

# LinkWidth: A Method to Measure Link Capacity and Available Bandwidth using Single-End Probes.

Sambuddho Chakravarty  
Department of Computer  
Science  
Columbia University  
sc2516@cs.columbia.edu

Angelos Stavrou  
Department of Computer  
Science  
Columbia University  
angel@cs.columbia.edu

Angelos D. Keromytis  
Department of Computer  
Science  
Columbia University  
angelos@cs.columbia.edu

We introduce LinkWidth, a method for estimating capacity and available bandwidth using single-end controlled TCP packet probes. To estimate capacity, we generate a train of TCP RST packets “sandwiched” between trains of TCP SYN packets. Capacity is computed from the end-to-end packet dispersion of the received TCP RST/ACK packets corresponding to the TCP SYN packets going to closed ports. Our technique is significantly different from the rest of the packet-pair based measurement techniques, such as *CapProbe*, *pathchar* and *pathrate*, because the long packet trains minimize errors due to bursty cross-traffic. Additionally, TCP RST packets do not generate additional ICMP replies, thus avoiding cross-traffic due to such packets from interfering with our probes. In addition, we use TCP packets for all our probes to prevent QoS-related traffic shaping (based on packet types) from affecting our measurements (eg. CISCO routers by default are known have to very high latency while generating to ICMP TTL expired replies).

We extend the *Train of Packet Pairs* technique to approximate the available link capacity. We use a train of TCP packet pairs with variable intra-pair delays and sizes. This is the first attempt to implement this technique using single-end TCP probes, tested on a range of networks with different bottleneck capacities and cross traffic rates. The method we use for measuring from a single point of control uses TCP RST packets between a train of TCP SYN packets. The idea is quite similar to the technique for measuring the bottleneck capacity. We compare our prototype with *pathchirp*, *pathload*, *IPERF*, which require control of both ends as well as another single end controlled technique *abget*, and demonstrate that in most cases our method gives approximately the same results if not better.

## Categories and Subject Descriptors

H.1.1 [Models and Principles]: Systems and Information Theory—*Value of Information*

## General Terms

Measurement, Experimentation

## Keywords

link capacity, available capacity, single-end tool, link measurement tool

## 1. INTRODUCTION

The scale and complexity of the Internet makes the task of understanding and analyzing its properties extremely difficult. The diffuse style of administration and operation means that even such basic information as topology or link capacity is spread across multiple entities with little incentive to share it. This knowledge, however, is necessary for the design and development of better management tools, communication protocols, and mechanisms that are network-related. As a concrete example, there is little information available on the distribution of access-link capacities and available bandwidth for end-users (*e.g.*, how fast can the average user send/receive data from the network). Although our interest is motivated by our work on distributed denial of service defense mechanisms [14, 24], such information is more generally useful. For example, if end users can estimate this information for different network paths then, given a choice between different mirrors, the user can choose the one with the best access characteristics.

This recognition has spurred research in this area, with several efforts focusing on estimating available capacity between two hosts [10, 17, 27, 19, 22], and measuring the capacity of an arbitrary network link respectively [15, 10, 22, 8]. Most of these tools (eg. *abget*, *PathLoad*, *PathChirp*, *IPERF* etc.) use a technique called Self Loading of Periodic Streams (SLoPS). Although accurate, tools that measure the available capacity require *two-ended control*, *i.e.*, that the two endpoints of a path collaborate in the measurement. Thus, the utility of such tools is limited, given our goal to conduct large-scale and/or opportunistic measurements of arbitrary links and paths. One notable exception is *abget* [4], which uses SLoPS with a remote TCP based

server to act as the remote measurement anchor point.

We present *LinkWidth*, a novel technique for measuring both installed and available capacity for Internet paths. We measure installed capacity by employing a modified version of the *Recursive Packet Train (PATH-NECK)* method [8]. In addition, in contrast to previous work that depends on a two-ended *Self-Loading of Periodic Streams (SLoPS)* [22] approach, we implemented an extension of the single-ended *Train of Packet-Pair* [22, 2] (TOPP) algorithm to measure available capacity.

