RPT: A Low Overhead Single-End Probing Tool for Detecting Network Congestion Positions

Ningning Hu, Peter Steenkiste

December 20, 2003
CMU-CS-03-218

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Detecting the points of network congestion is an intriguing research problem, because this information can benefit both regular network users and Internet Service Providers. This is also a highly challenging problem, because the Internet is designed to provide only end-to-end services, and its internals are in principal invisible to end users. Current techniques used to detect bottleneck positions have problems such as high probing overhead and low measurement accuracy. In this paper, we propose using Recursive Packet Trains (RPT) to detect the network congestion position. RPT combines two types of probing packets – measurement packets and load packets – in a single probing packet train. The idea is to let load packets generate a packet queue on the router, and to use the measurement packets at the beginning and the end of the train to measure the packet train length. By detecting the changes in the packet train length, we can derive the congestion points of the network path. RPT has the advantages that it only needs single-end control and that it has relatively low overhead. In this paper, we present the algorithm and evaluate it using both testbed experiments and Internet experiments.

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 .

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of DARPA or the U.S. Government.

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

1 Introduction

In this paper, the congestion position is defined as a network link or router that determines or significantly affects the data transmission throughput along a network path. Knowing the congestion positions is extremely useful for both the end users and the Internet Service Providers (ISPs). The 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.

Measuring network performance such as the end-to-end available bandwidth has been an active research area. However the proposed techniques 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 [10, 8, 12, 15, 17]. Second, for tools that do measure the hop-by-hop performance, the measurement overhead is often very high. This category includes Pathchar [9] and BFind [5].

We regard two properties as important for a network congestion point detection tool: single-end control and low overhead. In this paper, we propose using Recursive Packet Trains (RPT) to achieve these two goals. The key idea is to combine measurement packets and load packets in a single probing packet train. 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 derived.

In this paper, load packets emulate the behavior of regular data packets. That is because during their transmission, they interleave with the background traffic, which enables us to capture the network properties that we want. Measurement packets are small probing packets, similar to those used by standard network tools like ping, traceroute, etc. The measurement packets are used when we do not require any interaction between the probing packets and the background traffic packets.

This paper has two parts: the algorithm description and the performance evaluation. In Section 3, we describe the idea of Recursive Packet Trains, and present a preliminary algorithm to detect the congestion position. The rest of the paper is devoted to the performance evaluation, which includes both Emulab testbed experiments (Section 4) and Internet experiments (Section 5). We start with a discussion of related work.

2 Related Work

The most widely used active probing tools are ping and traceroute. Ping uses an ICMP echo packet to measure the round-trip time (RTT) to a specific destination. Traceroute sets the TTL in the IP header to trigger responses from the routers along the network path, thus collecting the hostname and RTT of the routers. However, the only performance information provided by these tools is RTT, which is not directly related to congestion.

Bandwidth estimation techniques, specifically available bandwidth estimation algorithms [10, 8, 12, 15, 17], measure network throughput, which is more closely related to congestion. However, they provide no location information for the congestion point. Also, all these tools, except cprobe, 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 the performance of TCP traffic [13]. Besides tools that can directly measure the network path loss rate such as Sting [16], there has been a tool Tulip [11] that can accurately pin point the packet loss position.

The tool is most closely related to RPT are BFind [5] and pathchar [9]. 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 [9] was designed earlier. It is used to estimate 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 out 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.

3 The Probing Algorithm

The key intuition that motivates this algorithm is: when a probing packet train passes through the routers along a network path, its total length changes with the change of the available bandwidth on each link. The change due to two reasons: the packet transmission time is different on links with different capacities, and the interaction with background traffic packets can increase or decrease the total packet train length. We expect the largest change to happen at the congestion points. To implement this idea, we need a probing technique that can measure the train length on *each* link, which is done by a novel packet train design — Recursive Packet Train (RPT), and an algorithm that can extract the exact congestion positions. In this section, we first describe the structure of the RPT, we then present the algorithm used to detect the congest positions, and finally we discuss the properties of the RPT technique.

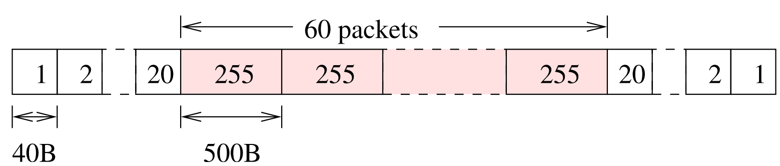


Figure 1: Recursive Packet Train (RPT). The number in each packet is the TTL value.

3.1 Recursive Packet Train

An example of a Recursive Packet Train is shown in Figure 1. In this figure, every box is a UDP packet, the numbers in the boxes are the TTL values. The whole probing packet train is composed of two types of packets — *measurement packets*, and *load packets*:

1. Measurement packets are standard traceroute packets, i.e., they are 40 bytes UDP packets, with properly filled-in payload fields. There are 20 measurement packets, at either end of the packet train. The TTL value of each measurement packet is linearly incremented from the head/tail packet, and the head/tail packet has TTL value 1. The train in Figure 1 can only measure a network path with no more than 20 hops, because it has only 20 measurement packets on both ends, but we can easily add more measurement packets for a longer path.

2. Load packets are used to generate a packet train with a measurable length along the network path. The load packets should be large packets. The exact size is configurable in our implementation. In the following experiments, we set it to 500 bytes.

   The number of packets in the packet train determines the amount of background traffic that it can interact with. As a result, this number should be fairly large. In our experiment, we set it empirically in the range of 60 to 100. Automatically configuring the number of probing packets is future work.

RPT works as follows. The user sends out all the packets back-to-back. When they arrive at the first router, the first and the last packet will expire and be dropped because their TTL values are 1, resulting in two ICMP packets being generated [14]. The other packets in the train are forwarded to the next router, with their TTL decremented by 1. Since the TTL values in a RPT are set recursively, the above process is repeated on each subsequent router. This method of probing is called *Recursive Probing*.

The key to the probing procedure is as follows: *the time gap between the two ICMP packets from each router is a close approximation of the time length of the packet train on the incoming link of that router.* This is because: (1) each router only drops the head and the tail measurement packets, and (2) the measurement packet size is much smaller than the total size of the train, i.e., the change of packet train length due to the dropping of the measurement packets can be neglected. In the following, we refer to the interval between two ICMP packets from the same router as the *gap value*.

3.2 The Detection Algorithm

RPT provides a way to get the probing packet train length on each network link along the path, i.e., a sequence of gap values. With these gap values, it is possible to find out the exact congestion position — we expect the train length to change significantly at the congestion point. We now present an algorithm that can detect the congestion position from a sequence of gap values.

