UKAIRO: Internet-Scale Bandwidth Detouring

Thom Haddow
Imperial College London

Sing Wang Ho
Imperial College London

Cristian Lumezanu
Georgia Tech

Moez Draief
Imperial College London

Peter Pietzuch
Imperial College London

Abstract

The performance of content distribution on the Internet is crucial for many services. While popular content can be delivered efficiently to users by caching it using content delivery networks, the distribution of less popular content is often constrained by the bandwidth of the Internet path between the content server and the client. Neither can influence the selected path and therefore clients may have to download content along a path that is congested or has limited capacity.

We describe UKAIRO, a system that reduces Internet download times by using *detour paths* with higher TCP throughput. UKAIRO first discovers detour paths among an overlay network of potential detour hosts and then transparently diverts HTTP connections via these hosts to improve the throughput of clients downloading from content servers. Our evaluation shows that by performing infrequent bandwidth measurements between 50 randomly selected PlanetLab hosts, UKAIRO can identify and exploit potential detour paths that increase the median bandwidth to public Internet web servers by up to 80%.

1 Introduction

Bandwidth is a critical performance metric for many Internet services. Operating system vendors that push large patch files to customers [28], video sharing sites like YouTube, and HD streaming services such as iTunes and Netflix require high-bandwidth Internet paths to send content quickly to users. In addition, as cloud computing applications become popular, access to high-throughput paths between their users and the cloud data centre becomes crucial for efficient operation [2].

Historically, the "last-mile" network path to the user was the bandwidth bottleneck. However, the recent widespread deployment of faster access technologies such as ADSL+ [34], DOCSIS 3 [8] and fibre-to-the-kerb means that the bottleneck is shifting away form the edge towards the core of the network [22]. As a consequence, obtained bandwidth depends on the capacity and congestion of the traversed Internet path, which is determined by routing protocols such as OSPF [15] and BGP [30]. These protocols select paths based on the financial gain and performance of the *Internet service providers* (ISPs) rather than directly using user-based performance metrics such as available bandwidth. This often results in paths for clients with unnecessarily low bandwidth to certain destinations.

Existing solutions to improve content distribution performance include the use of *content delivery networks* (CDNs) such as Akamai, Amazon CloudFront [7] and Coral [10]. CDNs replicate content and place it at locations "closer" to the clients to which they have high bandwidth paths. Since they require substantial storage and network resources, their usage is cost-effective only for popular content; content providers must find other solutions to handle the long tail of less popular content [3]. Downloading content along multiple paths using Bittorrent [4], Bullet [19], SplitStream [5] or multipath TCP [13] also increases download bandwidth but requires modifications to clients and content servers and may be less efficient due to the transfer of redundant data. More radical proposals such as content-centric networking [16] require changes to the Internet architecture, which are difficult given the existing investment in network equipment.

In contrast, we propose to enable users to select Internet paths with higher bandwidth by "detouring" traffic via a set of third-party nodes. Researchers have shown that *detour routing* [31] can reduce path latencies [23, 14] or increase availability [1, 11] without requiring changes to Internet routing protocols or content servers. An open challenge, however, is how to discover *detour paths* that have higher bandwidth than the direct path while keeping the measurement overhead low [12].

We provide a practical solution to this problem that

relies on the observation that a small number of good detour nodes on the Internet is sufficient to provide substantial bandwidth increases to a set of clients for a majority of destinations. Based on experiments on the global PlanetLab test-bed, we show that 50 randomly chosen detour nodes can provide 71.4% of the relative bandwidth improvement obtained by using a larger set of 150 PlanetLab nodes. In addition, a client can get almost all of the detouring benefit by strategically selecting 5 good detour nodes from this set of 50 detour nodes.

We describe the design and evaluation of UKAIRO[1], a system that allows web clients to exploit detour paths with higher TCP throughput on the Internet. UKAIRO maintains an overlay network of detour nodes that act as SOCKS proxies for TCP connections. The system relays TCP connections of external clients through a detour node for improved bandwidth.

The UKAIRO system operates in two separate phases: *detour discovery* and *client detouring*. In the detour discovery phase, the system collects all-pairs bandwidth measurements between a set of overlay nodes that can act as potential detour nodes and finds bandwidth detours among them. It then ranks the overlay nodes according to their average relative bandwidth improvement when acting as detour nodes. To combine good detouring performance with load-balancing of detour nodes, each client is assigned a small client-specific subset of detour nodes from the ranked overlay node list through biased random sampling.

In the second phase, a client finds a detour for a given destination by probing each of its client-specific detour nodes and selecting the path that results in the highest bandwidth when compared to the direct path. This path selection is done by performing a short parallel HTTP download until each path is likely to have reached steady-state bandwidth. Over time, a client can refine its client-specific detour node set by discarding detour nodes that do not perform well and replacing them with fresh ones from the ranked overlay node list.

The results from deployment of UKAIRO on PlanetLab show that it can increase the median bandwidth of routes to public webservers by 80%. Its load-balanced detour selection scheme can support simultaneous users, and the bandwidth measurement overhead can be reduced to 40% of all-pairs measurements in the overlay network.

In summary, the contributions of the paper and the outline of its remainder are:

1. an empirical study of bandwidth detours on PlanetLab, demonstrating that a small number of detour nodes can provide the bulk of achievable bandwidth improvement (Section 2);
2. a practical approach for bandwidth detouring on the Internet by discovering good detours among a set of overlay nodes and using them for third-party clients through path selection (Section 3); and
3. an evaluation of the UKAIRO system on PlanetLab with external web servers that shows the feasibility of bandwidth detouring system (Section 4).

[1]Japanese romaji for "detour"

2 Bandwidth Detouring

In this section, we provide evidence that bandwidth detouring is both *effective* and *feasible*. First, we study the properties of bandwidth detours and show that they provide significant bandwidth improvement and are stable. Second, we show that only a small number of carefully selected detour nodes is necessary to improve bandwidth significantly to most destinations.

2.1 Detour properties

To study the properties of detours, we measure the bandwidth of all network paths between 201 nodes on PlanetLab [35], a global test-bed for networking research, on 26 Oct, 2010. We include only one PlanetLab node per site and ensure that each node participates in at most one concurrent measurement. To measure bandwidth, we use `iperf`, a standard tool for observing TCP throughput of a path by creating an elastic TCP transfer between two hosts [36]. We use iperf because it is accurate and provides fast measurements [12]. Note that we use the terms bandwidth and TCP throughput interchangeably throughout the paper.

Benefit. We first show the potential benefit of detouring for bandwidth. A path (i, j) between the two nodes i and j benefits from detouring through a node l if $\hat{b}_{ilj}$, the minimum of the bandwidth of the two legs of the detour b_{il} and b_{lj}, is larger than the bandwidth b_{ij} of the direct path, i.e.,

$$\hat{b}_{ilj} = \min(b_{il}, b_{lj}) > b_{ij} .$$

In the remainder of the paper, we denote by $\hat{b}_{ilj}$ the bandwidth of the path between i and j when detoured via l as given by $\min(b_{il}, b_{lj})$. This assumes that the bandwidth of a detour path is determined by the bottleneck link [12]. We also denote by $d_l(i, j)$ the percentage of (or relative) bandwidth increase if the path (i, j) uses node l as a detour, i.e.,

$$d_l(i,j) = \frac{\max\left(\hat{b}_{ilj} - b_{ij}, 0\right)}{b_{ij}} , \qquad (1)$$

and we set $d_l(i, i) = 0$.

