienced correlated packet loss. The upper bound $\frac{w_0}{\sum w_j}$ guarantees that the congestion window of the direct path will not be more than w_0 . In other words, mPath will decrease proxied path i ’s congestion window w_i to half for a normal loss on that path, but decrease it to $\frac{w_i}{2}(1 - \min(\alpha, \frac{w_0}{\sum w_j}))$ for a correlated packet loss and add $\frac{w_i}{2} \min(\alpha, \frac{w_0}{\sum w_j})$ back to the congestion window of the direct path. We call this operation *load aggregation*. We found that $\alpha = 0.5$ achieves a good trade-off between the utilization of good proxied paths and the direct path, and reduces the utilization of bad paths relatively quickly.

We choose to gradually decrease the congestion window of a proxied path instead of dropping the path completely in order to prevent *bottleneck oscillation*. We illustrated this with an example in Fig. 4. In this example, the transmission is initially limited by the 10 Mbps core link bottleneck on the direct path. As new proxied paths that can route around the core link bottleneck are found (e.g. proxy A), the common access link with capacity 15 Mbps will become the new bottleneck. When a correlated packet loss is detected at the new bottleneck, the naive approach of dropping proxy A completely will cause the bottleneck to shift back to the core. If mPath then uses proxy A (or some other good proxy) to improve throughput, the bottleneck will eventually shift back to the 15 Mbps access link and the system will oscillate. By aggregating the congestion windows of the proxied paths to the direct path, the stability of the system is improved.

Handling Timeouts. Like TCP, mPath detects timeout for all paths with a mechanism based on the estimated RTT for each path. The direct path reacts to a timeout by reverting to the slow-start state. However, when a timeout occurs for a proxied path, we drop it instead of reverting to slow-start since it is easy for mPath to either find a replacement among the unused proxied paths

or redistribute the load across existing paths. The path can be dropped temporarily or permanently depending on the historical contribution of the path. If the path's contribution to the throughput is significantly below average compared to other proxied paths in its lifetime, it will be marked as unavailable and dropped permanently. Otherwise, it will only be dropped temporarily and may be reused at a later time.

3 ANALYSIS OF MULTIPATH AIMD

In this section, we extend the classic Chiu and Jain AIMD model [11] to analyze multipath AIMD. We show that our algorithm (i) is TCP-friendly, (ii) maximizes the utilization of the access link without under-utilizing the direct path when there is free core link capacity, and (iii) rapidly eliminates any redundant proxied paths.

Following the notation in [11], we obtain the multipath model for a single user, which is illustrated in Fig. 5. The user imposes a total load of x on the system, with a load of x_i on each path i . Path 0 refers to the direct path, which has a core link capacity of X_{goal_0} , while paths 1 to n are the proxied paths available to the user. Without loss of generality, we assume that k out of the n proxied paths are limited by core (or AS-AS peering) link bottlenecks, each with a capacity of X_{goal_i} , $1 \leq i \leq k$, while the remaining paths from $k+1$ to n are free of congestion and can accept more load. The capacity at the access link bottleneck is denoted with X_{goal} . We also define Y to be the feedback vector to the user, which is a tuple comprising of binary feedback values y_i for each path i . A positive feedback ($y_i = 0$) implies that there is no packet loss on path i , while a negative feedback ($y_i = 1$) implies that congestion has occurred.

When a negative feedback is received, mPath will halve the load on the associated path, and if it is a correlated packet loss, mPath will perform load aggregation. When a positive feedback is received, mPath will increase the load by one with probability $\frac{x_i}{\sum_{i=0}^n x_i}$, denoted by γ_i . Clearly, γ_i is less than 1 for each path i and $\sum_{i=0}^n \gamma_i = 1$ at the access link bottleneck where all paths are aggregated. Thus, for one mPath flow, the additive-increase value on any path is always less than or equal to that of TCP and the multiplicative-decrease value is always equal to that of TCP. This implies that mPath is TCP-friendly. In addition, because the behavior of mPath will be similar to that of TCP at the access link, mPath will compete for resources fairly and efficiently [11].

There are three scenarios under which a bottleneck can occur: (i) the direct path is limited by a core link bottleneck but there is insufficient free capacity on the available proxied paths to saturate the access link; (ii) the direct path is limited by a core link bottleneck and enough free capacity via other paths can be found to saturate the access link; or (iii) the bottleneck is entirely at the access link.

(i) Core Link Bottleneck, Insufficient Capacity. The first case is where the direct path experiences congestion

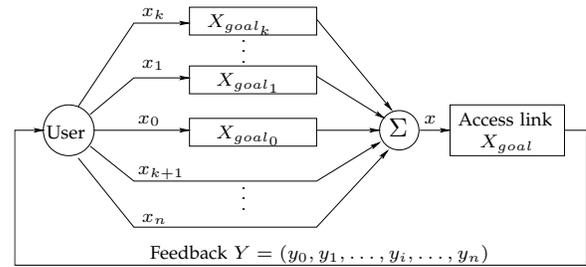


Fig. 5. Model for a single user using multiple paths.

on a core link and mPath cannot find a set of alternative paths to fully saturate the common access link. Load aggregation will ensure that any proxied paths sharing the same core link bottleneck as the direct path will eventually be dropped, i.e. only the proxied paths that do *not* share a bottleneck with the direct path will be retained. In the steady state, the congestion windows of the active paths would be oscillating in a manner that is equivalent to the *semicoupled algorithm* [7] for $a = 1$, where a is a constant that determines the aggressiveness of the congestion control algorithm. Raiciu et al. [7] determined that only the paths with loss rates satisfying the following condition will be used:

$$(1 - p_r) \frac{1}{\hat{x}} = p_r \frac{\hat{x}_r}{2} \quad (2)$$

where \hat{x} is the average load of the user, and p_r and \hat{x}_r are the loss rate and average load on path r respectively. This condition follows from the intuition that paths must either reach an equilibrium for the average increase and decrease of their load or converge to zero and get dropped. In other words, mPath will distribute as large a load as possible to paths with low loss rates and drop all the paths that have loss rates too high to satisfy condition (2).

(ii) Core Link Bottleneck, Excess Capacity Sufficient. The second case is where mPath is most effective. When there is a bottleneck on the direct path and sufficient core link capacity exists, the additive-increase phase will fully saturate the access link by utilizing the free capacity on proxied paths $k+1$ to n . This occurs even with a small additive-increase value of $\gamma_i < 1$, for $k+1 \leq i \leq n$, since these paths experience minimal packet loss. On the other hand, the congested paths 1 to k will experience packet loss and their loads will undergo multiplicative decrease, eventually converging to zero.

(iii) Access Link Bottleneck. Finally, if the flow is limited by an access link bottleneck, using multiple paths will not help. The proxied paths will experience correlated packet losses with the direct path, and, over time, load aggregation will move most of the traffic onto the direct path and cause the load for all the proxied paths to converge to zero.

