Available bandwidth measurement as simple as running wget

Demetres Antoniades[1], Manos Athanatos[1], Antonis Papadogiannakis[1], Evangelos P. Markatos[1], Constantine Dovrolis[2]

[1] Institute of Computer Science (ICS)
Foundation for Research & Technology Hellas (FORTH)
{danton,athanat,papadog,markatos}@ics.forth.gr
[2] College of Computing, Georgia Institute of Technology
dovrolis@cc.gatech.edu

Abstract. Although several available bandwidth measurement tools exist, they usually require access at both ends of the measured path. This important requirement significantly limits the usefulness, applicability, and ease of deployment of existing tools. This work presents a novel available bandwidth measurement tool, called *abget*, that runs in "single-end" mode. Our measurement tool can connect to any TCP-based (usually web) server in the Internet, pretending that it is a normal client, and then estimate the variation range of the available bandwidth from the server to the client within a few seconds. Contrary to existing available bandwidth tools, which are based on UDP and ICMP protocols, our methodology is based on the widely prevalent TCP protocol, which enables us to perform accurate measurements even in environments where ICMP and UDP packets are blocked by firewalls or rate-limited.

1 Introduction

The area of end-to-end available bandwidth (*avail-bw*) estimation has recently received significant attention. The average avail-bw of a network path is determined by the link with the minimum "residual capacity", which is equal to the link capacity minus the average traffic load. For more precise definitions, as well as for a survey of the related work in this area, we refer the reader to [4].

Despite the large number of avail-bw tools and estimation techniques (TOPP, Pathload, Spruce, PathChirp, IGI/PTR, and others), measuring avail-bw is still considered more difficult than measuring the round-trip time or loss rate of a path. One of the main reasons is that most existing avail-bw estimation tools require the execution of measurement code at both path ends. This constraint limits the applicability of these tools in just a few paths where the user has access at both the sender and the receiver. An additional problem with existing avail-bw estimation tools is that they rely on UDP/ICMP probing packets. Such traffic is often blocked, rate-limited, or handled differently than TCP traffic. A measurement tool that only uses TCP traffic would be ideal.

In this paper, we present a new avail-bw estimation tool called *abget*. There are three key points about *abget*. First, it can be run in single-end mode, requiring access only at the path's receiving host. The sender can be any TCP-based server. Second, *abget* uses TCP packets, and it appears as a normal client for the corresponding server. The server cooperates indirectly by servicing the client with normal TCP-based transfers. In the current version of the tool, the *abget* client connects to Web servers, but it is straightforward to change the client so that it works with any other TCP-based server that can send relatively large files (more than 50-100KB). Third, the *abget* estimation methodology is very similar to that of *pathload* [2]. *Pathload* has been validated by several research studies and, in comparison with other avail-bw estimation tools, it was shown to be the most accurate [7]. Also, *abget* is able to estimate the variation range of the avail-bw, rather than just the average, similar to what *pathload* does.

Three related tools are *Sting* [6], *SProbe* [5], and *Pathneck* [1]. *Sting* and *SProbe* are also single-end tools using TCP. *Sting* measures the packet loss rate on both the forward and reverse paths from an instrumented client to any TCP-based server, relying on TCP's loss recovery algorithms. *SProbe* estimates the bottleneck bandwidth (i.e., capacity, rather than avail-bw) in both the upstream and downstream directions. In the downstream direction, *SProbe* uses TCP SYN and RST packets to force the server to send packet pairs. In the upstream direction, from the client to the server, *SProbe* performs a normal HTTP GET request, and it then analyzes the dispersion of the received packet pairs during TCP's slow start. *abget* also performs HTTP GET requests, but it generates "fake ACKs", with appropriate ACK numbers and advertised window values, so that the server will transmit periodic packet trains at a certain rate that the client chooses. *Pathneck* is also a single-end measurement tool, but it relies on ICMP and it attempts to detect the location of the avail-bw bottleneck along the path. *Pathneck* cannot estimate end-to-end avail-bw.