Based on our bandwidth measurement dataset described above, we associate with each path (i, j) the best detour node, i.e., the node l with the highest value of $d_l(i, j)$. We find that 80.2% of the 35,240 paths measured benefit from an increase of at least 20% and 1 Mbps of their bandwidth with one-hop detours. We restrict our analysis to one-hop detours because adding more hops yields little additional improvement, as observed previously [12].

Longevity. Detour longevity, or the time for which a detour persists, is crucial for the feasibility of bandwidth detouring. Measurements to discover detours take a significant amount of time and bandwidth to perform. Long-lasting detours reduce the overhead of the system.

We study how detours evolve on PlanetLab. We measure paths between 100 randomly chosen PlanetLab nodes at 4am (GMT) everyday between Dec 9, 2010 and Jan 5, 2011 (except for Dec 18, 2010 due to a failure). For each set of measurements, we exhaustively search for the detours for each path and validate the detours' improvement, $d_l(i, j)$, on a later dataset. We find that 79.6% of the discovered detours are still valid after one day, providing a 53.2% median increase in bandwidth. The improvement provided by a discovered detour decreases over time—the median increase reduces to 34.3% by the 14th day. After 28 days, 71.1% of detours still exist but the increase has dropped to 31.4%.

We conclude that measurements for detour discovery have to be repeated roughly every 2 weeks in order to avoid substantially stale data. This bodes well for the practical use of detour nodes in a real deployment. In fact, a relatively high measurement overhead to discover detour nodes can be amortised over the lifetime of an application because detour paths, once discovered, remain effective for a prolonged period of time.

2.2 Finding detours scalably

We showed that bandwidth detours can provide significant and lasting improvements. However, to operate effectively, a system exploiting these detours must discover them quickly and with as little measurement overhead as possible, in particular, without relying on all-to-all path measurements. Next, we investigate the effectiveness of three techniques to reduce the number of measurements required to discover detours yet preserve their quality.

2.2.1 Using latency detours instead

Existing proposals for detouring to reduce latency use network coordinate embedding [24] or AS path similarity [14] to discover latency detours with low measurement overhead. Ideally, we could use such approaches to find latency detours and use them to improve path bandwidth.

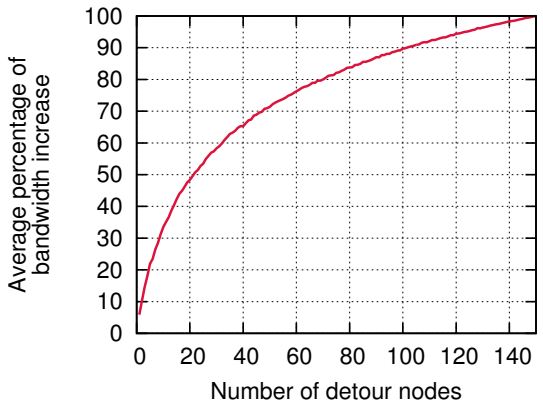

Figure 1: Percentage of bandwidth improvement suffers from diminishing returns as the size of the detour node set increases

However, as we show in Section 4 and as confirmed by previous results [23], latency detours do not improve bandwidth. We also found that the use of bandwidth detours does not necessarily reduce latency, as measured using the `ping` tool. In our dataset, only 15.6% of paths experience lower latency when detoured via the best detour node, i.e., the detour l that maximises the ratio $d_l(i, j)$ for the path (i, j). The median latency increased by 10.9 ms or 8.3% when detouring through the best bandwidth detour.

2.2.2 Reducing the number of detour nodes

Another way to lower the number of required bandwidth measurements is to restrict the number of detour nodes in the system. To evaluate whether this preserves good detours, we divide our n PlanetLab nodes into a *detour set* $\mathcal{D}$ containing k nodes, chosen uniformly at random, and a *client set* $\mathcal{C}$ containing the remaining $n - k$ nodes. For each path between a pair of nodes i and j in the client set $\mathcal{C}$, we find the best detour node l_{ij} in the detour set $\mathcal{D}$, i.e., the one with the largest improvement as given by

$$d_{l_{ij}}(i, j) = \max_{l \in \mathcal{D}} d_l(i, j) \; .$$

In this setting, we let the bandwidth of a path (i, j), $i, j \in \mathcal{C}$, be the maximum between $\hat{b}_{il_{ij}j}$ and b_{ij}. In other words, we fall back to using the direct path when no better bandwidth detour exists.

To understand how the size of the detour set affects the *quality* of detours, we consider the average percentage of improvement over all paths between nodes in the client

set $\mathcal{C}$, given by

$$\frac{1}{|\mathcal{C}|\,(|\mathcal{C}|-1)} \sum_{i,j \in \mathcal{C}} d_{l_{ij}}(i,j)\,. \qquad (2)$$

as a function of the size of the detour set $\mathcal{D}$. We repeat the experiment 1000 times, each time selecting a fresh set of detour nodes at random.

Figure 1 shows that increasing the size of the detour set $\mathcal{D}$ offers diminishing returns beyond a certain point. For example, for a detour set of 50 nodes, we preserve 71.4% of the optimal bandwidth improvements obtained from the all-to-all measurements described in Section 2.1. Therefore, a key insight is that even a limited number of randomly chosen nodes that act as potential detour nodes can achieve the bulk of the improvement from bandwidth detouring.

2.2.3 Choice of detour nodes

To reduce the number of path measurements required when searching for a good detour node, clients can first try detour nodes that are more likely to provide good detours. We analyse two strategies for ranking the nodes in the detour set $\mathcal{D}$, with the goal of identifying detour nodes that are more likely to lead to high detouring bandwidth: a *client-oblivious* and a *client-specific* strategy.

Client-oblivious ranking. In this strategy, the rank of a detour node does not depend on the client that uses it.

Given a set $\mathcal{P}$ of measured paths, we define $d_l(\mathcal{P})$, the *score* of node $l \in \mathcal{D}$, as the average improvement generated if we used l as the only detour for all paths in $\mathcal{P}$, i.e.,

$$d_l = \frac{1}{|\mathcal{P}|} \sum_{(i,j) \in \mathcal{P}} d_l(i,j)\,. \qquad (3)$$

Here we assume that we have measurements for all the paths between nodes in $\mathcal{D}$, i.e., $\mathcal{P} = \{(i,j) \in \mathcal{D} \times \mathcal{D}, i \neq j\}$. The nodes in $\mathcal{D}$ are then ranked in decreasing order of their score d_l. We define $\mathcal{D}_m$ as the set consisting of the first m nodes from this ranked list.

Client-specific ranking. In the client-specific ranking strategy, we compute the average improvement of each node $l \in D$, for each client $i \in C$. In this case, the score of node $l \in \mathcal{D}$, given a set $\mathcal{P}_i$ of measured paths between node i and some other client nodes in $\mathcal{C}$ is defined by

$$d_l(i) = \frac{1}{|\mathcal{P}_i|} \sum_{j:(i,j) \in \mathcal{P}_i} d_l(i,j)\,. \qquad (4)$$

Here we assume that we have measurements for all nodes in $\mathcal{D}$, i.e., $\mathcal{P}_i = \{(i,j)\ ,\ j \in \mathcal{D}, j \neq i\}$. Similarly to the client-oblivious strategy, each client i ranks the nodes in D in decreasing order of their $d_l(i)$ scores and then uses the first m nodes from this list as potential detours.

