Detecting and Measuring Asymmetric Links in an IP Network

Wenyu Jiang
wenyu@cs.columbia.edu
Columbia University, Computer Science Dept.

Technical Report: CUCS-009-99

The rapid growth of the Web has caused a lot of congestion in the Internet. Pinpointing bottlenecks in a network is very helpful in congestion control and performance tuning. Measuring link bandwidths can help identifying such bottlenecks. Existing tools for bandwidth measurement like Pathchar, Bing and Bprobe assume symmetric links. Hence their results will be incomplete or incorrect in the presence of asymmetric links. It becomes important to consider asymmetric links, as they are gaining popularity in recent years. Examples are ADSL lines, Cable modems, Satellite links, and 56K modems. In this paper, we present an algorithm that can measure each hop's link bandwidth in both directions in an IP network. Therefore, it is trivial to detect asymmetry of a link. We performed several experiments to validate our algorithm. We also discuss some factors that can adversely affect the precision and/or correctness of bandwidth measurement, and suggest possible solutions.

Additional Key Words and Phrases: Network measurement, bandwidth measurement, asymmetric link

1. INTRODUCTION

With the exponential growth of the Web, the Internet has seen far more congestion than in the past. On one hand, protocols such as TCP has been enhanced to adapt to and avoid congestion. On the other hand, it is valuable to identify bottlenecks in a network.

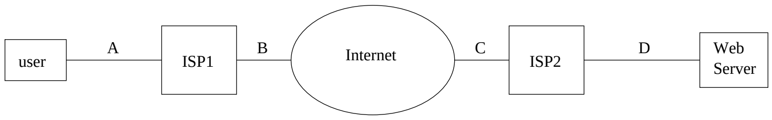

Fig. 1. Slow connection to a Web server

For example, when an Internet user perceives a slow connection to a Web server, as shown in Figure 1, it is usually due to queuing congestion at some locations, or slow speed of link A, B, C, D, or some links in the Internet.

Link speed can be further divided into raw link bandwidth (its full capacity) and available link bandwidth (what's available after sharing). In this paper we focus on measuring raw link bandwidth.

Various bandwidth estimation tools have been developed. Pathchar [Jacobson 1997] and Bing [Beyssac 1995] measure *per hop* link bandwidth by analyzing a packet's Round Trip Time (RTT) linearity with respect to its size, at a given hop. Bprobe [Carter and Crovella 1996] and Tcpanaly [Paxon 1997] measures *bottleneck* link bandwidth based on packet-pair technique.

These tools assume that the links are symmetric along the network path. While this was true for most of the Internet a few years ago, recent deployment of ADSL lines, Cable modems, Satellite links and 56K modems [1] is rapidly changing this assumption. ADSL, Cable modems, and 56K modems are mostly changing end user connectivity, while Satellite links are particularly useful in building transoceanic backbones.

Being able to detect and measure asymmetrical links is a very desirable capability. Imagine a university uses an ADSL line for their Internet connection. At first the students may feel satisfied about their download speed, but as some of them try to create their own Web service on it, outside users will likely experience slow access to these Web pages. If the students or even the outsiders can measure their link speeds, they will discover the asymmetry in their Internet connection. Then they can demand the university/corporation to upgrade its Internet service.

As an another example, satellite links are normally unidrectional, with a large download link and a small or zero upload link. Since satellite links are often used in building worldwide backbones, such asymmetric property must be considered when deploying the network service.

Since the above tools embeds no knowledge of asymmetry in a network, their output will contain only 1 number (instead of 2) for a particular link. Consequently their results will be incomplete or incorrect [2].

In the next section, we present an algorithm that measures *per hop* link bandwidth in both directions. Section 3 shows experiments we performed on asymmetric links in the real word. Section 4 discusses some factors that may alter the result. Before conclusion, we compared the above tools in Related Works.

2. THE ALGORITHM

Our algorithm is also based on RTT linearity analysis, like Pathchar and Bing. Mathematically, it is a combination of the techniques used by Pathchar and the one used by Bing. We start with an algorithm that computes per hop link bandwidth assuming link symmetry. It is essentially the same method used by Bing, but we were not aware of Bing when we developed it. However, we still describe it as a basis for our final algorithm. Then, we extend it to consider asymmetric links.

2.1 Algorithm 1: Symmetric case

The most common way of measuring RTT is by sending an ICMP echo [Postel 1981b] packet and wait for a reply packet, as in the *ping* program. That's also what we will use in our first algorithm.

Let T_i denote the RTT for an ICMP echo/reply packet of B bytes, v_i denote the speed of link between hop $i-1$ and hop i, where i is the number of hops between sender (hop 0) and receiver (hop i).