Recursive Packet Train (RPT), used originally in path-neck, uses a train of large sized ( $\approx 500\text{bytes}$ ) UDP packets (which the authors called the load packets), sandwiched between trains of smaller UDP packets ( $\approx 60\text{bytes}$ ) at the beginning (head measurement packets) and the end of the train (tail measurement packets) with increasing and decreasing TTL values respectively. At each hop of this entire train the sender receives two ICMP TTL Expired packets. The time difference between two such consecutive ICMP TTL Expired packets is accepted as the received dispersion of the train for the particular hop. *LinkWidth* employs a modified version of RPT, where the head and tail packets are replaced by TCP SYN packets going to closed ports on consecutive hosts along the path to the host that is being probed. Later sections of this paper elaborate how this modified RPT is being used for estimation of end-to-end installed/bottleneck capacity.

The Train of Packet Pairs is quite similar to SLoPS. In fact rather than measuring the end-to-end OWD of packet train arriving the receiver, TOPP goes of increasing the sending rate to reach a point where the sender is attempting to send faster than the available capacity. Further increasing the sending rate above the available capacity, the packets get queued at the intermediate routes while the receiver cannot receive faster than the available capacity (bandwidth) of the slowest link (i.e most congested link, in case the most congested link's present/instantaneous available capacity is lower than the available capacity of that of the slowest physical link). Thus, as long as the sender is still sending within the available capacity, the receiving rate is not more than that the available capacity. Thus the ratio of sending rate to the receiving rate is close to unity. When the sending rate is more than the available capacity, the sender goes on trying to send faster while the receiver cannot receive faster than the available capacity.

*LinkWidth* doesn't adhere to the TOPP model verbatim. Instead of increasing the sending rate, as described by the authors, a binary search procedure is used to converge to a value of sending rate which is within the end-to-end available capacity. The ratio of received dispersion to sending dispersion is used the search criteria.

The search is contingent upon convergence parameters which are user defined thresholds, depending upon how much opportunistic/reactive we want our probe trains to be. Detailed explanation of this techniques is presented in later sections of this paper.

We evaluate *LinkWidth* using a variety of tests, both over controlled environments (with controlled degrees of cross-trafficburst ness) and on the Internet (where there is no control over the cross traffic). We compare the results obtained via *LinkWidth* with the ground truth (when known) and against the measurements obtained from other tools. We thereby conclude empirically that for most cases we are accurately able to determine the capacity and closely reach the bottleneck available capacity when compared other known tools such as IPERF [27] which set up an end-to-end TCP connection and estimate the available capacity.

In summary, the contributions of our work are:

- A novel extension of Recursive Packet Train and Train of Packet Pair techniques for single-ended estimation of installed capacity and available capacity of network links and paths, without collaboration from the network or the remote endpoint.
- An implementation of *LinkWidth* in the form of a tool for Linux 2.6.

- An experimental evaluation of the measurement accuracy of *LinkWidth* using various scenarios both in controlled network environments and on the Internet.

**Paper Outline** Section 2 outlines the related work. In section 3, we introduce our extensions to TOPP and RPT techniques and how we used them to measure both installed and available capacity of Internet paths. We analyze our experimental results and present performance comparisons of our approach to known tools in section 4. Section 5 concludes the paper.

## 2. RELATED WORK

There exists a wealth of tools for measuring *Installed or Bottleneck Capacity* of an end-to-end path. Dovrolis et. al. have presented an accurate definition of this metric [3]. Consider a network path  $P$  as a sequence of first-come first-served (FCFS) store-and-forward links that transfer packets from a sender  $S$  and receiver  $R$ . Each link  $i$  transmits data with a constant rate of  $C_i$  bits per second, referred to as *link capacity* or *transmission rate*. The bottleneck link capacity or installed capacity is the minimum transmission capacity of all links in  $P$ . More formally, if  $H$  is the number of hops(links) in  $P$ ,  $C_i$  is the capacity of the link  $i$ , and  $C_0$  is the transmission rate of the sender, then the path capacity is:

$$C = \min(C_i) \quad i \in \{0, \dots, H\}$$

Note that the link capacity is independent of the traffic load on the path.