2 Measurement Methodology and Tool

2.1 Basic idea

We use an iterative algorithm that is similar to the Self-Loading Periodic Streams (SLoPS) technique used in *pathload* [2]. However, in *pathload*, the sender transmits periodic UDP packet streams to the client at a certain rate that is controlled by the latter. In our case, the sender is a TCP-based server that sends packets based on TCP's self-clocking, flow control and congestion control algorithms. Fortunately, there is a way to force a TCP server to send packets at a given rate. The idea is based on the use of a limited advertised window and on the generation of paced "fake" ACKs. Specifically, if the client acknowledges only one MSS (Maximum Segment Size) with each ACK and it advertises a window of only one MSS, then the server will be forced to send one MSS upon receiving each ACK, as long as the server has at least MSS bytes available in the send socket buffer. In order to achieve a certain rate R, the client's "fake" ACKs should be generated periodically with a period $T = MSS/R$.

abget emulates the TCP protocol, through the raw IP socket interface, and sends fake ACKs to the server. The ACKs are "fake" because they are generated by the client before the corresponding data segments have been received. Each ACK advances the acknowledge number by one MSS, and it sets the advertised window to one MSS. To verify this idea we used an instrumented server, passively monitored with tcpdump. Figure 1 shows a sample flow of ACKs and data segments between an *abget* client and a TCP server. ACKs are generated with a period of 247 microseconds at the *abget* client, and the transmission of data segments by the server has the same period. Figure 2 plots the interarrivals of data packets for three different ACK generation periods, and validates that the server transmits most segments, with just a few exceptions, very close to the desired rate.

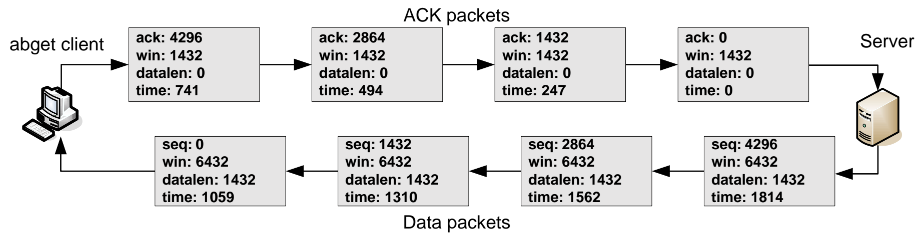

Fig. 1. An example of the flow of ACK and data segments between the *abget* client and a TCP server.

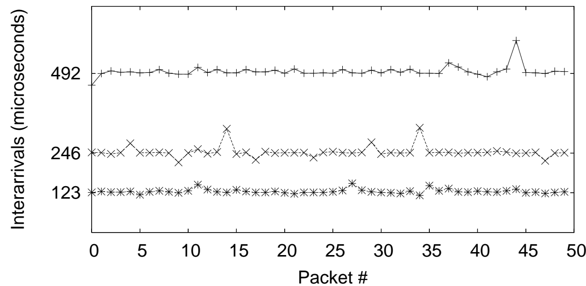

Fig. 2. Segment interarrivals at an instrumented TCP server when *abget* sends ACKs every 123, 246 or 492 microseconds.

To detect if a rate is higher than the avail-bw, we need to measure the relative One-Way Delay (OWD) of each packet. If the OWDs increase, then the probing rate is higher than the avail-bw; otherwise, the rate of that packet stream is less than the avail-bw. Since we cannot rely on the server to provide fine-resolution

timestamps at each segment, we estimate the OWDs from the interarrivals of the received segments. Specifically, after forcing the server to send a stream of packets at a certain rate, the *abget* client captures the data segments through *libpcap* recording their receive timestamps. These timestamps provide us with the packet interarrivals at the receiver. Let $s(i)$ be the time (with the sender's clock) that the sender transmitted the i'th packet, $r(i)$ the time (with the receiver's clock) that the receiver got the i'th packet, o the clock offset between the two hosts, $d(i)$ the OWD of packet i, $t(i)$ the interarrival between packets i and $i-1$ at the receiver, and T the (assumed) constant interarrival between packets i and $i-1$ at the sender. Then, it is easy to see that $s(i) = s(i-1) + T$, $r(i) = s(i) + d(i) + o$ and $t(i) = r(i) - r(i-1) = T + d(i) - d(i-1)$. So, we can reconstruct the OWD time series as follows

$$\begin{aligned} d(i) &= r(i) - s(i) - o = t(i) + r(i-1) - s(i-1) - T - o = \\ &= d(i-1) + t(i) - T \end{aligned} \tag{1}$$

Starting the recursion with $d(1) = 0$, we can estimate the sequence of OWDs, and then use the techniques and fine-tuned thresholds developed for *pathload* in the detection of increasing OWD trends.

