Locating Internet Bottlenecks: Algorithms, Measurements, and Implications

Ningning Hu
Carnegie Mellon University
hnn@cs.cmu.edu

Li (Erran) Li
Bell Laboratories
erranlli@bell-labs.com

Zhuoqing Morley Mao
University of Michigan
zmao@eecs.umich.edu

Peter Steenkiste
Carnegie Mellon University
prs@cs.cmu.edu

Jia Wang
AT&T Labs – Research
jiawang@research.att.com

ABSTRACT

The ability to locate network bottlenecks along end-to-end paths on the Internet is of great interest to both network operators and researchers. For example, knowing where bottleneck links are, network operators can apply traffic engineering either at the interdomain or intradomain level to improve routing. Existing tools either fail to identify the *location* of bottlenecks, or generate a large amount of probing packets. In addition, they often require access to both end points. In this paper we present *Pathneck*, a tool that allows end users to efficiently and accurately locate the bottleneck link on an Internet path. Pathneck is based on a novel probing technique called Recursive Packet Train (RPT) and does not require access to the destination. We evaluate Pathneck using wide area Internet experiments and trace-driven emulation. In addition, we present the results of an extensive study on bottlenecks in the Internet using carefully selected, geographically diverse probing sources and destinations. We found that Pathneck can successfully detect bottlenecks for almost 80% of the Internet paths we probed. We also report our success in using the bottleneck location and bandwidth bounds provided by Pathneck to infer bottlenecks and to avoid bottlenecks in multihoming and overlay routing.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations — Network Monitoring

General Terms

Algorithms, Measurement, Experimentation

Keywords

Active probing, packet train, bottleneck location, available bandwidth

1. INTRODUCTION

The ability to locate network bottlenecks along Internet paths is very useful for both end users and Internet Service Providers (ISPs). End users can use it to estimate the performance of the network path to a given destination, while an ISP can use it to quickly locate network problems, or to guide traffic engineering either at the interdomain or intradomain level. Unfortunately, it is very hard to identify the location of bottlenecks unless one has access to link load information for *all* the relevant links. This is a problem, especially for regular users, because the design of the Internet does not provide explicit support for end users to gain information about the network internals. Existing active bandwidth probing tools also fall short. Typically they focus on end-to-end performance [20, 18, 26, 30, 36], while providing no *location* information for the bottleneck. Some tools do measure hop-by-hop performance [19, 10], but their measurement overhead is often very high.

In this paper, we present an active probing tool – *Pathneck* – based on a novel probing technique called Recursive Packet Train (RPT). It allows end users to efficiently and accurately locate bottleneck links on the Internet. *The key idea is to combine measurement packets and load packets in a single probing packet train.* Load packets emulate the behavior of regular data traffic while measurement packets trigger router responses to obtain the measurements. RPT relies on the fact that load packets interleave with competing traffic on the links along the path, thus changing the length of the packet train. By measuring the changes using the measurement packets, the position of congested links can be inferred. Two important properties of RPT are that it has low overhead and does not require access to the destination.

Equipped with Pathneck, we conducted extensive measurements on the Internet among carefully selected, geographically diverse probing sources and destinations to study the diversity and stability of bottlenecks on the Internet. We found that, contrary to the common assumption that most bottlenecks are edge or peering links, for certain probing sources, up to 40% of the bottleneck locations are within an AS. In terms of stability, we found that inter-AS bottlenecks are more stable than intra-AS bottlenecks, while AS-level bottlenecks are more stable than router-level bottlenecks. We also show how we can use bottleneck location information and rough bounds for the per-link available bandwidth to successfully infer the bottleneck locations for 54% of the paths for which we have enough measurement data. Finally, using Pathneck results from a diverse set of probing sources to randomly selected destinations, we found that over half of all the overlay routing attempts improve bottleneck available bandwidth. The utility of multihoming in improving available bandwidth is over 78%.

This paper is organized as follows. We first describe the Pathneck design in Section 2 and then validate the tool in Section 3. Using Pathneck, we probed a large number of Internet destinations to obtain several different data sets. We use this data to study the properties of Internet bottlenecks in Section 4, to infer bottleneck locations on the Internet in Section 5, and to study the implications for overlay routing and multihoming in Section 6. We discuss related work in Section 7. In Section 8 we summarize and discuss future work.

2. DESIGN OF PATHNECK

Our goal is to develop a light-weight, single-end-control bottleneck detection tool. In this section, we first provide some background on measuring available bandwidth and then describe the concept of Recursive Packet Trains and the algorithms used by Pathneck.

2.1 Measuring Available Bandwidth

In this paper, we define the *bottleneck link* of a network path as the link with the smallest available bandwidth, *i.e.,* the link that determines the end-to-end throughput on the path. The *available bandwidth* in this paper refers to the residual bandwidth, which is formally defined in [20, 18]. Informally, we define a *choke link* as any link that has a lower available bandwidth than the partial path from the source to that link. The upstream router for the choke link is called the *choke point* or *choke router*. The formal definition of choke link and choke point is as follows. Let us assume an end-to-end path from source $S = R_0$ to destination $D = R_n$ through routers $R_1, R_2, ..., R_{n-1}$. Link $L_i = (R_i, R_{i+1})$ has available bandwidth $A_i (0 \leq i < n)$. Using this notation, we define the set of *choke links* as:

$$CHOKE_L = \{L_k | \exists j, 0 \leq j < n, k = argmin_{0 \leq i \leq j} A_i\}$$

and the corresponding set of *choke points* (or *choke routers*) are

$$CHOKE_R = \{R_k | L_k \in CHOKE_L, 0 \leq k < n\}$$

Clearly, choke links will have less available bandwidth as they get closer to the destination, so the last choke link on the path will be the *bottleneck link* or the primary choke link. We will call the second to last choke link the *secondary choke link*, and the third to last one the *tertiary choke link*, *etc.*

Let us now review some earlier work on available bandwidth estimation. A number of projects have developed tools that estimate the available bandwidth along a network path [20, 18, 26, 30, 36, 13]. This is typically done by sending a probing packet train along the path and by measuring how competing traffic along the path affects the length of the packet train (or the gaps between the probing packets). Intuitively, when the packet train traverses a link where the available bandwidth is less than the transmission rate of the train, the length of the train, *i.e.,* the time interval between the head and tail packets in the train, will increase. This increase can be caused by higher packet transmission times (on low capacity links), or by interleaving with the background traffic (heavily loaded links). When the packet train traverses a link where the available bandwidth is higher than the packet train transmission rate, the train length should stay the same since there should be little or no queuing at that link. By sending a sequence of trains with different rates, it is possible to estimate the available bandwidth on the bottleneck link; details can be found in [18, 26]. Using the above definition, the links that increase the length of the packet train correspond to the choke links since they represent the links with the lowest available bandwidth on the partial path traveled by the train so far.

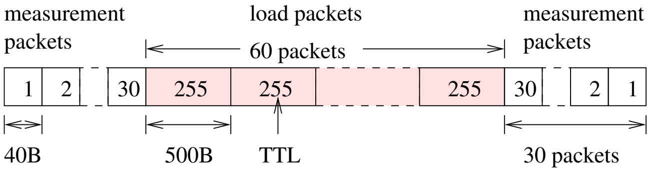


Figure 1: Recursive Packet Train (RPT).

Unfortunately, current techniques only estimate end-to-end available bandwidth since they can only measure the train length at the destination. In order to identify the bottleneck location, we need to measure the train length on *each* link. This information can be obtained with a novel packet train design, called a Recursive Packet Train, as we describe next.

2.2 Recursive Packet Train

Figure 1 shows an example of a Recursive Packet Train (RPT); every box is a UDP packet and the number in the box is its TTL value. The probing packet train is composed of two types of packets: measurement packets and load packets. *Measurement packets* are standard traceroute packets, *i.e.,* they are 60 byte UDP packets with properly filled-in payload fields. The figure shows 30 measurement packets at each end of the packet train, which allows us to measure network paths with up to 30 hops; more measurement packets should be used for longer paths. The TTL values of the measurement packets change linearly, as shown in the figure. *Load packets* are used to generate a packet train with a measurable length. As with the IGI/PTR tool [18], load packets should be large packets that represent an average traffic load. We use 500 byte packets as suggested in [18]. The number of load packets in the packet train determines the amount of background traffic that the train can interact with, so it pays off to use a fairly long train. In our experiment, we set it empirically in the range of 30 to 100. Automatically configuring the number of probing packets is future work.

The probing source sends the RPT packets in a back-to-back fashion. When they arrive at the first router, the first and the last packets of the train expire, since their TTL values are 1. As a result, the packets are dropped and the router sends two ICMP packets back to the source [7]. The other packets in the train are forwarded to the next router, after their TTL values are decremented. Due to the way the TTL values are set in the RPT, the above process is repeated on each subsequent router. The name "recursive" is used to highlight the repetitive nature of this process.

At the source, we can use *the time gap between the two ICMP packets from each router* to estimate the packet train length on the incoming link of that router. The reason is that the ICMP packets are generated when the head and tail packets of the train are dropped. Note that the measurement packets are much smaller than the total length of the train, so the change in packet train length due to the loss of measurement packets can be neglected. For example, in our default configuration, each measurement packet accounts for only 0.2% the packet train length. We will call the time difference between the arrival at the source of the two ICMP packets from the same router the *packet gap*.

2.3 Pathneck — The Inference Tool

RPT allows us to estimate the probing packet train length on each link along a path. We use the gap sequences obtained from a set of probing packet trains to identify the location of the bottleneck link. Pathneck detects the bottleneck link in three steps:

Step 1: *Labeling of gap sequences*. For each probing train, Path-

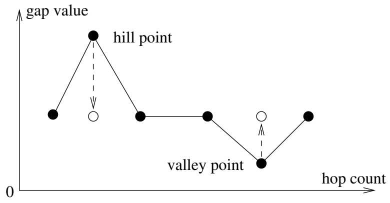


Figure 2: Hill and valley points.

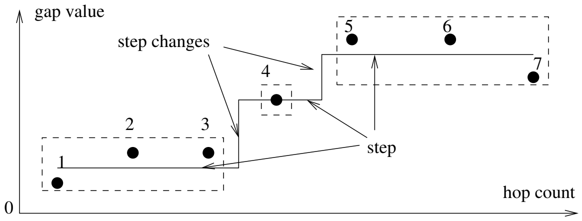


Figure 3: Matching the gap sequence to a step function.

neck labels the routers where the gap value increases significantly as candidate choke points.
Step 2: *Averaging across gap sequences*. Routers that are frequently labeled as candidate choke points by the probing trains in the set are identified as actual choke points.
Step 3: *Ranking choke points*. Pathneck ranks the choke points with respect to their packet train transmission rate.
In the remainder of this section, we describe in detail the algorithms used in each of the three steps.