Then for $i = 1$, as shown in Figure 2, $T_1 = \frac{2B}{v_1} + const$

[1] 56K modem downloads at about 56Kbit/s, and uploads at 33.6Kbit/s

[2] Pathchar and Tcpanaly are, to some degree, exceptions. We will compare them in Section 5

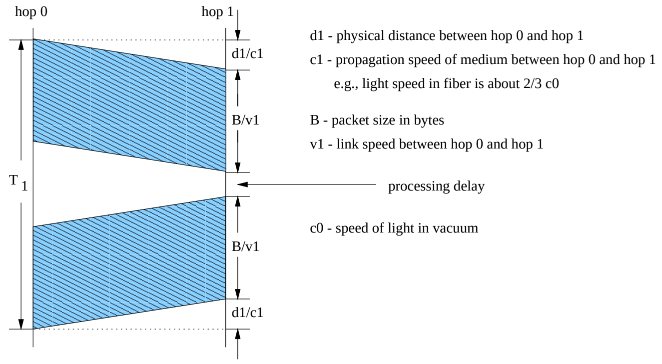

Fig. 2. 1 Hop, symmetric case

where *const* is a constant that may vary for different i. It is the sum of propagation delay, processing delay, and queuing delay. Here for simplicity, we assume the queuing delay is 0 [3]. The processing delay is a constant because it takes almost the same number of machine instructions (independent of B) to inspect any ICMP echo packet and modify it to generate a reply packet. Interrupt service time is relatively fixed. Further, there is generally no copying overhead from kernel memory [4] to Network Interface Card, since DMA is now almost ubiquitous.

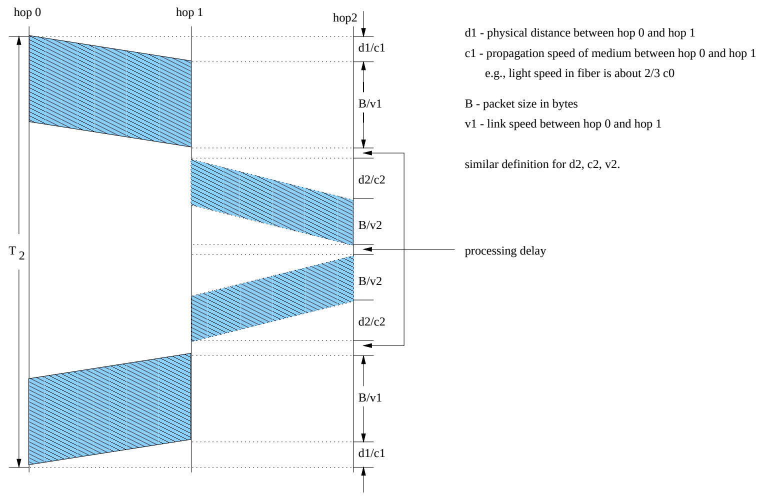

Fig. 3. 2 Hop, symmetric case

So, for $i = 2$, as in Figure 3, $T_2 = \frac{2B}{v_1} + \frac{2B}{v_2} + const$

[3]same if a constant queuing delay is assumed; while there is a lot of statistical techniques to filter out queuing delay in the real world, it is not the focus of this paper

[4]however, the copying overhead from user memory to kernel memory is proportional to B. Any disturbance that depends on B will affect the result. To avoid this problem, tcpdump can be used, which captures kernel time instead of application time.

In general, we get: $$T_i = \sum_{j=1}^{i} \frac{2B}{v_j} + const = B \sum_{j=1}^{i} \frac{2}{v_j} + const \qquad (1)$$

Let k_i denote the slope of linear function $T_i(B)$, then $$k_i = \sum_{j=1}^{i} \frac{2}{v_j} \quad (2)$$

To obtain k_i, we can send many packets of different sizes, which creates many sample points for function $T_i(B)$. Then, a linear regression is all that's needed to obtain k_i from the sample points.

So $$k_i - k_{i-1} = (\sum_{j=1}^{i} \frac{2}{v_j}) - (\sum_{j=1}^{i-1} \frac{2}{v_j}) = \frac{2}{v_i}$$

$$\Longrightarrow v_i = \frac{2}{k_i - k_{i-1}} \qquad (3)$$

Therefore, Formula (3) is our Algorithm 1, which computes symmetric link bandwidths.

2.2 How asymmetric links affect Formula (3)

Formula (3) does not deal with asymmetric links. In order to derive the algorithm for asymmetric links, let us first look at the Figure 4 and 5, and see what T_i becomes.