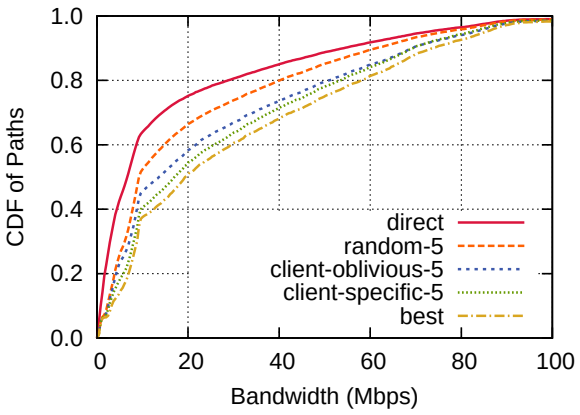

Figure 2: Comparison of the effectiveness of different ranking strategies to reduce the set of detour nodes to be considered by clients

We now evaluate the strategies by considering their detouring improvement. For each path (i,j) between nodes in $\mathcal{C}$, we associate the highest bandwidth that can be obtained by detouring through one of the detour nodes provided by a given ranking strategy, or set it to $b_{i,j}$ if none beats the direct path.

Figure 2 shows the results in terms of the distribution of bandwidths of obtained detour paths, based on our PlanetLab dataset. We compare this to the bandwidth distribution of the direct paths (direct), the best out of 5 randomly chosen detour paths (random-5) and the best possible by exhaustively considering all detour nodes in $\mathcal{D}$ (best).

The curve labeled client-oblivious-5 represents the distribution of bandwidth between nodes in $\mathcal{C}$ using the client oblivious ranking of $\mathcal{D}_5$, as described above. We observe that a value of $m = 5$ ensures enough diversity of detour nodes while keeping the size of the set small. Although the detour nodes are ranked based on their performance on the set $\mathcal{D}$, according to Equation (3), they offer good detours between the nodes in the set $\mathcal{C}$.

Similarly, the curve labeled client-specific-5 shows the bandwidth for the client-specific strategy with $m = 5$. Its improvement is close to the one obtained through best. Bandwidth is also higher than for client-oblivious-5, which illustrates that, by selecting detours based on their suitability for specific client nodes, we get improvements similar to the ones obtained through an exhaustive search for the best detour nodes.

2.3 Summary

We showed, using real world bandwidth measurements on PlanetLab, that bandwidth detours can provide significant and lasting improvements between Internet hosts.

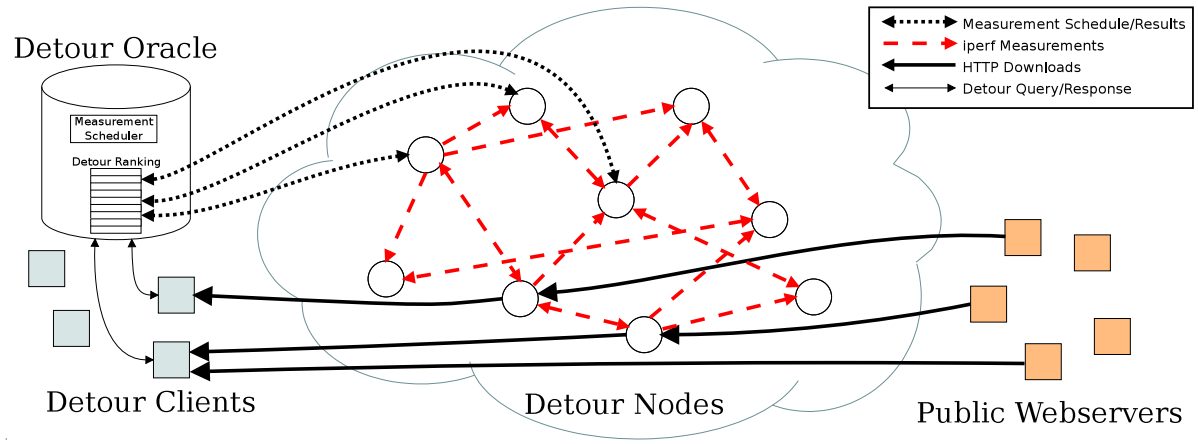

Figure 3: Overview of the UKAIRO system architecture

Discovering such detours does not require all-to-all measurements and most benefit can be recovered by restricting to detours through a small subset of detour nodes selected based on their average bandwidth improvement. These findings are crucial for providing a scalable and efficient service for bandwidth detouring on the Internet.

3 Ukairo Architecture

Next we describe UKAIRO, an Internet service for scalable bandwidth detouring using a set of overlay nodes. UKAIRO provides two main mechanisms: *detour discovery* and *client detouring*. Detour discovery identifies which nodes can act as bandwidth detours (Section 3.1), while client detouring enables clients to relay their traffic through these detour nodes (Section 3.2.1).

Figure 3 presents an overview of our system. UKAIRO consists of an overlay network of **detour nodes**, $\mathcal{D}$, which provide higher bandwidth paths to clients, and a **detour oracle**, which is responsible for discovering good detour nodes. The detour oracle collects bandwidth measurements between the nodes in the overlay using a **measurement scheduler** and ranks them based on their detouring potential (i.e., the average percentage of bandwidth increase that they provide). To take advantage of UKAIRO, users install **detour clients** which query the oracle to obtain a set of candidate detours for a given destination. Using on-demand measurement, detour clients then choose the best detour out of all candidates.

3.1 Detour discovery

Next we describe the detour discovery part of UKAIRO in detail. It is based on the observation that discovered detours through a smaller set of overlay detour nodes can be used effectively for detouring traffic of external clients and destinations.

At the heart of the detour discovery process are two components: a *measurement scheduler* and a *detour oracle*. The measurement scheduler periodically collects bandwidth measurements between the overlay nodes by instructing nodes to measure the bandwidth of paths. The detour oracle then uses the collected measurements to rank the detour nodes based on their estimated potential for bandwidth detouring.

3.1.1 Measurement Scheduler

The goal of the measurement scheduler is to collect bandwidth measurements among the set of detour nodes $\mathcal{D}$. The measurement scheduler can instruct detour nodes to measure bandwidth to other nodes using the `iperf` tool. Nodes report back successfully completed measurements to the scheduler, which records them in a database. Because measuring all-to-all bandwidth incurs significant overhead, the scheduler uses a two-phase heuristic to reduce the number of measurements. First, it collects the measurements needed to determine the detouring potential of each node. Once it acquires sufficient data, it continues to perform measurements with preference for paths that traverse nodes with high detouring potential. We describe the two phases in detail next (Algorithm 1).

In the first phase, the scheduler iteratively measures a set of paths to gather partial information about the nodes' detouring potential (lines 4–8). At each step, it picks the node with the lowest number of measured paths. In other words, it finds the node l with the smallest count of measured paths (a, b) where (a, l) and (l, b) are both measured. Next, two other nodes i, j are chosen at random from the detour node set $\mathcal{D}$ and the scheduler measures all six paths between them.

The second phase starts as soon as each node l in $\mathcal{D}$

Algorithm 1: Measurement scheduler

```
1  D := measurement set
2  p := target percentage of paths to measure
3  t := min. frequency before entering second phase
4  while min{∀l ∈ D Freq. l } < t do
5      l ← NodeWithLowestFrequency
6      i ← RandomNodeThatIsNot(l)
7      j ← RandomNodeThatIsNot(l, j)
8      MeasureAllSixPathsBetween(i, l, j)
9  while Proportion of Measured Paths < p do
10     R ← RankDetoursIn(D)
11     RemoveFullyMeasuredNodes(R)
12     l ← NodeWithHighestRankIn(R)
13     i ← RandomNodeThatIsNot(l)
14     j ← RandomNodeThatIsNot(l, j)
15     MeasureAllSixPathsBetween(i, l, j)
```

collects values of $d_l(i,j)$ for t distinct paths (i,j) in the set of measured paths $\mathcal{P}$. The scheduler then ranks each detour node in decreasing order of their scores given by Equation (3). Subsequently, it selects nodes according to their rank and, for each node, measures detour paths that traverse the node and have not yet been measured (lines 9–15).