On the other hand, the bottleneck link capacity is measured by injecting two back-to-back packets into the network and thereby measuring the dispersion (time delay between the end of reception of the first packet to the end of reception of the last packet) [23, 3]. Pathchar [7, 10] appears to be one of the most accurate tools to estimate the slowest link of an end-to-end IP Path. Figure 1 displays the measuring principle behind this technique: two back-to-back packets are sent from a one link to another via a low capacity bottleneck link (we shall refer to such bottleneck link as the narrow link or the slow link). When this packet pair leaves the narrow link for a faster/higher capacity link, a gap is introduced between the packet pair which is proportional to the slow link's capacity.

The *Packet Train* method enlarges the packet-pair dispersion by increasing the number of packets. The received dispersion is measured between the end of the first packet and the end of the last packet of the train [22]. Thus, the end-to-end capacity is measured as:

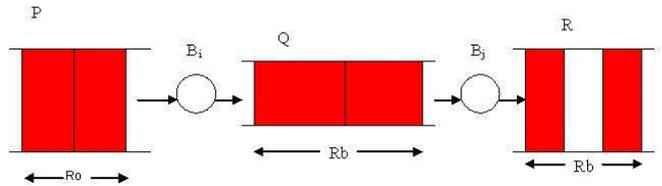
$$C = (N - 1) * \frac{L}{\Delta_R(N)}$$

for a train of  $N$  packets, where the  $L$  is the size of each packet (in bits) and  $\Delta_R(N)$  is the end-to-end dispersion of the entire train [22, 13, 20].

LinkWidth uses a variation of the Recursive Packet Train [8] (originally RPT was used in Pathchar [8, 9]). Packet train has its obvious advantages over the packet pair method: IP, inherently unreliable connectionless protocol, can potentially fragment large size packet and may even reorder them. This is especially true for large sized packets. Packet train method overcomes such an effect by sending long trains of packets. Smaller cross traffic packets in such long trains don't add significant end-to-end dispersion to cause much perturbation in the measurements. Packet pair suffer from size dependent over-estimation and under-estimation effects [13].

Other tools which measure bottleneck capacity such as pchar [15], clink [6] and bing [21] are not as accurate as pathchar under various/adverse link and cross-traffic conditions. The experimental results which justify our claim, have been presented in later sections in this paper. Although all previous tools implement packet pair method in some form, they suffer from capacity under-estimation and capacity-overestimation effects due to various link and cross traffic conditions [13, 20]. LinkWidth uses long trains of TCP RST packets (sent back-to-back in real-time) can be used effectively to compute the bottleneck link capacity under various link and traffic conditions. Moreover, this value is capacity is used as an upper bound in our measurement of available capacity (using a modified version of Train of Packet Pair method).

[3] further defines *Available or Un-utilized Capacity*



**Figure 1: Basic Packet Pair Dispersion Technique: The two packets go from a high-capacity link to a slow one and then to a high-capacity link introducing a delay between the pairs**

as a function of the link's *utilization*. If  $u_i$  is the link utilization for link  $i$  (where  $0 \leq u_i \leq 1$ ) over a certain time interval, the average spare capacity of link  $i$  is  $C_i(1 - u_i)$ . Thus, the available capacity of  $P$  in the same interval can be defined as:

$$A = \min([C_i(1 - u_i)]), \quad i \in \{0, \dots, H\}$$

Available capacity has been further classified as *Available Bottleneck Link Capacity, Surplus Available Capacity and Proportional Share Capacity* by Melander, Bjorkman et. al. [2]. Most research [2, 22, 3, 11, 28] have acknowledged available capacity to be defined as something quite similar to this definition.