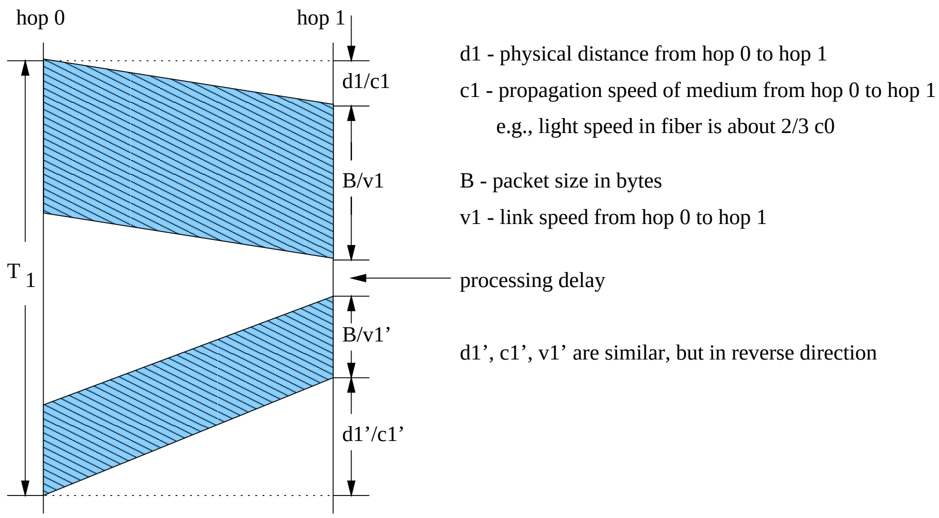

Fig. 4. 1 Hop, asymmetric case

When $i = 1$, $T_1 = \frac{B}{v_1} + \frac{B}{v_1'} + const$

When $i = 2$, $T_2 = \frac{B}{v_1} + \frac{B}{v_1'} + \frac{B}{v_2} + \frac{B}{v_2'} + const$

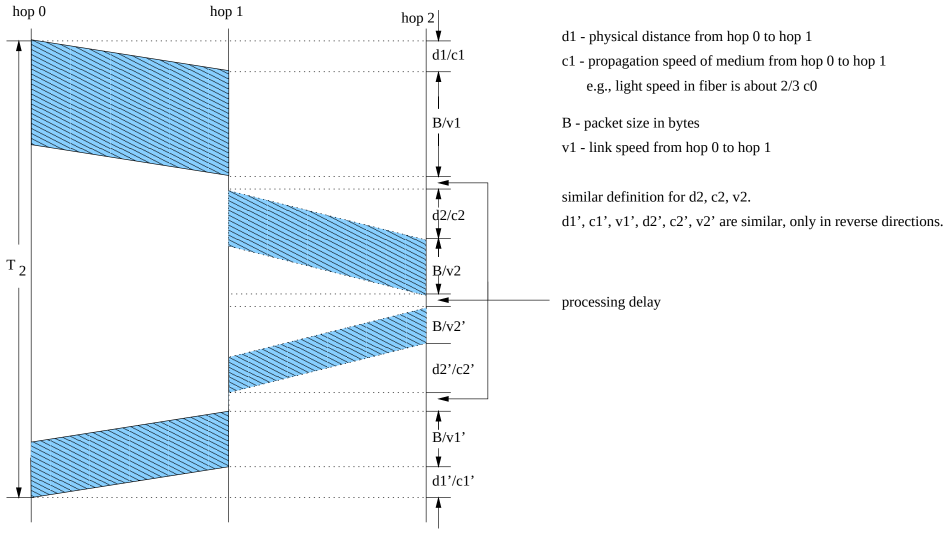

Fig. 5. 2 Hop, asymmetric case

In general, we have $$T_i = \sum_{j=1}^{i}(\frac{B}{v_j} + \frac{B}{v'_j}) + const = B\sum_{j=1}^{i}(\frac{1}{v_j} + \frac{1}{v'_j}) + const \quad (4)$$

Therefore, $$k_i = \sum_{j=1}^{i}(\frac{1}{v_j} + \frac{1}{v'_j}) \qquad (5)$$

$$\Longrightarrow k_i - k_{i-1} = (\sum_{j=1}^{i}(\frac{1}{v_j} + \frac{1}{v'_j})) - (\sum_{j=1}^{i-1}(\frac{1}{v_j} + \frac{1}{v'_j})) = \frac{1}{v_i} + \frac{1}{v'_i}$$

If we still use Formula (3) to calculate v_i, we would get

$$v_i = \frac{2}{k_i - k_{i-1}} = \frac{2}{\frac{1}{v_i} + \frac{1}{v'_i}} \qquad (6)$$

That is, the v_i we obtained is the *harmonic average* of the real v_i and v'_i.

2.3 Algorithm 2 - The final algorithm

Although Formula (6) does not give us what we want, if we know either v_i or v'_i, the other value becomes easily solvable. Formula (6) $\neq v_i$ because a packet's RTT is the sum of transit time in both directions, which depends on link speed in both directions and packet size B, as shown in the left of Figure 6. If we can force the returning packet to have a constant size D, in the right of Figure 6, then the returning transit time no longer depends on B but only v'_i, for any i. Since v'_i are constant for any i (link speed do not change on the fly), the returning transit time is also a constant. This means v_i is directly solvable.