2.2 Measurement tool: abget

abget uses an iterative algorithm, based on linear probing. Specifically, the user specifies the probing range `[Rmin, Rmax]` (e.g., from almost zero to the capacity of the client's network interface) and the estimation resolution `w`. The *abget* algorithm, summarized in Figure 3, starts probing at rate `Rmin`, gradually increasing the probing rate in increments of `w` until the latter exceeds the `Rmax`. So, *abget* probes at $\lfloor (Rmax - Rmin)/w + 1 \rfloor$ rates.

In each iteration, the *abget* client connects to the remote server and initiates a download operation for a sufficiently large file[3]. Next, *abget* starts sending ACKs with a period `T` that corresponds to the desired probing rate, with an advertised window of one MSS. Each ACK advances the acknowledge number by one MSS[4]. The number of ACKs is equal to the stream length parameter `K`, which determines the number of packets in each stream. This parameter is related to the variability of the estimated avail-bw, as discussed in more detail in [3].

After receiving K segments from the server, *abget* estimates their OWDs, as described earlier. The analysis of OWDs is similar to *pathload*. Specifically, we split them in groups of successive values, calculate the median of each group, and ignore the rest of the measurements. This is a useful technique for removing outliers. From the remaining median values, we calculate the *Pairwise Comparison*

[3] To help users locate large files on the web server, we have developed a crawler that finds suitable files through google searches and recursive crawling

[4] It should be noted that not all data segments carry MSS bytes. We handle those cases by sending again the corrected remaining ACKs.

```
for ( rate = Rmin; rate <= Rmax; rate += w )  {
    for ( currStream=0; currStream<N; currStream++ )  {
        TCP_Handshake( server );
        GET_Request( server, filename );
        send_fake_ACKs( stream length K, period T );
        OWD_vector = compute_OWDs();
        median_vector = get_medians( OWD_vector );
        PCT = pairwise_comparison_test( median_vector);
        if ( PCT > 0.65 ) increasing_streams++;
        else if ( PCT < 0.54) non_increasing_streams++;
        else grey_streams++;
    }
    if ( increasing_streams > N/2 )
        if ( rate < high_bound ) high_bound = rate;
    else if ( non_increasing_streams > N/2 )
        if ( rate > low_bound ) low_bound = rate;
}
```

Fig. 3. *abget* pseudocode.

Test (PCT) statistic, given in [2]. The PCT measures the fraction of consecutive one-way delay pairs that are increasing. Based on the PCT value, we then classify that packet stream as higher than the avail-bw (*increasing_stream*), or lower than the avail-bw (*non_increasing_stream*). It is also possible that we cannot reliably classify that stream (*grey_stream*).

To decide whether a probing rate is larger than the avail-bw, *abget* repeats the previous process N times for each probing rate. The parameter N corresponds to the number of streams per probing rate. The final classiffication of a probing rate is based on majority counting, i.e., if more than $N/2$ of the streams are increasing (non-increasing), we infer that the corresponding probing rate is higher (lower) than the avail-bw. The *abget* client stays idle for a user-specified time period T_i between iterations, to control the measurement overhead.

Finally, *abget* reports a variation range `[low_bound, high_bound]`. This is the range from the maximum probing rate that was estimated as lower than the avail-bw (`low_bound`) to the minimum probing rate that was estimated as higher than the avail-bw (`high_bound`). If it turns out that `high_bound` is less than `low_bound`, the tool reports that the avail-bw process showed signs of non-stationary behavior during the measurement.

2.3 Measurement duration and overhead

Reducing the measurement duration, we can achieve faster estimation and (typically) lower overhead. On the other hand, increasing the measurement duration often leads to better accuracy, as the tool can probe each rate with more streams or with longer streams. In *abget*, the trade-offs between measurement duration, overhead, and accuracy can be controlled by the user through the selection of

the following parameters: stream length K, number of streams N, estimation resolution w, idle time between streams T_i, and probing range [$Rmin$,$Rmax$]. Specifically, the measurement duration is:

$$\left\lfloor \frac{Rmax - Rmin}{w} + 1 \right\rfloor \times N \times \left(\frac{K \times MSS}{R_{avg}} + T_i \right) \tag{2}$$

where $R_{avg} = (Rmax + Rmin)/2$. The measurement overhead, in terms of rate, is

$$\frac{K \times MSS}{(K \times MSS)/R_{avg} + T_i} \tag{3}$$

We do not have a mathematical expression for the accuracy of the tool, as that would greatly depend on the characteristics of the avail-bw process, together with the previous parameters.