We further analyze the performance of mPath for scenarios (ii) and (iii) in more detail. We consider a simple flow where there is only one proxied path, with a core link capacity of X_{goal_p} , sharing a common access

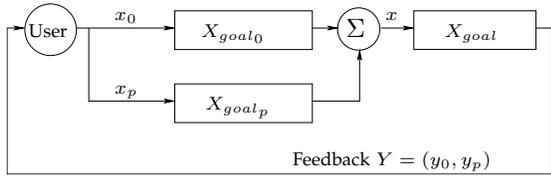


Fig. 6. Model of a shared access link bottleneck.

link with the direct path, as shown in Fig. 6. We define the load at time t on the direct path's core, proxied path's core and access link to be $x_0(t)$, $x_p(t)$ and $x(t)$ respectively. Depending on the feedback values received at time t , given by $Y(t) = (y_0(t), y_p(t))$, mPath will derive the loads at time $t + 1$ as follows:

- 1) $Y(t) = (0, 0)$. When positive feedback is received on both paths, the load on the direct and proxied path is increased probabilistically:

$$\begin{aligned} x_0(t+1) &= x_0(t) + 1 \\ &\text{with probability } \gamma_0 = \frac{x_0(t)}{x_0(t) + x_p(t)} \\ x_p(t+1) &= x_p(t) + 1 \\ &\text{with probability } \gamma_p = \frac{x_p(t)}{x_0(t) + x_p(t)} \end{aligned} \quad (3)$$

- 2) $Y(t) = (1, 0)$. When only the direct path has negative feedback, we obtain:

$$\begin{aligned} x_0(t+1) &= \frac{x_0(t)}{2} \\ x_p(t+1) &= x_p(t) + 1 \end{aligned} \quad (4)$$

- 3) $Y(t) = (0, 1)$. When only the proxied path has negative feedback, we obtain:

$$\begin{aligned} x_0(t+1) &= x_0(t) + 1 \\ x_p(t+1) &= \frac{x_p(t)}{2} \end{aligned} \quad (5)$$

- 4) $Y(t) = (1, 1)$. When negative feedback is received on both paths, the loads will be halved and load aggregation (with parameter α) will be performed to shift part of the load from the proxied path to the direct path:

$$\begin{aligned} x_0(t+1) &= \frac{x_0(t)}{2} + \alpha \frac{x_p(t)}{2} \\ x_p(t+1) &= (1 - \alpha) \frac{x_p(t)}{2} \end{aligned} \quad (6)$$

Now assume that X_{goal} is saturated at time t . This implies that there will be shared loss (i.e. $Y(t+1) = (1, 1)$), which will cause the overall load to be halved ($x(t+1) = x(t)/2$) and load aggregation to be performed according to Equation (6). If bottleneck conditions do not change and there are no other losses, it will take approximately $x(t)/2$ time intervals for X_{goal} to be saturated again to produce another shared loss, since the average increase for $x(t)$ is 1 at each time interval, as described by Equation (3). We can then deduce that after k intervals of shared losses under these conditions, at time $u = t + (k - 1)x(t)/2 + 1$, the loads would be:

$$\begin{aligned} x_0(u) &= \frac{x_0(t)}{2} + [1 - (1 - \alpha)^k] \frac{x_p(t)}{2} \\ x_p(u) &= (1 - \alpha)^k \frac{x_p(t)}{2} \end{aligned} \quad (7)$$

For scenario (ii), where the bottleneck is in the core and sufficient capacity exists (i.e. X_{goal_p} is large), x_0 and x_p would increase until either X_{goal} or X_{goal_0} is achieved. If X_{goal} is achieved first, correlated packet losses will be detected and load will aggregate to the direct path at an exponential rate until there are no more correlated losses as described by Equation (7). This behavior ensures that the direct path is never under-utilized. The overall performance in this case is the same as that for a normal TCP flow competing for resources constrained by the access link bottleneck. If X_{goal_0} is achieved first, mPath has over-utilized the direct path and caused congestion, so x_0 will be halved and x_p will increase (as described by Equation (4)) until X_{goal} is achieved. This means that mPath will always maximize the utilization of the access link without under-utilizing the direct path when sufficient core link capacity is available.

For scenario (iii), where the access link is the bottleneck, the proxied path is redundant. All losses will occur at the shared access link and load aggregation will always be performed to shift traffic to the direct path. Thus, as described by Equation (7), the load of the proxied path (x_p) will be aggregated to the direct path at an exponential rate until x_p is reduced to zero (i.e. the path is dropped).

Like the classic AIMD model [11], we assumed a synchronous feedback/control loop and omitted the RTTs of the paths in our analysis in order to keep the model tractable. We can extend the model to consider paths with different RTTs, but doing so will not shed significantly more insight. An extension of the classic AIMD model incorporating the RTTs [15], showed that the system would simply be biased against paths with longer RTTs. As mPath is a variant of AIMD, we can expect similar results. We show in Section 4.1 that even with paths of varying RTTs, mPath behaves as expected.

4 PERFORMANCE EVALUATION

We evaluated mPath by running experiments on both Emulab and PlanetLab to show that mPath (i) behaves in a manner consistent with the model described in Section 3, (ii) is practical and can often achieve significant improvements in throughput, and (iii) is scalable and that end-hosts can be used as proxies. We also investigated the tradeoffs associated with the choice of system parameters.

In our experiments, we compared mPath with TCP by sending continuous streams of randomly generated data. However, we did not use the native implementation of TCP due to two issues: (i) bias towards different transport protocols (e.g. by firewalls or routing policies) may skew the results; and (ii) PlanetLab limits the TCP window size, which limits the maximum throughput attainable. Therefore, we used an implementation of TCP (New Reno) based on UDP, which is available in UDT [16]. This implementation has been shown by its authors to have similar performance to native TCP.

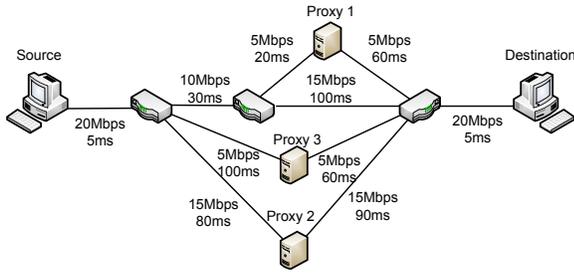


Fig. 7. An Emulab topology where mPath is able to find good proxied paths.

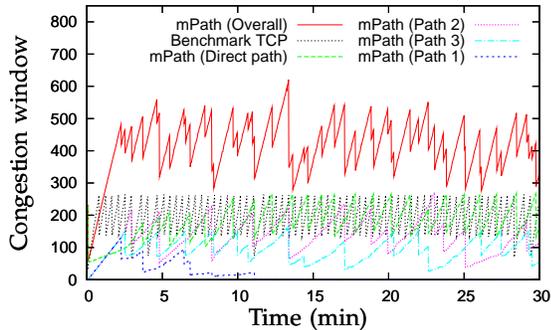


Fig. 8. Plot of congestion window over time for the topology in Fig. 7.