In an IP network, the most obvious case for a constant-sized returning packet is

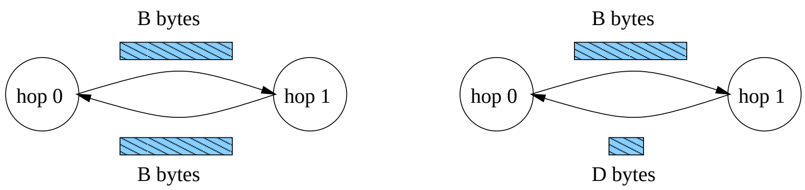

Fig. 6. 1 hop, left: B bytes to/from; right: B bytes to, D bytes from

the ICMP time-exceeded message [5] [Postel 1981b]. Therefore, our 2nd and final algorithm is:

(1) send an ICMP echo packet to the ith router with a TTL of i, this causes a normal reply packet from the ith router;

(2) *also* send an ICMP echo packet to $i+1$th router with a TTL of i, then at ith router, the packet will be bounced back as an ICMP time-exceeded error packet.
When the ith hop is not really a router, but the end host, there would be no $i+1$th router. So we have to send a UDP packet with a bad destination port number, forcing an ICMP port unreachable message [Postel 1981b], which also has a constant size. This was first used by *traceroute*, and also in Pathchar.

Let T'_i denote the RTT for the ICMP echo/time exceeded packet.
Then, Figure 7 gives $T_1 = \frac{B}{v_1} + \frac{D}{v'_1} + const = \frac{B}{v_1} + const.$

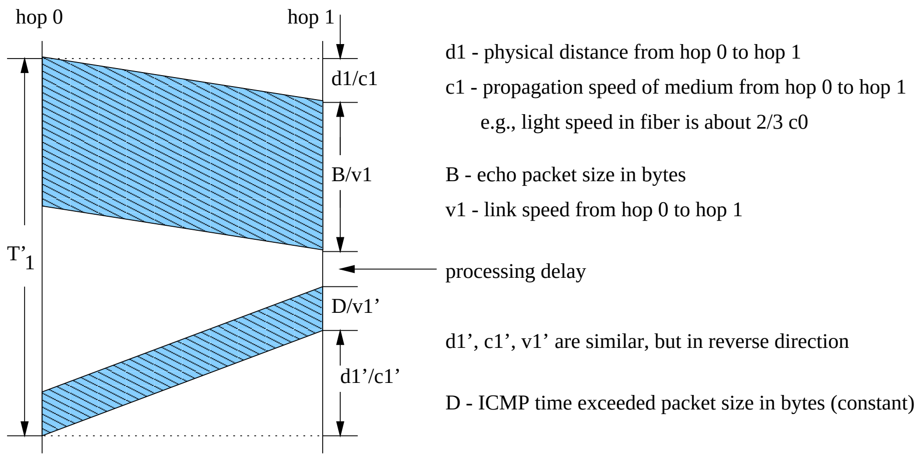

Fig. 7. 1 Hop, asymmetric case, with time exceeded packet

From Figure 8, we have $T_2 = \frac{B}{v_1} + \frac{D}{v'_1} + \frac{B}{v_2} + \frac{D}{v'_2} + const = \frac{B}{v_1} + \frac{B}{v_2} + const.$

[5] RFC 792 defines that an ICMP time-exceeded message to contain the original packet's IP header/options and ≥ 8 bytes of data. But in practice, nearly all implementations use exactly 8 bytes of data. By avoiding the use of IP header options, we easily get a constant-sized packet.

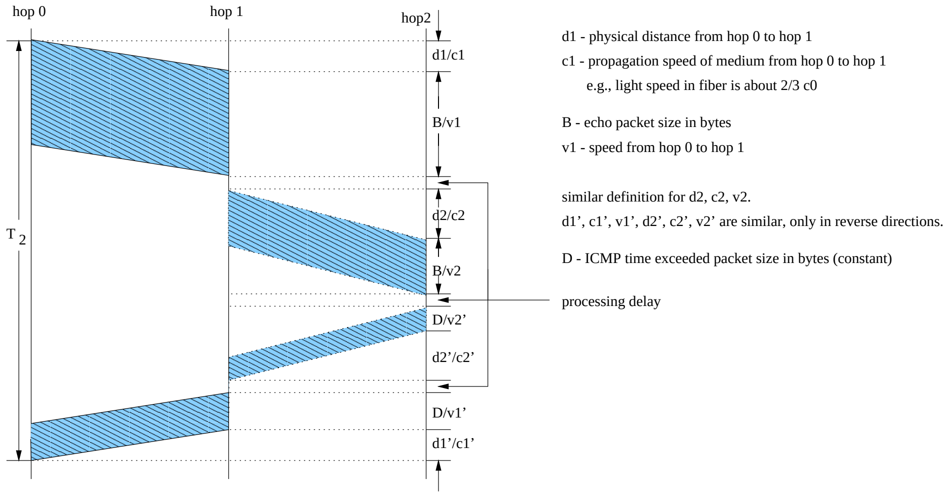

Fig. 8. 2 Hop, asymmetric case, with time exceeded packet

In general, $$T_i' = \sum_{j=1}^{i}\left(\frac{B}{v_j} + \frac{D}{v_j'}\right) + const = B\sum_{j=1}^{i}\frac{1}{v_j} + const \qquad (7)$$