3 Validation Results

This section presents validation results for *abget* in a completely controlled testbed and in an operational instrumented network path. In all the experiments of this section, unless stated otherwise, the *abget* parameters were set as follows: N=5 streams per probing rate, K=50 packets per stream, w=5Mbps estimation resolution, probing range $Rmin$=0 and $Rmax$=100Mbps, and idle time T_i=500msec. The duration of a measurement with these parameters is about 50 seconds.

3.1 Validation in a local testbed

We set up a fully instrumented testbed to create a single-hop network with a 100Mbps tight link between two Ethernet switches. Note that the capacity of the tight link at the IP layer (with Ethernet MTU packets) is about 97Mbps. A pair of hosts (sender and receiver) were used to generate the cross traffic. Another host was used as a web server, while a fourth host was running the *abget* client. The cross-traffic sink and the *abget* client were connected to the first switch, while the cross traffic source and the web server were connected to the second switch.

In the first experiment, the sender generates constant-rate UDP traffic with *Iperf*. We varied the UDP rate in the 10-90Mbps range. The results, both with *abget* and *pathload*, are presented in Figure 4. We observe that *abget* produces accurate estimates, compared to the actual avail-bw in the link. The results with *pathload* are generally more accurate, but we should keep in mind that *pathload* requires access at both ends of the path.

In the second experiment, we generated realistic traffic by "replaying" a packet trace that was previously collected from the University of Crete access link. The average rate of the original trace was about 11Mbps. So, to emulate different values of avail-bw, we scaled the packet interarrivals by a certain factor

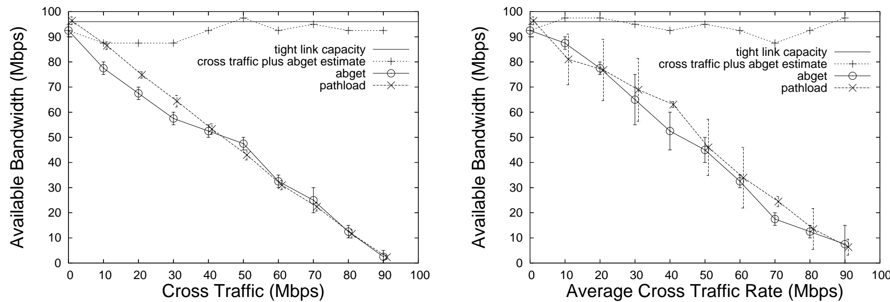

Fig. 4. Validation in testbed with constant-rate UDP cross traffic.

Fig. 5. Validation in testbed with trace-driven cross traffic emulation.

for each desired cross traffic rate. The results, presented in Figure 5, show that *abget* provides accurate measurements, in the sense that the sum of the cross traffic average rate and the center of the *abget* estimation range are close to the path capacity (97Mbps). The results with *pathload* are of comparable accuracy.

3.2 Validation in a passively monitored operational network path

The next set of validation experiments was performed at an operational Internet path in which we could passively monitor what we expect to be the tight link. Specifically, the passive monitor is a packet collector that we placed at the access link of the University of Crete (UoC, in Heraklion, Greece), as shown in Figure 6. The capacity of this link is 34 Mbps. Figure 7 shows *abget* measurements in the

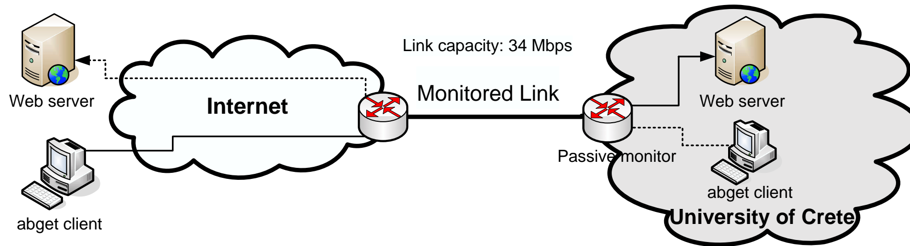

Fig. 6. The monitored network path.

path from an external Web server (*www.nytimes.com*) to a client within the University. Figure 8 shows similar results when we ran *abget* at a client located outside the monitored network, at Georgia Tech (in Atlanta, USA), to a Web server inside UoC. An *abget* measurement was performed every five minutes, while the duration of each measurement was 50 seconds.