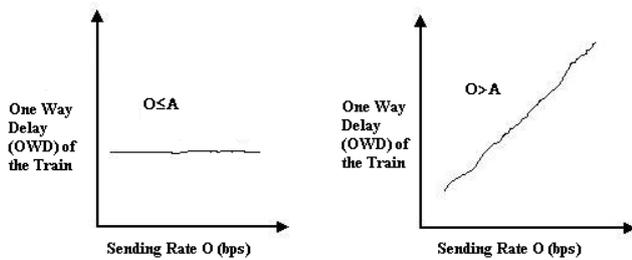
There are two broad types of available capacity estimation techniques, viz. Self Loading of Periodic Streams (SLoPS) and Train of Packet Pair (TOPP) [22]. The idea behind both these techniques is not radically different. In SLoPS one of the hosts (the sender  $S$ ) send to the other host (the receiver  $R$ ) a stream of packets over a fixed path  $P$ . The receiver measures the One-Way-Delay (OWD) of such trains in succession [18]:

$$\Delta_R = T_{arrive}^R - T_{send}^S = T_{arrive} - T_{send} + Offset(S, R)$$

As long as  $S$  sends to  $R$  within the available capacity, the OWD such trains remains constant. When the sender injects packets at a rate faster than the available capacity (minimum spare capacity), the OWD starts to grow linearly (see figure 2).

To measure the OWD of each of the trains, SLoPS requires both sender  $S$  and receiver  $R$  to have either synchronized clocks or add to the difference in time of reception and sending, the clock drift offset the hosts. This requirement makes SLoPS difficult to be used in situations where we do not have control over the clock of the host at the other end of path  $P$ . Some of the most commonly known available capacity estimation techniques use the SLoPS technique.

TOPP [2] is not very different from SLoPS: the sender iteratively sends a train of packet pair at some rate  $O_i$  (typically  $0 \leq O_i \leq C(\text{bottleneck capacity})$ ) over the fixed path  $P$  to receiver  $R$ . Assuming  $R$  receives these train of packet pairs at some rate  $M_i$ , the ratio  $O_i/M_i$  is calculated. Theoretically, this ratio should be exactly



**Figure 2: Variation of One Way Packet Delay(OWD) versus Number of Packets Sent in SLoPS: OWD remains constant as long as the sending rate is within the available capacity; grows linearly when the sending rate exceeds the available capacity**

one as long as the sender sends within the available capacity (figure 3 shows a variation of  $O/M$  versus  $O$ ). When the sender sends faster than what the receiver can receive, the ratio grows linearly with respect to the sending rate  $O_i$  (the receiver cannot receive faster than the available capacity; this is shown by the knee point,  $\tau$ , in figure 3). A train of packet pair is definitely better than just a pair.

Since TOPP relies only on the sending and receiving rates (sending and receiving dispersions) of the entire train, it is a suitable of measurement technique for single-end point controlled technique. SLoPS requires either the clocks of the hosts to be synchronized or the clock offset to be taken into account during the measurement of the OWD of each train. This makes SLoPS difficult to be implemented as a single-end point controlled technique.

LinkWidth is the first tool to implement a single-end controlled tool for measuring end-to-end capacity using an optimized Train of Packet Pair. We use a binary search over the range (0 to  $C$  (the end-to-end capacity);  $A$ , the available capacity cannot possibly more than the  $C$ ). Other tools commonly known for measuring available capacity are pathrate [3], pathchirp [28], pathload [11], IPERF and lately abget [4] (which is the only other single-end controlled available capacity estimation technique which gives a measure of both the upload and download capacity). Most of these available capacity estimation tools use SLoPS technique for measurement of available capacity; LinkWidth, to the best of the knowledge of the authors, is the only tool that implements single-end controlled tool that uses an optimized version of TOPP to compute available capacity.

### 3. MEASUREMENT APPROACH

This section discusses our measurement methodology for both installed and available capacity from a single

host. We extend some of the existing tools and described in the previous section 2. We begin this section with some background information about of TOPP and RPT and then we analyses the design and implementation details of our modified RPT and TOPP and how we integrated them into LinkWidth.