As we predicted, T_i' is not dependent upon link speed in the returning direction.

$$k_i' = \sum_{j=1}^{i}\frac{1}{v_j} \Longrightarrow k_i' - k_{i-1}' = B\sum_{j=1}^{i}\frac{1}{v_j} - B\sum_{j=1}^{i-1}\frac{1}{v_j} = \frac{1}{v_j}$$

$$\Longrightarrow v_j = \frac{1}{k_i' - k_{i-1}'} \qquad (8)$$

Similarly, we have $k_i - k_{i-1} = \frac{1}{v_j} + \frac{1}{v_j'} = k_i' - k_{i-1}' + \frac{1}{v_j'}$

$$\Longrightarrow v_j' = \frac{1}{k_i - k_{i-1} - (k_i' - k_{i-1}')} \qquad (9)$$

Therefore, Formula (8) and (9) compute the speed of ith link in both directions.

2.4 How to use Algorithm 2 in measurement

To actually measure v_i and v_i', we need to send packets of different sizes (B), and measure their RTTs T_i, T_i'. To eliminate queuing delay noise, we chose the simplest approach, i.e., by sending many packets of the same size, and takes its min. Then, we perform a linear regression for $T_i(B)$ and $T_i'(B)$, which gives us v_i and v_i'.

There are many ways to eliminate queuing delay noise. Pathchar uses *ensemble min* [Jacobson 1997]. Although not described in detail, it may mean taking all the values that are close to the min. We have found that in some cases, such as ethernet, which is quite stable, *ensemble min* shows better results, but other times, like in satellite links, where delay variation is low in percentage but very high in absolute value, the plain min seems to give better precision. However, a detailed analysis and critique of these different statistical methods are beyond the scope of this paper.

3. REAL WORLD EXPERIMENTS

Now, we validate Algorithm 2 by real world tests. We performed tests on 4 kinds of links. First we show a simple network topology if necessary. Then we present the $k_i, k_{i-1}, k'_i, k'_{i-1}$ obtained from linear regression, and finally the result v_i, v'_i. We also compare them with actual values, if available. $Longrightarrow$ means outgoing, $Longleftarrow$ means incoming. These directions are from the test initiator's perspective. All the k's have the unit msec/byte.

3.1 Dialup modem, possibly mixed with Ethernet

The first test was done in our lab using a Windows 95 laptop computer with 2 network interfaces: a 14.4kb/s dialup modem, and a 10Mb/s ethernet card. The routing tables on both end hosts (the laptop and the test initiator) were properly modified to control the routes. We used PPP as the modem's link layer protocol.

As shown in Figure 9, 10, 11, machine A, B, C are the test initiator, laptop, and dialup server, respectively. All 3 machines are connected to the same ethernet.

Although the modem is 14.4kb/s, the Dialup Networking control can only specify 9.6kb/s and 19.2kb/s. This results in a download speed of 19.2kb/s and an upload link of 14.4kb/s, which is both interesting and useful to our experiment.

(1) Outgoing: PPP Dialup, Incoming: Ethernet

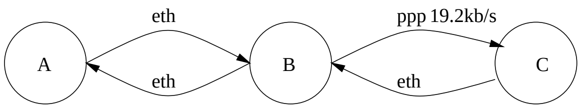

Fig. 9. Asymmetric link with dialup outgoing and ethernet incoming

In this case, $i = 2$,
$k_1 = 0.004369, k'_1 = 0.002171, k_2 = 0.520233, k'_2 = 0.5160,$
$\Longrightarrow v_2 = \frac{1}{k'_2 - k'_1} = \frac{1}{0.513829} = 1.946$ kB/s (kilo-byte/sec) $\Leftrightarrow$ 19.46 kb/s [6]
$\Longrightarrow v'_2 = \frac{1}{k_2 - k_1 - (k'_2 - k'_1)} = \frac{1}{0.515864 - 0.513829} = \frac{1}{0.002035} = 491.4kB/s = 3.93$ Mb/s

(2) Outgoing: Ethernet, Incoming: PPP Dialup

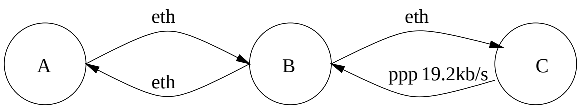

Fig. 10. Asymmetric link with ethernet outgoing and dialup incoming

$k_1 = 0.004347, k'_1 = 0.002193, k_2 = 0.715773, k'_2 = 0.003932,$

[6]in asynchronous transmission, the most typical configuration is 1 start bit, 1 stop bit, no parity, which means 10 signaling bits for each data byte. Therefore 1.946 kB/s is equivalent to 19.46 kb/s