Figure 2 shows the pesudo-code for the detection algorithm. Intuitively, the algorithm searches for the positions where the gap value has a significant change. To do this, we classify the gap value changes into two categories: *switch* points, and *stage* points. A switch point is defined as a change that lasts for at least two hops; while a stage point is just a change for a single hop, whose gap value may or may not be the same in the next hop. Figure 3(a) illustrates this difference.

The reason for making this distinction is to filter out gap value changes due to non-congestion factors. As will be discussed in the next section, there are two major non-congestion factors — the time used by routers to generate ICMP packets and reverse path queueing. To distill the real congestion induced changes, we rely on the observation that congestion induced gap values will

```
Algorithm Congestion_Detect(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 outliers;
    fix the hill/valley point;

    /* search for switch points */
    if (len >= 12) { /* look for 3 switch points */
        switch[0..2] = SEGMENT3(gap);
    }   else   if (len >= 6) { /* look for 2 switch points */
        switch[0..1] = SEGMENT2(gap);
    }   else   { /* only look for 1 switch point */
        switch[0] = SEGMENT1(gap);
    }

    /* search for the stage point */
    for i in (1..(len − 1)) {
        diff[i − 1] = abs(gap[i] − gap[i − 1]);
    }
    stage = get the largest 3 elements from diff;

    compare with the previous 4 stage record, if a hop is labelled
        for less than 4 times, it will not be included in the comparison in the next step;

    /* output */
    compare switch with stage, output the 3 hops with the largest change;
}
```

Figure 2: The Congestion Detection Algorithm

typically be maintained by subsequent routers. Therefore, the gap value sequence is expected to have a square wave shape. Otherwise, the gap change due to non-congestion reasons is more likely to be inconsistent. So if we detect a square wave in the sequence of gap values, as defined by the switch point, we have a higher confidence that it is caused by congestion.

To search for the switch point, we use a simple brute-force algorithm. By default, we search for 3 switch points. We arbitrarily split the gap sequence into 4 segments, with at least 2 points in each segment. This requires the sequence to have at least 8 points. In our algorithm, we only search sequences that have at least 12 points, so that we can compare among several combinations. For each segment, we compute its average, and the distance of each point to this average. We use the sum of the computed distances to indicate how good the segmentation approaches a square wave — the smaller, the better. We output the segmentation with the minimum sum. As just noted, we need at least 12 points to search for 3 switch points. For shorter sequences, we only look for 2 (for

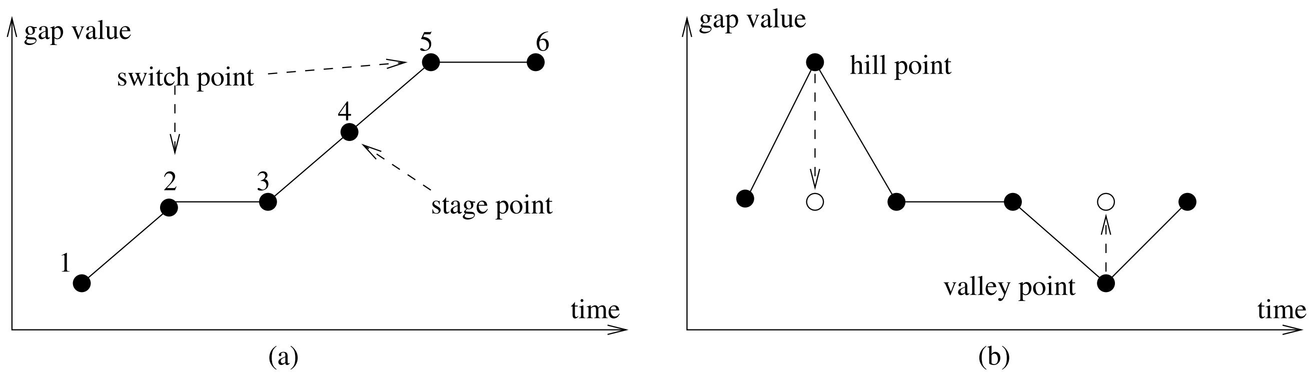


Figure 3: Switch/stage point & hill/valley point

a 6-11 point sequence) or 1 (for a sequence with less than 6 points) switch points.

For the stage point, since it is for only one hop, it is very hard to use the probing result of a single RPT to determine whether it is congestion induced. Instead, we compare multiple consecutive probing results to decide whether a stage point on a particular hop occurs consistently. For this reason, we need to maintain a *stage point history*. In our algorithm, we compare 5 probings. If a stage point at a hop occurs at least 4 times, we label it as congestion induced.

So far, we did not consider measurement noise, which is unavoidable in probing. Here, we list two of the important techniques that deal with noise and packet loss:

1. We need two ICMP packets from each router to compute the gap value. If at least one of them is lost, we will have a 0 gap value. If over half of the gap values are 0, we discard the whole sequence.

2. We need to modify the *hill/valley* point (Figure 3(b)). A hill point is defined as a point p_2 in a triple-point group: p_1, p_2, p_3, with gap values satisfying $2g_1 < g_2 > 2g_3$. A valley point is defined similarly, except the condition is changed to $g_1 > 2g_2 < g_3$. Intuitively, a hill/valley point is a burst in a time series measurement. But in the RPT probing, this type of burst is not expected, since the gap value on one router tends to be maintained or increased by the next router. For this reason, we regard a burst as an indication of measurement noise. In our algorithm, we replace g_2 with the closer gap value of its two neighbors.

3.3 Properties of RPT

From the above discussion, we can see that the main properties of RPT include:

1. Time consistent measurements. As described above, the gap values from different routers are triggered by the *same* RPT, and the measurement times are very close. That allows us to directly compare the measurements from adjacent routers. This property also enables us to catch all the congestion points along the path that change the packet train length. That is, RPT is potentially capable of detecting multiple congestion points. Note that RPT is biased to early congestion points, because a congestion point in an early part of the path can hide a later congestion with similar properties.

2. Single-end control. RPT does not need the cooperation of the destination, and can thus be easily used by a regular network user.

3. Low overhead. For the case in Figure 1, one probing only needs 100 packets. For better measurement accuracy, we may need multiple RPTs to do the probing. Even so, this is an extremely light-weight probing technique, comparing with pathchar and BFind,

In terms of the measurement accuracy, the following factors need to be considered:

1. ICMP packet generation time. A router needs time to generate the ICMP response packets. That 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 equal the packet train length when passing that router. Fortunately, measurements in [7] and [6] show that the ICMP packet generation time is pretty small; in most cases it is between 200us and 500us. So if the packet train length is much larger than 500us, it is reasonable to neglect this generation time. Since most Internet paths have a bottleneck link with a capacity of less than 100Mbps, and we use 100 load packets for the Internet experiment, the corresponding packet train length is larger than 4ms, which we regard as large enough to ignore the ICMP packet generation time.