#### 3.1 Train of Packet Pair (TOPP)

In Train of Packet Pair (TOPP) method the sender  $S$  sends trains of packet pairs at gradually increasing rates from to the receiver  $R$ . Assuming a train of packet pair sent with an initial dispersion of  $R_o$ . The probe packets have size of  $L$  bytes and thus the offered rate of the packet pair is  $O = L/R_o$ . As long as  $O \leq A$ , TOPP assumes that the train of packet pair will arrive at the receiver with the same rate with which it was injected into the network by the sender, i.e.  $M = O$ . If  $O > A$ , the measured rate at the receiver will be  $M < O$ . As per [2] that the received rate  $M$  is a fraction of the sending rate  $O$  when the sending rate  $O$  exceeds the available capacity  $A$ . Typically,  $M = (O/(O + X)) * C$

Thus,

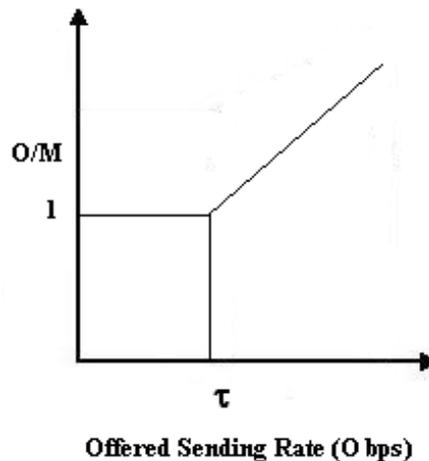
$$M = (O/(O + X)) * C$$

$$\text{or } O/M = (O + X)/C$$

$$\text{or } O/M = (O + C - A)/C$$

$$\text{or } O/M = O/C + (1 - A/C)$$

This gives a linear variation of  $O/M$  versus  $O$  in the case where  $O > A$ . This is illustrated in figure (Figure 3)



**Figure 3: Variation of the Ratio (Offered Rate ( $O$ ) / Measured Rate ( $M$ )) versus Offered Rate ( $O$ ) :  $O = \tau$  is available capacity and the ratio  $O/M = 1$ .  $O > A$  is indicated by the linear increase of  $O/M$  with increase in  $O$**

Here the graph displays the available capacity achieved at  $O=\tau$ . The very idea of TOPP isn't far from SLoPS

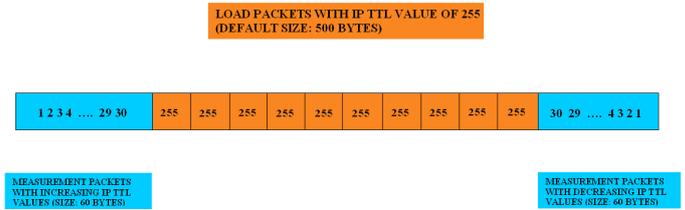
but we neither SLoPS nor TOPP verbatim. We propose a modification of the original TOPP to include a binary search technique to estimate the value of available capacity continent upon two convergence parameters  $\theta$  and  $\epsilon$  described in a later subsection 3.4).

As long as the sender injects packets at a rate within available capacity  $A$ , it must be received at the same rate. In case of cross-traffic, the probe packets may have to wait in queues, thereby possibly incurring additional queuing delay(s) between the first and last packet of the train (possibly resulting in additional end-to-end dispersion). This is due to best effort routing of the Internet which may multiplex probe traffic with cross-traffic. Thus, less cross-traffic would result in more available capacity for the probe packets. The greater the available capacity, the higher the chances of the train of packet pairs to be forwarded with rates at which it was injected into the network by the sender. This justifies the fact why small cross traffic on high bandwidth links has less effect on our probes (as presented in section 4). The available capacity measure in such cases is usually close to the maximum installed link capacity. This is also the case for other available capacity estimation tools, such as IPERF, which set up an end-to-end TCP connection. Elastic TCP flows utilize all the un utilized available capacity; hence what we measure from LinkWidth (which uses such train of packet pair probes) is not very different from what we measure when using IPERF. These results are presented and explained in detail in a later section of this paper (section 4). Corollary to this observation, in case of a non-elastic Constant Bit Rate (CBR) UDP traffic, the available capacity (the “head room”) may not increase considerably. Thus, what we measure as the installed link capacity is actually the bandwidth un utilized by the aggressive CBR UDP cross traffic (section 4).