$\Longrightarrow v_2 = \frac{1}{0.001739} = 575$ kB/s $= 4.6$ Mb/s
$\Longrightarrow v_2' = \frac{1}{0.711426-0.001739} = \frac{1}{0.709687} = 1.41$ kB/s $\Leftrightarrow 14.1$ kb/s

(3) Outgoing: PPP Dialup, Incoming: PPP Dialup

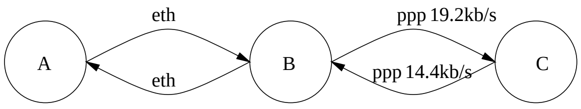

Fig. 11. Asymmetric link with dialup outgoing and incoming

$k_1 = 0.004348, k_1' = 0.002176, k_2 = 1.2274, k_2' = 0.5160,$
$\Longrightarrow v_2 = \frac{1}{0.513824} = 1.946$ kB/s $\Leftrightarrow 19.46$ kb/s
$\Longrightarrow v_2' = \frac{1}{1.223052-0.513824} = \frac{1}{0.709228} = 1.41$ kB/s $\Leftrightarrow 14.1$ kb/s

Note: Our measured ethernet speed is consistently around 4-5 Mb/s instead of 10 Mb/s. It is due to existence of switches in the ethernet. In our lab, in order for B to send a packet to C via ethernet, it needs to pass through a 10Mb/s switch, then onto several 100Mb/s switches, and finally reaching host C's 10Mb/s switch. So the measured speed will be lower than 5Mb/s, as explained in the next section how bridges, switches affect the measurements. Also, with the same signal to noise ratio, a slow PPP link can inject much higher noise in time measurement to an ethernet, that's why we got 2 quite different results: 3.9 Mb/s and 4.6 Mb/s. This is often called noise amplification. (The ideal value should be about 3.67 Mb/s, which is v_1 and v_1', because it is known path (A,B) and path (B,C) should pass through the same switches).

3.2 Cable Modem

Host C below is connected to the Internet via Cable modem service. It has a faster download speed than upload [7].

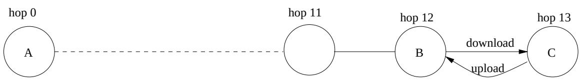

Fig. 12. Asymmetric link with Cable modem

$k_{12} = 0.0083, k_{12}' = 0.00325, k_{13} = 0.0264, k_{13}' = 0.0060,$
$\Longrightarrow v_{13} = \frac{1}{0.00275} = 363.6$ kB/s $= 2.91$ Mb/s
$\Longrightarrow v_{13}' = \frac{1}{0.0181-0.00275} = \frac{1}{0.01535} = 65.15$ kB/s $= 521$ kb/s

Unfortunately we only know the connection is cable modem, but not its actual speed. A good guess may be the typical offerings from Cable providers, which is 3Mb/s download and 512 kb/s upload, both peak speed [8].

[7] Here the download/upload is from host C's perspective

[8] Although, many cable modems are capable of download signaling at 27Mb/s, so again, there could be hidden bridges behind the hop. Actually, one apparent hidden hop is the 10 Mb/s ethernet NIC which is always coupled with the cable modem. Of course, we don't know the real settings, but at least our results prove the algorithm in principle is working.

The author have also made an interesting comparison with Bing. We ran a Bing test (http://web.cnam.fr/cgi-bin/traceroute.html) on the same cable modem host. Since Bing uses essentially our Algorithm 1, according to Formula (6), Bing will measure the *harmonic average* of v_i and v'_i.

Therefore, we predict Bing's result to be $(\frac{1}{521} + \frac{1}{2910}) - 1 = 884$ kb/s for cable modem. The actual Bing test gives a range of about 716 to 992 kb/s for cable modem, and the average is roughly 843kb/s, well in range of expectation.

3.3 ADSL Line

The network path for this case is almost identical to Figure 12. Except $i = 15$.

$k_{14} = 0.00748, k'_{14} = 0.0038, k_{15} = 0.1021, k'_{15} = 0.028,$

$\Longrightarrow v_{15} = \frac{1}{0.0242} = 41.3$ kB/s $= 331$ kb/s

$\Longrightarrow v'_{15} = \frac{1}{0.09462-0.0242} = \frac{1}{0.07042} = 14.2$ kB/s $= 113.6$ kb/s

Again, we don't know for sure its real speed. From the result, it could have 384 kb/s download and 128 kb/s upload. But it is known in this example that there is an ATM network hidden behind it, which will cause a lower estimated result.

We have also made an similar comparison test with Bing on the same ADSL host. our prediction is $(\frac{1}{114} + \frac{1}{331})^{-1} = 170$ kb/s. The actual test range is about 164 to 173 kb/s, with an average of about 168 kb/s, nicely fulfilling our prophecy.