2.3.1 Labeling of Gap Sequences

Under ideal circumstances, gap values only increase (if the available bandwidth on a link is not sufficient to sustain the rate of the incoming packet train) or stay the same (if the link has enough bandwidth for the incoming packet train), but it should never decrease. In reality, the burstiness of competing traffic and reverse path effects add noise to the gap sequence, so we preprocess the data before identifying candidate choke points. We first remove any data for routers from which we did not receive both ICMP packets. If we miss data for over half the routers, we discard the entire sequence. We then fix the *hill* and *valley* points where the gap value decreases in the gap sequence (Figure 2). A hill point is defined as p_2 in a three-point group (p_1, p_2, p_3) with gap values satisfying $g_1 < g_2 > g_3$. A valley point is defined in a similar way with $g_1 > g_2 < g_3$. Since in both cases, the decrease is short-term (one sample), we assume it is caused by noise and we replace g_2 with the closest neighboring gap value.

We now describe the core part of the labeling algorithm. The idea is to match the gap sequence to a step function (Figure 3), where each step corresponds to a candidate choke point. Given a gap sequence with len gap values, we want to identify the step function that is the best fit, where "best" is defined as the step function for which the sum of absolute difference between the gap sequence and the step function across all the points is minimal. We require the step function to have clearly defined steps, *i.e.,* all steps must be larger than a threshold ($step$) to filter out measurement noise. We use 100 $microseconds$ (μs) as the threshold. This value is relatively small compared with possible sources of error (to be discussed in Section 2.4), but we want to be conservative in identifying candidate choke points.

We use the following dynamic programming algorithm to identify the step function. Assume we have a gap subsequence between hop i and hop j: $g_i, ..., g_j$ $(i \leq j)$, and let us define $avg[i,j] = \sum_{k=i}^{j} g_k/(j-i+1)$, and the distance sum of the subsequence as $dist\_sum[i,j] = \sum_{k=i}^{j} |avg[i,j] - g_k|$. Let $opt[i,j,l]$ denote the minimal sum of the distance sums for the segments between hops i and j (including hops i and j), given that there are at most l steps. The key observation is that, given the optimal splitting of a subsequence, the splitting of any shorter internal subsequence delimited by two existing splitting points must be an optimal splitting for this internal subsequence. Therefore, $opt[i,j,l]$ can be recursively defined as the follows:

$$opt[i,j,l] = \begin{cases} dist\_sum[i,j] & l = 0 \ \&\ i \leq j, \\ min\{opt[i,j,l-1], opt2[i,j,l]\} & l > 0 \ \&\ i \leq j. \end{cases}$$

$$\begin{aligned} opt2[i,j,l] = \quad & min\{opt[i,k,l_1] + opt[k+1,j,l-l_1-1] : \\ & i \leq k < j, 0 \leq l_1 < l, \\ & |LS[i,k,l_1] - FS[k+1,j,l-l_1-1]| > step\} \end{aligned}$$

Here $LS[i,k,l_1]$ denotes the last step value of the optimal step function fitting the gap subsequence between i and k with at most l_1 steps, and $FS[k+1,j,l-l_1-1]$ denotes the first step value of the optimal step function fitting the gap subsequence between $k+1$ and j with at most $l - l_1 - 1$ steps.

The algorithm begins with $l = 0$ and then iteratively improves the solution by exploring larger values of l. Every time $opt2[i,j,l]$ is used to assign the value for $opt[i,j,l]$, a new splitting point k is created. The splitting point is recorded in a set $SP[i,j,l]$, which is the set of optimal splitting points for the subsequence between i and j using at most l splitting points. The algorithm returns $SP[0, len-1, len-1]$ as the set of optimal splitting points for the entire gap sequence. The time complexity of this algorithm is $O(len^5)$, which is acceptable considering the small value of len on the Internet. Since our goal is to detect the primary choke point, our implementation only returns the top three choke points with the largest three steps. If the algorithm does not find a valid splitting point, *i.e.,* $SP[0, len-1, len-1] = \emptyset$, it simply returns the source as the candidate choke point.

2.3.2 Averaging Across Gap Sequences

To filter out effects caused by bursty traffic on the forward and reverse paths, Pathneck uses results from multiple probing trains (*e.g.,* 6 to 10 probing trains) to compute *confidence* information for each candidate choke point. To avoid confusion, we will use the term *probing* for a single RPT run and the term *probing set* for a group of probings (generally 10 probings). The outcome of Pathneck is the summary result for a probing set.

For the optimal splitting of a gap sequence, let the sequence of step values be $sv_i (0 \leq i \leq M)$, where M is the total number of candidate choke points. The confidence for a candidate choke point i $(1 \leq i \leq M)$ is computed as

$$conf_i = \left| \frac{1}{sv_i} - \frac{1}{sv_{i-1}} \right| / \frac{1}{sv_{i-1}}$$

Intuitively, the confidence denotes the percentage of available bandwidth change implied by the gap value change. For the special case where the source is returned as the candidate choke point, we set its confidence value to 1.

Next, for each candidate choke point in the probing set we calculate d_rate as the frequency with which the candidate choke point appears in the probing set with $conf \geq 0.1$. Finally, we select those choke points with $d\_rate \geq 0.5$. Therefore, *the final choke points for a path are the candidates that appear with high confidence in at least half of the probings in the probing set*. In

Section 3.4, we quantify the sensitivity of Pathneck to these parameters.

2.3.3 Ranking Choke Points

For each path, we rank the choke points based on their average gap value in the probing set. The packet train transmission rate R is $R = ts/g$, where ts is the total size for all the packets in the train and g is the gap value. That is, the larger the gap value, the more the packet train was stretched out by the link, suggesting a lower available bandwidth on the corresponding link. As a result, we identify the choke point with the largest gap value as the bottleneck of the path. Note that since we cannot control the packet train structure at each hop, the RPT does not actually measure the available bandwidth on each link, so in some cases, Pathneck could select the wrong choke point as the bottleneck. For example, on a path where the "true" bottleneck is early in the path, the rate of the packet train leaving the bottleneck can be higher than the available bandwidth on the bottleneck link. As a result, a downstream link with slightly higher available bandwidth could also be identified as a choke point and our ranking algorithm will mistakenly select it as the bottleneck.

Note that our method of calculating the packet train transmission rate R is similar to that used by cprobe [13]. The difference is that cprobe estimates available bandwidth, while Pathneck estimates the location of the bottleneck link. Estimating available bandwidth in fact requires careful control of the inter-packet gap for the train [26, 18] which neither tool provides.

While Pathneck does not measure available bandwidth, we can use the average per-hop gap values to provide a rough upper or lower bound for the available bandwidth of each link. We consider three cases:

Case 1: For a choke link, *i.e.,* its gap increases, we know that the available bandwidth is less than the packet train rate. That is, the rate R computed above is an upper bound for the available bandwidth on the link.

Case 2: For links that maintain their gap relative to the previous link, the available bandwidth is higher than the packet train rate R, and we use R as a lower bound for the link available bandwidth.

Case 3: Some links may see a decrease in gap value. This decrease is probably due to temporary queuing caused by traffic burstiness, and according to the packet train model discussed in [18], we cannot say anything about the available bandwidth.

Considering that the data is noisy and that link available bandwidth is a dynamic property, these bounds should be viewed as very rough estimates. We provide a more detailed analysis for the bandwidth bounds on the bottleneck link in Section 3.3.

2.4 Pathneck Properties

Pathneck meets the design goals we identified earlier in this section. Pathneck does not need cooperation of the destination, so it can be widely used by regular users. Pathneck also has low overhead. Each measurement typically uses 6 to 10 probing trains of 30 to 100 load packets each. This is a very low overhead compared to existing tools such as pathchar [19] and BFind [10]. Finally, Pathneck is fast. For each probing train, it takes about one roundtrip time to get the result. However, to make sure we receive all the returned ICMP packets, Pathneck generally waits for 3 seconds — the longest roundtrip time we have observed on the Internet — after sending out the probing train, and then exits. Even in this case, a single probing takes less than 5 seconds. In addition, since each packet train probes all links, we get a consistent set of measurements. This, for example, allows Pathneck to identify multiple choke points and rank them. Note however that Pathneck is biased towards early choke points— once a choke point has increased the length of the packet train, Pathneck may no longer be able to "see" downstream links with higher or slightly lower available bandwidth.

A number of factors could influence the accuracy of Pathneck. First, we have to consider the ICMP packet generation time on routers. This time is different for different routers and possibly for different packets on the same router. As a result, the measured gap value for a router will not exactly match the packet train length at that router. Fortunately, measurements in [16] and [11] show that the ICMP packet generation time is pretty small; in most cases it is between $100\mu s$ and $500\mu s$. We will see later that over 95% of the gap changes of detected choke points in our measurements are larger than $500\mu s$. Therefore, while large differences in ICMP generation time can affect individual probings, they are unlikely to significantly affect Pathneck bottleneck results.

Second, as ICMP packets travel to the source, they may experience queueing delay caused by reverse path traffic. Since this delay can be different for different packets, it is a source of measurement error. We are not aware of any work that has quantified reverse path effects. In our algorithm, we try to reduce the impact of this factor by filtering out the measurement outliers. Note that if we had access to the destination, we might be able to estimate the impact of reverse path queueing.

Third, packet loss can reduce Pathneck's effectiveness. Load packet loss can affect RPT's ability to interleave with background traffic thus possibly affecting the correctness of the result. Lost measurement packets are detected by lost gap measurements. Note that it is unlikely that Pathneck would lose significant numbers of load packets without a similar loss of measurement packets. Considering the low probability of packet loss in general [23], we do not believe packet loss will affect Pathneck results.

Fourth, multi-path routing, which is sometimes used for load balancing, could also affect Pathneck. If a router forwards packets in the packet train to different next-hop routers, the gap measurements will become invalid. Pathneck can usually detect such cases by checking the source IP address of the ICMP responses. In our measurements, we do not use the gap values in such cases.