### 3.2 Pathneck (Recursive Packet Train)

Pathneck proposed by Ninigin Hu et. al. [8], used Recursive Packet Train (RPT) technique to locate bottleneck links and measure the end-to-end installed bottleneck capacity. The recursive packet train uses a train of back-to-back UDP packet (called load packets) which are appended and prepended by another train of UDP packets called measurement packets. Figure 4 describes the arrangement of packets in the pathneck arrangement.

Evident from the arrangement of packets in RPT, the TTL values of each of the head and tail measurement packets would decrement to zero at each hop of the train. This would send ICMP TTL Expired packets back to the sender. The dispersion between two consecutive ICMP TTL expired packets at the source from the same router would give a good approximation of the dispersion of the packet train as perceived at the



**Figure 4: Arrangement of packets in RPT Pathneck: IP TTL values increase from 1...30 (head measurement packets);intermediate load packets TTL=255;IP TTL values decrease from 30...1 (tail measurement packets)**

router.

### 3.3 The TCP Variant of RPT

We modified the RPT method used by pathneck and incorporated TCP SYN packets in place of UDP/ICMP packets as was described in the previous section. TCP SYN packets being sent to a closed TCP port (on which no service is waiting for incoming connections) are used as the head and tail measurement packets. Any operating system, adhering to standards, would reply back to such with a TCP RST+ACK packet.

Our arrangement (Figure 5) somewhat attempts to emulate the UDP based RPT. The head measurement packets which are sent to even port numbers that match up with the tail measurement packets going to the next consecutive odd port numbers (the original RPT implementation uses the 16-bit IP Identification field of the IPv4 header for this purpose).



**Figure 5: TCP based variant of RPT:TCP SYN packets to even port offset (head measurement packets);Intermediate TCP RST (load packets);TCP SYN Packets to odd offset (tail measurement packets)**

From the arrangement (Figure 5), we see each of the consecutive SYN segments in the arrangement being sent of the consecutive routers along the path to the destination. Starting with a certain base port number BASE-PORT, each of the consecutive packets are sent to consecutive even destination port numbers BASE-PORT+2, BASE-PORT+4, BASE-PORT+6, BASE-PORT+(2 \* N). Thus, packet sent with destination port as BASE-PORT+2 is sent to router 1, with BASE-

PORT+4 is sent to router 2, with BASE-PORT+6 is sent to router 3 and so on such that the one going to BASE-PORT+(2 \* N) is sent to the destination hop. This arrangement of packets forms the head measurement packets. This is followed by the TCP RST load packets; and finally the tail measurement packets which are sent to consecutive odd destination port numbers BASE-PORT+3, BASE-PORT+5, ..., BASE-PORT+2\*N + 1 (each being sent to the consecutive routers along the path to the destination). The reason for using such port numbering is to match up each of the head measurement packet's TCP RST+ACK reply (from even TCP port number) with that due the tail measurement packet (from the next consecutive odd TCP port number). Thus, the measured time dispersion between two such consecutive TCP RST+ACK packet (coming from two consecutive port numbers) gives the end-to-end dispersion of the entire train measured for that particular router.

There are two inherent advantages of the above arrangement:-

1. Most of the single-ended controlled methods for measuring the capacity and / or available capacity make, use of ICMP TTL expired packets. Commercially available routers give very low priority to ICMP packet generation / forwarding resulting in large end-to-end gaps being inserted between the ICMP TTL expired packets. This leads to capacity underestimation. We avoid, as much as possible, such a situation using TCP packets.
2. The load packets used here are TCP RST packets which don't result in the ICMP Destination Unreachable packets which used to be generated in case of older UDP/ ICMP based techniques such as Pathneck and CapProbe. This avoids reverse cross-traffic for the forward probe traffic.