3.4 Satellite Link

The Belgium research network has an 8Mb/s unidirectional Satellite link from USA to Belgium. Unfortunately, we don't know for sure the speed of returning link. It could be a 155Mb/s OC-3 ATM link, or a 22Mb/s partial ATM link. Usually test results for very fast links are un-trustable, since the RTT difference between a min-sized and max-sized packet would be only a few micro-seconds, approaching the limit of most operating system clock resolution. Plus, noises like queuing delay would easily overwhelm the RTT portion attributable to link speed.

The network path is similar to Figure 12, except $i = 12$.

$k_{11} = 0.005919, k'_{11} = 0.002582, k_{12} = 0.007833, k'_{12} = 0.00444,$

$\Longrightarrow v_{12} = \frac{1}{0.001858} = 538.2$ kB/s $= 4.3$ Mb/s

$\Longrightarrow v'_{12} = \frac{1}{0.001914-0.001858} = \frac{1}{0.000056} = 178857$ kB/s $= 143$ Mb/s

The results show the satellite link to be 4.3 Mb/s, which is only half of 8Mb/s. Again there must be a hidden bridge of some sort. If we assume a 10 Mb/s ethernet is in between, since routers are more likely to have an ethernet interface as opposed to a satellite one, our predicted result as shown in next section, would be 1/(1/8 + 1/10) = 4.44 Mb/s, which is very close to 4.3 Mb/s. Of course, it is only a conjecture.

The result of 143Mb/s may indicate a 155Mb/s ATM link, but since there are enough noise to disturb the measurement, the only conclusion we can make is it is a fast link.

4. ISSUES THAT AFFECT LINK SPEED ESTIMATION

There are many factors that can interfere with our algorithm, many of which were mentioned in Pathchar. Here we classify them into Random distortions, Deterministic distortions, and Potential fatal problems.

(1) Random distortions

—Noise
The noise issue was also mentioned in Pathchar. In particular for high speed links above 100Mb/s, even a small noise of 100 μs can be devastating. The noise can come from an imprecise system clock, or queuing delay. The former can be remedied by better testing equipment, but the latter is much harder. Queuing delay can easily exceed 100ms or even 1sec. Coupled with large variation in queuing delay, it makes measurements very difficult.
In addition, as we have seen in Section 3.1, it is difficult to precisely measure a fast link (say ethernet) behind a slow link (say PPP), due to noise amplification.

(2) Deterministic distortions

—Hidden hops
Examples are ethernet bridges, switches, VSAT hubs [Tanenbaum 1996], ATM network underneath IP, etc. They increase RTTs by an amount proportionally to packet size B. Therefore it makes the result smaller than the real value. Let's say there are m hops inside an IP hop i, and the link speed is v_{ij} for $j = 1, ..., m$. Then, our algorithm will compute v_i as:

$$v_i = (\sum_{j=1}^{m} \frac{1}{v_{ij}})^{-1} \qquad (10)$$

So, it is easy to see that an extra ethernet bridge, switch or VSAT of the same speed will cut the result by half.

—Signaling speed vs. Data speed
A 155Mb/s ATM link refers to its signaling speed at the optical/electrical level, however, due to ATM cell's small size (48 bytes data, 53 bytes cell), the actual data speed is about 48/53 of 155Mb/s. Our Algorithm will only measure data speed.
One way to overcome this problem is mentioned in Bing [Beyssac 1995]: send packets at small incremental sizes, to detect the small jump in RTT when an extra cell is used. However, it is hard to say how useful this can be on a fast link such as OC-3 ATM. The jump, which is about 53*8/155M = 2.74 μs, would be barely noticeable, and easily overwhelmed by noise.
A more extreme case is SLIP/PPP dialup link. Nearly all asynchronous links use 1 start bit, 1 stop bit with no parity, therefore a 14.4kb/s modem will have a 1.44kB/s data speed. Even worse, we cannot use the above trick for ATM here, because for SLIP/PPP, the cell is simply 1 byte. This means we must interpret our results with great care, especially when the measured speed is slow (on the order of 10kb/s).

—Peculiarity of SLIP and PPP
Both PPP and SLIP have the notion of escape bytes. For instance, SLIP converts byte 0xc0 to 0xdbdc, and 0xdb to 0xdbdb. PPP converts 0x7e to 0x7d5e, and 0x7d to 0x7d5d. Finally, PPP by default escapes all bytes less than 0x20, e.g., 0x01 becomes 0x7d, 0x21 [Stevens 1994].
This can greatly affect the measurement. Initially our program fills the ICMP echo packet with mostly bytes of 0. Our results for the PPP test was about half of the real value. After we changed the filling byte to 'A', the results

became correct. We also tested Pathchar against our first experiment, which has a PPP link with 19.2 kb/s download speed. Pathchar consistently reported between 11.2 and 11.4 kb/s, with a 11.3 kb/s average. This implies Pathchar happened to choose a lot of escape bytes.