2. Queueing delay on the reverse path. When the ICMP packets are sent back to the sender, they can experience queueing delay due to 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 related work that has measured this value. In our algorithm, we try to reduce the impact of this factor by filtering out the measurement outliers.

RPT also has some structural limitations:

1. We find that network firewalls often only let through 40 bytes UDP packets that strictly conform to the traceroute packet format (with properly filled-in payload fields), and drop any other UDP probing packets, such as the load packets in a RPT. So if the sender is behind such a firewall, RPT will not work. Similarly, if the destination is behind a firewall, the measurements for those hops behind the firewall can not be obtained by RPT.

2. Even if there is no firewall on the destination side, RPT may not be able to measure the packet train length on the last link, because the ICMP packets sent by the destination host can not be used. Theoretically, the destination is supposed to generate an ICMP destination port unreachable message for each packet in the packet train. But 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.

4 Testbed Validation

We use both the Emulab testbed and Internet paths to evaluate RPT. The Emulab testbed provides a fully controlled environment to study different aspects of the technique, while Internet experiments are necessary for studying RPT's performance with real background traffic. In this section, we focus on the Emulab testbed experiments. The Internet experiments are discussed in the next section.

With a fully controlled testbed, we can separate the factors that determine the final measurements, focusing on obtaining a preliminary understanding of the algorithm properties. In this section, we study the performance of RPT with (1) a single flow of background traffic, (2) multi-flow background traffic, (3) queueing delay on the reverse path. In all three cases, the background traffic is generated using iperf [4] UDP flows.

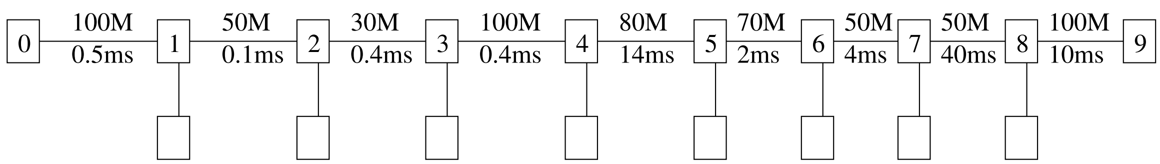


Figure 4: Testbed configuration. Hop 0 is the probing sender, hop 9 is the probing destination. Hop 1 - 8 work as routers, the blank boxes are used as iperf sender/receiver to generate the traffic load.

The testbed is created using the Emulab [3] facilities. Figure 4 shows the testbed configuration. Because the physical Emulab link capacity is 100Mbps, we set the bottleneck link capacity as 30Mbps in this experiment, using the dummynet functionality provided by Emulab. The link delays are roughly set based on a traceroute measurement from a CMU host to yahoo.com. Note that all the hosts on the testbed are PCs, not routers, so some properties such as the ICMP generation time are not exactly the same as those of a real router. As the result, the following testbed experimental results ignore some of the router related factors.

4.1 Single-Flow Background Traffic

In this section, we focus on two factors that determine the degree of a congestion: the link capacity, and the traffic load.

In Figure 5, we add no background traffic, and the congestion point is purely determined by the link capacity. We probe the path 10 times; the gap values on each hop are plotted in Figure 5. We can see that the maximum gap changes appear at R3, whose input link is the bottleneck link. There are also small gap increases at the subsequent routers. One possible reason is time measurement error. Another reason is that the gaps in the packet train are not evenly distributed, so when the link capacity decreases, some of the small packet gaps can still increase, thus increasing the total packet train length.

Figure 5 plots the base case of the probing results. In the following, we add different background traffic load on different links to study RPT's performance. Unless explicitly stated, the experiments are done using the following procedure. We gradually increase the iperf UDP traffic load, in increments of 10% of the link capacity. For each load, we probe the path 10 times, with 5 seconds sleeping time in between.

Figure 6 shows the results when adding background traffic on the link between R3 and R4. The other links are still free of load. Here, we choose to present the measurement from 40% to 90%

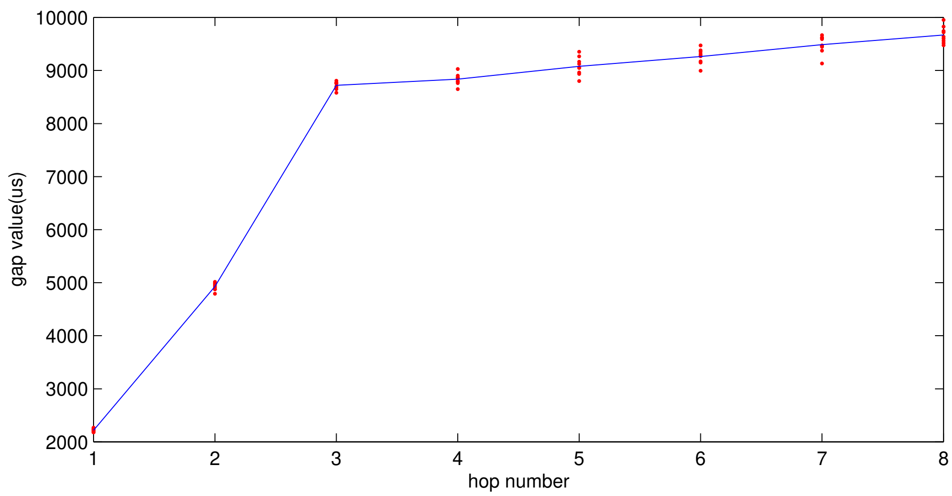


Figure 5: Measurements with no background traffic. The line connects the median values on each hop.

traffic load. We can see that, only when the available bandwidth on this link is less than 30Mbps (the 80% graph) does the congestion point change from R3 to R4, which correctly reflects the real bottleneck change.

Figure 7 presents the measurement when adding background traffic on the link between R6 and R7, while keeping the other links free of load. Similarly, only when the available bandwidth on this link is less than 30Mbps (starts from the 50% point in the figure) does the congestion point change to R7, which is again the real bottleneck change.

4.2 Multi-Flow Background Traffic

In this section, we study how RPT works with multiple competing traffic flows. We focus on two scenarios. In the first scenario, we add two background flows in the same direction, but on different links. This is done as follows: we first add a 20Mbps flow from R4 to R8, then we gradually increase the traffic between R6 and R7, in 3Mbps increments. As in the previous experiments, we do 10 probings for each setting, for 9 settings in total. In the second scenario, we configure the two traffic flows in opposite directions: we first fix the load from R8 to R4 to 20Mbps, and then gradually increase the load from R6 to R7, in increments of 10% of the link capacity.

