Locating Internet Bottlenecks: Algorithms, Measurements, and Implications

Ningning Hu [1], Li Erran Li [2], Zhuoqing Morley Mao [3]
Peter Steenkiste [4], Jia Wang [5]

April 27, 2004
CMU-CS-04-123

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1]Ningning Hu is with Computer Science Department, Carnegie Mellon University, hnn@cs.cmu.edu;
[2]Li Erran Li is with Bell Laboratories, erranlli@bell-labs.com;
[3]Zhouqing Morley Mao is with University of Michigan, zmao@eecs.umich.edu;
[4]Peter Steenkiste is with School of Computer Science and Department of Electrical and Computer Engineering, Carnegie Mellon University, prs@cs.cmu.edu;
[5]Jia Wang is with AT&T Labs — Research, jiawang@research.att.com.

This research was sponsored in part by DARPA under contracts F30602-99-1-0518, F30602-96-1-0287, and N66001-99-2-8918, and by NSF under award number CCR-0205266 .

Keywords: Network measurements, bottleneck location, active probing, packet train, congestion

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 bandwidth measurement tools fail to identify the *location* of bottleneck links. In addition, they often require access to both end points and generate huge amount of probing packets. These drawbacks make them impractical. In this paper, we present a novel light-weight, single-end active probing tool – *Pathneck* – based a novel probing technique called Recursive Packet Train (RPT), which allows end users to efficiently and accurately locate bottleneck points to destinations on the Internet. We evaluate Pathneck using trace-driven emulations and wide area Internet experiments. In addition, we conduct extensive measurements on the Internet among carefully selected, geographically diverse probing sources and destinations to study Internet bottleneck properties. We find that Pathneck can successfully detect bottlenecks for over 70% of paths, and most of the bottlenecks are fairly stable. We also report our success on bottleneck inference, using multihoming and overlay routing to avoid bottlenecks based on the bottleneck link location and bandwidth estimation provided by Pathneck.

1 Introduction

The ability to locate network bottlenecks along end-to-end paths on the Internet is very useful for both the end users and the Internet Service Providers (ISPs). End users can use it to estimate the performance of an ISP, while an ISP can use it to quickly locate the position of 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 links along the path. 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 bandwidth measurement tools fall short in at least two ways. First, they focus on end-to-end performance, while providing no *location* information for the performance bottleneck. Typical examples include the work on available bandwidth measurements [1, 2, 3, 4, 5]. Second, for tools that do measure hop-by-hop performance, the measurement overhead is often very high. This category includes Pathchar [6] and BFind [7].

In this paper, we present a novel active probing tool – *Pathneck* – based a novel probing technique called Recursive Packet Train (RPT). It allows end users to efficiently and accurately locate bottleneck points 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, and RPT relies on the fact that congestion builds up as load packets queue on the router interface, thus changing the packet train length on the link. By measuring this change using the measurement packets, the position of the congestion 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 conduct 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. Our main findings include

1. Pathneck is capable of locating the bottleneck for 70% - 95% of paths from most of our probing sources.

2. Unlike the common knowledge that bottleneck locations are mostly on the edge links or peering links, we find that roughly over 50% of the bottleneck locations are within a single AS.

3. In terms of stability, intra-AS bottlenecks are more stable than inter-AS bottlenecks, while AS-level bottlenecks are more stable than router level bottlenecks.

4. With the bottleneck location information, and the rough estimation for the absolute available link bandwidth, we can successfully infer the bottleneck locations for 40% of arbitrary paths for which we do not have measurement data.

5. 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 to avoid bottlenecks are successful. The success of multihoming in avoiding bottleneck links is over 78%.

This paper is organized as the following. We first present the details of the Pathneck design and the algorithms' details (Section 2), followed by the tool's validation (Section 3). Using Pathneck, we probed a large number of destinations to obtain several different data sets. Based on these data, we study the properties of Internet bottlenecks (Section 4), how to avoid the bottlenecks on the Internet (Section 5), and the implications on multihoming and overlay routing (Section 6). Related work is discussed in Section 7. Finally, Section 8 conclude the paper together with a discussion of future work.

2 Inferring Bottleneck Location

Our goal is to develop a tool that is light-weight, does not require access to the destination, and provides a ranking of the detected bottlenecks. In this section we first provide some background in available bandwidth measurement techniques and we then describe the concept of Recursive Packet Trains and the Pathneck tool.

2.1 Measuring Available Bandwidth

In this paper, we define the "bottleneck link" of a network path as the link with the minimum available bandwidth, i.e. it is the link that determines the end-to-end throughput on the path. In our algorithm description below, we will also use the concept of "choke point", which we define as follows. Assume an end-to-end path from source $S = R_0$ to destination $D = R_n$ passes routers: $R_1, R_2, ..., R_{n-1}$. Link $L_i = (R_{i-1}, R_i)$ has available bandwidth $A_i (1 \leq i \leq n)$. We define the set of *choke links* as:

$$CHOKE_L = \{L_k | A_k = min\{A_1, ..., A_k\} \& A_k < A_{k-1}, 1 < k \leq n\}$$

and the corresponding set of *choke routers* are

$$CHOKE_R = \{R_k | L_k \in CHOKE_L, 1 < k \leq n\}$$

We also use *choke point* as an equivalent term for choke router. Intuitively, the choke points on a network path are the links with the minimum available bandwidth from the source to that link's downstream router. Based on this definition, the *bottleneck link* is the choke link that has the smallest available bandwidth. Clearly, choke points will have less available bandwidth as they get closer to the destination. We will refer to the choke point has the smallest available bandwidth as "the primary choke point" (the bottleneck), the next choke point is the "second choke point", then the "third choke point", 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 path [1, 2, 4, 5, 8]. This is typically done by sending a probing packet train along a network path and by measuring how competing traffic along the path affects the length of the packet train (or the gaps between the packet pairs). 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 will increase. This increase can be caused by higher packet transmission times (on low capacity links), or by the interleaving between the probing packets and the background traffic packets (heavily loaded links). When the packet train traverses a link where the available bandwidth is higher than the packet train rate, the train length should stay the same since there should be little or no queuing at that link. As a result, the packet train length can be used to estimate the available bandwidth on the bottleneck link; details can be found in [2]. Using the definition introduced above, the links that increase the length of the packet train correspond to the choke points since they represent the links with the lowest available bandwidth on the partial path traveled by the train so far.

Unfortunately, current techniques can 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 would like to know the available bandwidth on each link along the path, so we need a probing technique that can measure the train length on *each* link. In this section, we introduce a novel packet train design — Recursive Packet Train (RPT), that provides train length estimates for each hop.

2.2 Recursive Packet Train

An example of a Recursive Packet Train is shown in Figure 2.2. In this figure, 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. First, we have the *measurement packets*, which are standard traceroute packets, i.e., they are 60 bytes UDP packets, with properly filled-in payload fields. The figure shows 20 measurement packets at each end of the packet train, which allows us to measure network paths with up to 20 hops; more measurement packets should be used for a longer path. The TTL values of the measurement packets changes linearly, as is shown in the figure.

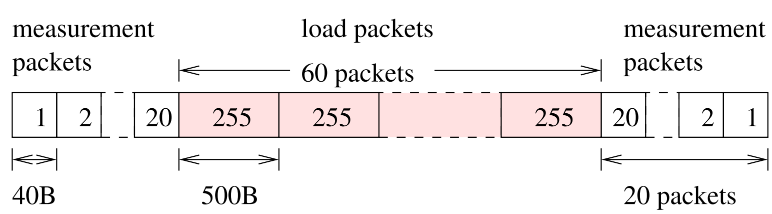

Figure 1: Recursive Packet Train (RPT). The number in each packet is the TTL value.

Second, we have the *load packets* that are used to generate a packet train with a measurable length along the network path. Similar to the PTR method [2], the load packets should be large packets that represent an average traffic load. We use 500 byte packets as suggested in [2]. The number of 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.