The measurements stop after a fraction p of the paths was measured. The value of p depends on the cost of measuring an overlay path and the benefits of a more accurately measured overlay—we evaluate different choices of p in Section 4.3.

3.1.2 Detour Oracle

The detour oracle has access to the bandwidth measurements collected by the measurement scheduler. Each client queries the oracle to receive a small set of candidate detour nodes. The oracle determines these sets by ranking the detour nodes based on the available bandwidth measurements. It implements two different strategies for returning candidate detour node sets to clients.

Client-oblivious ranking. The aim of this strategy is to benefit from the properties of detour nodes highlighted in the analysis of client-oblivious-5 in Section 2.2.3. For each detour node $l \in \mathcal{D}$, the oracle computes its score according to Equation (3) where $\mathcal{P}$ corresponds to the proportion p of paths measured by the scheduler. Nodes are then ranked in a list $\mathcal{L}$ in decreasing order of this metric.

The list $\mathcal{L}$ is then made available to client nodes, which can then choose their detours from it. If some detour nodes are particularly effective for a large number of paths, then they may be exploited by too many paths at once resulting in a degradation of their detouring capability. Next, we propose a load balancing strategy to avoid overloading individual detour nodes.

Load-balanced ranking. Our modified strategy supports load-balancing of detour nodes when used concurrently by many clients to avoid overloading particularly effective detour nodes. The oracle ranks detour nodes as in the client-oblivious strategy. It then only provides clients with a small subset of detour candidates. These candidates are chosen uniformly at random from a truncated list containing the first half of detour nodes in the ranked list $\mathcal{L}$.

3.2 Client detouring

The detour oracle supplies clients with a set of candidate detour nodes that are more likely to provide bandwidth improvement to *any* destination. The goal of the client detouring mechanism is to decide which of the candidate detour nodes the client should use for a *specific* destination. Next we describe how clients choose a detour node and how the overlay carries traffic across the detour path to the destination.

3.2.1 Path selection

Each client host participating in the UKAIRO system runs a detouring client component. The detouring client intercepts HTTP connections from an application, such as a web browser or a file downloader, and transparently detours them through an appropriate detour node. After acquiring a list of candidate detour nodes, the detouring client must perform path selection, i.e., choose the best detour node for the given destination, or fall back to the direct path if no better detour is found.

Path selection requires predicting the bandwidth to a destination through a set of detour nodes. To preserve the performance benefits of bandwidth detouring, path selection must be fast, transparent to applications, and have little overhead. However, even when a client is restricted to choosing from a small set of detour nodes, it is still difficult to accurately and quickly predict which detour node to use for a particular path. Previous work [27] has attempted to use latency or jitter as inexpensive predictors for path bandwidth. However, we have found that these methods tend to be too unpredictable or resource-intensive to be of practical use on PlanetLab nodes.

UKAIRO performs path selection by starting parallel HTTP transfers on each candidate path and simultaneously measuring their bandwidth to select the best path. Figure 4 shows how clients perform path selection on the candidate detour set obtained from the oracle. Path selection has two phases: a *sampling* phase, in which the best

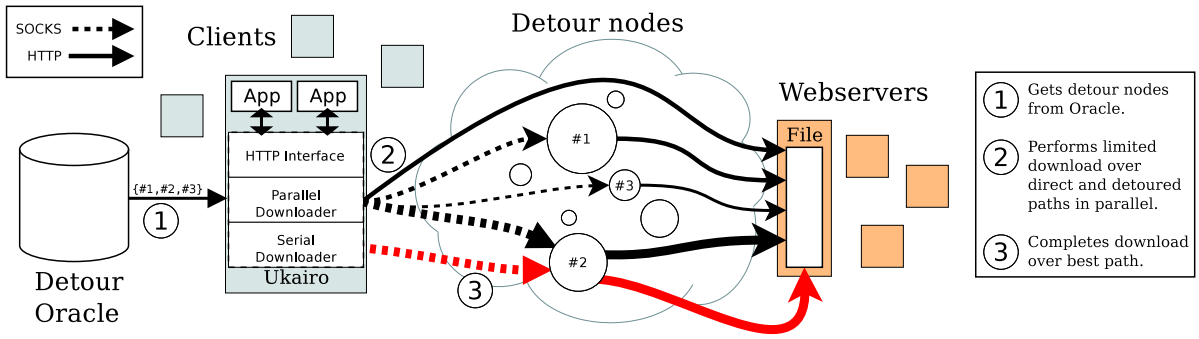

Figure 4: A client performs a parallel path selection download to choose the best path from the supplied detours.

available path is discovered by performing a limited content download, and a *completion phase*, in which the best detour path is used for the rest of the download.

In the sampling phase, given s candidate detour nodes and a content size N, the parallel path-selecting HTTP downloader allocates a chunk of size $N/(s+1)$ to each potential detour path and the direct path using the HTTP "range" header [9]. HTTP as a protocol is particularly suited for this because of its idempotent operation semantics—downloads can be terminated and restarted on different paths as required.

UKAIRO downloads each chunk in parallel until a timeout of 5 seconds is reached or the chunk is completed. This is based on an empirical observation that most paths achieve TCP steady-state behaviour within 5 seconds. The path with the fastest download rate is then used to download the remainders of any incomplete chunks by constructing a multiple-range final HTTP request in the completion phase. If none of the detour paths have higher bandwidth than the direct path, the download completes using the direct path.

To reduce load on the detouring overlay and to avoid an unfair advantage on congestion due to multiple TCP flows, we want to minimise the amount of time that parallel downloading occurs. We observe that, due to the short timeout period and the TCP slow startup, the combined parallel rate in the short sampling phase rarely exceeds the serial rate of the completion phase. This suggests that the sampling phase has a limited negative influence over competing flows.

3.2.2 Adaptive detour selection

Clients can improve their choice of detour nodes over time by taking advantage of additional measurements collected during path selection. In this way, the list of client-oblivious nodes provided by the oracle becomes client-specific. We have already shown in Section 2.2.3 that detour nodes specific to clients provide better performance than client-oblivious ones.

Algorithm 2: Adaptive detour selection

1 $\mathcal{L}_i :=$ ranked detour list
2 $\ell_i :=$ candidate detour set
3 $l_i :=$ test detour
4 run parallel path selection with ℓ_i, l_i as detours
5 $\mathcal{L}_i \leftarrow$ RankDetoursUsingAvailableMeasurements
6 $\ell_i \leftarrow$ get top-s detours from $\mathcal{L}_i$
7 **if** $b_{il_ij} < b_{ij}$ **then**
8 $\quad l_i \leftarrow$ RandomlySelectNodeUsing($\mathcal{L}_i$)

We describe how a client can incrementally improve its choice of detours in Algorithm 2. Initially, each client i downloads the ranked list of detour nodes together with their scores from the oracle; the detour nodes are based on the client-oblivious strategy described in Section 3.1.2. We refer to this list as $\mathcal{L}_i$. After each download that client i performs, it updates the list $\mathcal{L}_i$ to incorporate the new measurements that it acquired with the goal of customising its choice of candidate detour nodes, as follows.

Given an instance of the list $\mathcal{L}_i$, client i draws a set ℓ_i consisting of s distinct candidate detour nodes from $\mathcal{L}_i$ where each node is chosen with probability proportional to its current score. In addition to the set ℓ_i, the client chooses a distinct *test detour* l_i from $\mathcal{L}_i$ with a probability proportional to its score. The idea behind adding a test node is to speed up the convergence of our set ℓ_i to the set of client-specific detours (cf. Section 2.2.3).