Pathneck also has some deployment limitations. First, we discovered that network firewalls often only forward 60 byte UDP packets that strictly conform to the packet payload format used by standard Unix traceroute implementation, while they drop any other UDP probing packets, including the load packets in our RPT. If the sender is behind such a firewall, Pathneck will not work. Similarly, if the destination is behind a firewall, no measurements for links behind the firewall can be obtained by Pathneck. Second, even without any firewalls, Pathneck may not be able to measure the packet train length on the last link, because the ICMP packets sent by the destination host cannot be used. In theory, the destination should generate a "destination port unreachable" ICMP message for each packet in the train. However, due to ICMP rate limiting, the destination network system will typically only generate ICMP packets for some of the probing packets, which often does not include the tail packet. Even if an ICMP packet is generated for both the head and tail packets, the *accumulated* ICMP generation time for the whole packet train makes the returned interval worthless. Of course, if we have the cooperation of the destination, we can get a valid gap measurement for the last hop by using a valid port number, thus avoiding the ICMP responses for the load packets.

3. VALIDATION

We use both Internet paths and the Emulab testbed [3] to evaluate Pathneck. Internet experiments are necessary to study Pathneck

Table 1: Bottlenecks detected on Abilene paths.

Probe destination	d_rate (Utah/CMU)	Bottleneck router IP	AS path ($AS1$-$AS2$)†
calren2§	0.71/0.70	137.145.202.126	2150-2150
princeton§	0.64/0.67	198.32.42.209	10466-10466
sox§	0.62/0.56	199.77.194.41	10490-10490
ogig§	0.71/0.72	205.124.237.10 198.32.8.13	210-4600 (Utah) 11537-4600 (CMU)

† $AS1$ is bottleneck router's AS#, $AS2$ is its post-hop router's AS#.
§ calren = `www.calren2.net`, princeton = `www.princeton.edu`,
§ sox = `www.sox.net`, ogig = `www.ogig.net`.

with realistic background traffic, while the Emulab testbed provides a fully controlled environment that allows us to evaluate Pathneck with known traffic loads. Besides the detection accuracy, we also examine the accuracy of the Pathneck bandwidth bounds and the sensitivity of Pathneck to its configuration parameters. Our validation does not study the impact of the ICMP generation time[1].

3.1 Internet Validation

For a thorough evaluation of Pathneck on Internet paths, we would need to know the actual available bandwidth on all the links of a network path. This information is impossible to obtain for most operational networks. The Abilene backbone, however, publishes its backbone topology and traffic load (5-minute SNMP statistics) [1], so we decided to probe Abilene paths.

The experiment is carried out as follows. We used two sources: a host at the University of Utah and a host at Carnegie Mellon University. Based on Abilene's backbone topology, we chose 22 probing destinations for each probing source. We make sure that each of the 11 major routers on the Abilene backbone is included in at least one probing path. From each probing source, we probed every destination 100 times, with a 2-second interval between two consecutive probings. To avoid interference, the experiments conducted at Utah and at CMU were run at different times.

Using $conf \geq 0.1$ and $d\_rate \geq 0.5$, we only detected 5 non-first-hop bottleneck links on the Abilene paths (Table 1). This is not surprising since Abilene paths are known to be over-provisioned, and we selected paths with many hops inside the Abilene core. The d_rate values for the 100 probes originating from Utah and CMU are very similar, possibly because they observed similar congestion conditions. By examining the IP addresses, we found that in 3 of the 4 cases (`www.ogig.net` is the exception), both the Utah and CMU based probings are passing through the same bottleneck link close to the destination; an explanation is that these bottlenecks are very stable, possibly because they are constrained by link capacity. Unfortunately, all three bottlenecks are outside Abilene, so we do not have the load data.

For the path to `www.ogig.net`, the bottleneck links appear to be two different peering links going to AS4600. For the path from CMU to `www.ogig.net`, the outgoing link of the bottleneck router 198.32.163.13 is an OC-3 link. Based on the link capacities and SNMP data, we are sure that the OC-3 link is indeed the bottleneck. We do not have the SNMP data for the Utah links, so we cannot validate the results for the path from Utah to `www.ogig.net`.

3.2 Testbed Validation

We use the Emulab testbed to study the detailed properties of

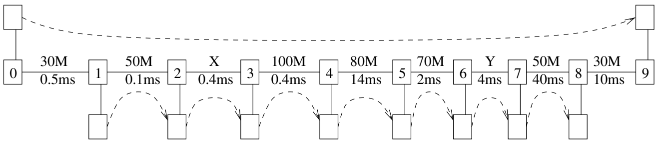


Figure 4: Testbed configuration.

Table 2: The testbed validation experiments

#	X	Y	Trace	Comments
1	50	20	*light-trace* on all	Capacity-determined bottleneck
2	50	50	35Mbps *exponential-load* on Y, *light-trace* otherwise	Load-determined bottleneck
3	20	20	*heavy-trace* on Y, *light-trace* otherwise	Two-bottleneck case
4	20	20	*heavy-trace* on X, *light-trace* otherwise	Two-bottleneck case
5	50	20	30% *exponential-load* on both directions	The impact of reverse traffic

Pathneck. Since Pathneck is a path-oriented measurement tool, we use a linear topology (Figure 4). Nodes 0 and 9 are the probing source and destination, while nodes 1-8 are intermediate routers. The link delays are roughly set based on a traceroute measurement from a CMU host to `www.yahoo.com`. The link capacities are configured using the Dummynet [2] package. The capacities for links X and Y depend on the scenarios. Note that all the testbed nodes are PCs, not routers, so their properties such as the ICMP generation time are different from those of routers. As a result, the testbed experiments do not consider some of the router related factors.

The dashed arrows in Figure 4 represent background traffic. The background traffic is generated based on two real packet traces, called *light-trace* and *heavy-trace*. The *light-trace* is a sampled trace (using prefix filters on the source and destination IP addresses) collected in front of a corporate network. The traffic load varies from around 500Kbps to 6Mbps, with a median load of 2Mbps. The *heavy-trace* is a sampled trace from an outgoing link of a data center connected to a tier-1 ISP. The traffic load varies from 4Mbps to 36Mbps, with a median load of 8Mbps. We also use a simple UDP traffic generator whose instantaneous load follows an exponential distribution. We will refer to the load from this generator as *exponential-load*. By assigning different traces to different links, we can set up different evaluation scenarios. Since all the background traffic flows used in the testbed evaluation are very bursty, they result in very challenging scenarios.

Table 2 lists the configurations of five scenarios that allow us to analyze all the important properties of Pathneck. For each scenario, we use Pathneck to send 100 probing trains. Since these scenario are used for validation, we only use the results for which we received all ICMP packets, so the percentage of valid probing is lower than usual. During the probings, we collected detailed load data on each of the routers allowing us to compare the probing results with the actual link load. We look at Pathneck performance for both probing sets (*i.e.,* result for 10 consecutive probings as reported by Pathneck) and individual probings. For probing sets, we use $conf \geq 0.1$ and $d\_rate \geq 0.5$ to identify choke points. The real background traffic load is computed as the average load for the interval that includes the 10 probes, which is around 60 seconds. For individual probings, we only use $conf \geq 0.1$ for filtering, and the load is computed using a $20ms$ packet trace centered around the probing packets, *i.e.,* we use the instantaneous load.

[1] A meaningful study of the ICMP impact requires access to different types of routers with real traffic load, but we do not have access to such facilities.

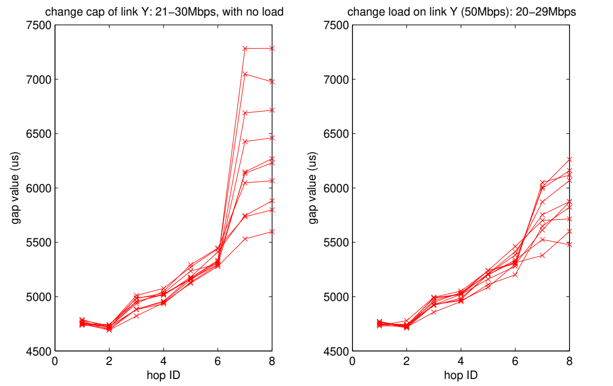


Figure 5: Comparing the gap sequences for capacity (left) and load-determined (right) bottlenecks.

3.2.1 Experiment 1 — Capacity-determined Bottleneck

In this experiment, we set the capacities of X and Y to 50Mbps and 20Mbps, and use *light-trace* on all the links; the starting times within the trace are randomly selected. All 100 probings detect hop 6 (*i.e.,* link Y) as the bottleneck. All other candidate choke points are filtered out because of a low confidence value (*i.e.,* $conf < 0.1$). Obviously, the detection results for the probing sets are also 100% accurate.

This experiment represents the easiest scenario for Pathneck, *i.e.,* the bottleneck is determined by the link capacity, and the background traffic is not heavy enough to affect the bottleneck location. This is however an important scenario on the Internet. A large fraction of the Internet paths fall into this category because only a limited number of link capacities are widely used and the capacity differences tend to be large.

3.2.2 Experiment 2 — Load-determined Bottleneck

Besides capacity, the other factor that affects the bottleneck position is the link load. In this experiment, we set the capacities of both X and Y to 50Mbps. We use the 35Mbps *exponential-load* on Y and the *light-trace* on other links, so the difference in traffic load on X and Y determines the bottleneck. Out of 100 probings, 23 had to be discarded due to ICMP packet loss. Using the remaining 77 cases, the probing sets always correctly identify Y as the bottleneck link. Of the individual probings, 69 probings correctly detect Y as the top choke link, 2 probings pick link $\langle R7, R8\rangle$ (*i.e.,* the link after Y) as the top choke link and Y is detected as the secondary choke link. 6 probings miss the real bottleneck. In summary, the accuracy for individual probings is 89.6%.

3.2.3 Comparing the Impact of Capacity and Load

To better understand the impact of link capacity and load in determining the bottleneck, we conducted two sets of simplified experiments using configurations similar to those used in experiments 1 and 2. Figure 5 shows the gap measurements as a function of the hop count (x axis). In the left figure, we fix the capacity of X to 50Mbps and change the capacity of Y from 21Mbps to 30Mbps with a step size of 1Mbps; no background traffic is added on any link. In the right figure, we set the capacities of both X and Y to 50Mbps. We apply different CBR loads to Y (changing from 29Mbps to 20Mbps) while there is no load on the other links. For each configuration, we executed 10 probings. The two figures plot the median gap value for each hop; for most points, the 30-70 percentile interval is under $200\mu s$.