We attempted to estimate the variation range of the actual avail-bw at the tight link as follows. During each 50-second *abget* measurement, we passively

measured the actual avail-bw at the tight link in consecutive time intervals of length 12msec; this is the average duration of an *abget* packet stream in this path. The variation range during the 50-second period is the range between the minimum and the maximum 12-msec avail-bw measurement. Figures 7 and 8 show the variation range of the actual avail-bw together with the corresponding *abget* estimated variation range.

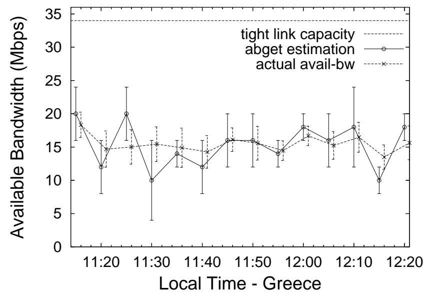

Fig. 7. Available bandwidth from *www.nytimes.com* to UoC client.

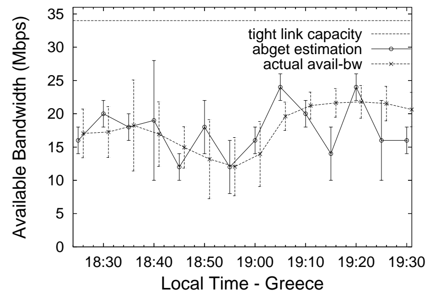

Fig. 8. Available bandwidth from a UoC Web server to an *abget* client at Georgia Tech.

A first observation is that the *abget* central estimates (the center of the estimated variation range) are, for the most part, within the corresponding avail-bw variation range at the UoC access link. The accuracy of the tool is not as good in the path of Figure 8. One possible explanation is that the spesific path includes more links, other than the UoC access link, that occasionally limit the end-to-end avail-bw. Also, these results show that the *abget* variation range does not always follow the variation range of the UoC access link. A plausible explanation is that the variation range of *abget* depends on the avail-bw variability in the entire path, not just at the UoC access link.

3.3 Robustness to reverse path traffic

So far, we have assumed that the ACKs can reach the server periodically, as sent by the client. This will not be the case when the reverse (non-measured) path from the client to the server is significantly loaded with traffic. Specifically, if L_D is the size of a data segment (sent at the forward path) and L_A is the size of the corresponding ACK segment (sent at the reverse path), then the load imposed by the *abget* ACKs in the reverse path will be L_D/L_A times less than the probing rate at the forward path. The load due to ACKs does not create a problem as long as the avail-bw in the reverse path is no less than L_D/L_A times the avail-bw in the forward path; otherwise, the ACKs saturate the reverse path and so they do not arrive at the server periodically. Typically L_D=1500B and L_A ≈40B, which means that the ratio L_D/L_A is about 40. We expect that only few paths

will have such a high degree of avail-bw asymmetry. We have also examined these effects experimentally, with testbed measurements, verifying that *abget* is indeed robust to significant traffic load in the reverse path.

4 Available Bandwidth Variability

This section presents timeseries for avail-bw measurements using *abget*, focusing on the temporal variability of the available bandwidth process. We used two different client hosts, one located at the University of Crete (UoC, in Heraklion, Greece), and another located at the Georgia Institute of Technology (Gatech, in Atlanta, USA). We measured paths from two popular[5] Web servers: *www.nero.com* (in Germany) and *www.chez.com* (in France). The *abget* parameters were set as in the previous section. Figure 9 shows the timeseries of avail-bw measurements in the four paths. A new measurement is performed every 10 minutes during a 24-hour period.

In the case of the UoC client, both paths are limited by the 34Mbps access link of the University of Crete. Also, both paths show the same diurnal pattern, with the avail-bw reaching its maximum (around 20Mbps) in the early morning hours and its minimum (around 5Mbps) in the afternoon and evening hours. The variation range of the avail-bw is typically wider when the avail-bw is lower, as described in [3].