Before downloading from some node j, client i runs the sampling phase of parallel path selection on paths through the candidate detour nodes in ℓ_i, the test detour l_i, and the direct path. As a by product, the list $\mathcal{L}_i$ is updated to incorporate the new measurements acquired by running parallel path selection. More precisely, the score of each detour node l is recalculated by averaging its current score with the new value $d_l(i,j)$ that it acquired. The nodes are then re-ranked in decreasing order of these new scores to obtain a fresh list $\mathcal{L}_i$ (lines 5–6).

A fresh detour candidate set ℓ_i is drawn, and if detouring through l_i does not beat the direct path, a new test detour l_i is chosen (lines 7–8).

The random procedure above of drawing the set ℓ_i and the node l_i ensures that the set of candidate detour nodes quickly converges, as we perform more and more downloads, to the set of client-specific detours as illustrated in Section 4.1.2.

3.2.3 Detouring

To carry traffic from clients to destinations on detour paths, detour nodes relay TCP connections on demand, which is illustrated in Figure 4. This is done using the standard SOCKS protocol [21], as implemented by the lightweight `srelay` SOCKS server [37]. SOCKS is essentially an application-layer protocol for proxying the standard "network socket" programming interface via a remote host over TCP. From an application perspective, SOCKS provides a similar interface to that of the standard sockets API and lightweight wrappers are easily deployed to make it completely transparent to applications.

In our architecture, we utilise the `libcurl` HTTP library [32] to interface with the remote SOCKS server. HTTP connection attempts are first relayed to the detour node over the TCP-based SOCKS protocol. The detour then establishes a new TCP connection to the intended destination server. Since two separate sequential TCP handshakes must be performed to complete this process, connecting via SOCKS typically doubles the connection setup time. From the perspective of the destination server, the connection appears as a regular HTTP connection, except that it comes from the detour node rather than the original client node.

Since detouring results in two TCP connections, the system benefits from the *split-TCP* effect [18]. The TCP throughput of a long IP path can be improved by splitting the connection into two independent TCP connections. Each split TCP connection becomes more responsive to packet loss due to its lower round-trip time, and thus has higher throughput. However, split-TCP alone does not account for the improvement due to bandwidth detouring—many bandwidth detours have intermediate leg latencies that are larger than the direct path latency [12].

Another approach for redirecting traffic is IP detouring: tunneling IP packets to the detour nodes, which in turn redirect them to the corresponding destination. However, as we showed in our earlier work [12], IP detouring is not viable when detour nodes are at the edge of the Internet. The longer paths to edge nodes (in terms of IP hops) incur higher loss rates and therefore exhibit lower bandwidth

However, TCP detouring, as opposed to IP detouring, is less transparent to destinations because TCP connections appear to originate from the wrong source IP address. As this is a common side effect of many Internet systems such as HTTP proxies or gateway NATs, we believe that it is an acceptable restriction.

4 Evaluation

We evaluate UKAIRO using realistic bandwidth-intensive HTTP workloads on the PlanetLab test-bed. The goal is to demonstrate that UKAIRO can improve download performance for bulk HTTP downloads for many destinations, while exhibiting acceptable overheads.

First, we evaluate the download performance of a single client in isolation, from both PlanetLab nodes and public Internet web servers, using various detouring strategies (Section 4.1). Second, we present performance results from multiple clients utilising the system concurrently to observe the impact of load balancing (Section 4.2). Finally, we explore the bandwidth measurement overhead of the system (Section 4.3).

Experimental setup. We consider a population of geographically-dispersed 201 PlanetLab nodes, such that there are no two nodes from the same PlanetLab site. We choose 50 nodes at random to act as the UKAIRO detour nodes and configure a machine at our university as the detour oracle.

We select two independent sets of 30 nodes each to act as detour clients and destination web servers. The destination nodes run the lightweight `thttpd` HTTP server [29] serving pre-generated random binary files. We choose a download file size of 40 MB to obtain statistically significant results on high bandwidth paths. To avoid measuring low bandwidth paths, we limit any single serial HTTP transfer to 20 seconds.

Since we have observed that we cannot sustain detour connections over 40 Mbps reliably on PlanetLab, we attempt to find detours only for direct paths with bandwidth below this rate. This restricts us to around the 85$^{\text{th}}$ percentile of Internet paths, as shown in Figure 2.

The client nodes download a predefined list of URLs as follows. First, clients download each URL via the direct path (i.e., without any detouring). Clients then attempt to download each URL via a detour path with higher bandwidth than the direct path by obtaining a candidate list from the oracle (Section 3.1.2) and performing path selection (Section 3.2.1).

4.1 Single-client detouring

Our first experiment shows that the bandwidth improvement due to detouring using the two oracle ranking strategies, client-oblivious-5 and load-balanced-5, with

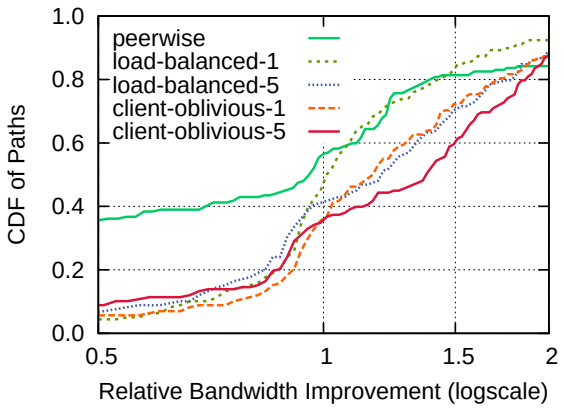

Figure 5: Detouring improves bandwidth on the majority of lower-bandwidth Internet paths

5 candidate detour nodes for each client (cf. Section 3.1.2). To demonstrate the benefits of having more candidate detours, we also compare to similar strategies that use only one detour node (client-oblivious-1 and load-balanced-1). The 10 client, 25 destination, and 50 detour nodes are all located on PlanetLab. We run client benchmarks serially to avoid interactions between measurements.

In addition, we compare to an approach, in which bandwidth detours are chosen based on latency. To test this, we use PeerWise [23] to discover latency detours from clients to destinations and return the detour node that provides largest latency reduction. These detours are validated without using the parallel path selection mechanism. We resort to the scalable network coordinate based mechanism of PeerWise to find latency detours, and not to less scalable all-to-all measurements, because it better matches the overhead constraints of a real deployment.

In Figure 5, we show the distribution of detouring path performance expressed as a fraction of the direct path bandwidth on a logscale. Results to the right of the vertical line at 1 indicate an improvement in detoured download performance. The points to the left of 1 represent paths that appear to be good detours based on measurements, but which in fact have lower throughput than the direct path, due to measurement noise causing mistakes in path selection and the detouring overhead. The results are based on around 200 path measurements, in which around 75% of the direct path bandwidths are under 40 Mbps. During the parallel path selection download, we observed that the overall rate of the parallel sampling phase was on average only 3% faster than the completion phase, suggesting the parallel aspect of the downloading does not gain an unfair share of the network resources.

When each client uses the same set of 5 highest-ranked detour nodes (client-oblivious-5), the median bandwidth improvement is around 1.4× that of the direct path. 65% of paths see no reduction in bandwidth, while 40% of paths are improved by at least 50%. The load-balanced-5 strategy leads to a smaller bandwidth improvement—the median improvement is 1.2× compared to the direct path. However, it still provides at least as much bandwidth as the direct path on the majority (60%) of detours.