RPT works as follows. The user sends out the probing packets back-to-back. When they arrive at the first router, the first and the last packet of the train will expire, since their TTL values are 1. As a result, the packets are dropped and the router will send two ICMP packets back to the

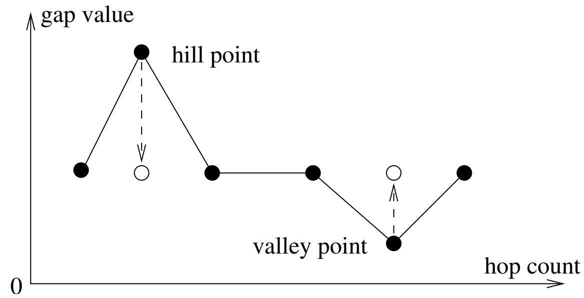

Figure 2: Hill/valley point

source [9]. The other packets in the train are forwarded to the next router, after their TTL is decremented. Since the TTL values in a RPT are set recursively, the above process is repeated on each subsequent router. The source can now 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. This is because: (1) the ICMP packets were generated when the head and the tail packets of the train were dropped, and (2) the measurement packet size is much smaller than the total length of the train, i.e., the change in packet train length due to the dropping of the measurement packets can be neglected. We will refer to the time interval between the arrival of the two ICMP packets from a router as the *gap value*.

2.3 Pathneck — The Bottleneck Location Inference Tool

RPT provides a way to estimate the probing packet train length on each link along the path. We can now use this sequence of gap values to identify the location of bottleneck links — we expect the train length to change significantly at the bottleneck. This is the basis for the Pathneck tool.

Pathneck uses three steps to detect and rank bottlenecks along a path:

1. *Labeling of gap sequences*: we use a sequence of RPT trains to collect gap sequences, and identify links where the gap value changes significantly. These are at the candidate choke points.

2. *Averaging across gap sequences*: using the labeled gap sequences from step (1), we identify links that frequently generate significant gap changes as choke links.

3. *Ranking*: since network paths can have multiple choke points, Pathneck ranks these choke points with respect to the available bandwidth.

In the remainder of this section, we describe the algorithms that are used in each of the three steps in more detail.

Labeling of Gap Sequences

Under ideal circumstances, a sequence of gap values would 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 drop. In reality, the burstiness of competing traffic and reverse path effects add noise to the gap sequence, and before we can identify candidate choke points we have to clean up gap sequence. The first step is to remove any data for routers from which we do not receive both ICMP packets. If we miss over half of the gap values due to that, we discard the entire sequence.

The second step is to modify the *hill* and *valley* points in the gap sequence (Figure 2). A hill point is defined as a point 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 similarly, except the condition is changed to $g_1 > g_2 < g_3$. Both hill and valley points contain a drop in gap value, which should not happen. Since in both cases, it is a short-term (one sample) disturbances in the sequence, we assume they are caused by noise, and we replace the hill or valley point (g_2) with the closer gap value of its two neighbors.

We are now ready to run the core part of the labeling algorithm (Figure 3). The idea is to match the gap sequence to a graph consisting of a sequence of steps (Figure 4), where each step corresponds to a candidate choke point. Easy to see, this is a typical clustering problem. But since the number of hops in our problem is very limited, generally less than 30 points, we use a simple brute-force algorithm to identify the candidate choke points. Given a gap sequence with len gap values, we generate all possible step functions with $n = round(len/2)$ steps. We pick the step function that is the best fit for the gap sequence. The "best fit" is defined as the step function for which the sum of difference between the gap sequence and the step function across all point is minimal (refer to the computation of $dist\_sum$ in Figure 3). If that step function has clearly defined steps (i.e. all n steps are larger than 100 $microseconds$ (μs)) then we take this as our fit for the gap sequence, and we identify these steps as a set of candidate choke points. If not, we repeat the process with a function with $(n-1)$-steps. This process is repeated until we find a segmentation where each step has a gap change larger than $100\mu s$, or when $n = 0$. In the latter case we mark the first hop as the bottleneck router. The requirement that steps must be larger than $100\mu s$ is used to filter out noise. The threshold value is relatively small compared with possible sources of error (see Section 2.4). However, at this point we want to be conservative in discarding candidate choke points.

Averaging across gap sequences

In order to filter out effects caused by bursty traffic on the forward and reverse path, we typically use the results from multiple probing trains (e.g. 6-10) to compute *confidence* information for each detected choke point. In this paper, we will use the term "probing" to refer to a probing with a single RPT, i.e. one train. We will use the term "probing set" for a group of probings. The outcome of Pathneck is the summary result of the probings in the probing set; we will sometimes refer to this as the probing set result.

Intuitively, the confidence is denoted as the percentage of available bandwidth change implied by the gap value change. The reason is that a large gap value change is less likely to be caused by short-term burstiness in the traffic, so the link is more likely to be a real bottleneck. We compute the confidence for each candidate choke point as the follows:

$$conf_i = \frac{abs(1/g_i - 1/g_{i-1})}{1/g_{i-1}}$$

For $i = 1$, we let $conf_1 = 1$.

```
algorithm Labeling(gap)
/* gap is an gap sequence with len values */
{
        return if len < 4;
        return if over half of the gap values is 0;
        fix the hill/valley point;

        /* brute force search for the choke points */
        n = round(len/2);
        while (n > 1) {
              for any segmentation with n splitting points {
                dist_sum = 0;
                for each segment between two adjacent splitting points {
                      g_avg = avg(g_i in the current segment);
                      dist_sum+ = sum(|g_i - g_avg|);
                }
                record the dist_sum;
              }
              pick the segmentation with the minimum dist_sum;
              if (all the splitting points in this segmentation
                      have a gap value change > 100us)
                return the splitting points as the set of choke points;
              else
                n = n-1;
        }
        if (n == 0) {
              return the first hop as the choke point;
        }
}
```

Figure 3: Labeling Algorithm for a gap sequence.

For the set of choke points detected in each probing, we pick out the candidate choke points with $conf \geq 0.1$, for which we further calculate the detection rate d_rate. Here d_rate is defined as the frequency with which a candidate choke point appears in the probing set. Finally, we select those choke points with $d\_rate \geq 0.5$, i.e. *the final choke points for a path are the high confident candidates that appear in at least half of the probings in the same probing set*.

Ranking

For each path, we rank the choke points based on the average gap values in the probing set. Because the packet train transmission rate R has the following relationship with the gap value g:

$$R = data\_size\_of\_train/g$$

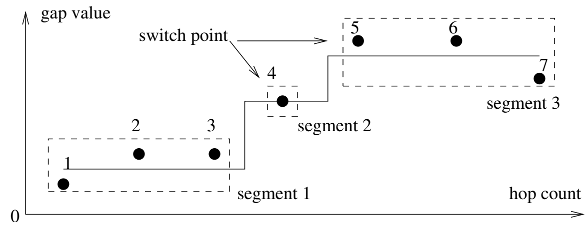

Figure 4: Matching the gap sequence to a step function.

where $data\_size\_of\_train$ is the total size for all the packets in the train. That is, the larger the gap value, the more the packet train was stretched out by the link, thus the lower the available bandwidth on the corresponding link. The link with the lowest available bandwidth is the bottleneck of the path.

The average gap values can also provide a rough upper and lower bound on the available bandwidth. We have to consider three cases:

1. For a link which is identified as a choke point, i.e. its gap change is an increase, we know that the available bandwidth is less than the packet train transmission rate. That is, the rate R computed above is an upper bound for the available bandwidth on the link.

2. For a link which is not a choke point and has a decrease in gap value, we cannot say anything about the available bandwidth, because the decrease is probably caused by traffic burstiness.

3. For a link which is not a choke point and maintains its gap, the available bandwidth is higher than the packet train transmission rate R, i.e., R is a lower bound for the available bandwidth.