Table 1 presents the experimental results. Since our probing algorithm uses the first 4 probings as the stage point history, we can obtain 6 final measurements, and each hop can be labelled at most 6 times. The numbers in the table are the number of times that a router is labelled as a congestion point.

For the first experiment, the real bottleneck is on $< R6, R7 >$, and the probing algorithm starts to identify it when the traffic load reaches 20% of the residual capacity, i.e. when the real available bandwidth is 24Mbps. The measurement misses the real congestion point at R7 for 3 times in the 10% case, due to the small difference in available bandwidth of 27Mbps on $< R6, R7 >$ and 30Mbps on $< R2, R3 >$. These results exemplifies an important property of RPT: the measure-

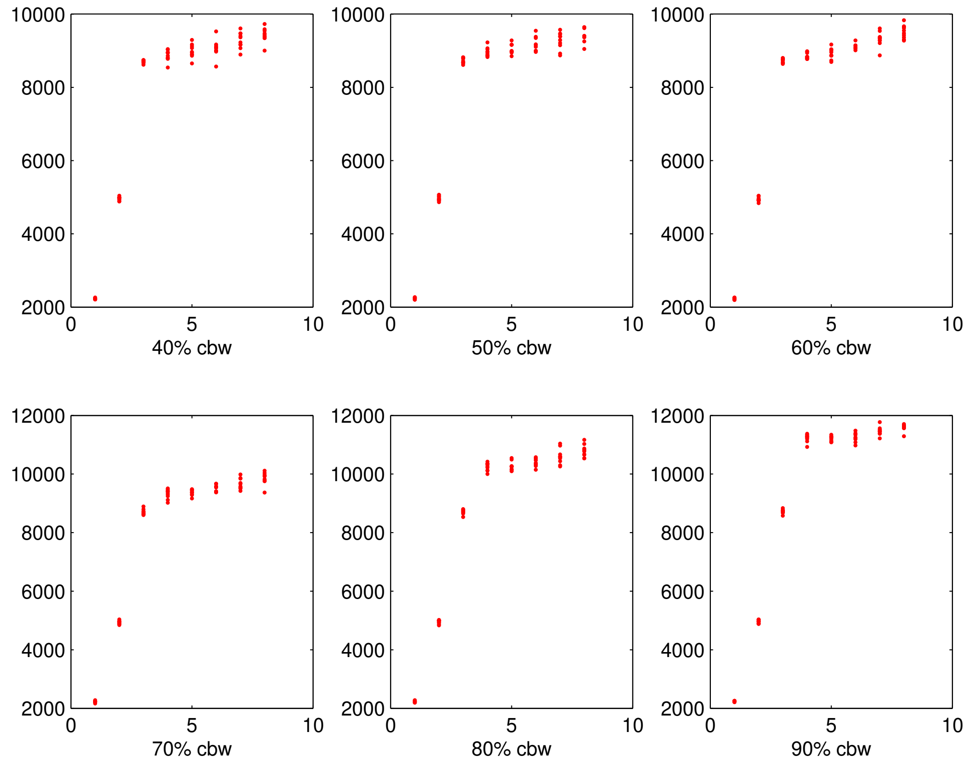


Figure 6: Measurement with different background traffic on the link <R3, R4>. "cbw" mean background traffic load. In each graph, the x-axis is the hop number, the y-axis is the gap value in micro-seconds.

ment is biased to the congestion point in the early part of the path, i.e., a congestion point can hide a later point with a similar level of congestion. This is also why the tool misses the congestion point $< R6, R7 >$ in the second experiment when the load is 40% and 50% (corresponding to available bandwidths of 30Mbps and 25Mbps).

4.3 Queueing Delay on the Reverse Path

During the probing of RPT, when ICMP packets are sent back to the sender, they may experience queueing delay due to the traffic in the reverse direction, thus increasing measurement error. To understand this effect, we experiment with traffic on links $< R3, R2 >$ and $< R7, R6 >$. That is, we add traffic in the reverse direction on each of these links, and gradually increase the load, as we did in the experiment for Figure 6.

The experimental results are listed in Table 2. We can see that, R2 and R3 are always labelled as congestion points in all the probings, which means that the probing technique never misses the real congestion point.

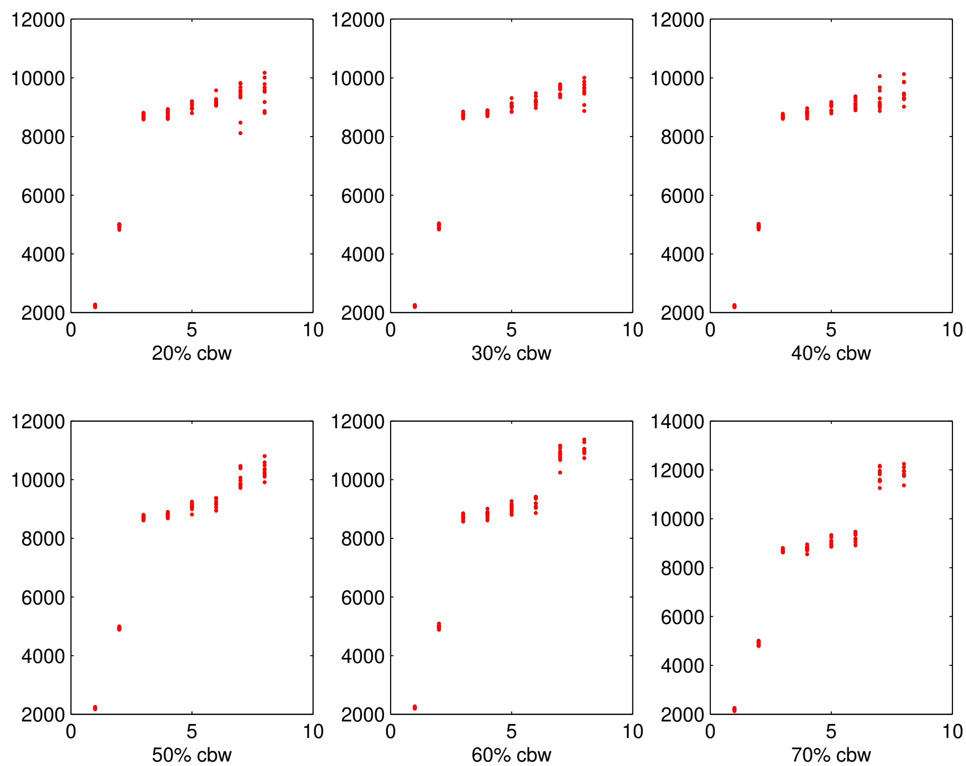


Figure 7: Measurement with background traffic on the link <R6, R7>. In each graph, the x-axis is the hop number, the y-axis is the gap value in micro-seconds.