4.1 Is our model accurate?

We first verify the behavior of mPath with a series of experiments on Emulab, to show that mPath (i) can significantly improve throughput when there is sufficient core capacity and automatically distribute load over the paths according to the available path capacities, performs no worse than TCP when (ii) there is insufficient capacity in the core or (iii) the access link is the bottleneck, (iv) is TCP-friendly, and (v) can dynamically adapt to changing network conditions.

(i) Core Link Bottleneck, Excess Capacity Sufficient.

Our first set of experiments was conducted on a topology containing a core link bottleneck as shown in Fig. 7. In this topology, we created three proxied paths, with one proxy sharing a 10 Mbps core link bottleneck and all three proxies sharing a 20 Mbps access link bottleneck along the direct path. We ran mPath for 30 minutes and compared its performance to a TCP benchmark that was run for the same duration. As expected, mPath achieved an average throughput of 14.31 Mbps, and the benchmark TCP flow achieved an average throughput of 7.21 Mbps.

The congestion windows of the various paths (direct path and three proxied paths) used by mPath are shown in Fig. 8. The congestion windows of the proxied paths are labeled according to the proxies that they pass through. We also plot the congestion window for the benchmark TCP flow and the overall mPath congestion window over time for reference.

Initially, the three proxied paths compete for bandwidth, with the congestion window of the direct path increasing slowly to allow those of the proxied paths to

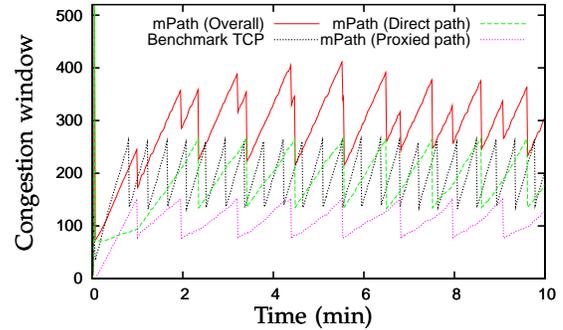


Fig. 9. Plot of congestion window over time for the topology in Fig. 7 when only proxy 3 is used.

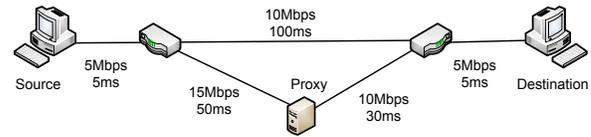


Fig. 10. An Emulab topology where the access link is the bottleneck and the proxied path is useless.

fully expand. In the process, mPath detects the shared bottleneck between path 1 and the direct path and applies load aggregation, causing path 1 to be dropped 11 minutes into the transfer when its congestion window is reduced to zero. This leaves the system in a stable state where paths 2 and 3 efficiently exploit the access link capacity that cannot be utilized by the direct path alone. Path 2 carries more traffic because it has a larger core link capacity. Observe that load aggregation also ensures that the direct path is fully utilized throughout the transfer, handling a load of about 7 Mbps.

(ii) Core Link Bottleneck, Insufficient Capacity. To investigate how mPath performs when there is insufficient core link capacity to saturate the access link, we use the topology shown in Fig. 7 but with proxy 1 and 2 removed. That is, we have a 20 Mbps access link, a 10 Mbps core link bottleneck on the direct path and a 5 Mbps alternative path via proxy 3. Our results from running mPath and TCP individually for 10 minutes are shown in Fig. 9. Clearly, mPath can still effectively utilize all available capacity on both direct and proxied paths.

(iii) Access Link Bottleneck. For this scenario, we designed a simple topology where the access link is the only bottleneck in the system, as shown in Fig. 10, and ran mPath and TCP individually for 10 minutes. The results in Fig. 11 show that mPath and TCP produce similar patterns for their congestion windows and achieve about the same throughput: 4.36 Mbps for mPath and 4.40 Mbps for TCP. Some traffic is sent along the proxied path at first, but this drops significantly when mPath determines it is of no benefit and the proxied path's congestion window is aggregated to the direct path. This example verifies that when a good proxies are not available due to an access link bottleneck, mPath behaves much like TCP.

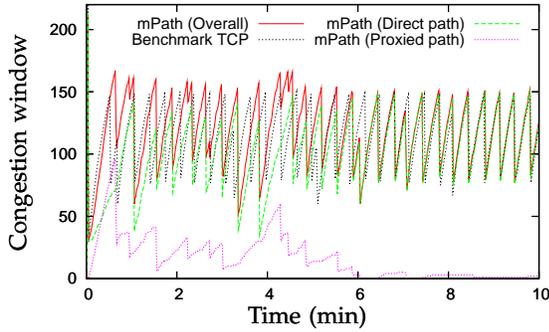


Fig. 11. Plot of congestion window over time for the topology in Fig. 10.

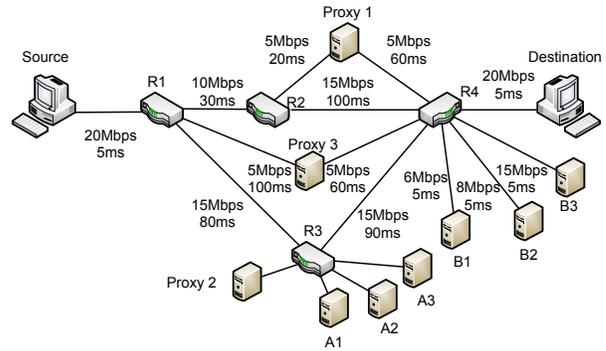


Fig. 13. An Emulab topology to investigate how mPath reacts to changing path conditions.

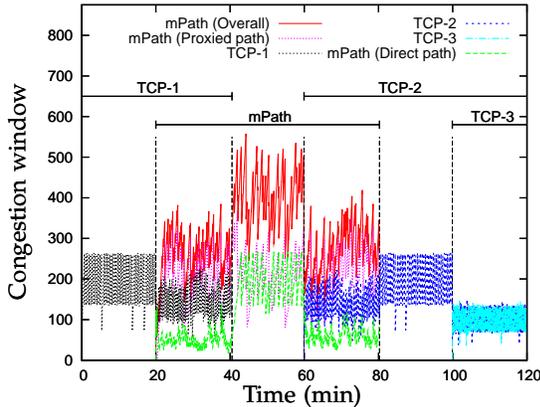


Fig. 12. Plot of congestion window over time with competing mPath and TCP flows for the topology in Fig. 7.

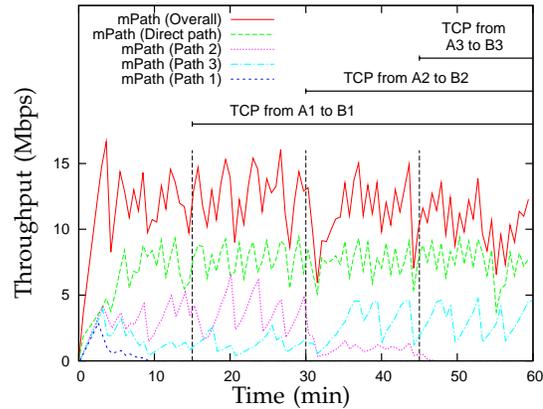


Fig. 14. Plot of throughput over time with interfering TCP flows on proxied path 2 for the topology in Fig. 13.