Considering that we cannot control the format of the probing train at every link in the path and that the available bandwidth on a link is a dynamic property, these are only very rough bounds. However, they proved to be useful in our analysis in Section 5.

2.4 Properties of Pathneck

Since a single packet train is used to estimate the available bandwidth on all links along a path, we get a consistent set of measurements. This, for example, allows Pathneck to identify multiple choke points and to rank them. Note however Pathneck is biased towards early choke points: once a choke point early in the path has increased the length packet train, it may no longer be possible to "see" links downstream with similar or higher available bandwidth.

Pathneck also meets the design goals we identified in the beginning of this section. Pathneck does not need the cooperation of the destination, so it can be widely used by regular network user. Pathneck also has low overhead. Each measurement typically uses 6 to 10 probing trains of 60 to 100 packets each. This is very low overhead compared with tools such as pathchar [6] and BFind [7]. Finally, Pathneck is fast. For each probing train, it takes about a 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 RTT we have observed on Internet — after sending out the probing trains, and then exits. As a result, one measurement generally takes less than 5 seconds.

A number of factors 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 [10] and [11] show that the ICMP packet generation time is pretty small; in most cases it is between $100\mu s$ and $500\mu s$. Since most Internet paths have a bottleneck link with a capacity of less than 100Mbps, if we use 100 load packets, then the corresponding packet train length is larger than 4ms, which is large enough to ignore the ICMP packet generation time. Second, as ICMP packets travel to the source, they may encounter queue 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 measured this value. In our algorithm, we try to reduce the impact of this factor by filtering out the measurement outliers.

Pathneck also has some deployment limitations. First, we discovered that network firewalls often only let through 60 bytes UDP packets that strictly conform to the traceroute packet format, while they drop any other UDP probing packets, such as the load packets in a 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 the tail packet, the *accumulated* ICMP generation time for the whole packet train makes the returned interval worthless.

3 Validation

We use both the Emulab testbed [12] and Internet paths to evaluate Pathneck. The Emulab testbed provides a fully controlled environment that allows us to evaluate Pathneck with known traffic loads, while Internet experiments are necessary to study Pathneck with realistic background traffic.

3.1 Testbed Validation

Figure 5 shows our testbed configuration. The physical Emulab link capacity is 100Mbps, and we set the bottleneck link capacity to 20Mbps in the experiments, using the dummynet [13] functionality provided by Emulab. The link delays are roughly set based on a traceroute measurement from a CMU host to yahoo.com.

The background traffic is generated based on two real packet traces, *light-trace* and *heavy-trace*. The *light trace* is sampled from a outgoing link of a data center connected to a Tier-1 ISP. Its load varies from around 500Kbps to 6Mbps, with a median load 2Mbps. The *heavy-trace* is

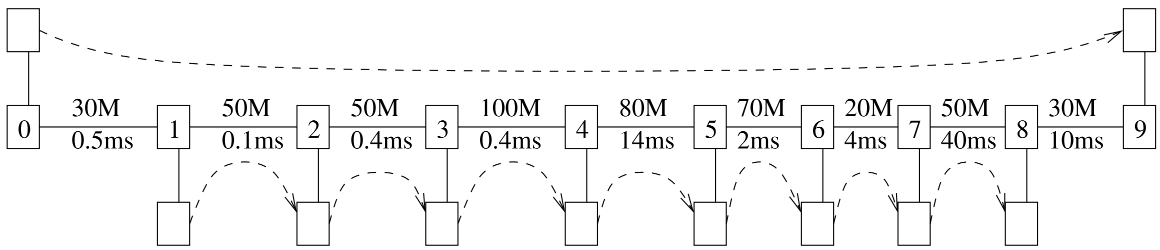

Figure 5: Testbed configuration. Hop 0 is the probing sender, hop 9 is the probing destination. Hop 1 - 8 work as routers, and we use $R1 - R8$ to denote them in the paper. The blank boxes are used for background traffic generation. The dashed lines show the background traffic flow directions for the evaluations in "single bottleneck" and "two bottlenecks".

sampled from a trace collected in front of a corporation network. Its bandwidth varies from 4 Mbps to 36Mbps, with a median load 8Mbps. Since the traces are very bursty, this is a particularly challenging scenario. We send traffic between each pair of routers and also between routers 0 and 9, as is shown by the dashed arrows in the figure. By assigning different traces to different links, we can emulate different scenarios to evaluate Pathneck. Note that all the hosts on the testbed are PCs, not routers, so the properties such as the ICMP generation time are different from those of a real router. As the result, the testbed results ignore some of the router related factors.

We ran three sets of experiments using this configuration:

1. **Single Bottleneck:** In this experiment, we use *light-trace* for all the load generators, but the starting times within the trace are randomly selected. For the 100 single-train probings that we did, we always detect hop 7 (i.e., link $\langle R6, R7 \rangle$) as the bottleneck. The other candidate choke points detected are all filtered out due to small confidence value (less than 0.1).

2. **Two Bottlenecks:** In this experiment, we reduce the link capacity of $\langle R2, R3 \rangle$ from 50Mbps to 20Mbps, and use the *heavy-trace* for link $\langle R6, R7 \rangle$, the other links keep using the *light-trace*. We probed 100 times; 14 probings had to be discarded due to ICMP packet loss. Among the remaining valid 86 single-train probings, 72 probings correctly detected these two links as the top two choke points, in the correct order. The other 14 probings only identified $\langle R2, R3 \rangle$ as the choke point. Careful examination of the probing data shows that, for these 14 cases, $\langle R6, R7 \rangle$ actually is the second choke point detected, but with a confidence less than 0.1. The reason is that the probing packet train has already been stretched by the first choke point, so the second choke point is easily hidden. This is the result of the biasing of Pathneck in favor of the earlier choke point.

3. **Reverse Path Queueing:**

   To study the effect of reverse path queueing, we replaced the trace-based traffic generator by a simple UDP traffic generator so that we can control the *average* load placed on a link. The instantaneous load generated by this generator follows an exponential distribution, which is used to emulate the burstiness. We use the topology of the "single bottleneck" experiment,

Table 1: Per-link results of reverse-path traffic experiment on Emulab

router_id	detected_times	d_rate
2	24	0.245
3	18	0.184
4	5	0.051
5	21	0.214
6	20	0.204
7	75	0.765
8	34	0.347

i.e., the bottleneck link is $\langle R6, R7 \rangle$. On all links (except the two edge links) we sent background traffic in both directions, with the average load set to 30% of the link capacity. With this setup, we got 98 valid probing results. The d_rate for each router, i.e., the frequency of that router being detected as a candidate choke point with $conf \geq 0.1$, is shown in Table 1. We see that, while reverse path queueing disturbs the detection to some extend, only the real bottleneck hop (R7) has a $d\_rate \geq 0.5$. That is, Pathneck will output R7 as the only choke point, thus the bottleneck.

3.2 Internet Validation

This section evaluates the performance of Pathneck on Internet paths. For a thorough evaluation we would need to know the actual available bandwidth on all the links of the network path. Of course, this information is impossible to obtain for most operational networks. The Abilene backbone [14], however, publishes its backbone topology and the traffic load (5-minute SNMP statistics) [15], so we decide to probe Abilene paths. We ran experiments from two sources: a CMU machine and a host at the University of Utah.

The experiment is carried out as the follows. 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. We insert a 2 seconds sleeping time between two consecutive probings. To avoid interference, the CMU and the Utah based experiments were run at different times.

Using the $conf \geq 0.1$ and $d\_rate \geq 0.5$ requirements, we only detected 4 none-first-hop bottleneck routers on the Abilene paths. This is not surprising since Abilene paths are well known to be over-provisioned, and we selected paths that were as much as possible in the Abilene core. It turns out that the probes from both sources identify exactly the same four bottleneck routers (Table 2). The d_rate for probes originating in Utah and CMU are very similar, possibly because we took all measurements in the same 24 hours period so experiments saw 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;

Table 2: Bottlenecks detected on Abilene Paths. Here the last column "AS Path" has the format AS1-AS2, where AS2 is the bottleneck router's AS, AS1 is its pre-hop router's AS.