The performance of the client-oblivious-1 strategy shows the impact that a single well-chosen detour node can have on general Internet end-to-end bandwidth. It provides a median improvement of around 20% over all paths and has a significant 50% improvement on around 30% of paths. The load-balanced-1 approach demonstrates that any single detour can provide a few significant gains—e.g., at least some improvement on half of the paths tested. However, its relatively poor performance in comparison to client-oblivious-1 highlights the importance of a good choice of detour node.

Both path selection strategies with single detour nodes, namely client-oblivious-1 and load-balanced-1, perform reasonably well because the probability of selecting the wrong detour path is lower with only two (direct and single detour) paths to choose from. All the path-selection strategies show similar performance when they fail to provide a detour path with higher bandwidth than the direct path. This suggests that any reduction in performance is then independent of the detour node given by path selection, and is more likely due to general overheads, such as the increased connection set-up time and limitations in the forwarding capacity of detour nodes.

When choosing bandwidth detours based on latency (peerwise), the overall performance drops, with more than half of the paths showing a substantial reduction in bandwidth when detoured. That the peerwise strategy performs worse means that, as described in Section 2.2, using latency to find bandwidth detours without a means of verifying the suggested path is not a viable solution.

4.1.1 Detouring to public web servers

To explore the potential of bandwidth detouring on a more realistic subset of the Internet, we run an experiment that uses public Internet web servers as detouring destinations. We use around 100 software mirror sites hosting the Linux kernel source archive, which is around 40 MB in size. The majority of mirror servers are hosted at commercial sites. We use 50 detour nodes and 10 client nodes on PlanetLab. To minimise any negative impact of our experiment on the web servers, each client requests the file from a single site and we use only the client-oblivious-5 detouring strategy, which per-

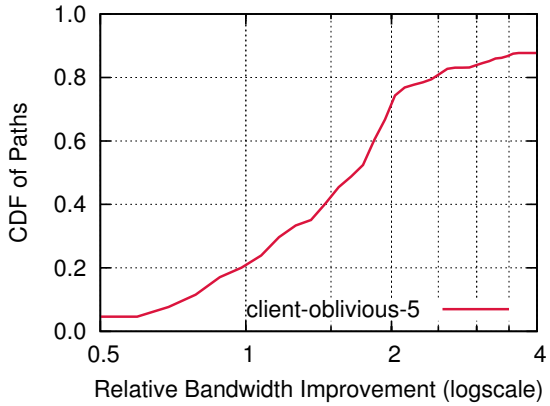

Figure 6: Detouring improves bandwidth even on arbitrary public Internet paths

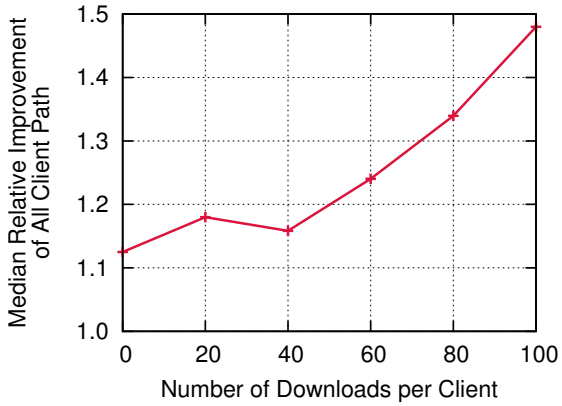

Figure 7: Adaptive detouring improves the quality of the detour set selection over time

formed best in our previous experiments.

Figure 6 shows the bandwidth detouring potential expressed as relative download improvement over the direct path. The detouring performance results are significantly better than those for the previous experiment using just PlanetLab destinations: the median bandwidth improvement factor is around 1.8×, and we manage to improve bandwidth successfully on 80% of paths. Significantly, 30% of paths find their bandwidth doubled or better. A possible explanation is that the performance of bandwidth detouring in the previous experiment is constrained by the destination web servers running on overloaded PlanetLab nodes.

4.1.2 Adaptive detour selection

By using adaptive detour selection, clients can improve upon the client-oblivious ranking of detours provided by the oracle to create client-specific sets. We evaluate our adaptive detouring selection process by having clients randomly download from our set of PlanetLab destinations using adaptive detouring (cf. Section 3.2.2). Each client performs up to 100 file downloads to customise the rank list $\mathcal{L}_i$. To test the improvement that adaptive detouring provides, we randomly choose 5 PlanetLab web servers as destination and observe the detouring improvement after an increasing number of file downloads per client. We use the top-5 detours from $\mathcal{L}_i$ to evaluate the performance of our adaptive selection detour mechanism in improving the bandwidth between the clients and the web servers.

In Figure 7, we plot the median improvement from the testing phase for 10 clients. The median relative improvement observed increases with the number of downloads performed. It reaches about 1.48× the direct path bandwidth after 100 downloads; without adaptive detouring the client would expect just a 12% increase. We also find that, eventually, our adaptive detouring strategy converges and provides the set of client-specific detours of Section 2.2.3.

4.2 Load balancing

The previous experiments have evaluated the performance of UKAIRO with a single client at a time. This gives a baseline view of the detouring potential on the Internet but it is not a realistic usage model. Next, we investigate how UKAIRO supports multiple detour clients simultaneously and load-balances between detour nodes.

We run between 1 and 20 simultaneous clients, which use 20 PlanetLab nodes as destinations, always testing all possible paths. We evaluate both the client-oblivious (client-oblivious-5) and the load-balanced (load-balanced-5) detour ranking strategies and plot the average relative bandwidth improvement across all paths in Figure 8.

The client-oblivious-5 approach initially performs better than the load-balanced-5 strategy, confirming our first experiment. As we increase the number of parallel clients there is, however, a clear downward trend in its performance. With 16 simultaneous clients, it fails to offer any performance improvements through detouring.

The performance of load-balanced-5 remains relatively stable over the tested range of parallel downloads, consistently representing a moderate average performance improvement. This matches our intuition that any ranking strategy that focuses on a small set of good detour nodes would suffer from limited scalability due to hot-spots, even with a small number of simultaneous clients.

We restricted ourselves to detouring paths of less than 40 Mbps because we found that only few of our de-

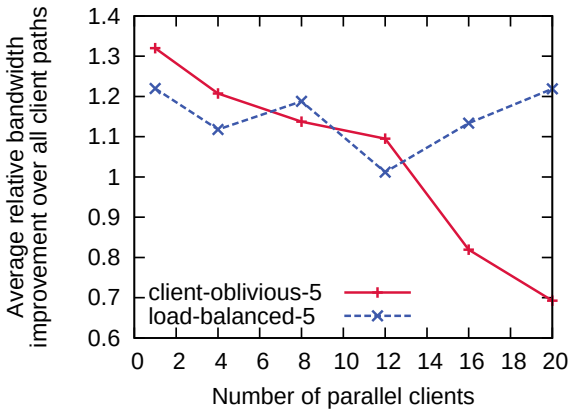

Figure 8: A diversified detour strategy can be effective for many clients simultaneously

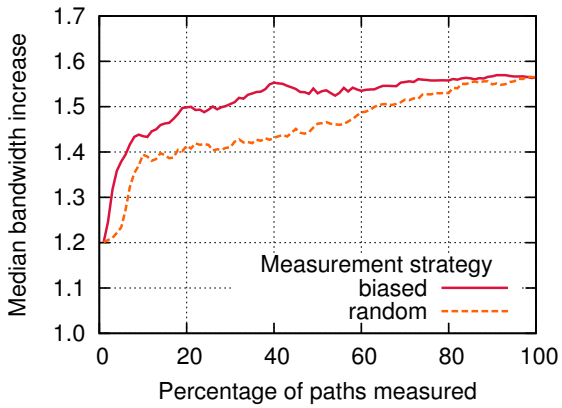

Figure 9: The biased measurement scheduler requires fewer measurements to achieve a given bandwidth increase

tour nodes are capable of forwarding connections at a faster rate, regardless of the path being measured. This suggests that any given detour node has a maximum throughput, which is likely to be the limiting factor when attempting to service higher numbers of clients simultaneously. This does imply that it should be possible to support greater numbers of parallel clients by adding more detouring nodes at popular locations, such as Internet peering points, subject to network capacity.