(iv) **TCP-friendliness.** We ran another experiment on the topology in Fig. 7 to evaluate mPath in the presence of competing TCP flows. We started a TCP flow (TCP-1) in the background, after which we started mPath to observe its influence on TCP-1. Next, TCP-1 is terminated and another TCP flow (TCP-2) is started to observe how mPath reacts to the new flow. Finally, mPath is stopped completely and a third TCP flow (TCP-3) is started to provide us with a benchmark for two competing TCP flows. After starting/stopping a flow, we give the system 20 minutes to stabilize before making the next change. The congestion windows of the various flows over time are shown in Fig. 12.

We found that running mPath in parallel with TCP causes the TCP congestion window to drop by about 25% on average, which is better than the TCP-3 benchmark which causes a 50% drop as expected. When TCP-1 is terminated, the direct path for mPath quickly soaks up all the excess bandwidth freed by the departure of TCP-1. When TCP-2 is started, it is able to achieve a steady state congestion window that is equivalent to that for TCP-1. The congestion windows of the proxied paths remain fairly stable throughout because they do not share the core link bottleneck with the direct path. These results show that, in the presence of good proxied paths, mPath can achieve an overall throughput surpassing that

of TCP while remaining TCP-friendly.

(v) **Adapting to changing proxied path conditions.** To show that mPath can dynamically adapt to congestion on the proxied paths, we created a new topology by adding some new nodes to the topology in Fig. 7, as shown in Fig. 13. Under this new scenario, path 2 of the mPath flow is disrupted by incrementally introducing three TCP flows that all use the path segment R3 to R4 to reduce the available capacity of the segment. The results are shown in Fig. 14. The mPath flow is given 15 minutes to stabilize before we start the first TCP flow from A1 to B1, which has an access link capacity of 6 Mbps. In this state, proxied path 2 still has sufficient capacity to allow mPath to saturate the access link. After another 15 minutes, we add a second TCP flow with an access link capacity of 8 Mbps from A2 to B2. Now path 2 does not have sufficient capacity and mPath automatically redistributes some load to path 3. mPath’s overall throughput in the steady state drops from 12.0 Mbps to 11.4 Mbps. When the final TCP flow from A3 to B3 is started, the segment R3 to R4 becomes highly congested and causes significant packet losses on path 2. This leads to path 2 being dropped completely within 2 minutes and an overall drop in the steady state throughput of mPath from 11.4 Mbps to 10.6 Mbps.

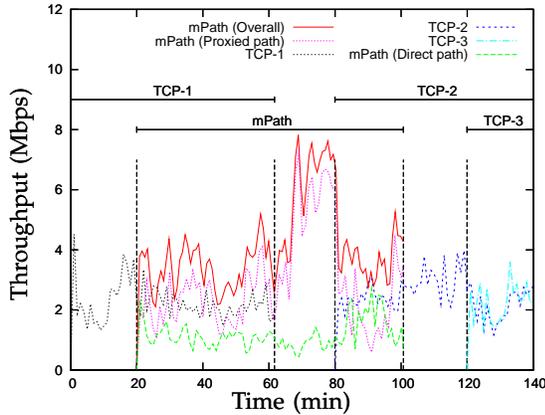


Fig. 15. Plot of throughput against time for the path from `pads21.cs.nthu.edu.tw` to `planetlab1.cs.uit.no`.

4.2 Does mPath work over the Internet?

To evaluate the performance of mPath over the Internet, we ran a series of experiments on PlanetLab using approximately 450 proxies. In this section, we present results from two representative experiments that demonstrate mPath (i) can improve throughput while maintaining TCP-friendliness when good proxied paths exist and (ii) performs no worse than TCP when a good proxied path cannot be found.

Performance with good proxied paths. The first experiment follows the same procedure as that of the Emulab experiment for TCP-friendliness described in Section 4.1. The only difference is that we now give mPath an additional 20 minutes to stabilize, since we now have significantly more proxied paths and it might take longer for good proxied paths to be found. The results from running this experiment on PlanetLab nodes `pads21.cs.nthu.edu.tw` and `planetlab1.cs.uit.no` are shown in Fig. 15.

We observed similar behavior as that for the earlier corresponding Emulab experiment. In both cases, mPath achieves a relatively large increase in throughput. However, when TCP-1 terminates, the increase in throughput occurs on the proxied paths, rather than on the direct path as observed on Emulab. This is because the sender is given exclusive access to the topology in Emulab, while many users may have flows passing through the same core link bottleneck on the Internet for the PlanetLab experiment. Flows that pass through the same link will be given a share of the freed bandwidth when a flow leaves. In this case, stopping the flow of TCP-1 will only increase mPath’s share of the direct path by a small amount. However, stopping TCP-1 also frees up bandwidth on the access link, which can then be used to increase the congestion windows of the proxied paths. Since the direct path is unable to supply enough bandwidth to fully utilize the access link, the proxied paths will take up most of the slack.

To better understand the improvement in throughput achieved by mPath, we did *traceroutes* for the direct and

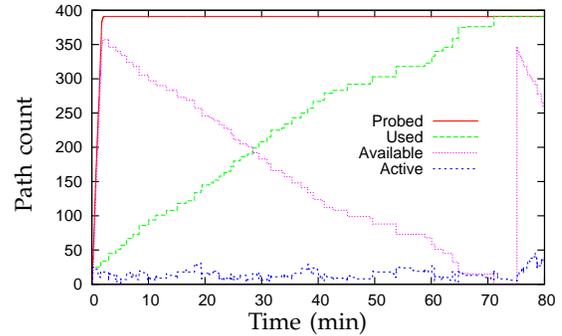


Fig. 16. Plot of proxied path usage over time.

proxied paths used in the experiment. The default route from `pads21.cs.nthu.edu.tw` to `planetlab1.cs.uit.no` used a direct path from 211.79.48.190 to 109.105.98.41, which is a route across the Indian Ocean to Europe. We found that the proxied paths that contributed most to the throughput did not intersect with this route. In particular, most of the proxied paths used crossed the Pacific Ocean, continental America, and the Atlantic Ocean before reaching Europe. To some extent, this is not surprising because the Earth is round and there are generally two ways to connect any two points on the planet: clockwise and anti-clockwise.

We also examined how mPath finds and uses the proxies in the system to establish a stable set of proxied paths. Fig. 16 is a plot of the distribution of the proxies over time. The probing phase to determine the available proxies completes relatively quickly and takes approximately 2 minutes to build an available list of approximately 400 proxies out of the 450 registered proxy nodes. mPath attempts to use all the available proxies in 75 minutes while maintaining an active set of between 10 to 20 proxies at any one time. Comparing these results with the evolution of throughput in Fig. 15, it is clear that the system finds a good working set of proxies long before it tries out all the available proxies. In fact, enough good proxied paths were found almost immediately after starting the transfer.