Table 1: Multiple flows

	$< R4, R8 >$ & $< R6, R7 >$								$< 8, 4 >$ & $< 6, 7 >$							
cbw	r1	r2	r3	r4	r5	r6	r7	r8	r1	r2	r3	r4	r5	r6	r7	r8
10%	0	6	6	0	1	2	3	0	0	6	6	0	3	1	2	0
20%	0	6	6	0	0	0	6	0	0	6	6	0	2	1	3	0
30%	0	6	6	0	1	1	4	0	0	6	6	0	5	1	0	0
40%	0	6	6	0	0	0	6	0	0	6	6	0	1	1	4	0
50%	0	6	6	0	0	0	6	0	0	6	6	0	1	2	3	0
60%	0	6	6	0	0	0	6	0	0	6	6	0	1	0	5	0
70%	0	6	6	0	0	0	6	0	0	6	6	0	0	1	5	0
80%	0	6	6	0	0	0	6	0	0	6	6	0	0	0	6	0
90%	0	6	6	0	0	0	6	0	0	6	6	0	0	0	6	0

5 Internet Validation

This section evaluates the performance of RPT on Internet paths. For a complete evaluation on the Internet, we need to know the real available bandwidth on all the links of a network path. But that information is hard to obtain. The Abilene backbone [1] can partly solve this problem,

Table 2: Return path queueing on link $< R3, R2 >$ and $< R7, R6 >$

	$< R3, R2 >$								$< R7, R6 >$							
cbw	r1	r2	r3	r4	r5	r6	r7	r8	r1	r2	r3	r4	r5	r6	r7	r8
10%	0	6	6	0	4	1	0	0	0	6	6	0	3	2	0	0
20%	0	6	6	0	3	3	0	0	0	6	6	0	4	0	2	0
30%	0	6	6	0	3	0	3	0	0	6	6	0	5	1	0	0
40%	0	6	6	0	2	3	1	0	0	6	6	0	5	1	0	0
50%	0	6	6	0	1	3	1	0	0	6	6	0	5	1	0	0
60%	0	6	6	0	3	2	1	0	0	6	6	0	4	0	2	0
70%	0	6	6	0	3	0	3	0	0	6	6	0	6	0	0	0
80%	0	6	6	0	4	1	1	0	0	6	6	0	3	2	1	0
90%	0	6	6	0	2	2	2	0	0	6	6	0	5	1	0	0

since it publishes its backbone topology and the traffic load (5-minute SNMP statistics) [2]. For this reason, we did all our Internet experiments over Abilene paths. Because the tool needs root permission to send raw packets, we only did the experiment from two sources: a CMU machine and an Emulab host.

The experiment is carried out as the follows. Based on Abilene's backbone topology, from each probing source node, we choose 22 probing destinations. For each of the 11 major routers on the Abilene backbone, we make sure that it is used by at least one probing path. From either probing source, we keep probing all the destinations, with 2 seconds sleeping time in between. For each destination, we obtained 33 sets of measurements from the CMU source, and over 200 sets of measurements from the Emulab source.

Table 3 and Table 4 list the experimental results. For each probing destination, we list the total number of probing results that we collected (excluding the first 4 that are used as the stage point history), and the top 3 routers that are labelled as the congestion points. For each congestion point, we present the number of times it is labelled, and the corresponding percentage over all the probings, which is referred to as the *detection rate*.

The primary congestion point for CMU sourced paths is abilene-psc.abilene.ucaid.edu, corresponding to a link within Pittsburgh Supercomputing Center. The top congestion point for the Emulab sourced paths is 205.124.237.10. It is a router in the Utah Educational Network. In both cases, the detection rate is over 90%, which shows the stability of RPT.

The second congestion point in each path is also valuable, as indicated by the detection rate, which is often over 50%. We believe this is the second bottleneck on the path. For example, the path from both sources to www.ogig.net identified pos-6-3.core0.eug.oregon-gigapop.net as the second congestion point, both with a 70% detection rate. Using the MRTG data from Abilene (refer to Figure 8), we find that that router corresponds to an OC-3 link, which indeed has the smallest capacity for the partial path starting from PSC.

6 Conclusion and Future Work

In this paper, we proposed the idea of RPT, a tool that can be used to detect the congestion position of a network path. It has attractive properties such as single-end control and low overhead. We also report the design and evaluation of a preliminary algorithm that analyzes the probing results from RPT and identifies the congestion point of a network path. We use both Emulab testbed and Internet experiments to evaluate this algorithm. The preliminary results show that RPT is effective in detecting the position of network congestion.

We are currently working on improving the following aspects of the RPT technique presented in this paper:

1. *Algorithm improvement*. We need to improve the congestion point detection algorithm discussed in Section 3.2, to make it more complete.

2. *More realistic testbed evaluation*. In this paper, we studied several aspects of the probing technique, trying to understand its basic properties. But the traffic loads that we used are not always realistic. Experiments with more realistic background traffic are needed.

3. *More solid and extensive Internet evaluation*. We only study the performance of the tool on Abilene Internet paths. We need to use more diverse Internet paths to evaluate this tool. A challenge in Internet experiment is to know the real traffic load on *all* relevant links.

4. *Understanding what this tool really measures*. We discussed the notion of "congestion point" in this paper. But in some cases, this may not be the link with the lowest available bandwidth. We need to have a more complete understanding on what factors affect our measurement and how to distinguish them.

5. *The impact of the probing packet size and the number of packet packets*. In this paper, the probing packet size and the number of load packets in the train were empirically selected. We need to know how the measurement change with different configurations, and eventually design a mechanism to automatically configure the probing packet train.

References

[1] Abilene. http://abilene.internet2.edu/.

[2] Abilene network monitoring. http://www.abilene.iu.edu/noc.html.

[3] Emulab. http://www.emulab.net.

[4] Iperf. http://dast.nlanr.net/Projects/Iperf/.

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

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

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

[8] 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*, 21(6), August 2003.

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

[10] Manish Jain and Constantinos Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. In *Passive and Active Measurements*, Fort Collins CO, March 2002.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

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

[17] 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.