In both configurations, the bottleneck available bandwidth changes in exactly the same way, *i.e.,* it increases from 21Mbps to 30Mbps. However, the gap sequences are quite different. The gap increases in the left figure are regular and match the capacity changes, since the length of the packet train is strictly set by the link capacity. In the right figure, the gaps at the destination are less regular and smaller. Specifically, they do not reflect the available bandwidth on the link (*i.e.,* the packet train rate exceeds the available bandwidth). The reason is that the back-to-back probing packets compete un-fairly with the background traffic and they can miss some of the background traffic that should be captured. This observation is consistent with the principle behind TOPP [26] and IGI/PTR [18], which states that the probing rate should be set properly to accurately measure the available bandwidth. This explains why Pathneck's packet train rate at the destination provides only an upper bound on the available bandwidth. Figure 5 shows that the upper bound will be tighter for capacity-determined bottlenecks than for load-determined bottlenecks. The fact that the gap changes in the right figure are less regular than that in the left figure also confirms that capacity-determined bottlenecks are easier to detect than load-determined bottlenecks.

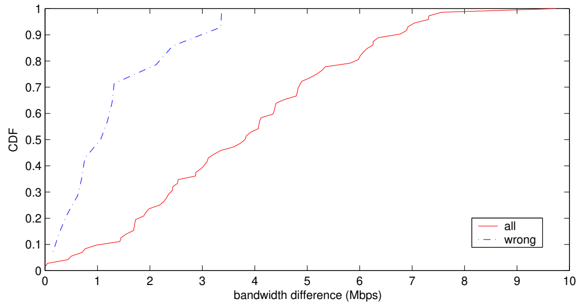


Figure 6: Cumulative distribution of bandwidth difference in experiment 3.

3.2.4 Experiments 3 & 4 — Two Bottlenecks

In these two experiments, we set the capacities of both X and Y to 20Mbps, so we have two low capacity links and the bottleneck location will be determined by load. In experiment 3, we use the *heavy-trace* for Y and the *light-trace* for other links. The probing set results are always correct, *i.e.,* Y is detected as the bottleneck. When we look at the 86 valid individual probings, we find that X is the real bottleneck in 7 cases; in each case Pathneck successfully identifies X as the *only* choke link, and thus the bottleneck. In the remaining 79 cases, Y is the real bottleneck. Pathneck correctly identifies Y in 65 probings. In the other 14 probings, Pathneck identifies X as the only choke link, *i.e.,* Pathneck missed the real bottleneck link Y. The raw packet traces show that in these 14 incorrect cases, the bandwidth difference between X and Y is very small. This is confirmed by Figure 6, which shows the cumulative distribution of the available bandwidth difference between X and Y for the 14 wrong cases (the dashed curve), and for all 86 cases (the solid curve). The result shows that if two links have similar available bandwidth, Pathneck has a bias towards the first link. This is because the probing packet train has already been stretched by the first choke link X, so the second choke link Y can be hidden.

As a comparison, we apply the *heavy-trace* to both X and Y in experiment 4. 67 out of the 77 valid probings correctly identify X as the bottleneck; 2 probings correctly identify Y as the bottleneck; and 8 probings miss the real bottleneck link Y and identify X as the only bottleneck. Again, if multiple links have similar available bandwidth, we observe the same bias towards the early link.

Table 3: **The number of times of each hop being a candidate choke point.**

Router	1	2	3	4	5	6	7
$conf \geq 0.1$	24	18	5	21	20	75	34
$d\_rate \geq 0.5$	6	0	0	2	0	85	36

3.2.5 Experiment 5 — Reverse Path Queuing

To study the effect of reverse path queuing, we set the capacities of X and Y to 50Mbps and 20Mbps, and apply *exponential-load* in both directions on all links (except the two edge links). The average load on each link is set to 30% of the link capacity. We had 98 valid probings. The second row in Table 3 lists the number of times that each hop is detected as a candidate choke point (*i.e.,* with $conf \geq 0.1$). We observe that each hop becomes a candidate choke point in some probings, so reverse path traffic does affect the detection accuracy of RPTs.

However, the use of probing sets reduces the impact of reverse path traffic. We analyzed the 98 valid probings as 89 sets of 10 consecutive probings each. The last row of Table 3 shows how often links are identified as choke points ($d\_rate \geq 0.5$) by a probing set. The real bottleneck, hop 6, is most frequently identified as the actual bottleneck (last choke point), although in some cases, the next hop (*i.e.,* hop 7) is also a choke point and is thus selected as the bottleneck. This is a result of reverse path traffic. Normally, the train length on hop 7 should be the same as on hop 6. However, if reverse path traffic reduces the gap between the hop 6 ICMP packets, or increases the gap between the hop 7 ICMP packets, it will appear as if the train length has increased and hop 7 will be labeled as a choke point. We hope to tune the detection algorithm to reduce the impact of this factor as part of future work.

3.3 Validation of Bandwidth Bounds

A number of groups have shown that packet trains can be used to estimate the available bandwidth of a network path [26, 18, 21]. However, the source has to carefully control the inter-packet gap, and since Pathneck sends the probing packets back-to-back, it cannot, in general, measure the available bandwidth of a path. Instead, as described in Section 2.3, the packet train rate at the bottleneck link can provide a rough upper bound for the available bandwidth. In this section, we compare the upper bound on available bandwidth on the bottleneck link reported by Pathneck with end-to-end available bandwidth measurements obtained using IGI/PTR [18] and Pathload [21].

Since both IGI/PTR and Pathload need two-end control, we used 10 RON nodes for our experiments, as listed in the "BW" column in Table 4; this results in 90 network paths for our experiment. On each RON path, we obtain 10 Pathneck probings, 5 IGI/PTR measurements, and 1 Pathload measurement[2]. The estimation for the upper bound in Pathneck was done as follows. If a bottleneck can be detected from the 10 probings, we use the median packet train transmission rate on that bottleneck. Otherwise, we use the largest gap value in each probing to calculate the packet train rate and use the median train rate of the 10 probings as the upper bound.

Figure 7 compares the average of the available bandwidth estimates provided by IGI, PTR, and Pathload (x axis) with the upper bound for the available bandwidth provided by Pathneck (y axis). The measurements are roughly clustered in three areas. For low bandwidth paths (bottom left corner), Pathneck provides a fairly tight upper bound for the available bandwidth on the bottleneck link, as measured by IGI, PTR, and Pathload. In the upper left region, there are 9 low bandwidth paths for which the upper bound provided by Pathneck is significantly higher than the available bandwidth measured by IGI, PTR, and Pathload. Analysis

[2] We force Pathload to stop after 10 fleets of probing. If Pathload has not converged, we use the average of the last 3 probings as the available bandwidth estimate.

Table 4: Probing sources from PlanetLab (PL) and RON.

ID	Probing Source	AS Number	Location	Upstream Provider(s)	Test-bed	BW	GE	ST	OV	MH
1	aros	6521	UT	701	RON	√	√		√	
2	ashburn	7911	DC	2914	PL		√		√	
3	bkly-cs	25	CA	2150, 3356, 11423, 16631	PL		√		√	√
4	columbia	14	NY	6395	PL		√		√	
5	diku	1835	Denmark	2603	PL		√		√	
6	emulab	17055	UT	210	–		√		√	
7	frankfurt	3356	Germany	1239, 7018	PL		√		√	
8	grouse	71	GA	1239, 7018	PL		√		√	
9	gs274	9	PA	5050	–		√		√	
10	bkly-intel	7018	CA	1239	PL		√		√	
11	intel	7018	CA	1239	RON		√		√	
12	jfk1	3549	NY	1239, 7018	RON	√	√		√	
13	jhu	5723	MD	7018	PL		√		√	√
14	nbgisp	18473	OR	3356	PL		√		√	
15	nortel	11085	Canada	14177	RON	√	√		√	
16	nyu	12	NY	6517, 7018	RON	√	√		√	
17	princeton	88	NJ	7018	PL		√		√	√
18	purdue	17	IN	19782	PL		√		√	
29	rpi	91	NY	6395	PL		√		√	√
20	uga	3479	GA	16631	PL		√		√	
21	umass	1249	MA	2914	PL		√		√	
22	unm	3388	NM	1239	PL		√		√	
23	utah	17055	UT	210	PL		√		√	
24	uw-cs	73	WA	101	PL		√		√	
25	vineyard	10781	MA	209, 6347	RON		√		√	
26	rutgers	46	NJ	7018	PL			√		
27	harvard	11	MA	16631	PL			√		
28	depaul	20130	CH	6325, 16631	PL			√		√
29	toronto	239	Canada	16631	PL			√		
30	halifax	6509	Canada	11537	PL			√		
31	unb	611	Canada	855	PL			√		
32	umd	27	MD	10086	PL			√		√
33	dartmouth	10755	NH	13674	PL			√		√
34	virginia	225	VA	1239	PL			√		
35	upenn	55	PA	16631	PL			√		
36	cornell	26	NY	6395	PL				√	
37	mazu1	3356	MA	7018	RON				√	
38	kaist	1781	Korea	9318	PL					√
39	cam-uk	786	UK	8918	PL					√
40	ucsc	5739	CA	2152	PL					√
41	ku	2496	KS	11317	PL					√
42	snu-kr	9488	Korea	4766	PL					√
43	bu	111	MA	209	PL					√
44	northwestern	103	CH	6325	PL					√
45	cmu	9	PA	5050	PL					√
46	mit-pl	3	MA	1	PL					√
47	stanford	32	CA	16631	PL					√
48	wustl	2552	MO	2914	PL					√
49	msu	237	MI	3561	PL					√
50	uky	10437	KY	209	PL					√
51	ac-uk	786	UK	3356	PL					√
52	umich	237	MI	3561	PL					√
53	cornell	26	NY	6395	RON	√				
54	lulea	2831	Sweden	1653	RON	√				
55	ana1	3549	CA	1239, 7018	RON	√				
56	ccicom	13649	UT	3356, 19092	RON	√				
57	ucsd	7377	CA	2152	RON	√				
58	utah	17055	UT	210	RON	√				

BW: measurements for bandwidth estimation; GE: measurements for general properties;
ST: measurements for stability analysis; OV: measurements for overlay analysis;
MH: measurements for multihoming analysis. "–" denotes the two probing hosts obtained privately.

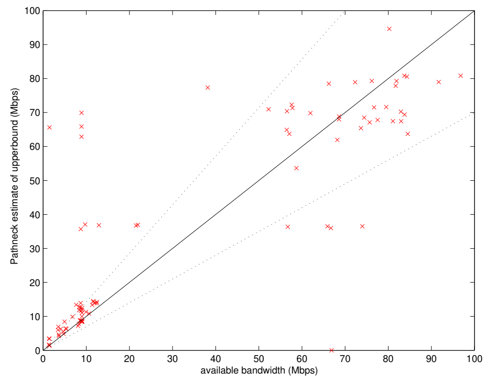


Figure 7: Comparison between the bandwidth from Pathneck with the available bandwidth measurement from IGI/PTR and Pathload.