### 3.4 TCP Based Train of Packet Pairs (TOPP) - Available Capacity Estimation

A certain variation of the Train of Packet Pair method, described earlier, is used to measure the available capacity of an end-to-end IP path. A single-end controlled technique, requires the sender to send packets at varying sending rates and to measure the varying receiving rates, and thereby converging to a value of available capacity ( $A = O_i$  (some sending rate less than the capacity  $C$ )) depending on whether  $O/M \approx 1$  or  $O/M > 1$ . The arrangement of packet train (figure 6). Rather than using an iterative linear increment to  $O_i$  (where  $0 \leq O_i \leq C$ ) we perform a binary search over the entire range  $[0, C]$  to select values for  $O_i$  and decide if we are sending within available capacity or more than that.

### Binary Search Procedure Used to Compute the Available Capacity

The value of available capacity converges to a value  $O_i$ , such that  $O_{MIN} \leq O_i \leq O_{MAX}$  ( $O_{MIN}$  being zero in all cases and  $O_{MAX}$  being the upper bound -  $C$ ). LinkWidth iteratively performs a binary search over the range  $[O_{MIN}, O_{MAX}]$ . At each iteration, LinkWidth send the Train of Packet Pairs at  $O_{MID}$ , where  $O_{MID} = (O_{MIN} + O_{MAX})/2$ . At each iteration, the ratio of end-to-end dispersion of the packet train as received  $R_m$  versus the end-to-end dispersion with which the sender injects it into the network  $R_o$ , is compared with the convergence parameter  $\epsilon$ . If this ratio  $(R_m/R_o) > (1 - \epsilon)$  we go on trying with faster sending rate by setting  $O_{MIN} = O_{MID}$ . However, if the ratio exceeds unity, we know we have sent more than the available capacity and we slow down our sending rate by setting  $O_{MAX} = O_{MID}$ .

There is also a possibility of the received dispersion less than the sending dispersion; resulting in capacity over estimation. This phenomenon is explained more clearly by Kapoor et. al [13]. This is experienced when the ratio  $R_m/R_o < 1 - \epsilon$  (the convergence parameter). In such a situation we set  $O_{MAX} = O_{MID}$  to reduce our sending rate to a new value which is half of the previous value. The algorithm converges when the difference  $O_{MAX} - O_{MIN}$  is less than the granularity parameter  $\theta$ .

The following algorithm summarizes the procedure to compute available capacity

1. Start with a range of sending rate 0 bps -  $C$  bps (the end-to-end bottleneck capacity) ( $O_{MIN} = 0$  bps and  $O_{MAX} = C$  bps).
2. If  $|O_{MAX} - O_{MIN}| < \theta$  then report  $A = (O_{MIN} + O_{MAX})/2$  and stop the search procedure
3. Perform the experiment by sending the train of packet pair with appropriate sending rate and compute  $O/M = R_m/R_o$ ; ( $R_o$  = end - to - end dispersion of the train at the sender-side;  $R_m$  = end - to - end dispersion of the train at the receiver-side)
4. If  $((R_m/R_o) \geq 1 - \epsilon)$  and  $((R_m/R_o) \leq 1)$  (sending rate is within available capacity and we can try sending faster), set  $O_{MIN} = O_{MID}$  and go back to step 2
5. Else if  $(R_m/R_o) > 1$  (we are above the available capacity and hence need to back off), set  $O_{MAX} = O_{MID}$  and go back to step 2
6. Else if  $((R_m/R_o) < 1 - \epsilon)$  (we are above the available capacity and hence need to decrease our send-

ing rate  $O$ ), set  $O_{MAX} = O_{MID}$  and go back to step 2

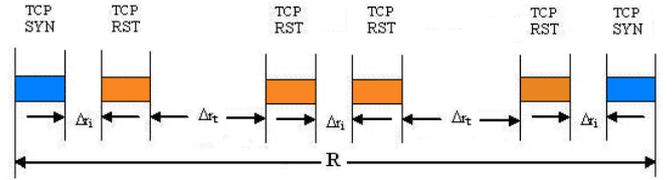
This is what we had selected to “tune” LinkWidth’s convergence conditions, based upon the results when we measured the end-to-end available capacity with a IPERF, which sets up an end-to-end TCP connection. We know we have sent more than the available capacity and we slowdown the sending rate by setting  $O_{MAX} = O_{MID}$ .