Probe Dst	d_rate (Utah)	d_rate (CMU)	Bottleneck Router IP	AS Path
www.calren2.net	0.71	0.70	137.145.11.46	2150-2150
www.princeton.edu	0.64	0.67	198.32.42.66	10466-10466
www.sox.net	0.62	0.56	199.77.194.6	10490-10490
www.ogig.net	0.71	0.72	198.32.163.13	210-4600 (Utah) 11537-4600 (CMU)

an explanation is that the bottlenecks are very stable, possibly because they are constrained by link capacity.

Unfortunately, except for the bottleneck to www.ogig.net, all three bottlenecks are outside of Abilene, so we cannot get the load data. For the path to www.ogig.net, the bottlenecks appear to be two different peering links. For the path from CMU to www.ogig.net, the incoming link to the bottleneck router 198.32.163.13 is an OC-3 link. Based on the SNMP data that we have, which includes all links on that path except one link inside PSC (with a capacity of at least 1Gbps), we are sure that the OC-3 link is indeed the bottleneck.

4 Internet Bottleneck Measurement

The primary function of the Pathneck tool is to report the location of the bottlenecks along end-to-end paths. It has been a common assumption in many studies that the bottlenecks often occur at edge links and the peering links. In this section, we evaluate this widely used assumption using Pathneck, which is sufficiently light-weight and non-intrusive that it allows us to conduct large scale measurements on 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 [16] and RON [17] as the probing sources. Table 3 lists all the probing nodes that we used for this paper: node 1-25 are used for Section 4.2 and 4.3, node 26-35 are used for Section 4.4, node 36-58 and some nodes from 1-35 are used for Section 6. They reside in 47 distinct ASes and are connected to 31 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, using the following simple sampling algorithm. The key idea is making use of the local BGP routing table information of the probe sources to select destination IP addresses. In most cases, we do not have access to the local BGP table; however, we almost always have the BGP table from the corresponding upstream provider from public BGP data sources such as RouteViews [18]. The upstream provider information can be obtained by performing traceroute to a few randomly chosen locations such as `www.google.com` and `www.cnn.com` from the probe sources. Note that we may not be able to obtain the complete set of upstream providers in case of multihomed customers. Given the routing table, we first pick a ".1" or ".129" IP address for each prefix possible. The prefixes that are completely covered by its subnets are not selected. We then subsequently reduce the set of IPs by eliminating the ones whose AS paths starting from the probe source is part of other AS paths. Here we make the simplification that there is only a single inter-AS link. This assumption does not hurt, as the core of the Internet is repeatedly traversed for the roughly 3500 destinations we selected from each source. For instance, some links between tier-1 providers such as AT&T and UUnet are traversed several thousand times in our probing.

We run Pathneck on each source node as follows. For each destination, Pathneck continuously probes 10 times, with 2 seconds idle time in between. These 10 probings form a probing set, for which Pathneck reports the location of the choke points as well as a rough estimation of the available bandwidth for the corresponding choke links. Due to the small measurement time, we were able to finish probing around 3500 destinations within 2 days. In this section, we set $conf \geq 0.1$ and $d\_rate \geq 0.5$ in Pathneck as the thresholds to select choke points.

4.2 Popularity

As described in previous sections, Pathneck is able to detect multiple choke points for a network path. In our measurements, we observed that up to 5 choke points can be detected. Figure 6 shows the number of paths that have 0 to 5 choke points. We found that, for all probing sources, very few probe sets report more than 3 choke points. Fewer than 2% of the paths have 4 or more choke points. We also noticed that a good portion of the paths have no choke point. This number varies from 3% to 60% across the different probing sources. This is generally because the traffic on those paths are bursty enough that Pathneck could not reach a decision under the $conf \geq 0.1$ and $d\_rate \geq 0.5$ requirements.

In our measurements, we observe that some links are detected as choke points in a large number of paths. For a given link b, we define *positive probes* of b to be the subset of probes in a probe set for which b is a choke point. Let $NumProbe(b)$ denote the total number of probes that traverse the link b and $NumPositiveProbe(b)$ denote the total number of positive probes of b in all probe sets. We compute the $Popularity(b)$ of a link b as follows:

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

Figure 7 shows the cumulative distribution of the popularity of a link being detected as a bottleneck (the solid curve) and as a choke point (the dashes curve) for all the links observed as choke points in our measurements. We observed in Figure 6 that about 30% of the links never become choke points in the probes that traverse them. Half of the choke point links have the probability of 20% or less to be a choke point in the probes that traverse them. About 5% of the choke point links are detected in all the probes. The same observation holds on the cumulative distribution of the popularity of a link being detected as a bottleneck (i.e., primary choke point).

Table 3: Probing sources from PlanetLab (PL) and RON (RON). “-” denotes two probing hosts obtrained privately.

ID	Probing Source	AS Number	Location	Upstream Provider(s)	Testbed
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	PL
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	depaul-p	20130	CH	6325, 16631	PL
37	kaist	1781	Korea	9318	PL
38	cam-uk-p	786	UK	8918	PL
39	ucsc	5739	CA	2152	PL
40	princeton-p	88	NJ	7018	PL
41	jhu-p	5723	MD	7018	PL
42	ku	2496	KS	11317	PL
43	snu-kr	9488	Korea	4766	PL
44	bu	111	MA	209	PL
45	bkly-cs2	25	CA	2150, 3356, 11423, 16631	PL
46	northwestern	103	CH	6325	PL
47	bkly-cs3	25	CA	2150, 3356, 11423, 16631	PL
48	cmu	9	PA	5050	PL
49	dartmouth2	10755	NH	13674	PL
50	mit-pl	3	MA	1	PL
51	umd	27	MD	10886	PL
52	rpi	91	NY	6395	PL
53	stanford	32	CA	16631	PL
54	wustl	2552	MO	2914	PL
55	msu	237	MI	3561	PL
56	uky	10437	KY	209	PL
57	ac-uk	786	UK	3356	PL
58	caltech	31	CA	226	PL

4.3 Location

We define a link b as an *intra-AS link* if both ends of b belong to the same AS; otherwise, b is an *inter-AS link*. Figure 8 shows the ratios of intra-AS bottlenecks vs. inter-AS bottlenecks (the top figure) and that of intra-AS choke points vs. inter-AS choke points (the bottom figure) across different probing sources. We found that, for both bottlenecks and choke points, over half of them occur at intra-AS links. This is in contrast to the widely used assumption that bottlenecks often occur at the boundary links between networks.

For a choke point b in a probe set P, we compute its *normalized location* (denoted by $NL(b, P)$) in the corresponding network path in the following way. Let $A_1, A_2, ..., A_k$ denote the AS-level

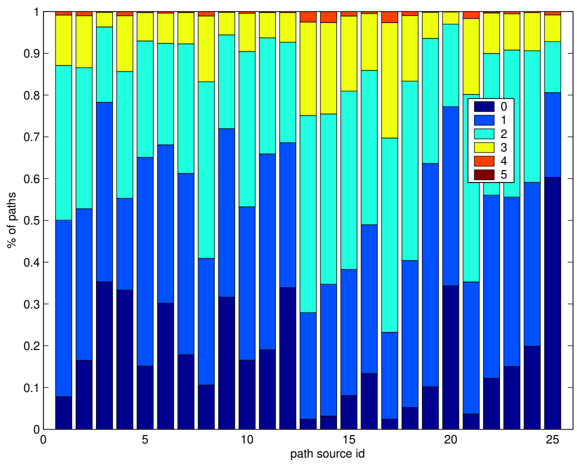

Figure 6: For each probing source, the number of probe sets that have 0 to 5 choke points.