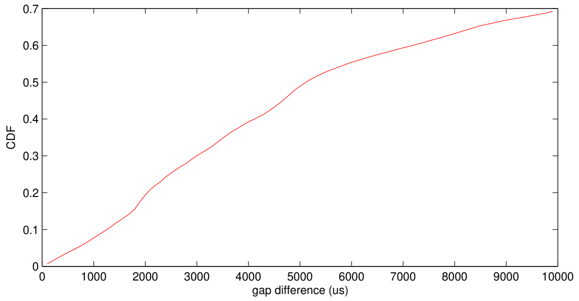


Figure 8: Distribution of step size on the choke point.

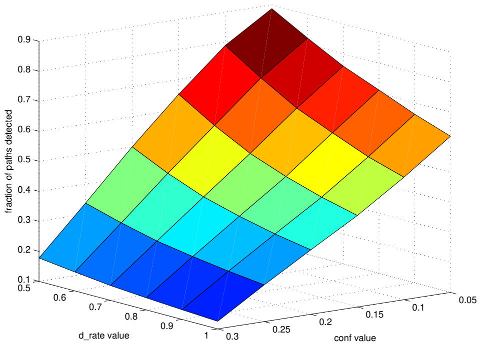


Figure 9: Sensitivity of Pathneck to the values of $conf$ and d_rate.

shows that the bottleneck link is the last link, which is not visible to Pathneck. Instead, Pathneck identifies an earlier link, which has a higher bandwidth, as the bottleneck.

The third cluster corresponds to high bandwidth paths (upper right corner). Since the current available bandwidth tools have a relative measurement error around 30% [18], we show the two 30% error margins as dotted lines in Figure 7. We consider the upper bound for the available bandwidth provided by Pathneck to be *valid* if it falls within these error bounds. We find that most upper bounds are valid. Only 5 data points fall outside of the region defined by the two 30% lines. Further analysis shows that the data point above the region corresponds to a path with a bottleneck on the last link, similar to the cases mentioned above. The four data points below the region belong to paths with the same source node (lulea). We have not been able to determine why the Pathneck bound is too low.

3.4 Impact of Configuration Parameters

The Pathneck algorithms described in Section 2.3 use three configuration parameters: the threshold used to pick candidate choke points ($step$ = 100μs), the confidence value ($conf$ = 0.1), and the detection rate (d_rate = 0.5). We now investigate the sensitivity of Pathneck to the value of these parameters.

To show how the 100μs threshold for the step size affects the algorithm, we calculated the cumulative distribution function for the step sizes for the choke points detected in the "GE" set of Internet measurements (Table 4, to be described in Section 4.1). Figure 8 shows that over 90% of the choke points have gap increases larger than 1000μs, while fewer than 1% of the choke points have gap increases around 100μs. Clearly, changing the step threshold to a larger value (*e.g.,* 500μs) will not change our results significantly.

To understand the impact of $conf$ and d_rate, we reran the Pathneck detection algorithm by varying $conf$ from 0.05 to 0.3 and d_rate from 0.5 to 1. Figure 9 plots the percentage of paths with at least one choke point that satisfies both the $conf$ and d_rate thresholds. The result shows that, as we increase $conf$ and d_rate, fewer paths have identifiable choke points. This is exactly what we would expect. With higher values for $conf$ and d_rate, it becomes more difficult for a link to be consistently identified as a choke link. The fact that the results are much less sensitive to d_rate than $conf$ shows that most of the choke point locations are fairly stable within a probing set (short time duration).

The available bandwidth of the links on a path and the location of both choke points and the bottleneck are dynamic properties. The Pathneck probing trains effectively sample these properties, but the results are subject to noise. Figure 9 shows the tradeoffs involved in using these samples to estimate the choke point locations. Using high values for $conf$ and d_rate will result in a small number of stable choke points, while using lower values will also identify more transient choke points. Clearly the right choice will depend on how the data is used. We see that for our choice of $conf$ and d_rate values, 0.1 and 0.5, Pathneck can clearly identify one or more choke points on almost 80% of the paths we probed. The graph suggests that our selection of thresholds corresponds to a fairly liberal notion of choke point.

4. INTERNET BOTTLENECK MEASUREMENT

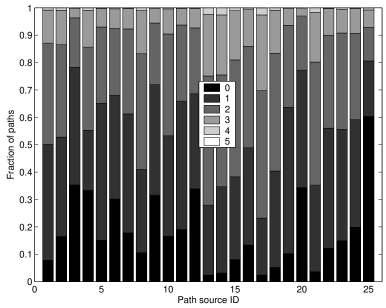
It has been a common assumption in many studies that bottlenecks often occur at edge links and peering links. In this section, we test this popular assumption using Pathneck, which is sufficiently light-weight to conduct large scale measurements on the Internet. Using the same set of data, we also look at the stability of Internet bottlenecks.

4.1 Data Collection

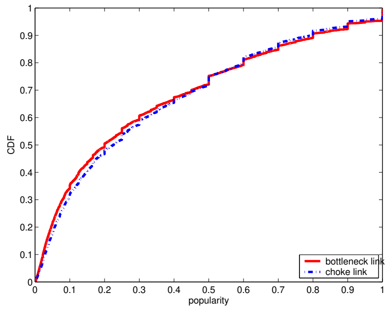
We chose a set of geographically diverse nodes from Planetlab [4] and RON [31] as probing sources. Table 4 lists all the nodes that we used for collecting measurement data for the analysis in this paper. Among them, "GE" is used in Sections 4.2, 4.3, and 5, "ST" is used in Section 4.4, "OV" is used in Section 6.1, and "MH" is used in Section 6.2. These nodes reside in 46 distinct ASes and are connected to 30 distinct upstream providers, providing good coverage for north America and parts of Europe.

We carefully chose a large set of destinations to cover as many distinct inter-AS links as possible. Our algorithm selects destination IP addresses using the local BGP routing table information of the probe source, using a similar method as described in [24]. In most cases, we do not have access to the local BGP table for the sources, but we almost always can obtain the BGP table for their upstream provider, for example from public BGP data sources such as RouteViews [6]. The upstream provider information can be identified by performing traceroute to a few randomly chosen locations such as `www.google.com` and `www.cnn.com` from the probe sources. In the case of multihomed source networks, we may not be able to obtain the complete set of upstream providers.

Given a routing table, we first pick a ".1" or ".129" IP address for each prefix possible. The prefixes that are completely covered by their subnets are not selected. We then reduce the set of IP addresses by eliminating the ones whose AS path starting from the probe source are part of other AS paths. Here we make the simplification that there is only a single inter-AS link between each pair of adjacent ASes. As the core of the Internet is repeatedly traversed for the over 3,000 destinations we selected for each source, we would expect that each of these inter-AS links is traversed many times by our probing packets. Note that the destination IP addresses



(a) Distribution of number of choke links per source.



(b) Popularity of choke links and bottleneck links.

Figure 10: Distribution and popularity of choke links.

obtained from this procedure do *not* necessarily correspond to real end hosts.

In our experiments, each source node probes each destination once using Pathneck. Pathneck is configured to use a probing set of 10 probing trains and it then uses the results of the probing set to calculate the location of the choke points as well as a rough estimate for the available bandwidth for the corresponding choke links. We again use the $conf \geq 0.1$ and $d\_rate \geq 0.5$ thresholds to select choke points. Due to the small measurement time, we were able to finish probing to around 3,500 destinations within 2 days.

4.2 Popularity

As described in previous sections, Pathneck is able to detect multiple choke links on a network path. In our measurements, Pathneck detected up to 5 choke links per path. Figure 10(a) shows the number of paths that have 0 to 5 choke links. We found that, for all probing sources, fewer than 2% of the paths report more than 3 choke links. We also noticed that a good portion of the paths have no choke link. This number varies from 3% to 60% across different probing sources. The reason why Pathneck cannot detect a choke link is generally that the traffic on those paths is too bursty so no link meets the $conf \geq 0.1$ and $d\_rate \geq 0.5$ criteria.

In our measurements, we observe that some links are detected as choke links in a large number of paths. For a link b that is identified as a choke link by at least one Pathneck probe, let $NumProbe(b)$ denote the total number of probes that traverse b and let $NumPositiveProbe(b)$ denote the total number of probes that detect b as a choke link. We compute the $Popularity(b)$ of link b as follows:

$$Popularity(b) = \frac{NumPositiveProbe(b)}{NumProbe(b)}$$

The popularity of a bottleneck link is defined similarly. Figure 10(b) shows the cumulative distribution of the popularity of choke links (dashed curve) and bottleneck links (solid curve) in our measurements. We observe that half of the choke links are detected in 20% or less of the Pathneck probings that traverse them. About 5% of the choke links are detected by all the probes. The same observations hold for the popularity of bottleneck links.

4.3 Location

In general, a link b is considered to be an *intra-AS link* if both ends of b belong to the same AS; otherwise, b is an *inter-AS link*. In practice, it is surprisingly difficult to identify a link at the boundary between two ASes due to the naming convention [24] that is currently used by some service providers. In our experiments, we first use the method described in [24] to map an IP address to its AS. We then classify a link b into one of the following three categories: (i) *Intra-AS link*. A link b is intra-AS if both ends of b and its adjacent links belong to the same AS. Note that we are very conservative in requiring that intra-AS links fully reside inside a network. (ii) *Inter0-AS link*. A link b is inter0-AS if the ends of b do not belong to the same AS. The link b is likely to be an inter-AS link, but it is also possible that b is one hop away from the actual inter-AS link. (iii) *Inter1-AS link*. A link b is inter1-AS if both ends of b belong to the same AS and it is adjacent to an inter0-AS link. In this case, b appears to be one hop away from the link where AS numbers change, but it might be the actual inter-AS link. Note that, using our definitions, the inter0-AS links and inter1-AS links should contain all the inter-AS links and some intra-AS links that are one hop away from the inter-AS links.

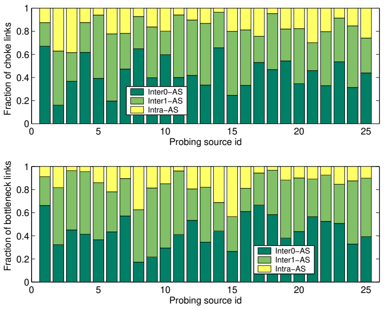
Figure 11(a) shows the distribution of choke links and bottleneck links across these three categories. We observe that for some probing sources up to 40% of both the bottleneck links and choke links occur at intra-AS links. Considering our very conservative definition of intra-AS link, this is surprising, given the widely used assumption that bottlenecks often occur at the boundary links between networks.