Performance without good proxied paths. In the second experiment, we used a pair of nodes (`planetlab2.cs.ucla.edu` and `planetlab2.unl.edu`) for which mPath failed to find any good proxied paths. The throughput achieved in this experiment is shown in Fig. 17. For this pair of nodes, we found that all the proxied paths experienced a bottleneck at the same access link. We can see from Fig. 17 that the throughput achieved by mPath and TCP are similar. mPath achieves its steady state throughput within 2 minutes and spends only about 12 minutes assessing the 450 available proxies before giving up.

4.3 How often and how well does mPath work?

We investigated the throughput achieved by mPath for approximately 500 source-destination pairs (distinct from the proxies) on PlanetLab and compared it to

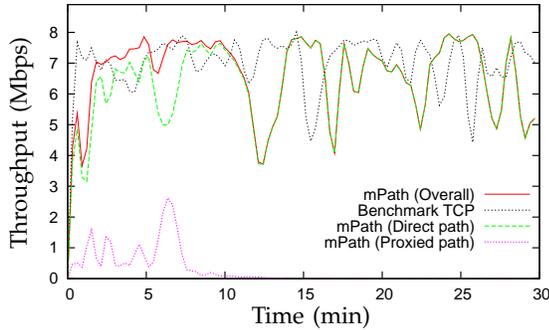


Fig. 17. Plot of throughput against time for the path from planetlab2.cs.ucla.edu to planetlab2.unl.edu.

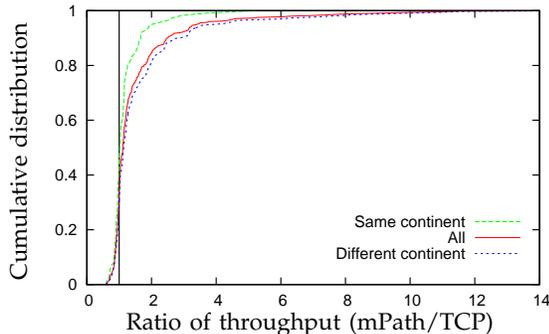


Fig. 18. Cumulative distribution of the ratio of mPath throughput to TCP throughput for 500 source-destination pairs.

the throughput achieved by TCP. Fig. 18 shows the cumulative distribution of the ratio of the throughput achieved by mPath to that of TCP over all the node pairs tested. Each data point was obtained by running mPath for 30 minutes followed by TCP for another 30 minutes on each pair of nodes.

We found that mPath performs at most 20% worse than TCP for a small number of node pairs, which we believe can be attributed to the natural temporal variance of the available bandwidth on the Internet due to congestion and cross-traffic. To verify this, we ran a large number of TCP flows back-to-back for 30 minutes on random node pairs and plot the ratio between these two flows in Fig. 18 as “TCP benchmark”. The line provides us with a benchmark for what would be considered performance equivalent to TCP. In this light, we consider mPath to have achieved an improvement over TCP if it achieves a distribution that is to the right of this benchmark line. We see that about 40% of the node pairs seem to achieve a non-trivial improvement in throughput, with about half of these pairs achieving more than twice the throughput achieved by TCP. This is a significant proportion and it verifies our hypothesis that many of the bottlenecks for the direct paths are in the core, at least for PlanetLab nodes. From our observations, we found that the remaining 60% of node pairs could not improve their throughput using mPath, possibly because they were limited by their access links.

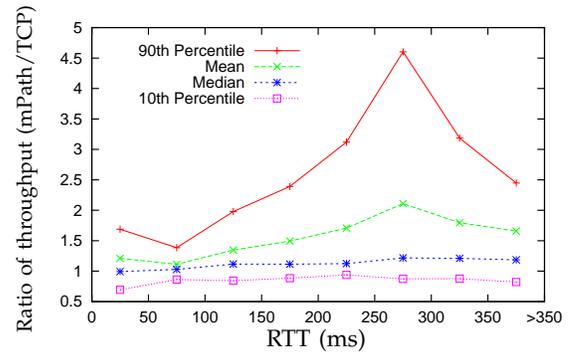


Fig. 19. Plot of ratio of mPath throughput to TCP throughput against RTT.

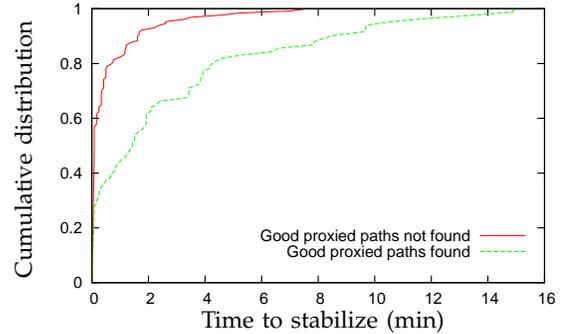


Fig. 20. Cumulative distribution of the time taken for mPath to stabilize.

Even for cases where mPath seems to perform more poorly than TCP, the distribution is still on the right of the TCP benchmark line, suggesting that even when network conditions deteriorate, mPath is likely able to ameliorate the degradation.

Intuitively, the distance between the sender and receiver would have a significant impact on how much mPath can improve the throughput. We expect that if the sender and receiver are very close (e.g. in the same AS), the throughput gains would only be marginal. This is evident in Fig. 18, where we plot the improvement ratio of node pairs that have been categorized according to whether they are located on the same continent or on different continents. From these results, it is clear that the pairs located on different continents can achieve larger improvements and this conforms with our intuition. In addition, we plot the throughput improvement ratio against the direct RTT between the sender and receiver (for 500 node pairs) in Fig. 19. As expected, node pairs with a higher RTT have a greater chance of benefiting from mPath and this suggests that mPath should use proxied paths more aggressively if the direct path RTT is larger, but this remains as future work.

To determine how quickly mPath can find a good set of proxies, we plot the time taken for mPath to reach its steady state throughput in Fig. 20. We see that if good proxied paths exist, mPath can find them within 5 minutes for 80% of the node pairs. If good proxied paths cannot be found, mPath gives up within 1 or 2

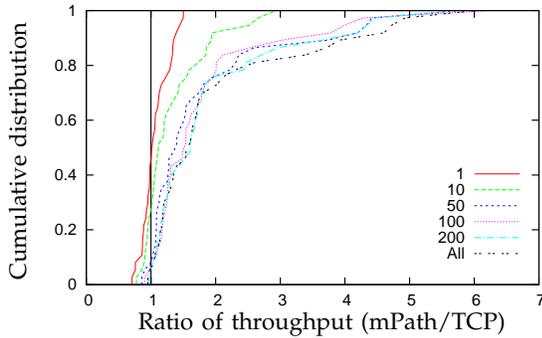


Fig. 21. Cumulative distribution of the ratio of mPath throughput to TCP throughput when different numbers of proxies are provided by the RS.

minutes 90% of the time.