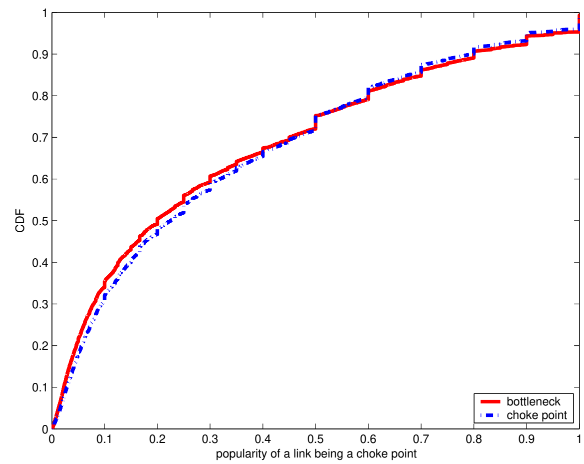

Figure 7: The popularity of a link being detected as a bottleneck/choke point.

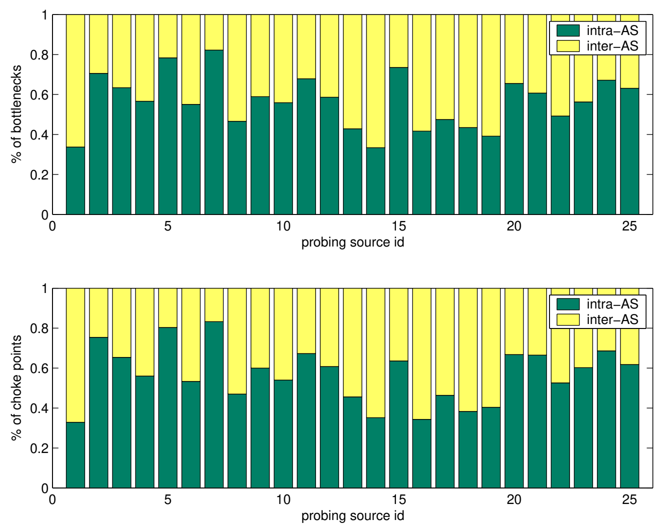

Figure 8: For each probing source, the ratio of intra-AS bottlenecks vs. inter-AS bottlenecks (the top figure) and that of intra-AS choke points vs. inter-AS chock points (the bottom figure).

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 $NL(b, P)$ is, the closer the choke point b is to the probing source. Thus, the normalized location of b in all its positive probes $P_1, P_2, ..., P_m$ $(0 \leq m \leq 10)$ is computed as

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

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

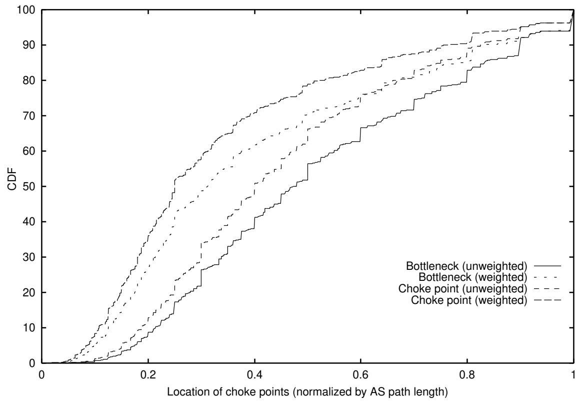

Figure 9: Cumulative distribution of the normalized locations of bottlenecks and choke points.

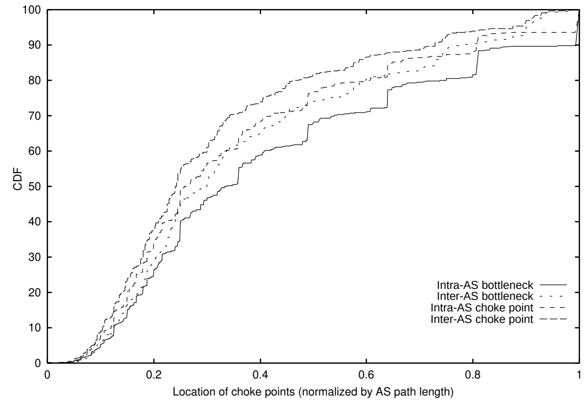

Figure 10: Cumulative distribution of the normalized location of intra-AS and inter-AS choke points.

Figure 9 shows the cumulative distribution of the normalized locations for both bottlenecks and chock points. The curves labeled "(unweighted)" show the distribution of the normalized location. The curves labeled "(weighted)" in Figure 9 show the distribution of the normalized location weighted by the number of probe sets in which a link is detected as a bottleneck or a choke point, because we have observed in Figure 7, some links are much more likely to be a bottleneck or a choke point than others.

We first observe that about 65% of the choke points appear in the first half of an end-to-end path (i.e., $NL(b, P) \leq 0.5$). Second, both the bottleneck and the choke point that are close to source are likely to be detected. This is possibly because Pathneck is biased to the earlier choke points. Third, by compared the curves for the choke points and the curves for the bottlenecks, we found that bottleneck location are more evenly distributed along the end-to-end path.

Figure 10 shows the cumulative distribution of the normalized location for intra-AS and inter-AS choke points, weighted by the number of probe sets in which a link is detected as a choke point. We observe that there is no significant bias on the location of choke points for inter-AS and intra-AS. Compared to the inter-AS choke points, the intra-AS choke points are slightly likely to appear later in an end-to-end path. This observation also holds on the intra-AS and inter-AS bottleneck links.

4.4 Stability

Due to the burstiness of the Internet traffic and occasional routing changes, the bottlenecks on an end-to-end path may change over time. In this section, we study the stability of the bottlenecks. In our measurements, we randomly selected 10 probing sources from the PlanetLab nodes (node 26 - 35 in Table 3). We sampled 30 destinations randomly from the set of destinations obtained in Section 4.1. Figure 11 shows the pseudo-code for the experiments conducted at each probing source. The measurement lasts three hours. From a given source, there are 45 probes to each of the

```
For each epoch {
    Repeat {
        Randomize the destination sequence;
        For each destination {
            Probe once;
            Sleep 1 second;
        }
    }
    if (end of epoch) {
        For each destination {
            Detect choke points based on probes
             in the current epoch;
        }
    }
}
```

Figure 11: Experiments for stability evaluation.

destinations. We divide these 45 probes into 9 epochs of length 20 minutes, the probe set in each epoch contains 5 probes between a pair of source and destination. Pathneck then reports choke points for each probe set.

Let $NumPositiveProbe_i(b)$ denote the number of probes where b is a choke point in probe set i. The stability of the choke point b over a period of n epochs is defined as

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

The same definition applies to bottlenecks.

Note that the range of $Stability(b)$ is [0.5, n] because $d\_rate \geq 0.5$. The dash curve in Figure 12 shows the cumulative distribution of the stability (at router level) for choke points over 9 measurement epochs. We observed that there are few links that are choke points in all of the 9 epochs. Compared with the stability of choke points, the bottlenecks (the solid curve in Figure 12) shows similar stability over measurement epochs.

Figure 13 shows the stability (at router level) of the intra-AS and inter-AS choke points over time. We found that inter-AS choke points are more stable than the intra-AS choke points. Comparing the top and the middle curves in Figure 13, we found that the intra-AS choke points are more stable at the AS level (the curve labeled “intra-AS-level”) than at the router level. Similar observations apply to the bottlenecks (not shown in Figure 13).

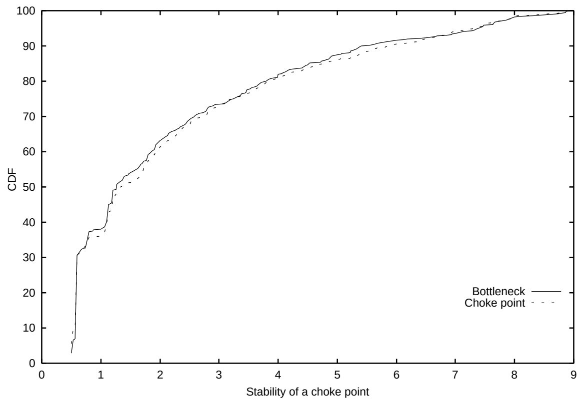

Figure 12: The stability of bottlenecks and choke points.