APPENDIX

A The MRTG for the Abilene Path

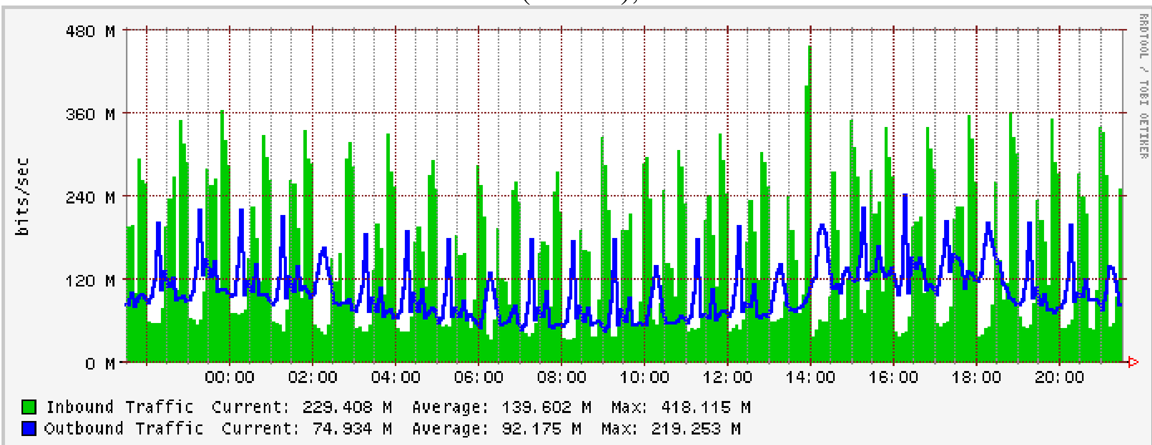
Table 3: Probing results over Abilene paths using a CMU host as the source

www.anl.gov 33			www.ogig.net 33		
30	0.91	abilene-psc.abilene.ucaid.edu	25	0.76	abilene-psc.abilene.ucaid.edu
25	0.76	bar-cmu-ge-4-0-0-1.psc.net	23	0.70	pos-6-3.core0.eug.oregon-gigapop.net
5	0.15	chinng-nycmng.abilene.ucaid.edu	16	0.48	bar-cmu-ge-4-0-0-1.psc.net
www.apan.net 34			www.onenet.net 33		
33	0.97	abilene-psc.abilene.ucaid.edu	31	0.94	abilene-psc.abilene.ucaid.edu
20	0.59	bar-cmu-ge-4-0-0-1.psc.net	19	0.58	bar-cmu-ge-4-0-0-1.psc.net
5	0.15	losang-hstnng.abilene.ucaid.edu	9	0.27	164.58.10.209
www.arc.nasa.gov 33			www.pnw-gigapop.net 33		
27	0.82	abilene-psc.abilene.ucaid.edu	24	0.73	abilene-psc.abilene.ucaid.edu
16	0.48	bar-cmu-ge-4-0-0-1.psc.net	10	0.30	nycmng-washng.abilene.ucaid.edu
9	0.27	chinng-nycmng.abilene.ucaid.edu	10	0.30	HYPER-VL502.GW.CMU.NET
www.arizona.edu 33			www.rutgers.edu 34		
19	0.58	abilene-psc.abilene.ucaid.edu	26	0.76	abilene-psc.abilene.ucaid.edu
16	0.48	bar-cmu-ge-4-0-0-1.psc.net	17	0.50	bar-cmu-ge-4-0-0-1.psc.net
10	0.30	HYPER-VL502.GW.CMU.NET	11	0.32	nycmng-washng.abilene.ucaid.edu
www.calren2.net 34			www.sox.net 34		
23	0.68	abilene-psc.abilene.ucaid.edu	31	0.91	abilene-psc.abilene.ucaid.edu
20	0.59	bar-cmu-ge-4-0-0-1.psc.net	25	0.74	bar-cmu-ge-4-0-0-1.psc.net
10	0.29	beast-bar-g4-0-1.psc.net	5	0.15	gw2-sox.sox.gatech.edu
www.hawaii.edu 33			www.tamu.edu 34		
24	0.73	abilene-psc.abilene.ucaid.edu	31	0.91	abilene-psc.abilene.ucaid.edu
17	0.52	bar-cmu-ge-4-0-0-1.psc.net	22	0.65	bar-cmu-ge-4-0-0-1.psc.net
12	0.36	205.166.205.218	7	0.21	atla-washng.abilene.ucaid.edu
www.iastate.edu 33			www.ttu.edu 34		
26	0.79	abilene-psc.abilene.ucaid.edu	32	0.94	abilene-psc.abilene.ucaid.edu
13	0.39	HYPER-VL502.GW.CMU.NET	23	0.68	bar-cmu-ge-4-0-0-1.psc.net
10	0.30	nycmng-washng.abilene.ucaid.edu	5	0.15	hstnng-atlang.abilene.ucaid.edu
www.louisville.edu 33			www.udel.edu 34		
29	0.88	abilene-psc.abilene.ucaid.edu	31	0.91	abilene-psc.abilene.ucaid.edu
24	0.73	bar-cmu-ge-4-0-0-1.psc.net	26	0.76	bar-cmu-ge-4-0-0-1.psc.net
11	0.33	lou-belknap-9-0-0-p.kec.net	7	0.21	chp-br4-p-0-0-0.nss.udel.edu
www.magpi.net 34			www.usf.edu 34		
32	0.94	abilene-psc.abilene.ucaid.edu	32	0.94	abilene-psc.abilene.ucaid.edu
29	0.85	bar-cmu-ge-4-0-0-1.psc.net	28	0.82	bar-cmu-ge-4-0-0-1.psc.net
6	0.18	local1.abilene.magpi.net	8	0.24	atla-washng.abilene.ucaid.edu
www.npt.nren.nasa.gov 33			www.wisc.edu 33		
25	0.76	abilene-psc.abilene.ucaid.edu	24	0.73	abilene-psc.abilene.ucaid.edu
21	0.64	bar-cmu-ge-4-0-0-1.psc.net	21	0.64	bar-cmu-ge-4-0-0-1.psc.net
8	0.24	dnvrng-kscyng.abilene.ucaid.edu	16	0.48	r-peer-WNMadison-gw.net.wisc.edu
www.oar.net 33			www.wpi.edu 34		
29	0.88	abilene-psc.abilene.ucaid.edu	27	0.79	abilene-psc.abilene.ucaid.edu
24	0.73	bar-cmu-ge-4-0-0-1.psc.net	26	0.76	bar-cmu-ge-4-0-0-1.psc.net
12	0.36	chinng-nycmng.abilene.ucaid.edu	6	0.18	nycmng-washng.abilene.ucaid.edu