Based on these results, we see that the practicality of using mPath largely depends on the location of the bottleneck and the duration of transmission. If the bottleneck is at the core, and the transmission takes longer than mPath’s stabilization time, we can expect the throughput improvements shown earlier. On the other hand, if the bottleneck is at the access link, or if the transmission duration is too short, throughput gains with mPath will be marginal (if any). This suggests that mPath is only suitable for high-volume data transfers.

4.4 How many proxies are minimally required?

We are also interested in the number of candidate proxies required for mPath to find a good proxy set. In this experiment, we limited the size of proxy lists returned by the RS and compared the throughput achieved by mPath to TCP. The sizes of proxy sets investigated are 1, 10, 50, 100, 200, and all the available proxies (approximately 450). As shown in Fig. 21, the performance of mPath improves up till about 50 proxies, after which the performance gains of having more available proxies become negligible. This suggests that the RS should provide source nodes with at least 50 proxies. As these results are for a system where only one mPath flow is active, it is possible that more proxies will be required in practice.

4.5 Is mPath scalable?

Since mPath is expected to support a large number of users, we want to understand how the performance of mPath will scale as the number of users in the system increases. We used 200 PlanetLab nodes (distinct from the proxies), partitioned them into 100 disjoint pairs of senders and receivers, and ran experiments with 1, 50 and 100 pairs of nodes transmitting to each other simultaneously. By using four different random partitions, we obtained 400 data points for each of the three scenarios. These results are shown in Fig. 22.

Since mPath is useful only if it improves the throughput for a node pair significantly, we focus on the proportion of node pairs for which mPath can achieve

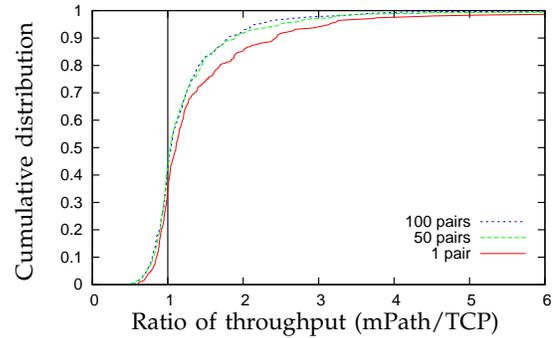


Fig. 22. Cumulative distribution of mPath throughput to TCP throughput with n disjoint source-destination pairs transmitting simultaneously when proxies and end-hosts are distinct nodes.

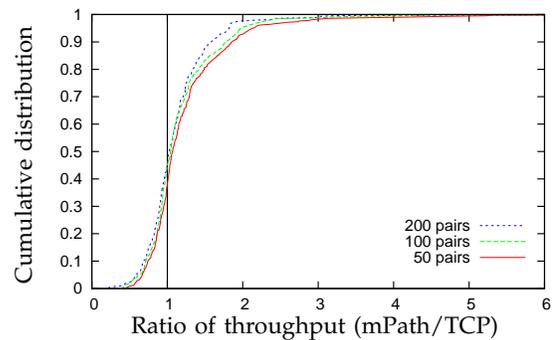


Fig. 23. Cumulative distribution of mPath throughput to TCP throughput with n disjoint source-destination pairs transmitting simultaneously when the end-hosts are themselves proxies.

throughput that is at least twice that of TCP. We had shown earlier that when there is only one user, about 20% of the source-destination pairs can achieve twice the throughput of TCP. As shown in Fig. 22, for 50 and 100 concurrent users, this number drops to about 10% of the users. This can be explained as follows: mPath consumes the unused bandwidth of proxies to improve throughput and users who are concurrently sending or receiving data would compete for this same bandwidth. If the amount of unused bandwidth is kept constant, then as the number of concurrent users increases, the number of users who see an improvement in throughput would decrease. Hence, the scalability of mPath depends on the amount of unused bandwidth available in the system, as expected. As we are limited by the PlanetLab nodes available, we are not able to conduct larger scale experiments to study this in greater detail.

Using client nodes as proxies is one potential way of improving the scalability of mPath. To evaluate the feasibility of this approach, we devised an experiment where 450 PlanetLab nodes are used as proxies and 50, 100 and 200 source-destination pairs are randomly selected from these proxies to concurrently send/receive data. As shown in Fig. 23, mPath is still able to improve the throughput for some node pairs. For 50 and 100

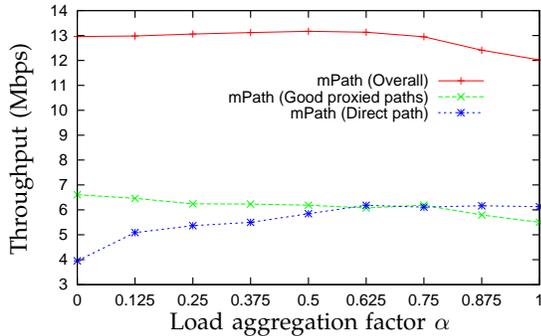


Fig. 24. Plot of throughput against load aggregation factor α .

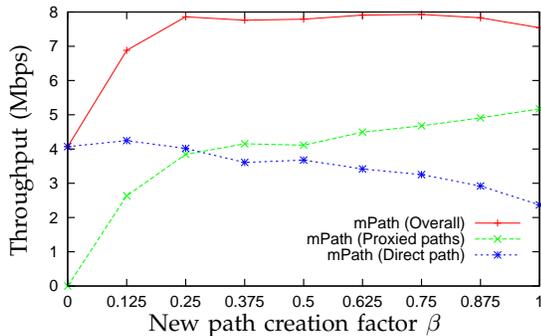


Fig. 25. Plot of throughput against new path creation factor β .

pairs, about 10% of the pairs can achieve at least twice the throughput of TCP. This drops to a very small number for 200 pairs, but this is not entirely surprising as this means that some 400 out of the 450 proxies are sending/receiving data. Since these proxies do not have much unused bandwidth to be exploited, mPath is unlikely to be able to use them to improve throughput. Our results suggests that if the number of users concurrently sending/receiving data is less than 50% of the total number of proxies, it is feasible to use client nodes as proxies.

4.6 How should the parameters be tuned?

mPath is characterized by the parameters α , β and τ . In this section, we investigate the tradeoffs for each of these parameters. We investigate the effect of α on Emulab because of the controlled environment and the effect of β and τ on PlanetLab because it was not practical to create hundreds of proxies on Emulab.