4.3 Measurement overhead

Measurements across the Internet consumes both bandwidth and time. In Section 3.1.1, we described a *biased* measurement approach to reduce the number of measurements within the overlay. To evaluate the performance of this measurement strategy, we deploy $k = 50$ detour nodes and compare the performance of the measurement scheduler with $t = 25$ to the following *random approach*. Given the set $\mathcal{P}$ of paths already measured, we uniformly choose a node i that is the end-point of a path in $\mathcal{P}$ and randomly choose a new path (i, j) to measure such that (i, j) is not in $\mathcal{P}$.

In both strategies, we calculate, after each measurement, the score d_l for each detour node to create the client-oblivious ranking. We then pick the top-5 detours from the ranked list of detour nodes to form the set of candidate detour nodes $\mathcal{D}_5$. We then perform parallel path selection on the 151 client nodes that download from each other. The bandwidth of a given path (i, j) is set to be the maximum of b_{ij} and $\hat{b}_{ilj}$ for all $l \in \mathcal{D}_5$.

In Figure 9, we show the median relative improvement due to bandwidth detouring as a function of the percentage of all paths measured between detour nodes. Without any measurement, both strategies begin with an unranked list of detours, thus each client behaves as if randomly selecting a detour. This still provides a 20% increase in a client's median bandwidth. However, reaching a 50% increase requires the measurement of 30% of paths if using our biased strategy and 70% with the random strategy. More significantly, the biased strategy provides a measurement set that achieves good bandwidth detouring with only approximately 40% of all-to-all measurements.

4.4 Discussion

We showed that a bandwidth detouring system can improve bandwidth to popular Internet destination. Due to the small number of required detour nodes and the longevity of detours, the bandwidth measurement overhead to discover good detour nodes for any client is acceptable. Parallel path selection among candidate detour nodes allows clients to choose the best detour path as part of an ongoing HTTP download. Our approach for load-balancing across a wider set of good detour nodes shows that it can alleviate hot-spots created by popular detour nodes.

Our system is not without its limitations. First, to achieve fast path selection by parallel downloads, UKAIRO handles exclusively HTTP traffic. We are currently looking into ways of scalably detouring more types of traffic. Second, using PlanetLab for our evaluation may raise concerns about the validity of the results. Because clients connect to a commercial site through PlanetLab, they may discover detours that would not be available had the detour node been on the commercial network. However, our measurements show that PlanetLab paths are not bandwidth optimal: we were able to discover bandwidth detours even between PlanetLab nodes.

Finally, an open question is what impact wide-spread UKAIRO deployment would have on the Internet as a whole. Detour routing disregards network-level routing policies because AS customers provide transit, which is forbidden in inter-domain routing. While this may seem undesirable at first, we already witness applications based on the overlay networks such as peer-to-peer file sharing systems [4] and CDNs [10] that follow a similar philosophy.

In the future, overlay networks are even more likely to make more intelligent path decisions taking path properties such as bandwidth into account. As a consequence, it is important for network operators to understand the implications of such decisions and provide mechanisms to handle their effects. Recent proposals for oracle services [38] operated by network providers that help peer-to-peer applications make ISP-friendly peer selection decisions are a first step towards a practical solution and could also be adopted for bandwidth detouring systems.

5 Related Work

Detouring was first proposed by Savage et al. [31]. They experimentally verify that detour paths with lower latency exist on the Internet due to violations of the triangle inequality in Internet routing. They outline an overlay architecture based on IP tunneling between edge nodes to redirect connections over such lower latency paths.

Another proposal for improving path properties using an overlay network is *OverQoS* [33]. OverQoS provides quality-of-service guarantees on network paths by changing the trade-off between bandwidth and packet loss. Instead of discovering alternate paths with desired properties, as done in our approach, it modifies the behaviour of the direct path through packet encoding.

Internet path measurement. Substantial effort has gone into the measurement of Internet path metrics. Several overlay networks aim to inform clients of the current condition of the Internet. Madhyastha et al. [25] propose *iPlane*, an Internet service that uses continuous network measurements from many vantage points to construct a global topology map of the Internet. iPlane uses this map to provide clients information about network path metrics such as latency, loss, and bandwidth. *iPlane nano* [26], a decentralised version of iPlane, partitions the map across multiple clients. Both systems leave it up to clients to decide how to use this information.

Detouring for availability. The *Resilient Overlay Network* (RON) [1] uses detour routing to improve the availability of Internet paths. The authors show that using an overlay network to route around Internet failures is practical for improving reliability. However, they discover detour paths through all-to-all measurements, which does not scale to larger system sizes. Gummadi et al. [11] make detouring for availability scalable by selecting detour nodes at random. However, their approach does not work for other path metrics such as latency or bandwidth.

Detouring for latency. Most successfully, detour routing can reduce Internet path latencies. *Peerwise* [24] discovers latency detours based the path embedding error in a network coordinate system. Users benefit from mutually advantages detours in exchange for their participation in the system and becoming detour nodes themselves. In our earlier work [14], we describe how the similarity of network-level AS paths can be exploited to discover detour paths with reduced latency. None of the latency detouring approaches work well for discovering bandwidth detours [24, 12].

Detouring for bandwidth. Recent work has investigated the use of overlay networks for improving path bandwidth. Jain et al. [17] use a technique for available bandwidth estimation to select paths for detouring in a video streaming application. To reduce measurement overheads, they describe a method for implicitly gathering measurements from the shape of traffic in a video stream. The *Bandwidth-Aware Routing Overlay Network* (BARON) [20] uses capacity measurements of network paths to discover good bandwidth detours. However, the effectiveness of this has not been evaluated experimentally as part of a deployed Internet system.

Su et al. [6] minimise the overhead of detour discovery by taking advantage of on-going measurements of Internet CDNs. While this approach works well in practice, it relies on the proprietary decision-making by commercial CDN operators without any control over properties of detour paths. In contrast, UKAIRO performs its own measurements and discovery of detour paths, operating transparently to clients.

6 Conclusions

UKAIRO is a bandwidth detouring system that reduces web download times by sending traffic along overlay paths with higher TCP throughput. UKAIRO operates in two phases: it first discovers detour paths among an overlay network of potential detour nodes and then it transparently diverts HTTP connections from its clients to their destinations via these detour nodes. Our evaluation on the PlanetLab test-bed shows that UKAIRO can provide paths of increased bandwidth to simultaneous Internet users who download content from popular web servers. UKAIRO discovers detours quickly and with little measurement overhead by keeping the set of potential detour nodes small and employing intelligent strategies, such as parallel path selection, for choosing the best de-

tour for each client.

An Internet bandwidth detouring system can help content providers that do not posses the resources to pay for a professional CDN. By shifting content delivery costs to the operators of detour nodes, it is possible to imagine a cooperative deployment of a bandwidth detouring service—in a similar spirit to the PeerWise system [23]—in which clients contribute resources by acting as detour nodes. In addition, by providing path diversity, UKAIRO may offer content providers more control in circumventing censorship (e.g., avoiding paths that cross certain geographic regions or autonomous systems), while still providing good download performance.