For a choke link b in a probing set P, we compute its *normalized location* (denoted by $NL(b, P)$) on the corresponding network path in the following way. Let $A_1, A_2, ..., A_k$ denote the AS-level path, where k is the length of the AS path. (i) If b is in the i-th AS along the path, then $NL(b, P) = i/k$. (ii) If b is the link between the i-th and $(i + 1)$-th ASes, then $NL(b, P) = (i + 0.5)/k$. Note that the value of $NL(b, P)$ is in the range of [0, 1]. The smaller the value of $NL(b, P)$, the closer the choke link b is to the probing source. Given a set of probing sets $P_1, P_2, ..., P_m$ $(m > 0)$ that detect b as a choke link, the normalized location of link b is computed as

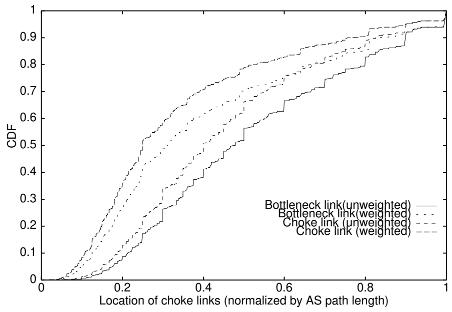
$$NL(b) = \frac{\sum_{j=1}^{m} NL(b, P_j)}{m}$$

Since the bottleneck link is the primary choke link, the definition of *normalized location* also applies to the bottleneck link.

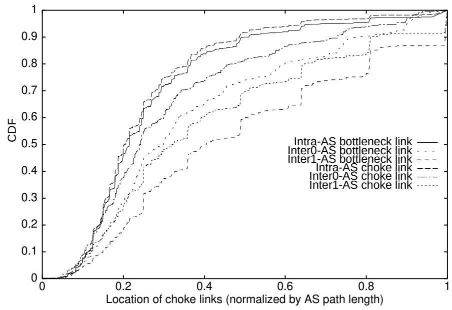
Figure 11(b) shows the cumulative distribution of the normalized locations of both bottleneck and choke links. The curves labeled "(unweighted)" show the distribution when all links have an equal weight, while for the curves labeled "(weighted)" we gave each link a weight equal to the number of probing sets in which the link is detected as a bottleneck or choke link. This is interesting because we observed in Figure 10(b) that some links are much more likely to be a bottleneck or a choke link than others. The results show that about 65% of the choke links appear in the first half of an end-to-end path (*i.e.,* $NL(b, P) \leq 0.5$). By comparing weighted with unweighted curves, we also observe that high-frequency choke links tend to be located closer to the source. Finally, by comparing the curves for choke links and bottleneck links, we observe that bottleneck locations are more evenly distributed along the end-to-end path. These observations are in part influenced by the definition of choke link and bottleneck, and by Pathneck's bias towards earlier



(a) Distribution across classes



(b) Cumulative distribution of normalized location



(c) Cumulative distribution per class

Figure 11: Location of bottleneck and choke links.

choke links.

Figure 11(c) shows the cumulative distribution for the normalized location for choke links and bottleneck links separately for the different classes of links; the results have been weighted by the number of probing sets in which a link is detected as a choke link or bottleneck link. We observe that intra-AS choke links and bottleneck links typically appear earlier in the path than inter0-AS and inter1-AS choke links and bottlenecks. The reason could be that some sources encounter choke links and bottlenecks in their home network.

4.4 Stability

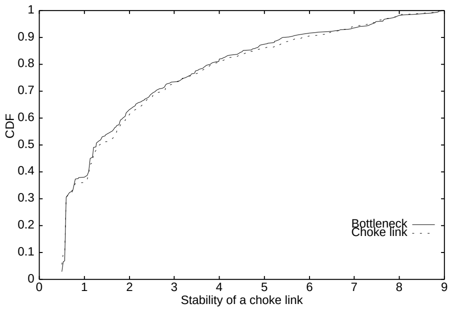
Due to the burstiness of Internet traffic and occasional routing changes, the bottleneck on an end-to-end path may change over time. In this section, we study the stability of the bottlenecks. For our measurements, we randomly selected 10 probing sources from PlanetLab ("ST" data set in Table 4). We sampled 30 destinations randomly chosen from the set of destinations obtained in Section 4.1. We took measurements for a three hour period and we divided this period into 9 epochs of 20 minutes each. In each epoch, we ran Pathneck once for each source-destination pair. Pathneck used probing sets consisting of 5 probing trains and reported choke links for each 5-train probing set.

Suppose link b is a choke link in probing set i. Let $DetectionRate_i(b)$ denote the frequency with which b is a candidate choke link in probing set i. For each path, we define the stability of choke link b over a period of n epochs as

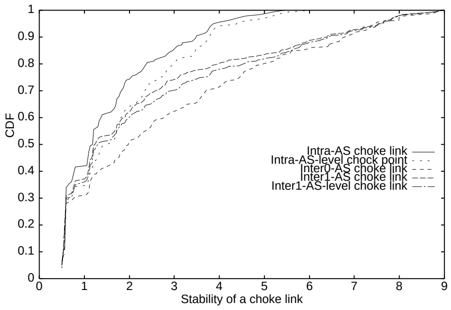
$$Stability(b) = \sum_{i=1}^{n} DetectionRate_i(b)$$

The same definition applies to bottleneck links. Note that the range of $Stability(b)$ is [0.5, n] because $d\_rate \geq 0.5$.

Figure 12(a) shows the cumulative distribution for stability over all measurements. We can see that bottlenecks and choke links have very similar stability, but this stability is however not very high. We speculate that the reason is that many bottlenecks are determined by the traffic load, not link capacity. Figure 12(b) shows the stability (at the router level) for intra-AS, inter0-AS and inter1-AS choke links. We see that intra-AS choke links are significantly less stable than inter-AS choke links. Comparing the two types of inter-AS choke links, inter0-AS choke links are more stable than inter1-AS choke links. We observe similar results at the AS level as shown by the curves labeled "intra-AS-level" and "inter1-AS-level": the intra-AS choke links are again less stable than the inter1-AS choke links. Moreover, we see, not surprisingly, that AS-level choke links are more stable than router-level choke links. Similar observations apply to bottlenecks (not shown in Figure 12(b)). Given the small number of destinations (*i.e.,* 30) and short duration of the experiment (*i.e.,* 3 hours), we cannot claim that these stability results are representative for the Internet. We plan to do more extensive experiments in the future.



(a) Bottlenecks vs. choke links



(b) Intra-AS vs. inter-AS choke links

Figure 12: The stability of choke links.

5. INFERRING BOTTLENECKS

In this section, we look at the problem of inferring the bottleneck location for a path that was not directly probed by Pathneck. The observation is that, if all or most of the links on a path are also part of paths that have already been probed, we may be able to derive the bottleneck location without actually probing the path. This could significantly reduce the amount of probing that must be done when studying bottleneck locations.

Methodology: We divide the "GE" data set we gathered in Section 4 into two parts — a training set and a testing set. The *training set* is used to label each link L with an upper bound $B_u(L)$ and/or a lower bound $B_l(L)$ for the available bandwidth; these bounds are calculated using the algorithm presented in Section 2.3.3. If there are multiple bounds on the available bandwidth of a link from multiple probing sets, we take the lowest upper bound as the upper bound of the link's available bandwidth and the highest lower

Table 5: Inference results

Class	Links covered	Correct	Incorrect	No upper bound	Not covered	Total
0	100%	10.2%	8.5%	9.9%	0%	28.6%
1	[80%, 100%)	11.4%	9.3%	9.8%	7.2%	37.7%
2	[70%, 80%)	2.7%	2.5%	2.4%	3.6%	11.2%
3	[60%, 70%)	1.4%	1.3%	1.3%	2.6%	6.6%
4	[0%, 60%)	0.9%	0.8%	0.6%	2.2%	4.5%
–	–	–	–	–	–	11.4%
Total	–	26.6%	22.4%	24%	15.6%	100%

bound as the lower bound of the link's available bandwidth. Since this type of calculation on bounds is very sensitive to measurement noise, we first preprocess the data: we include upper bounds only if the standard deviation across the probing set is less than 20% of the average upper bound.

The *testing set* is used for inference validation as follows. For each path P in the testing set, we try to annotate each link $L_i \in P$. If the link is covered in the training set, we associate the upper bound $B_u(L_i)$ and/or lower bound $B_l(L_i)$ derived from the training set with it. We identify the link L_i with the lowest upper bound $B_u(L_i)$ as the inferred bottleneck link $\hat{L_i}$; we ignore links that have no bounds or only lower bounds. We then compare $\hat{L_i}$ with the "true" bottleneck location, as identified by the Pathneck result in the testing set. If the location matches, we claim that the bottleneck location inference for the path P is successful. Paths in the testing set for which Pathneck cannot identify any choke link with high enough d_rate and $conf$ are excluded from the analysis. Note that routers may have multiple interfaces with different IP addresses, we use the tool *Ally* [35] to resolve router aliases.

When evaluating how successful we are in inferring bottleneck location, we need to account for the fact that for most paths, we miss information for at least some of the links. Obviously we would expect to have a lower success rate for such paths. For this reason, we classify the paths in the testing set into 5 classes based on the percentage of links that are covered by the training set. *Class 0* includes paths in the testing set for which we have some information (either upper bound, lower bound, or both) for every link in the path. *Class 1* includes paths for which we have information for over 80% of the links, but not for every link. Similarly, *Classes* 2, 3, and 4 include paths for which the percentage of covered links is [70%, 80%), [60%, 70%), and [0%, 60%), respectively.

Results: The probing data that we used in this section includes the results for 51,193 paths. We randomly select 60% of the probing sets as the training data, while the remaining 40% are used as testing data for inference evaluation. That gives us 20,699 paths in the testing set. Column "Total" in Table 5 lists the percentage of paths in each class; the "11.4%" entry corresponds to paths in the testing set on which we cannot identify a bottleneck.