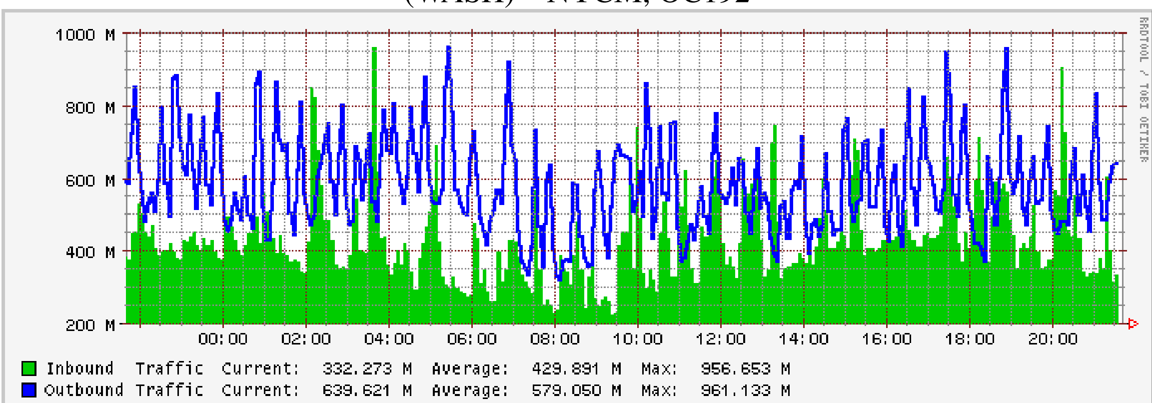
Table 4: Probing results over Abilene paths using an Emulab host as the source

www.anl.gov 207			www.ogig.net 207		
192	0.93	205.124.237.10	182	0.88	205.124.237.10
115	0.56	wr1ebc-crebc.net.utah.edu	145	0.70	pos-6-3.core0.eug.oregon-gigapop.net
101	0.49	anl-mren-gige.anchor.anl.gov	113	0.55	205.124.249.122
www.apan.net 208			www.onenet.net 206		
201	0.97	205.124.237.10	201	0.98	205.124.237.10
153	0.74	155.99.132.109	138	0.67	155.99.132.109
80	0.38	205.124.249.122	123	0.60	205.124.249.122
www.arc.nasa.gov 207			www.pnw-gigapop.net 207		
190	0.92	205.124.237.10	201	0.97	205.124.237.10
138	0.67	155.99.132.109	140	0.68	155.99.132.105
88	0.43	205.124.249.122	81	0.39	205.124.249.122
www.calren2.net 208			www.rutgers.edu 208		
199	0.96	205.124.237.10	198	0.95	205.124.237.10
110	0.53	wr1ebc-crpark.net.utah.edu	120	0.58	POS5-0-0-rutgers-gw.Rutgers.EDU
101	0.49	WestEdCAT6009POS-WestEdGSRPOS.CSU.net	116	0.56	wr1ebc-crebc.net.utah.edu
www.cmu.edu 207			www.sox.net 208		
197	0.95	205.124.237.10	201	0.97	205.124.237.10
132	0.64	wr1ebc-crpark.net.utah.edu	105	0.50	155.99.132.105
108	0.52	cmu-i2.psc.net	76	0.37	205.124.249.122
www.hawaii.edu 207			www.tamu.edu 208		
195	0.94	205.124.237.10	202	0.97	205.124.237.10
120	0.58	155.99.132.105	134	0.64	155.99.132.109
69	0.33	205.124.249.122	34	0.16	TAMU.GIGAPOP.GEN.TX.US
www.iastate.edu 207			www.ttu.edu 208		
198	0.96	205.124.237.10	198	0.95	205.124.237.10
75	0.36	155.99.132.109	128	0.62	155.99.132.105
43	0.21	wr1ebc-crebc.net.utah.edu	63	0.30	205.124.249.122
www.louisville.edu 207			www.udel.edu 208		
197	0.95	205.124.237.10	194	0.93	205.124.237.10
123	0.59	155.99.132.109	98	0.47	wr1ebc-crebc.net.utah.edu
22	0.11	205.124.249.122	60	0.29	chp-br4-p-0-0-0.nss.udel.edu
www.magpi.net 208			www.usf.edu 208		
191	0.92	205.124.237.10	204	0.98	205.124.237.10
117	0.56	wr1ebc-crpark.net.utah.edu	139	0.67	155.99.132.109
51	0.25	phl-01-02.backbone.magpi.net	58	0.28	205.124.249.122
www.npt.nren.nasa.gov 207			www.wisc.edu 207		
202	0.98	205.124.237.10	199	0.96	205.124.237.10
128	0.62	155.99.132.109	104	0.50	155.99.132.109
67	0.32	192.12.123.201	82	0.40	r-peer-WNMadison-gw.net.wisc.edu
www.oar.net 207			www.wpi.edu 208		
197	0.95	205.124.237.10	198	0.95	205.124.237.10
132	0.64	155.99.132.105	122	0.59	155.99.132.105
49	0.24	krc1-atm1-0-0s4.columbus.oar.net	119	0.57	goddard-wpi.goddard.gigapop.net