References

[1] ANDERSEN, D. G., BALAKRISHNAN, H., KAASHOEK, M. F., AND MORRIS, R. Resilient Overlay Networks. In *SOSP* (Chateau Lake Louise, Banff, Canada, Oct. 2001).

[2] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R. H., KONWINSKI, A., LEE, G., PATTERSON, D. A., RABKIN, A., STOICA, I., AND ZAHARIA, M. Above the Clouds: A Berkeley View of Cloud Computing. Internet draft, EECS University of California at Berkeley, Feb. 2009.

[3] BEAVER, D., KUMAR, S., LI, H. C., SOBEL, J., AND VAJGEL, P. Finding a Needle in Haystack: Facebook's Photo Storage. In *OSDI* (Vancouver, BC, Oct. 2010).

[4] BitTorrent. `http://www.bittorrent.com/`.

[5] CASTRO, M., DRUSCHEL, P., KERMARREC, A.-M., NANDI, A., ROWSTRON, A., AND SINGH, A. SplitStream: high-bandwidth multicast in cooperative environments. In *SOSP* (New York, NY, USA, 2003), ACM, pp. 298–313.

[6] CHOFFNES, D., AND BUSTAMANTE, F. On the Effectiveness of Measurement Reuse for Performance-Based Detouring. In *INFOCOM* (Rio de Janeiro, Brazil, Apr. 2009), pp. 693 –701.

[7] Amazon CloudFront. `http://aws.amazon.com/cloudfront/`.

[8] DOCSIS 3.0 Interface. `http://www.cablelabs.com/cablemodem/specifications/specifications30.html`.

[9] FIELDING, R., IRVINE, U., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T. Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616).

[10] FREEDMAN, M. J., FREUDENTHAL, E., AND MAZIÈRES, D. Democratizing content publication with coral. In *NSDI* (Berkeley, CA, USA, Mar. 2004), USENIX Association, pp. 18–18.

[11] GUMMADI, K. P., MADHYASTHA, H. V., GRIBBLE, S. D., LEVY, H. M., AND WETHERALL, D. Improving the Reliability of Internet Paths with One-hop Source Routing. In *OSDI* (San Francisco, CA, 2004).

[12] HADDOW, T., HO, S. W., LEDLIE, J., LUMEZANU, C., DRAIEF, M., AND PIETZUCH, P. On the Feasibility of Bandwidth Detouring. In *PAM* (Atlanta, GA, Mar. 2011).

[13] HAN, H., SHAKKOTTAI, S., HOLLOT, C. V., SRIKANT, R., AND TOWSLEY, D. Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in the internet. *IEEE/ACM Trans. Netw. 14* (Dec. 2006), 1260–1271.

[14] HO, S. W., HADDOW, T., LEDLIE, J., DRAIEF, M., AND PIETZUCH, P. Deconstructing internet paths: An approach for as-level detour route discovery. In *IPTPS* (Boston, MA, USA, 04/2009 2009).

[15] J. MOY. OSPF Version 2 (RFC 2328).

[16] JACOBSON, V., SMETTERS, D. K., THORNTON, J. D., PLASS, M. F., BRIGGS, N. H., AND BRAYNARD, R. L. Networking named content. In *CoNEXT* (Rome, Italy, 2009), ACM, pp. 1–12.

[17] JAIN, M., AND DOVROLIS, C. Path selection using available bandwidth estimation in overlay-based video streaming. *Comput. Netw. 52* (Aug. 2008), 2411–2418.

[18] JAN, R., AND OTT, T. J. Design and implementation of split TCP in the Linux kernel. In *Globecom* (San Franciso, CA, USA, 2006).

[19] KOSTIĆ, D., RODRIGUEZ, A., ALBRECHT, J., AND VAHDAT, A. Bullet: High Bandwidth Data Dissemination using an Overlay Mesh. In *SOSP* (Bolton Landing, NY, 2003).

[20] LEE, S.-J., BANERJEE, S., SHARMA, P., YALAGANDULA, P., AND BASU, S. Bandwidth-Aware Routing in Overlay Networks. In *INFOCOM* (Phoenix, AZ, 2008).

[21] LEECH, M., GANIS, M., LEE, Y., KURIS, R., KOBLAS, D., AND JONES, L. SOCKS Protocol Version 5 (RFC 1928).

[22] LEIGHTON, T. Improving performance on the internet. *Commun. ACM 52* (Feb. 2009), 44–51.

[23] LUMEZANU, C., BADEN, R., LEVIN, D., SPRING, N., AND BHATTACHARJEE, B. Symbiotic Relationships in Internet Routing Overlays. In *NSDI* (Boston, MA, USA, Apr. 2009).

[24] LUMEZANU, C., LEVIN, D., AND SPRING, N. PeerWise Discovery and Negotiation of Faster Paths. In *HotNets* (Atlanta, GA, Nov. 2007).

[25] MADHYASTHA, H. V., ISDAL, T., PIATEK, M., DIXON, C., ANDERSON, T. E., KRISHNAMURTHY, A., AND VENKATARAMANI, A. iPlane: An Information Plane for Distributed Services. In *OSDI* (Seattle, WA, Nov. 2006).

[26] MADHYASTHA, H. V., KATZ-BASSETT, E., ANDERSON, T., KRISHNAMURTHY, A., AND VENKATARAMANI, A. iPlane Nano: path prediction for peer-to-peer applications. In *NSDI* (Boston, MA, Apr. 2009), USENIX Association, pp. 137–152.

[27] MATHIS, M., SEMKE, J., MAHDAVI, J., AND OTT, T. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM 27* (July 1997), 67–82.

[28] Microsoft Security Updates. `http://www.microsoft.com/security/updates/bulletins/`.

[29] POSKANZER, J. thttpd-tiny/turbo/throttling HTTP server, Feb. 2000.

[30] REKHTER, Y., LI, T., AND HARES, S. Border Gateway Protocol 4 (RFC 4271).

[31] SAVAGE, S., ANDERSON, T., AGGARWAL, A., BECKER, D., CARDWELL, N., COLLINS, A., HOFFMAN, E., SNELL, J., VAHDAT, A., VOELKER, G., AND ZAHORJAN, J. Detour: Informed Internet Routing and Transport. *IEEE Micro 19*, 1 (January 1999), 50–59.

[32] STENBERG, D. libcurl - the multiprotocol file transfer library, 1996.

[33] SUBRAMANIAN, L., STOICA, I., BALAKRISHNAN, H., AND KATZ, R. H. OverQoS: An Overlay Based Architecture for Enhancing Internet QoS. In *NSDI* (San Francisco, CA, Mar. 2004).

[34] TELECOMMUNICATION STANDARDIZATION SECTOR. G.992.5 Annex C, Nov. 2010.

[35] THE PLANETLAB CONSORTIUM. PlanetLab. `http://www.planetlab.org`, 2003.

[36] TIRUMALA, A., QIN, F., DUGAN, J., FERGUSON, J., AND GIBBS, K. Iperf: The TCP/UDP bandwidth measurement tool. `http://dast.nlanr.net/Projects/Iperf/`, 2004.

[37] TOMO.M. Srelay: A Free SOCKS server for UNIX, 2000.

[38] XIE, H., YANG, Y. R., KRISHNAMURTHY, A., LIU, Y. G., AND SILBERSCHATZ, A. P4p: provider portal for applications. In *SIGCOMM* (Seattle, WA, USA, Aug. 2008), ACM, pp. 351–362.