Column "Correct" corresponds to the cases where inference was successful, while column "Incorrect" corresponds to the cases where we picked the wrong link as the bottleneck, even though the training set provided sufficient information about the real bottleneck link. Column "No upper bound" corresponds to the paths where we picked the wrong bottleneck link, but the training set only has lower bounds for the true bottleneck link; this is typically because the link was covered by very few probings in the training set. The column "Not covered" corresponds to paths for which the bottleneck link is not covered in the training set, so we can obviously not identify the link as the bottleneck link. For both the "No upper bound" and "Not covered" cases, inference fails because the training set does not offer sufficient information. A more carefully designed training set should reduce the percentage of paths in these categories.

Overall, inference is successful for 30% of the paths which we can identify bottleneck in the testing set, while the success rate increases to 54% when we have sufficient data in the training set. Note the diminishing trend in the inference success rate as we have information for fewer links in the path: the "Correct" cases account for 36%, 30%, 24%, 21% and 20% of the paths in *Classes* 0 through 4, respectively. This drop is expected since the less information we have on a path, the less likely it is that we can infer the bottleneck location correctly.

Discussion: The inference capability presented in this section shows that it is possible to infer the network bottleneck location without probing the path with some level of accuracy. However, we need sufficient information on the links in the path so it is important to properly design the training set to reduce the number of links for which we have little or no data. Ideally, we would be able to systematically probe a specific region of the Internet and put the results in a database. This information could then be used by applications to infer the bottlenecks for any path in that region of the network.

6. AVOIDING BOTTLENECKS

In this section we study how bottleneck information obtained by Pathneck can be used to improve overlay routing and multihoming.

6.1 Overlay Routing

Overlay routing, or application layer routing, refers to the idea of going through one or more intermediate nodes before reaching the destination. The intermediate nodes act as application layer routers or overlay nodes by forwarding traffic typically without any additional processing. Previous studies [33, 31] have shown that by going through an intermediate node, the round trip delay can be significantly improved and routing failures can be bypassed. In such cases, the part of the network experiencing congestion or routing problems is avoided. Note that between any two overlay nodes or between an overlay node and either the source or destination, regular IP routing is used to route traffic. One of the reasons why such "triangular" routing works is that BGP — the Inter-domain Routing Protocol, does not optimize for network performance in terms of delay, loss rate, or bandwidth. Shortest AS-path-based routing does not always yield the best performing paths because routing policies can cause path inflation [37, 34].

Overlay routing can thus be used to avoid bottleneck links in the underlying IP path, thereby improving application level performance in terms of throughput. So far, no studies have quantified the benefit overlay routing provides in avoiding bottleneck links. To the best of our knowledge, this study presents the very first large scale analysis of how overlay routing can improve the available bandwidth of a path. Other metrics such as delay, loss rate, and cost [15] are also important, and we plan to study the correlation between these metrics and the available bandwidth we consider here in a future study. Most of the nodes from which we performed probing are well connected, *i.e.,* they receive upstream Internet service from a tier-1 ISP. We would like to understand the utility of overlay routing when the probe nodes serve as overlay routers for paths destined to arbitrary locations in the Internet. We used the following probing methodology to gather the data for this study.

Methodology: We selected 27 RON and Planetlab nodes as both the source nodes and overlay nodes, as listed in the "OV" column in Table 4. Using a BGP table from a large tier-1 ISP, we sampled 200 random IP addresses from a diverse set of prefixes; each IP address originates from a different AS and ends with ".1" to minimize the

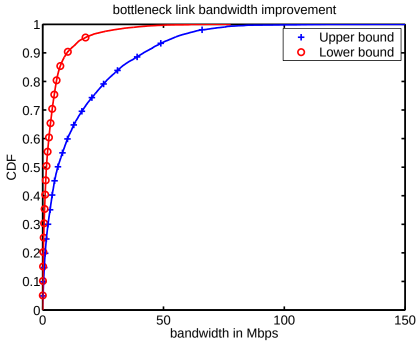


Figure 13: Improvements in the lower and upper bounds for available bandwidth using overlay routing (cutoff at 150Mbps).

chance of triggering alarms at firewalls. From each probing source we performed the probing process described below during the same time period to minimize the effect of transient congestion or any other causes for nonstationary bottleneck links. Given the list of 200 target IP addresses, each source node S probes each IP address 10 times using Pathneck. After probing each target IP address, S randomly selects 8 nodes from the set of 22 source nodes as its candidate overlay nodes and probes each of these 8 nodes 10 times. This probing methodology is designed to study the effectiveness of overlay routing in avoiding bottleneck links in a fair manner, as the probing of the following three paths occur very close in time: $S \to D$, $S \to S'$, and $S' \to D$, where S' is overlay node and D is destination node. The upper bound for the bottleneck link available bandwidth is calculated based on the largest gap value in the path across the 10 probing results.

For each destination from a given source, we calculate the lower bound and upper bound for the available bandwidth of the path as the lowest and highest upper bound on bottleneck link available bandwidth calculated by Pathneck in the 10 probings for that path. When composing two paths such as $S \to S'$ with $S' \to D$, the lower bound of this overlay path is assumed to be the minimum of the two lower bound values from the individual paths. The upper bound of an overlay path is calculated in the same manner to have a conservative estimate of the available bandwidth. We only consider the effectiveness of overlay routing by going through a single intermediate node, similar to previous studies.

Results: Of the $63,440$ overlay attempts, *i.e.,* routing to a destination by going through an intermediate node, 52.72% are *useful*, which means that overlay routing improves either the lower or the upper bound on the available bandwidth of the path. Note that we are not considering other metrics such as latency or packet loss. If we require both bounds to increase, the useful rate is 15.92%; the breakdown for improving only the lower bound or only the upper bound is 17.39% and 19.40%, respectively. The distribution of the improvement in upper and lower bounds for the available bandwidth is shown in Figure 13. The curves show that most improvements in the upper bound are less than 100Mbps while the limit for the lower bound is 20Mbps.

We now examine more closely how useful the overlay nodes are in improving the available bandwidth bounds from each source node to the 200 randomly selected destinations. We found that for most sources almost all 22 overlay nodes can be used for reaching *some* destinations with improved performance. A few exceptions stand out: the mazu1 site finds only 8 out of 22 nodes useful in terms of improving available bandwidth, and the cornell site finds 18 nodes useful. One possible reason is that the paths between these two sites and the chosen destinations already have good performance, hence overlay routing does not help. Another possible explanation is that the bottleneck is near the source, so again overlay routing cannot help. Among the other sites, where most of the randomly selected overlay nodes can help in improving available bandwidth, we studied the data in more detail to see whether any particular overlay nodes are always helpful for a given source node. Surprisingly, the answer is *yes*. In fact, for most source nodes, there are 2 to 3 overlay nodes that can improve performance for more than 90% of the cases examined. For example, when using vineyard as a source, jfk1, bkly-cs, and purdue all prove to be useful as overlay nodes for 92% of the destinations. Such information is very helpful in making overlay routing decisions, as we discuss below.

Discussion: The study presented here has several important implications for how to select overlay nodes and for improving overlay routing strategies. Overlay node selection typically involves continuous probing and monitoring between the source node and the overlay node, and between the overlay node and the destination node. This solution is not scalable if one has to probe exhaustively for every combination of destinations and candidate overlay nodes. To minimize measurement overhead, one can make use of the topology information to predict how likely an intermediate overlay node can help improve performance to a particular destination. Pathneck presents two opportunities here: (1) Pathneck is very helpful in identifying both the location of stable bottleneck links and overlay nodes that often seem helpful in avoiding such links. (2) Pathneck is light-weight enough to be used on-demand to decide which upstream provider to use for routing bandwidth-intensive applications or applications requiring a minimal amount of bandwidth to function, *e.g.,* multimedia streaming.

6.2 Multihoming

Large enterprise networks often *multihome* to different providers. The multihomed network usually has its own Autonomous System (AS) number and exchanges routing information with its upstream providers via the Border Gateway Protocol (BGP). The original motivation for multihoming was to achieve resilient network connectivity or redundancy in case the connectivity to one ISP fails or one of the ISPs experiences severe routing outages. Multihoming can not only increase the availability of network connectivity, but can also improve performance by allowing multihomed customers to route traffic through different upstream providers based on the routing performance to a given destination. A recent study [8] has shown that, by carefully choosing the right set of upstream providers, high-volume content providers can gain significant performance benefit from multihoming.

The performance benefit offered by multihoming depends highly on the routing path diversity and the location of failures or performance bottlenecks. For example, if a network is multihomed to two providers that route large portions of its traffic via paths with significant overlap, then the benefit of multihoming will be sharply diminished since it will not be able to avoid bottlenecks in the shared paths. As a result, we consider the following two problems: (1) Given the set of popular destinations a network frequently accesses, which upstream provider should the network consider using? (2) Given a set of upstream providers, which provider should be used to reach a given destination? Clearly we would like to do the selection without expensive probing. We show that Pathneck can help answer both these questions. To the best of our knowledge, this is the first study to examine the benefit of multihoming to avoid bottleneck links by quantifying the improvement in available bandwidth.

Methodology: To understand the effect of multihoming on avoid-

Table 6: Grouping based on coarse-grained geographic proximity.

Group name	Group member	Useful rate
sf	bkly-cs, ucsc, stanford	94%
nyc	princeton, jhu, bu, umd, rpi, mit-pl, dartmouth, cmu	99%
kansas	ku, wustl	90%
chicago	depaul, umich, uky, northwest, msu,	98%
britain	cam-uk, ac-uk	17%
korea	kaist-kr, snu-kr	74%

ing bottleneck links, one would ideally probe from the same source to each of several destinations through different upstream providers. A previous study [8] simulated this by probing from nodes within the same city but connected through different upstream providers. Unfortunately, very few of our probe nodes are located in the same city and have different upstream providers. We simulate this by choosing 22 probing sources belonging to different, but geographically close, organizations, as is shown in Table 6. We treat the members in the same group as nodes within the same city. While this is a simplification, we note that the geographic distance between any two nodes within the same group is small relative to the diverse set of $7,090$ destinations we selected for probing.

To evaluate the effectiveness of multihoming, for each geographic group, we examine the bounds on the available bandwidth of the paths from each member in the group to the same destination. If the improvement in the lower bound or the upper bound from the worst path compared with any other path in the group is more than 50% of original value, then we declare multihoming to be *useful*. Note that similar to the overlay routing study, our metric only considers available bandwidth; for some applications, other path properties such as latency and cost could be more important.