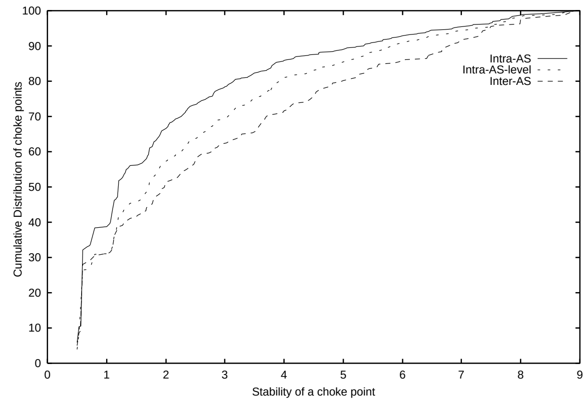

Figure 13: The stability of intra-AS vs inter-AS choke points over time.

5 Building a Bottleneck Map for Inference

In this section, we look at the problem of inferring a network path bottleneck without really probing that path. This ability can significantly reduce the amount of probing traffic.

5.1 Methodology

One naive approach is to first gather the layer-3 Internet topology, probe large number of destinations from varies vantage points to cover all links, and annotate each link with estimated bounds on available bandwidth. As exemplified by the Rocketfuel project [19], inferring the topology of a single ISP is already a difficult task. The scale of the Internet precludes us from using a complete layer-3 topology.

Although there are millions of layer-3 routers, the number of autonomous systems (AS) is significantly smaller. It is thus tempting to use an AS level graph and to annotate the edges to neighboring ASes with bandwidth estimates. However, by doing this, we over-simplify the diversity of links between a pair of ASes since we combine multiple peering links into a single path. Instead, we preserve all the peering links between a pair of ASes by treating them as parallel links, and we annotate separately them with available bandwidth measurement obtained from our tool. Since the number of intra-AS links can be huge, we cannot hope to get measurements for all of them. Instead, our data collection focuses on getting measurements for most of the peering links between ASes based on the BGP routing table information.

Since a router may have multiple links numbered by different IP addresses, we have to deal with the router alias issue. To figure out whether two IP addresses belong to the same router, we make use of the tool *ally* to resolve alias. The tool is described in detail in [19].

5.2 Inference method and experimental results

We divide the data we gathered into two parts. The first part of the probing data is used to annotate our inference graph. That is, we label links in the inference graph using an upper bound $B_u(L_i)$ and lower bound $B_l(L_i)$ for the available bandwidth of each link L_i obtained by using the algorithm presented in Section 2. The second part of the data is used for inference validation, i.e. we evaluate how well the available bandwidth information obtained from the inference graph matches the second set of probing results.

We use a pair of source and destination to identify a *path*. Let us denote the set of paths in the second part of the data as I, the *inference set*. For each path P in the inference set I, we look at $B_u(L_i)$ and $B_l(L_i)$ of each link $L_i \in P$ in our annotated topology. We use the link L_i with the lowest $B_u(L_i)$ as the inferred bottleneck link. We then compare the bound for the inferred bottleneck $\hat{L}_i$ with the probing result.

We define three types of inferences based on the information we have available in our annotated map. A *type-0 inference* corresponds to the case where we have upper bound estimates $B_u(L_i)$ for each IP link L_i in P in our annotated map. A *type-1 inference* corresponds to the case where we do not have upper bound estimation for at least one intra-AS link in P, but, we have upper bound estimation $B_u(L_i)$ for each inter-AS link L_i in P. Finally, a *type-2 inference* corresponds to the case where we do not have upper bound estimate for at least one inter-AS link in P, but we have $B_u(L_i)$ information for each intra-AS link L_i in P. In the remaining case we are missing both inter-AS and intra-AS links in the path and we do not try to infer the bottleneck because of lack of information. This eliminates 33% of the paths, leaving 67% to be used in the evaluation.

Table 4: Percentage of Correct and Incorrect Inferences

Type	Correct	Incorrect
0	19%	11%
1	15%	13%
2	6%	3%
Total	40%	27%

We first randomly select 60% of the probing sets (or paths) for annotating the AS-level topology map, leaving the remaining 40% for inference validation. There are 13,292 paths for which we have enough confidence that the bottleneck is inside the network (i.e. not on the first hop), according to the Pathneck algorithm. We focus on these paths that have bottlenecks. Table 4 shows the correct and incorrect percentages of our inference. We correctly inferred the bottleneck locations of 19% of the 13,292 paths for type-0 inference. Sum up all three types of inference, we can correctly infer 40% (out of 67%) of the total bottleneck paths, while in 27% (out of 67%) the inference is incorrect, i.e. the bottleneck link is off by one or more hops.

We decided to investigate the cases where we identified the wrong bottleneck more carefully. For type-0 inferences, 60% of the incorrect inferences are due to the fact that we do not have much information on the true bottleneck link, i.e. the true bottleneck link only appears as a bottleneck for fewer than 2 paths in the set of data we used to annotate the map because the link was covered by very few probings. For the incorrect inferences of type-1, 41% of them are due to the fact that we do not have any information of the true bottleneck link in our bottleneck map. Overall, this leaves only about 10% (out of 67%) of the paths with incorrect results.

6 Avoiding Bottlenecks

6.1 Overlay Routing

Overlay routing or application layer routing refers to the idea of going through one or more intermediate nodes before going to the destination. The intermediate nodes act as application layer routers or overlay nodes so they forward traffic but usually do not do any additional processing. Previous studies [20, 17] 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 of path inflation [21, 22].

Overlay routing can thus be used to avoid bottleneck links in the underlying IP path, thereby improving application level performance in terms of throughput or available bandwidth. 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. 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 usefulness 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 select 27 RON and Planetlab nodes as both the source nodes and overlay nodes. Using a BGP table from a large tier-1 ISP, we sampled 200 random IP addresses from a diverse set of prefixes in the BGP table; each IP address originates from a different AS and ends with ".1" to minimize the 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 non-stationary 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, it randomly selects 8 nodes from the set of 27 source nodes. For each of these 8 nodes, S again probes each of them 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: $S_1 \rightarrow D$, $S_1 \rightarrow S_2$, and $S_2 \rightarrow D$. The bottleneck link available bandwidth is calculated based on the largest gap value in the path across the 10 probing results.

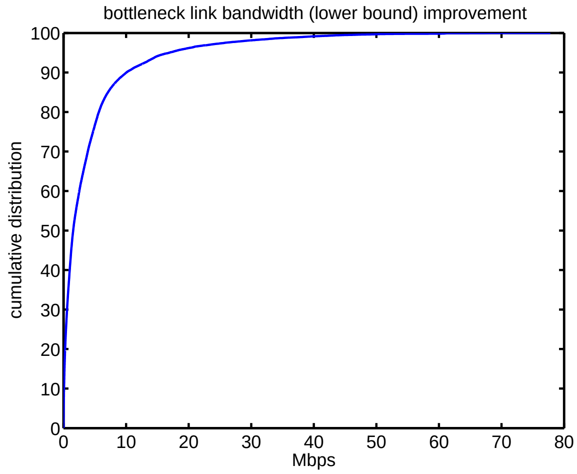

Figure 14: Improvement in reducing lower bound of the bottleneck link available bandwidth by overlay routing.