Load Aggregation (α). The first parameter, α , is the proportion of the congestion window moved from a proxied path to the direct path when a correlated packet loss is detected. If α is too large, it may result in low utilization of good proxied paths and reduced throughput; if α is too small, it may take a long time for bad proxied paths to converge to zero and cause the direct path to be under-utilized. As shown in Fig. 24, our experiments indicate that $\alpha > 0.75$ would lead to a decrease in overall throughput. We also found that when $\alpha \leq 0.25$, mPath

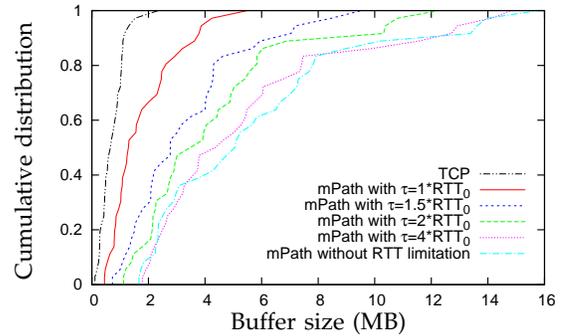


Fig. 26. Cumulative distribution of the maximum buffer size required for different maximum proxied path RTTs τ .

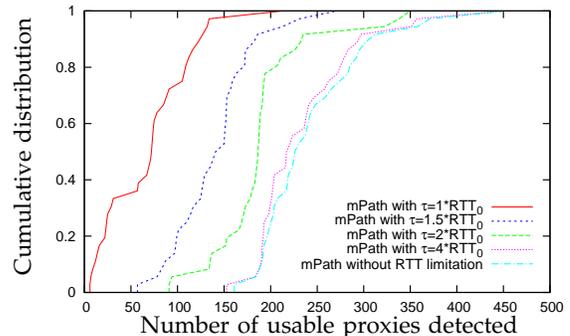


Fig. 27. Cumulative distribution of the number of usable proxies detected for different maximum allowable proxied path RTTs τ .

would take more than 30 minutes to eliminate the bad proxied paths. Thus, we set $\alpha = 0.5$.

Proxied Path Creation (β). Next, we investigate β , the number of new paths created as a fraction of the direct path congestion window when loss is detected. This number trades off the time taken by mPath to find a good proxy set against the utilization of the direct path. Using a pair of nodes observed to have good proxied paths (planetlab6.goto.info.waseda.ac.jp to planetlab2.wiwi.hu-berlin.de), we perform data transfers lasting 30 minutes and plot the throughput achieved against β in Fig. 25. The graph shows that when $\beta \geq 0.5$, the utilization of the direct path decreases, and when $\beta > 0.75$, there is even a slight drop in the overall throughput. We also found that when $\beta < 0.25$, the time taken to find good proxies increases significantly, and that beyond $\beta = 0.25$, there is no substantial reduction in this time. Therefore, we set $\beta = 0.25$.

Maximum Allowable Proxied Path RTT (τ). Intuitively, the maximum allowable RTT for the proxied paths is directly related to packet reordering and the number of proxied paths that can be used, and these factors will affect the achieved throughput. Fig. 26 shows the effect of τ on the maximum buffer size required at the receiver. Clearly, increasing τ results in greater reordering and thus larger buffering requirements. Fig. 27 shows the number of usable proxied paths as we increase τ . We pick $\tau = 2 \times RTT_0$ because this provides a

sufficient number of proxies and because we found that increasing it beyond this value did not yield significant improvements in throughput.

5 RELATED WORK

In this section, we provide an overview of prior work in the literature related to mPath.

Bandwidth Bottlenecks: Bottlenecks are commonly thought to occur at the access links. Akella et al. were the first to dispute this assumption, by highlighting that nearly half of the Internet paths they investigated had a non-access link bottleneck with an available capacity of less than 50 Mbps [3]. Hu et al. suggested that bottlenecks exist everywhere, at access links, peering links or even inside ASes [2]. Our experience with mPath seems to corroborate their findings.

Detour Routing: The benefits of detour routing using an overlay network has been demonstrated by many researchers [1, 2]. Anderson et al. built a resilient overlay network (RON) [10] based on detour routing and showed that it could recover from a significant number of outages and path failures. mTCP [8], built over RON, attempts to improve throughput by exploiting multiple paths but has scalability issues arising from the maintenance of the RON overlay. An overlay network is also used by Skype [17] to improve the latencies of VOIP. mPath differs from these systems in that it aims to maximize throughput by using hundreds of lightweight proxies to do one-hop source routing instead of depending on an overlay network [8, 10]. Gummadi et al. were the first to propose one-hop source routing as a means to address RON’s scalability issues [9].

Multi-homing and Multipath TCP: Another common mechanism that can provide path diversity is multi-homing [18], but it needs to be supported by the ISPs at the network-layer. Multipath TCP (MPTCP) [7] was developed to support multipath TCP over multi-homing and has been proposed for use in intra-datacenter bulk transfers [19]. The design of MPTCP is similar to mPath, e.g. the separation of the connection-level and subflow-level sequence numbers and a coupled congestion control algorithm to take into account shared bottlenecks and to maintain TCP-friendliness. The major difference between mPath and MPTCP is that MPTCP seeks only to allocate traffic optimally over a fixed (and small) set of available paths, while mPath needs to solve two separate problems simultaneously: (i) identify good proxied paths out of several hundred paths; and (ii) allocate the optimal amount of traffic to the good proxied paths. Also, mPath can exploit, but does not require, multi-homing.

Parallel TCP and Split TCP: mPath also differs from Parallel TCP [20, 21] and Split TCP [22]. Parallel TCP was proposed to increase throughput by exploiting multiple TCP flows at the expense of TCP-friendliness. In mPath, we strictly adhere to the AIMD mechanism to maintain TCP-friendliness and achieve greater throughput by simply routing around core link bottlenecks. Split TCP increases throughput by exploiting the pipeline parallelism

of multiple low-latency segments, which, unlike mPath, requires buffering of data at the proxies and breaks end-to-end guarantees.

Path Selection: Previous work on selecting good paths from a large pool includes random- k [9] and the earliest divergence rule [23]. mTCP’s path selection method of using *traceroute* to select disjoint paths is not adaptive and has been shown to be unscalable [8]. We believe that our approach of dynamically assessing path quality and adaptively adding and dropping paths depending on their performance will be more scalable in practice.

Congestion Control: The AIMD [11, 24] algorithm employed in TCP is easily implemented and works well in achieving fair bandwidth distribution between competing flows. Our congestion control algorithm is a variant of AIMD that uses information from multiple paths in a correlated manner. This is similar to Congestion Manager [25], where congestion control is performed for multiple applications for a single host.

In mTCP [8], congestion control is performed for each individual path without coordination among the paths. We found that this strategy is overly aggressive when there are a large number of paths. It has been shown that coordinated congestion control is better [26, 7], so we also adopted a coordinated approach. mPath is similar in many ways to the multi-path TCP algorithms proposed and analyzed by Raiciu et al. and we verified that our algorithm satisfies all the requirements proposed in [7]. Our key innovation is a *load aggregation* mechanism that attempts to maximize the utilization on the direct path and causes the congestion windows for redundant proxied paths to converge to zero.

There have also been a number of theoretical works on multipath congestion control algorithms based on fluid models [27] and control theory [28]. However, Raiciu et al. simulated these algorithms and found that they do not work well in practice [7].