(3) Potential fatal Problems

—Asymmetry of paths

Our algorithm can solve asymmetric links, but it assumes the network path is symmetric. If the returning path is completely different, the result would clearly be unreliable.

It is difficult to detect such asymmetry. IP's record route option [Postel 1981a] can only record 9 routers. Traceroute only gives us the path for the outgoing direction.

The conclusion is there isn't much we can do about it.

—Multiple paths

This was also mentioned in Pathchar, when there are more than one way to reach the destination and come back, the measurement again is clearly questionable.

One possible way to detect this is by analyzing RTT distribution, if multiple paths do exist, it will show up as several densely distributed regression lines. However, if there are too many paths available, this method will not scale.

5. RELATED WORKS AND COMPARISONS

5.1 Tcpanaly

Tcpanaly [Paxon 1997] calculates *bottleneck* link speed by measuring One-way Transit Time (OTT) as opposed to RTT. Since OTT depends only on links in one direction, it is not affected by asymmetric links. Therefore, it works for asymmetric links. In order to obtain OTT, it runs tcpdump at both ends. For bottleneck estimate, the packet pair technique is used, implying only the spacing of OTT is required and not a globally synchronized clock. Therefore, one can measure an asymmetric link by running 2 Tcpanaly tests, each in one direction.

However, our algorithm requires only one test initiator, and does not require human intervention. In contrast, Tcpanaly requires human co-operation at the other end to run tcpdump, an arrangement not easy to get by. Further more, if we want to extend Tcpanaly to measure *per hop* link speed, one needs to measure a packet's OTT at each router hop. It means human co-operation is required at each hop, thereby easily crossing multiple ISP domains. This would add a lot of political difficulty to perform a successful test.

The advantage of OTT vs. RTT is, as explained in [Paxon 1997], less noise in measurement. RTT can have some queuing delay noise on the returning path, even when the returning packet is constant size.

5.2 Pathchar

Pathchar uses RTT of UDP packets that bounce back as either ICMP time exceeded packets, or ICMP port unreachable packets. This idea is the same as we used in our final algorithm, although we used ICMP echo packets (except for the last hop). So, it can successfully measure outgoing link speed. However, Pathchar didn't combine its use with ICMP echo/reply packets, that's why it cannot measure speeds in both directions. If we run Pathchar from host A to B, and then from B to A, it will give the results for asymmetric links correctly. This is less demanding than Tcpanaly, but it is still more restrictive than our algorithm.

5.3 Bing

Bing measures *per hop* link bandwidth, using essentially our Algorithm 1. We have already compare our Algorithm 2's results with Bing's results, together with our prediction of Bing's results. Please see Section 3.2 and 3.3.

5.4 Bprobe

Bprobe measures *bottleneck* link speed also using packet-pair technique, but it analyzes spacing of RTTs instead of OTTs as in Tcpanaly. We haven't done any testing using Bprobe because it is only ported to SGI Irix. The author has access to an SGI machine, but not with a root account. However, we can predict that Bprobe will measure the thinner of the two thinnest links in each direction, due to its packet pair technique.

6. CONCLUSIONS

We describe an algorithm that measures *per hop* raw link bandwidth where asymmetric links may exist. The algorithm requires only one end to run the measurement test. It works by sending ICMP echo packets with different TTLs and different sizes, to cause sometimes ICMP reply packets and other times ICMP time exceeded packets. Then we perform linear regression on RTTs and packet sizes, one for echo/reply case, the other for echo/time-exceeded case. This produces 2 linear equations with 2 unknowns, giving an apparent solution.

Our algorithm is very similar to those used by Pathchar and Bing. In fact, mathematically, it is a combination of both. However, it was developed independently.

Link bandwidth estimate is not simple. There are many practical issues affecting the result. Bridges, switches, hubs, peculiarity of PPP/SLIP links, ATM links, path instability and asymmetry, are all obstacles to good results. We mentioned some solutions for part of the issues, but there remains a lot of challenge to achieve good measurement.

REFERENCES

Beyssac, P. 1995. Bing - bandwidth ping.

Carter, R. L. and Crovella, M. E. 1996. Measuring bottleneck link in packet-switched networks. Technical Report BU-CS-96-006, Boston University.

Jacobson, V. 1997. pathchar - a tool to infer characteristics of internet paths. In *April 1997 MSRI Workshop* (April 1997). Mathematical Sciences Research Institute.

Paxon, V. 1997. *Measurements and Analysis of End-to-End Internet Dynamics*. Ph. D. thesis, University of California at Berkeley.

Postel, J. 1981a. Rfc 791: Ip - the internet protocol.

Postel, J. 1981b. Rfc 792: Icmp - the internet control and message protocol.

Stevens, R. 1994. *TCP/IP Illustrated, Vol. 1*, Chapter 2, pp. 24–27. Addison Wesley.

Tanenbaum, A. S. 1996. *Computer Networks*, Chapter 2, pp. 165–166. Prentice Hall.