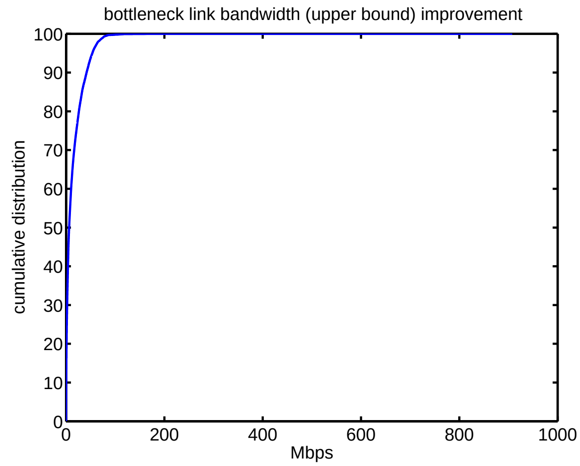

Figure 15: Improvement in reducing upper bound of the bottleneck link available bandwidth by overlay routing.

For each destination from a given source, we calculate the lower bound and upper bound of the bottleneck link available bandwidth. When composing two paths such as $S_1 \rightarrow S_2$ with $S_2 \rightarrow D$, 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. We only consider the effectiveness of overlay routing by going through a single intermediate node, similar to previous studies.

Results: We now present the results of our overlay routing study. Of the $63,440$ overlay attempts, *i.e.,* routing to a destination by going through an intermediate node, 52.72% are *successful*. We define success loosely as either the lower bound of the bottleneck link increasing or the upper bound increasing. If we require both bounds to increase, the success rate is 15.92%; 17.39% and 19.40% are the breakdown for the cases when only the lower bound of the bottleneck bandwidth value increases or only the upper bound increases. The biggest difference in the increase of the upper bound of the bottleneck link available bandwidth is more than 900Mbps; the corresponding value for the lower bound is 77Mbps. The distribution of the improvement in bottleneck available bandwidth values for upper bound and lower is shown in Figures 14 and 15. The two figures show that most improvement in the upper bound is below 100Mbps; that for the lower bound is 20Mbps.

We now examine more closely how useful the overlay nodes are for each source node for the 200 randomly selected destinations. We studied whether only specific overlay nodes are helpful for a given source node and found that at almost all 27 locations, more than 90% of the overlay nodes can be used for reaching *some* destinations with improved performance. A few exceptions stand out: mazu1 finds only 8 out of 27 nodes useful in terms of improving available bandwidth, and the Cornell site finds 67% or 18 nodes helpful. It is worthwhile to investigate these two sites further. Most likely the path between these two sites and the chosen destinations have quite good performance already, hence overlay routing does not help. Among the other sites, where most of the randomly selected overlay nodes seem to help in reducing the bottleneck link bandwidth, we studied the data in more details 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 at 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 achieve over 92% success rate when used as overlay nodes. 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. Typically overlay node selection requires 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 do probing 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 static bottleneck links and overlay nodes that always 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 it exchanges routing information with its upstream providers via Border Gateway Protocol (BGP). The original motivation for multihoming is to achieve resilient network connectivity or redundancy in case the connectivity to one ISP fails or one of the ISP experiences severe routing outages. Enterprise networks usually require higher level of reliability for their connectivity to the rest of the Internet. Multihoming can not only increase the availability of network connectivity, but it can also improve performance by allowing multihomed customers to route traffic through different upstream providers based the routing performance to a given destination. A recent study [23] has shown that, by carefully choosing the right set of upstream providers, high-volume content providers can gain significant performance benefit from multihoming.

The reliability 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 benefits of multihoming will be sharply diminished since it will not be able to recover from failures 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 on avoid bottleneck links by quantifying the reduction the bottleneck link available bandwidth.

Table 5: Grouping based on coarse-grained geographic close proximity.

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

Methodology: To understand the effect of multihoming on avoiding bottleneck links, one would ideally probe from the same location by going through different upstream providers to several selected destinations. A previous study [23] 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 24 geographically close probe sources belonging to different organizations as shown in Table 5. We approximate the members in the same group to be nodes within the same city. Arguably this is a simplification; however, the geographic distance between any two nodes within the same group is small enough 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 bottleneck link available bandwidth of the path to the same destination from each member in the group. If the improvement or increase 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 it a success.

Results: Among all $42,285$ comparisons we are able to make across all probe locations, more than 78% of them are successful cases. This is very encouraging and shows that multihoming significantly helps in avoiding bottleneck links. However, we emphasize that the result is artificially inflated by the way the probe destinations are selected in reducing the chance of discovering the last mile bottleneck link at the destination site. Many of the probe destinations selected are not stub networks and most of them do not correspond to addressable end hosts. Furthermore, firewalls often prevent outgoing ICMP packets and thus rendering Pathneck ineffective at identifying last mile bottleneck links near the destination site. We plan to improve the tool in this regard in the near future. Nevertheless, our results also indicate that multihoming is very effective at avoiding any last-mile bottleneck links near the source or bottleneck links inside the core. To be more conservative, if we require both the upper bound as well the lower bound to improve by 50%, then the success rate is reduced to exactly 50%.

Examining the success rate for each group in Table 5 reveals some interesting characteristics.

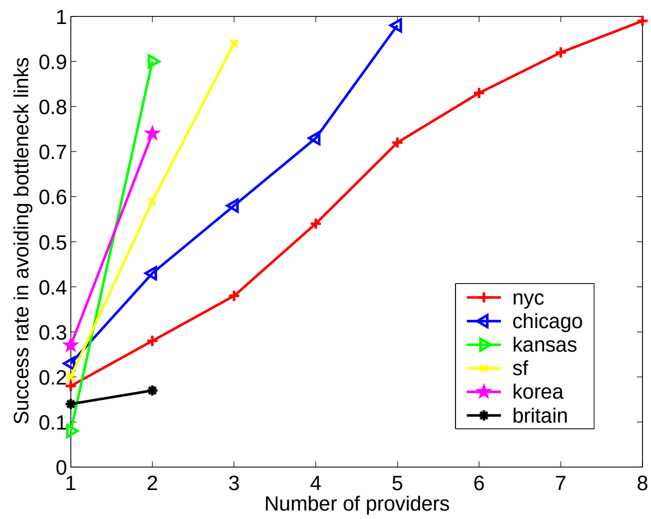

Figure 16: Improvement in avoiding bottleneck links giving increase in providers.

First of all, the bigger the group, the higher the success rate. For the two non North American sites – Britain and Korea, the success rate is significantly lower. Our conjecture to explain this is that the transoceanic links become the main bottleneck link and cannot be avoided by choosing a nearby source node within the same country.

It is intuitive that there is diminishing returns in multihoming to more providers in improving performance. A previous study [23] has shown this with respect to reducing Web object download time. We now examine such effect in reducing the bottleneck link bandwidth value. Figure 16 shows that there is *steady* improvement in reducing bottleneck link bandwidth values as the success rate continuously to increase. We plan to investigate this further with more probe source locations.

Discussion: The results in the multihoming study is quite encouraging in terms validating the usefulness of Pathneck in understanding the benefit of multihoming as well as the actual benefit. Similar to overlay routing, Pathneck is extremely useful in making route selection decisions.

7 Related Work

Bandwidth estimation techniques, specifically available bandwidth estimation algorithms [8, 1, 3, 2, 4, 5], measure network throughput, which is very closely related to congestion. However, they provide no location information for the congestion point. Also, all these tools, except cprobe [8], need the cooperation of the destination. That makes them very hard to deploy.

Packet loss rate is another metric that is related to user traffic performance, especially for TCP traffic [24]. Besides tools that can directly measure the network path loss rate, such as Sting [25], the tool Tulip [26] can accurately pin point the packet loss position.

The tools that are most closely related to Pathneck include Cartouche [27], Packet Tailgating [28], BFind [7] and pathchar [6]. Cartouche uses a packet train with different packet size to measure the bandwidth for any segment of the network path. Packet Tailgating works by setting the packet interval within the same packet train. It also combines the load packets and measurement packets, but instead of letting measurement packets expire, it lets load packets expire. Both Cartouche and Packet Tailgating need two end control.

BFind adds a steady UDP flow to the network path, and gradually increases its throughput. At the same time, traceroute is used to monitor the RTT changes from all the routers on the path. When the UDP flow throughput approaches the available bandwidth along the path, the RTT from the source to the bottleneck router is expected to change more significantly than that to non-bottleneck routers. One of the problems of BFind is that the UDP flow generates a heavy measurement overhead, which is undesirable for a general purpose probing tool.