6 CONCLUSION

We propose mPath, a practical massively-multipath source routing algorithm, that (i) is TCP-friendly, (ii) will maximize the utilization of the access link without under-utilizing the direct path when there is free core link capacity, and (iii) will rapidly eliminate any redundant proxied paths. This is achieved with a modified AIMD congestion control algorithm that uses *loss intervals* to infer shared bottlenecks and incorporates a *load aggregation* mechanism to maximize direct path usage.

Multipath routing is currently not widely used in practice due to the lack of infrastructure support and limited availability of multi-homing, which existing solutions depend on. Since mPath only requires stateless proxies to enable efficient multipath data transfers, we believe it is a more practical solution given the state of existing network infrastructure. Another factor that has traditionally hindered the adoption of multipath routing is the lack of use cases. However, recent work on using multipath

solutions to transfer bulk data between datacenters [6, 19] shows that there are useful applications for multipath routing, and mPath can potentially be applied to these and other scenarios as well.

ACKNOWLEDGMENTS

This work was supported by the Singapore Ministry of Education grant T208A2101.

REFERENCES

- [1] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and G. Zahorjan, "Detour: Informed Internet Routing and Transport," *IEEE MICRO*, pp. 50–59, 1999.
- [2] N. Hu, L. E. Li, Z. M. Mao, P. Steenkiste, and J. Wang, "Locating Internet Bottlenecks: Algorithms, Measurements, and Implications," in *Proceedings of SIGCOMM '04*, Sep. 2004.
- [3] A. Akella, S. Seshan, and A. Shaikh, "An Empirical Evaluation of Wide-area Internet Bottlenecks," in *Proceedings of IMC '03*, Oct. 2003.
- [4] iN2015 Infocomm Infrastructure, Services and Technology Development Sub-Committee, "Totally Connected, Wired and Wireless," Jun. 2006.
- [5] G. Kola and M. Livny, "DiskRouter: A Flexible Infrastructure for High Performance Large Scale Data Transfers," UW–Madison, Tech. Rep. CS-TR-2004-1518, 2003.
- [6] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-Datacenter Bulk Transfers with NetStitcher," in *Proceedings of SIGCOMM '11*, Aug. 2011.
- [7] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath tcp," in *Proceedings of NSDI '11*, Mar. 2011.
- [8] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson, and R. Wang, "A Transport Layer Approach for Improving End-to-end Performance and Robustness Using Redundant Paths," in *Proceedings of USENIX '04*, Jun. 2004.
- [9] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall, "Improving the Reliability of Internet Paths with One-hop Source Routing," in *Proceedings of OSDI '04*, Dec. 2004.
- [10] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient Overlay Networks," in *Proceedings of SOSP '01*, Oct. 2001.
- [11] D. M. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, vol. 17, no. 1, pp. 1–14, 1989.
- [12] D. Seah, W. K. Leong, Q. Yang, B. Leong, and A. Razeen, "Peer NAT proxies for peer-to-peer applications," in *Proceedings of NetGames '09*, Nov. 2009.
- [13] M. J. Freedman, K. Lakshminarayanan, S. Rhea, and I. Stoica, "Non-transitive connectivity and DHTs," in *Proceedings of WORLDS '05*, Dec. 2005.
- [14] Stanford Linear Accelerator Center, "The PingER project," <http://www-wanmon.slac.stanford.edu/cgi-wrap/pingtable.pl>.
- [15] Y. R. Yang, M. S. Kim, X. Zhang, and S. S. Lam, "Two Problems of TCP AIMD Congestion Control," Department of Computer Sciences, UT Austin, Tech. Rep. TR-00-13, 2000.
- [16] Y. Gu and R. L. Grossman, "UDT: UDP-based Data Transfer for High-speed Wide Area Networks," *Comput. Netw.*, vol. 51, no. 7, pp. 1777–1799, 2007.
- [17] W. Kho, S. A. Baset, and H. Schulzrinne, "Skype relay calls: Measurements and experiments," in *Proceedings of IEEE INFOCOM '08*, Apr. 2008.
- [18] A. Akella, J. Pang, B. Maggs, S. Seshan, and A. Shaikh, "A Comparison of Overlay Routing and Multihoming Route Control," in *Proceedings of SIGCOMM '04*, Sep. 2004.
- [19] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," in *Proceedings of SIGCOMM '11*, Aug. 2011.
- [20] H. Sivakumar, S. Bailey, and R. Grossman, "PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks," in *Proceedings of SC '00*, Nov. 2000.

- [21] T. Hacker, B. Athey, and B. Noble, "The End-to-end Performance Effects of Parallel TCP Sockets on a Lossy Wide-area Network," in *IPDPS '02*, 2002, pp. 434–443.
- [22] R. Jain and T. J. Ott, "Design and implementation of split TCP in the linux kernel," Ph.D. dissertation, Newark, NJ, USA, 2007.
- [23] T. Fei, S. Tao, L. Gao, and R. Guerin, "How to Select a Good Alternate Path in Large Peer-to-peer Systems," in *Proceedings of IEEE INFOCOM '06*, Apr. 2006.
- [24] Y. Gu, X. Hong, and R. Grossman, "An Analysis of AIMD Algorithms with Decreasing Increases," in *Proceedings of GridNets '04*, Oct. 2004.
- [25] H. Balakrishnan, H. S. Rahul, and S. Seshan, "An Integrated Congestion Management Architecture for Internet Hosts," in *Proceedings of SIGCOMM '99*, Sep. 1999.
- [26] P. Key, L. Massouli, and D. Towsley, "Path selection and multipath congestion control," in *Proceedings of IEEE INFOCOM '07*, May 2007.
- [27] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley, "Multi-Path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1260–1271, 2006.
- [28] W.-H. Wang, M. Palaniswami, and S. H. Low, "Optimal flow control and routing in multi-path networks," *Perform. Eval.*, vol. 52, pp. 119–132, Apr. 2003.



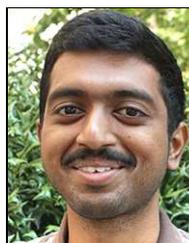
Yin Xu is pursuing his Ph.D. in Computer Science at the National University of Singapore. He received his Bachelor's degree in Computer Science in 2008 at Fudan University of China. His research interests include distributed systems and computer networking.



Ben Leong is an Assistant Professor of Computer Science at the School of Computing, National University of Singapore. He received his Ph.D., M.Eng. and S.B. degrees from the Massachusetts Institute of Technology in 2006, 1997 and 1997 respectively. His research interests are in the areas of computer networking and distributed systems.



Daryl Seah is pursuing his Ph.D. in Computer Science at the National University of Singapore. He received his Bachelor's degree in Computer Science in 2008 at the National University of Singapore. His research interests include distributed systems and computer networking.



Ali Razeen is pursuing his Ph.D. in Computer Science at Duke University. He obtained his Bachelor's degree in Computer Science from the National University of Singapore in 2011. His current research interests include distributed systems and networking.