Results: Among all $42,285$ comparisons we are able to make across all probing locations, more than 78% of them are useful cases. This is very encouraging and shows that multihoming significantly helps in avoiding bottleneck links. However, these results may be overly optimistic given our destination set and the difficulty in discovering bottlenecks at the destination site. First, many of the probe destinations selected are not stub networks and most of them do not correspond to addressable end hosts. Furthermore, firewalls often drop outgoing ICMP packets, thus rendering Pathneck ineffective at identifying bottleneck at some destination sites. Nevertheless, our results suggest that multihoming is very effective at avoiding bottleneck links near the source or inside the network core. When we are more conservative and we require both the upper bound and the lower bound to improve by 50%, then the useful rate is reduced to exactly 50%.

Examining the results for individual groups in Table 6 reveals some interesting characteristics. First of all, the bigger the group, the higher the useful rate. For the two sites outside North America – britain and korea, the useful rates are significantly lower. We conjecture that the transoceanic link is the main bottleneck link and it cannot easily be avoided by choosing a nearby source node within the same country. Also, these two groups have only two members, so they have fewer choices.

Intuitively one would expect that as one adds more service providers, there is a diminishing return for multihoming. An earlier study [8] has shown this with respect to reducing download time for Web objects. We now examine this effect using available bandwidth as the performance metric. Figure 14 shows how the useful rate increases with the number of providers. We see that there is a fairly steady increase, even for higher numbers of providers. We plan to investigate this further using more probe source locations.

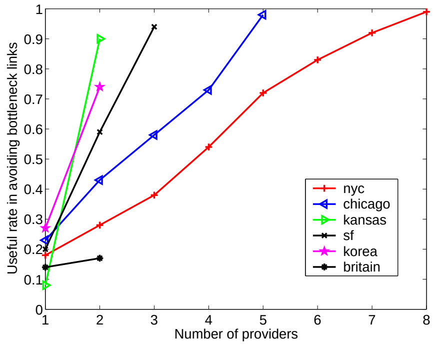


Figure 14: Improvement in avoiding bottleneck links with increase in providers.

Discussion: The results of the multihoming study are quite encouraging. Not only do they suggest that multihoming can yield significant benefits, but they also show that information collected by Pathneck can be used to select the upstream providers.

7. RELATED WORK

Bandwidth estimation techniques [14, 22], specifically available bandwidth estimation algorithms [13, 20, 26, 18, 30, 36], measure network throughput, which is closely related to congestion. However, they provide no location information for the congestion point. Also, all these tools, except cprobe [13], need the cooperation of the destination, which makes them very hard to deploy. Packet loss rate is another important metric that is related to user throughput, especially for TCP traffic [27]. Tools that focus on loss rate include Sting [32], which measures the network path loss rate, and Tulip [23], which can pinpoint the packet loss position.

The tools that are most closely related to Pathneck include Cartouche [17], Packet Tailgating [29], BFind [10] and Pathchar [19]. Cartouche [17] uses a packet train that combines packets of different sizes and exploits differences in how different-sized packets are handled to measure the bandwidth for any segment of the network path. The bottleneck location can naturally be deduced from its measurement results. Packet Tailgating [29] also combines load packets and measurement packets, but instead of letting measurement packets expire, it expires the load packets. Both Cartouche and Packet Tailgating require two end control.

BFind [10] detects the bottleneck position by injecting a steady UDP flow into the network path, and by gradually increasing its throughput to amplify the congestion at the bottleneck router. At the same time, it uses Traceroute to monitor the RTT changes to all the routers on the path, thus detecting the position of the most congested link. Concerns due the overhead generated by the UDP flow force BFind to only look for a bottleneck with available bandwidth of less than 50Mbps. Even then, considering its measurement time, its overhead is fairly high, which is undesirable for a general-purpose probing tool.

Pathchar [19] estimates the capacity of each link on a network path. The main idea is to measure the data transmission time on each link. This is done by taking the difference between the RTTs from the source to two adjacent routers. To filter out measurement noise due to factors such as queueing delay, pathchar needs to send a large number of probing packets, identifying the smallest RTT values for the final calculation. As a result, pathchar also has a large probing overhead.

Due to the lack of a measurement tool to identify bottleneck location, there has not been much work on analyzing Internet bottlenecks. We are only aware of the analysis in [10], which shows that most bottlenecks are on edge and peering links. Pathneck overcomes some of the limitations in BFind, thus allowing us to perform a more extensive bottleneck study. The large BGP database that we have access to also enables us to probe the Internet in a more systematic way, thus giving our analysis a broader scope.

Several studies have shown that overlay routing [31, 33], multipath routing [12, 25, 28], and multihoming [8, 5] benefit end user communication by reducing the packet loss rate and increasing end-to-end throughput. These projects primarily focus on link failure or packet loss, although some consider latency and throughput as well. A recent study compares the effectiveness of overlay routing and multihoming, considering latency, loss rate, and throughput as metrics [9]. In contrast, our work approaches the problem from a different angle— by identifying the location of the bottleneck we can study how overlay routing and multihoming can be used to avoid bottlenecks. Our work shows the benefit of overlay routing and multihoming and suggests efficient route selection algorithms.

8. CONCLUSION AND FUTURE WORK

In this paper, we present a novel light-weight, single-end active probing tool – *Pathneck* – based on a probing technique called Recursive Packet Train (RPT). Pathneck allows end users to efficiently and accurately locate bottleneck links on the Internet. We show that Pathneck can identify a clearly-defined bottleneck on almost 80% of the Internet paths we measured. Based on an extensive set of Internet measurements we also found that up to 40% of the bottlenecks are inside ASes, contrary to common assumptions. We showed how Pathneck can help infer bottleneck location on a path without probing. Finally, we illustrated how Pathneck can be used to guide overlay routing and multihoming.

This paper analyzes only some aspects of Internet bottlenecks and many issues require further study, including the stability of bottlenecks, the impact of topology and routing changes on bottlenecks, and the distribution of bottlenecks across different (levels of) ASes. We also hope to improve Pathneck, for example by studying how configuration parameters such as the load packet size, the number of load packets, and the initial probing rate of RPT affect the measurement accuracy.

Acknowledgments

We thank Dave Andersen for his help in setting up the experiments on RON, Jay Lepreau and the Emulab team for their quick responses during our testbed experiments, and Ming Zhang for his help in using the PlanetLab socket interface. We thank support from the Planetlab staff. We also thank our shepherd Christos Papadopoulos and the anonymous reviewers for their constructive comments.

Ningning Hu and Peter Steenkiste were in part supported by the NSF under award number CCR-0205266.

9. REFERENCES

[1] Abilene network monitoring. http://www.abilene.iu.edu/noc.html.
[2] Dummynet. http://info.iet.unipi.it/∼luigi/ip_dummynet/.
[3] Emulab. http://www.emulab.net.
[4] Planetlab. https://www.planet-lab.org.
[5] Routescience. http://www.routescience.com.
[6] University of Oregon Route Views Project. http://www.routeviews.org.
[7] RFC 792. Internet control message protocol, September 1981.
[8] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman. A Measurement-Based Analysis of Multihoming. In *Proc. ACM SIGCOMM*, September 2003.
[9] A. Akella, J. Pang, B. Maggs, S. Seshan, and A. Shaikh. Overlay routing vs multihoming: An end-to-end perspective. In *Proc. ACM SIGCOMM*, August 2004.
[10] A. Akella, S. Seshan, and A. Shaikh. An empirical evaluation of wide-area internet bottlenecks. In *Proc. ACM IMC*, October 2003.
[11] K. G. Anagnostakis, M. B. Greenwald, and R. S. Ryger. cing: Measuring network-internal delays using only existing infrastructure. In *Proc. IEEE INFOCOM*, April 2003.
[12] D. G. Andersen, A. C. Snoeren, and H. Balakrishnan. Best-path vs. multi-path overlay routing. In *Proc. ACM IMC*, October 2003.
[13] R. L. Carter and M. E. Crovella. Measuring bottleneck link speed in packet-switched networks. Technical report, Boston University Computer Science Department, March 1996.
[14] C. Dovrolis, P. Ramanathan, and D. Moore. What do packet dispersion techniques measure? In *Proc. of ACM INFOCOM*, April 2001.
[15] D. Goldenberg, L. Qiu, H. Xie, Y. R. Yang, and Y. Zhang. Optimizing Cost and Performance for Multihoming. In *Proc. ACM SIGCOMM*, August 2004.
[16] R. Govindan and V. Paxson. Estimating router ICMP generation delays. In *Proc. PAM*, March 2002.
[17] K. Harfoush, A. Bestavros, and J. Byers. Measuring bottleneck bandwidth of targeted path segments. In *Proc. IEEE INFOCOM*, April 2003.
[18] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, 21(6), August 2003.
[19] V. Jacobson. pathchar - a tool to infer characteristics of internet paths, 1997. Presented as April 97 MSRI talk.
[20] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. In *Proc. ACM SIGCOMM*, August 2002.
[21] M. Jain and C. Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. In *Proc. PAM*, March 2002.
[22] K. Lai and M. Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *Proc. of the USENIX Symposium on Internet Technologies and Systems*, March 2001.
[23] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level internet path diagnosis. In *Proc. SOSP*, October 2003.
[24] Z. M. Mao, J. Rexford, J. Wang, and R. Katz. Towards an Accurate AS-level Traceroute Tool. In *Proc. ACM SIGCOMM*, September 2003.
[25] N. Maxemchuk. *Dispersity Routing in Store and Forward Networks*. PhD thesis, University of Pennsylvania, May 1975.
[26] B. Melander, M. Bjorkman, and P. Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Proc. IEEE GLOBECOM*, November 2000.
[27] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proc. ACM SIGCOMM*, September 1998.
[28] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. of the ACM*, 36(2), April 1989.
[29] V. Ribeiro. Spatio-temporal available bandwidth estimation for high-speed networks. In *Proc. of the First Bandwidth Estimation Workshop (BEst)*, December 2003.
[30] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Proc. PAM*, April 2003.
[31] RON. Resilient Overlay Networks. http://nms.lcs.mit.edu/ron/.
[32] S. Savage. Sting: a TCP-based network measurement tool. In *Proc. of the 1999 USENIX Symposium on Internet Technologies and Systems*, October 1999.
[33] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: a case for informed internet routing and transport. *IEEE Micro*, 19(1), 1999.
[34] N. Spring, R. Mahajan, and T. Anderson. Quantifying the Causes of Path Inflation. In *Proc. ACM SIGCOMM*, August 2003.
[35] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proc. ACM SIGCOMM*, August 2002.
[36] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proc. ACM IMC*, October 2003.
[37] H. Tangmunarunkit, R. Govindan, and S. Shenker. Internet Path Inflation Due to Policy Routing. In *Proc. SPIE ITCOM*, August 2001.