Pathchar [6] 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 noises due to factors such as queueing delay, pathchar needs to send a large number of probing packets, picking out 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, there is not a lot of analysis on Internet bottlenecks. We are only aware the analysis in [7], which shows that most of the bottlenecks are in the edge links and peering links. Due to the limitation of BFind, the results are inconclusive. Pathneck overcomes many of the limitations in BFind. Based on the large BGP database that we can access, we can also probe the Internet in a more systematic way, thus giving our analysis a broader scope.

Several studies have shown that overlay routing [17, 20], multi-path routing [29, 30, 31], and multihoming [32, 33] benefit end user data transmission by reducing the packet loss rate and increasing end-to-end throughput. Their experiments mainly focused on link failure or packet loss. In contrast, our work takes a complete different angle looking at this problem by identifying the location of the tightest links and by examining the use of overlay routing and multi-homing to avoid bottlenecks. Our work shows the benefit of overlay routing and multihoming and suggest 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 a novel probing technique called Recursive Packet Train (RPT). Pathneck allows end users to efficiently and accurately locate bottleneck points to destinations on the Internet. We show that Pathneck can successfully detect bottlenecks for over 70% of paths from most of our Internet probing sources. Based on the extensive Internet measurements we find that over 50% of the bottlenecks are within a single ISP, and most of the bottlenecks are pretty stable. We also achieve fairly good performance on the bottleneck inference for those paths where we do not have probing data. With the bottleneck location information, we also find over half of the overlay routing attempts to avoid bottlenecks are successful, and multihoming can successfully avoid bottlenecks in over 78% of the cases.

In this paper, we only analyze some aspects of Internet bottlenecks. There are many other important research issues we can look at:

1. **The time properties of bottleneck changes**. All the results studied in this paper are collected within two weeks. Although we already see some interesting bottleneck dynamics, we can not make conclusive statement on the dynamic properties yet. We hope to do longer time scale probing in the future to study this property.

2. **The impact of Internet topology on bottleneck locations**. We know that the Internet topology keeps changing, and it would be very interesting to know how that impacts the bottleneck positions. Knowing that is very useful for network maintenance planning.

3. **Improvement on Pathneck**. An attractive feature of Pathneck is its low probing overhead. It allows us to quickly collect the probing results for a large amount of Internet paths. But in this paper, we did not study the properties related with the load packet size and the number of load packets in a RPT. For example, how do they affect the measurement accuracy, and how to automatically set their value based on different path properties. That will help us to further reduce the overhead both of the probing itself, and of the management of the probing. That is the key to make Pathneck a general purpose tool like ping and traceroute.

4. **AS-based bottleneck analysis**. We treat different tiers of ASes the same in this paper, focusing on understanding the bottleneck properties in the whole Internet. It would also be very intriguing to understand the bottleneck properties for different tiers of ASes, and the difference among different ASes in the same tier.

References

[1] Manish Jain and Constantinos Dovrolis, "End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput," in *SIGCOMM 2002*, Pittsburgh, PA, August 2002.

[2] Ningning Hu and Peter Steenkiste, "Evaluation and characterization of available bandwidth probing techniques," *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, vol. 21, no. 6, August 2003.

[3] Bob Melander, Mats Bjorkman, and Per Gunningberg, "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks," in *IEEE Globecom - Global Internet Symposium*, San Francisco, November 2000.

[4] Vinay Ribeiro, Rudolf Riedi, Richard Baraniuk, Jiri Navratil, and Les Cottrell, "pathchirp: Efficient available bandwidth estimation for network paths," in *Passive and Active Measurement Workshop 2003*, La Jolla, CA, April 2003.

[5] Jacob Strauss, Dina Katabi, and Frans Kaashoek, "A measurement study of available bandwidth estimation tools," in *Internet Measurement Conference (IMC) 2003*, Miami, Florida, USA, October 2003.

[6] Van Jacobson, "pathchar - a tool to infer characteristics of internet paths," 1997, presented as April 97 MSRI talk.

[7] Aditya Akella, Srinivasan Seshan, and Anees Shaikh, "An empirical evaluation of wide-area internet bottlenecks," in *IMC'03*, Miami, Florida, October 2003.

[8] Robert L. Carter and Mark E. Crovella, "Measuring bottleneck link speed in packet-switched networks," Tech. Rep., Boston University Computer Science Department, March 1996.

[9] J. Postel, "Internet control message protocol," September 1981.

[10] Ramesh Govindan and Vern Paxson, "Estimating router icmp generation delays," in *PAM'02*, March 2002.

[11] Kostas G. Anagnostakis, Michael B. Greenwald, and Raphael S. Ryger, "cing: Measuring network-internal delays using only existing infrastructure," in *INFOCOM 2003*, April 2003.

[12] "Emulab," http://www.emulab.net.

[13] "Dummynet," http://info.iet.unipi.it/ luigi/ip_dummynet/.

[14] "Abilene," http://abilene.internet2.edu/.

[15] "Abilene network monitoring," http://www.abilene.iu.edu/noc.html.

[16] "Planetlab," https://www.planet-lab.org.

[17] RON, "Resilient Overlay Networks," `http://nms.lcs.mit.edu/ron/`.

[18] "University of Oregon Route Views Project," `http://www.routeviews.org/`.

[19] Neil Spring, Ratul Mahajan, and David Wetherall, "Measuring isp topologies with rocketfuel," in *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*. 2002, pp. 133–145, ACM Press.

[20] Stefan Savage, Tom Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, , and John Zahorjan, "Detour: a case for informed internet routing and transport," *IEEE Micro*, vol. 19, no. 1, 1999.

[21] H. Tangmunarunkit, R. Govindan, and S. Shenker, "Internet Path Inflation Due to Policy Routing," in *Proceedings of SPIE ITCOM*, 2001.

[22] Neil Spring, Ratul Mahajan, and Tom Anderson, "H. Tangmunarunkit and R. Govindan and S. Shenker," in *SIGCOMM*, 2003.

[23] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman, "A Measurement-Based Analysis of Multihoming," in *Proc. ACM SIGCOMM*, September 2003.

[24] Jitendra Padhye, Victor Firoiu, Don Towsley, , and Jim Kurose (U. Mass), "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. of ACM SIGCOMM'98*, Vancouver, British Columbia, Canada, September 1998.

[25] Stefan Savage, "Sting: a TCP-based network measurement tool," in *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems*, Boulder, CO, October 1999, pp. 71–79.

[26] Ratul Mahajan, Neil Spring, David Wetherall, and Thomas Anderson, "User-level internet path diagnosis," in *SOSP'03*, The Sagamore, Bolton Landing (Lake George), New York, October 2003.

[27] Khaled Harfoush, Azer Bestavros, and John Byers, "Measuring bottleneck bandwidth of targeted path segments," in *Proc. IEEE INFOCOM*, San Francisco, CA, April 2003.

[28] Vinay Ribeiro, "Spatio-temporal available bandwidth estimation for high-speed networks," in *The First Bandwidth Estimation workshop (BEst)*, San Jose, CA, December 2003.

[29] David G. Andersen, Alex C. Snoeren, and Hari Balakrishnan, "Best-path vs. multi-path overlay routing," in *Internet Measurement Conference*, October 2003.

[30] N.F. Maxemchuk, *Dispersity Routing in Store and Forward Networks*, Ph.D. thesis, University of Pennsylvania, May 1975.

[31] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *J. of the ACM*, vol. 36, no. 2, April 1989.

[32] Aditya Akella, Bruce Maggs, Srinivasan Seshan, Anees Shaikh, and Ramesh Sitaraman, "A measurement-based analysis of multihoming," in *Proc. ACM SIGCOMM*, August 2003.

[33] "Routescience," http://www.routescience.com.