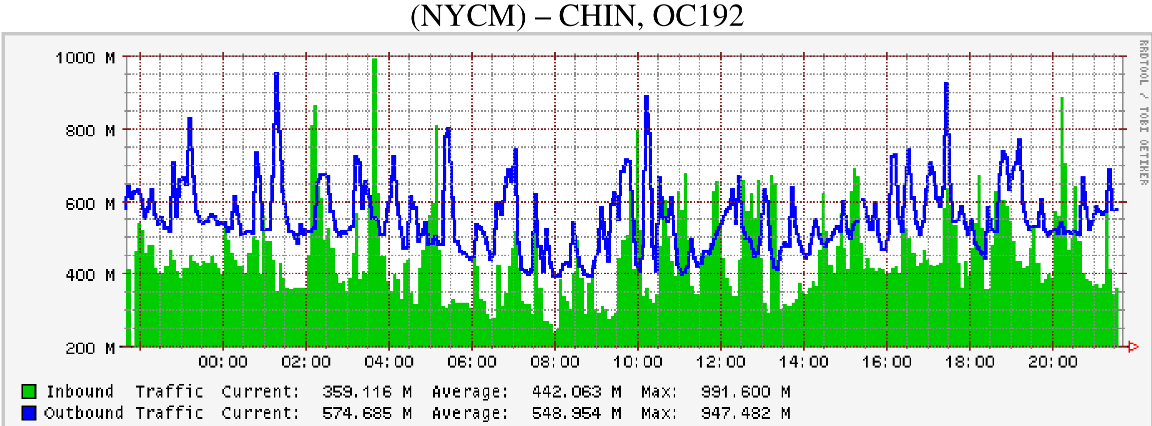
PSC – (WASH), OC48



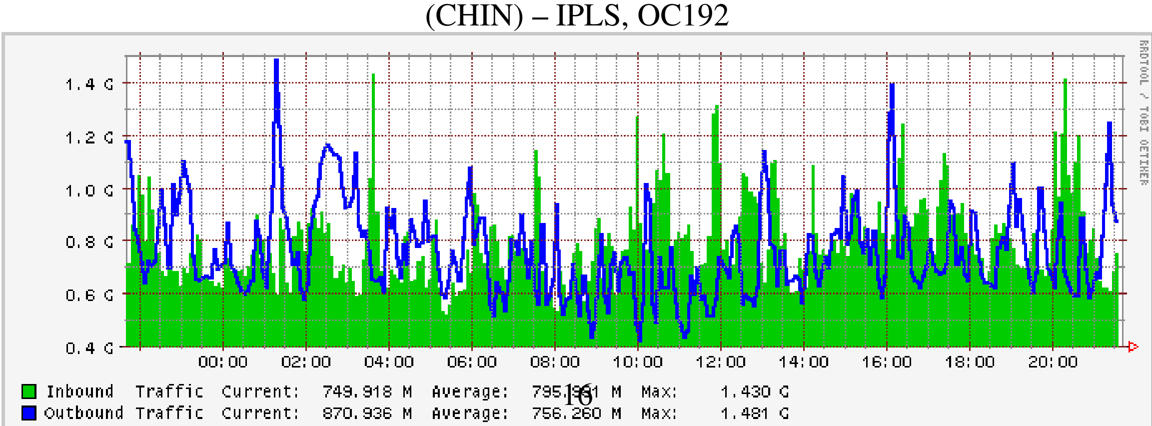
(WASH) – NYCM, OC192



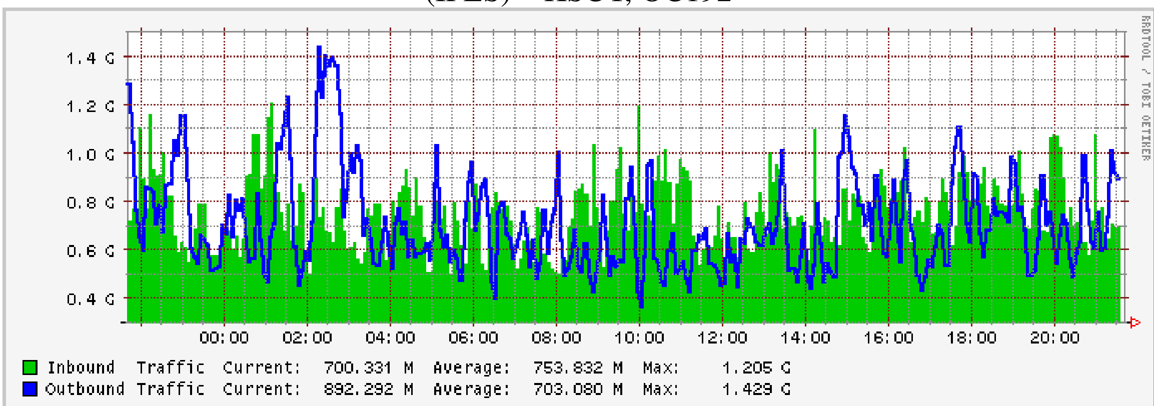
(NYCM) – CHIN, OC192



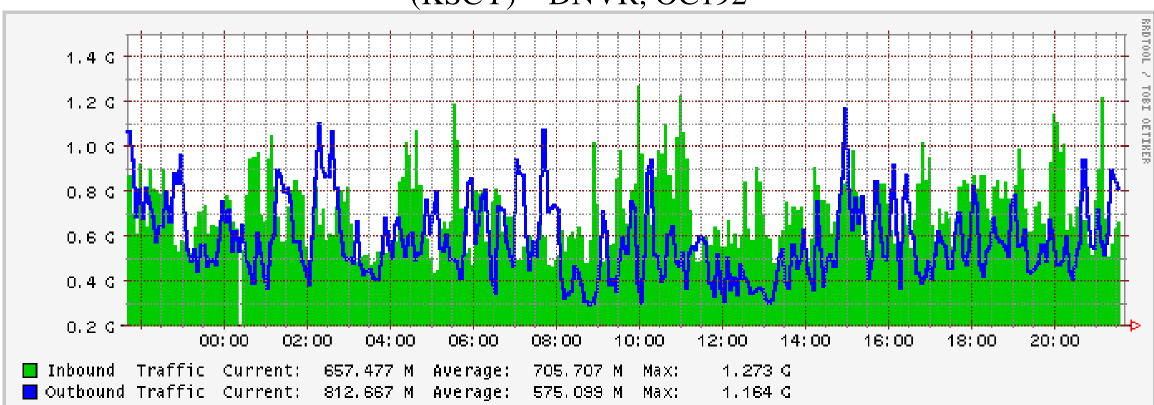
(CHIN) – IPLS, OC192



(IPLS) – KSCY, OC192



(KSCY) – DNVR, OC192



(DNVR) – OGIG, OC3

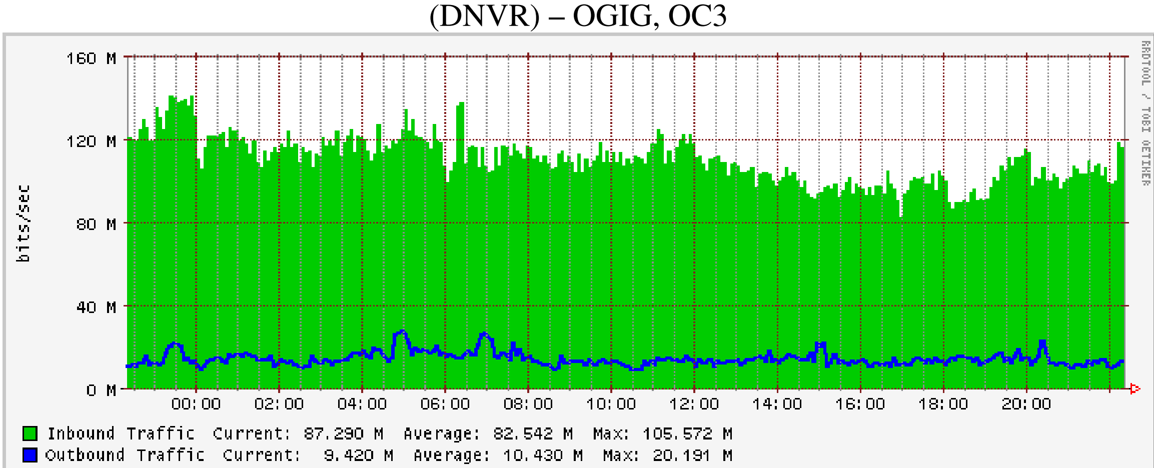


Figure 8: The MRTG traffic statistics for the Abilene routers on the path CMU $\rightarrow$ www.ogig.net.