The two rightmost graphs show avail-bw measurements from the two servers to the Gatech client. Here, the two paths seem to have different tight links and diurnal patterns. The avail-bw in the path from the Nero server seems to go through diurnal variations, with significantly lower avail-bw during the morning/afternoon hours (EST time). The path from the Chez server does not show such diurnal variations, and it has significantly higher avail-bw. Notice however that the avail-bw variation range in that path is often wider than 40-50Mbps. As shown in [3], a wide variation range should be expected when the tight link carries just a few flows (low degree of statistical multiplexing).

Acknowledgments

This work was supported in part by the IST project LOBSTER funded by the European Union under contract number 004336. This work was also supported in part by the GSRT project EAR (USA-022) funded by the Greek Secretariat for Research and Technology. A. Papadogiannakis, D. Antoniades, M. Athanatos and Evangelos P. Markatos are also with the University of Crete. The work of C. Dovrolis was supported in part by the NSF CAREER award ANIR-0347374.

[5] Based on *www.netcraft.com*.

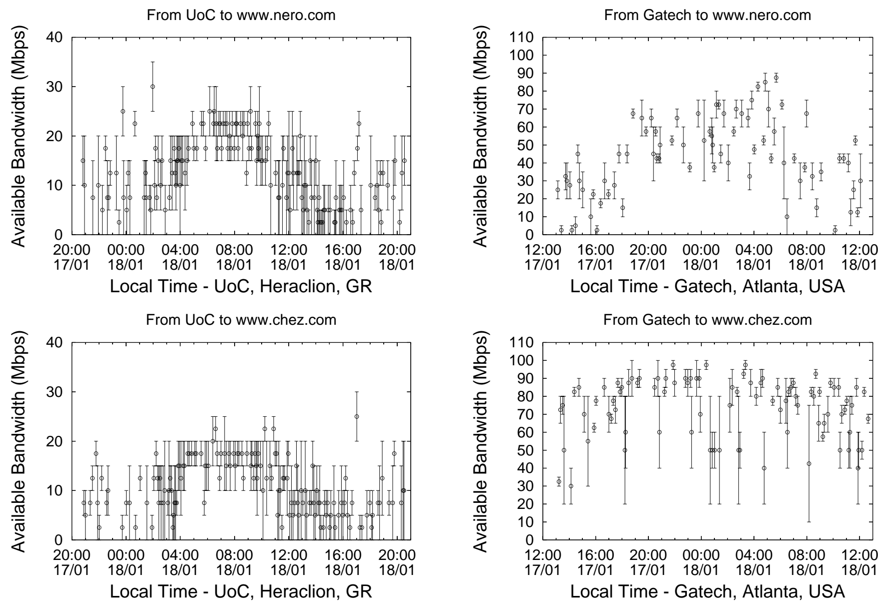

Fig. 9. Available bandwidth measurements from two servers to two different *abget* clients.

References

1. N. Hu, L. E. Li, Z. M. Mao, P. Steenkiste, and J. Wang. Locating Internet Bottlenecks: Algorithms, Measurements, and Implications. In *Proceedings ACM SIGCOMM*, 2004.
2. M. Jain and C. Dovrolis. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. *IEEE/ACM Transactions on Networking*, 11(4):537–549, Aug. 2003.
3. M. Jain and C. Dovrolis. End-to-end estimation of the available bandwidth variation range. In *Proceedings of ACM SIGMETRICS Conference, Banff, Canada*, June 2005.
4. R. S. Prasad, M. Murray, C. Dovrolis, and K. Claffy. Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. *IEEE Network*, Nov. 2003.
5. S. Saroiu, P. Gummadi, and S. Gribble. SProbe: A Fast Technique for Measuring Bottleneck Bandwidth in Uncooperative Environments. In *Proceedings of IEEE INFOCOM*, 2002.
6. S. Savage. Sting: a TCP-based Network Measurement Tool. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, 1999.
7. A. Shriram, M. Murray, Y. Hyun, N. Brownlee, A. Broido, and M. Fomenkov. Comparison of Public End-to-end Bandwidth Estimation Tools on High-Speed Links. In *Proceedings Passive and Active Measurements (PAM)*, 2005.