This is in accordance to the theoretical background described in [2]; sender and receiver dispersion is almost the same as long as the sender sends within the available capacity. Thus, ideally, the ratio should be exactly one as long as the sender sends within available capacity. However, we never get a ratio of exactly one due to errors in our time computation (inherent to Intel x86 platform). Since, we don’t have any knowledge of the cross-traffic, we needed some figures to determine what values could we approximate as “unity” for the purpose of our making the decision for the binary search (so as to decrease or increase the sending rate by half). Thus we “tuned” the convergence parameters by correlating with the results which we achieved by running IPERF. The real Internet traffic emulation being anyways a hard problem with the existing hardware and software within a lab environment, we required this “tuning” of LinkWidth to determine correctly the installed and available cross traffic for the emulated/shaped link capacities and the crude Internet workload generation we used. When running LinkWidth Accross hosts connected to the Internet, we required tuning the convergence parameters  $\epsilon$  and  $\theta$  with some idea of what to expect as the installed and available capacity (when compared with IPERF). As presented in the section 4, when the ratio lies in the range of 0.9000 to 0.9999 we assume that we are sending within our available capacity. This is also proven with correlating to what we achieve with IPERF. The  $\epsilon$  parameter for such experiments was set to 0.1. Thus, as long as the ratio is less than one and more than  $1 - \epsilon$  we assume we are within the available capacity and hence go on increasing the sending rate.

### 3.5 Implementation Methodology

The implementation methodology for implementing LinkWidth is simple. It involves merging the two techniques described above, namely the TCP based RPT with the TCP oriented TOPP implementation.

LinkWidth is implemented from an existing implementation of PATHNECK [8]. UDP / ICMP TTL Expired Packets used for measuring the end-to-end dispersion of the Train of Packet Pair at the destination is replaced with a TCP based implemented. The head measurement packets, the load packets and the tail measurement packets are such as that described earlier (Fig-



**Figure 6: Arrangement of Packets in TOPP:TCP SYN packet used as head measurement packet;Intermediate TCP RST load packets;TCP SYN packet used as tail measurement packets**

ure 5). The head measurement packets are so arranged such that the first head packet and the first tail measurement packet are destined to the first hop in the path. The second head and the second tail packet are destined to the second host and so on till the end of the train. The head measurement packets are sent to even port numbers while the tail measurement packets are sent to the next consecutive odd ports. This is as per the description of TCP based RPT (subsection 3.3).

The TCP based RPT procedure gives us a measure of the end-to-end installed/bottleneck capacity  $C$ . This estimation of  $C$  is used as an upper bound in the binary search procedure used by our TCP based Train of Packet Pair method. The figure 6 described the arrangement of packets in the TCP based TOPP. Our TCP based TOPP procedure uses an iterative binary search to try out capacities in the range  $[0, C]$  and converges (by using the algorithm outlined in the previous subsection 3.4) to a value of available capacity.

The following are the input parameters that LinkWidth takes:-

- $num\_load\_pkts$  = Number of load packets for the TCP based RPT
- $\theta$  = granularity parameter (in bps)
- $\epsilon$  = convergence parameter
- $payloadSize$  = Payload Size (in Bytes)
- $npackets$  = Number of packets per train
- $\Delta r_t$  = Inter-Pair gap (in microseconds)
- Inter-Train delay (in seconds)
- Output file

#### Arrangement of Packets in the TCP Based TOPP

The figure 6 describes the arrangement of packets used by the TCP based TOPP. Here

- $R$  = End-to-end dispersion of the train

- $\Delta r_i$  = Intra pair gap for a given sending rate  $O_i$ , ( $1 \leq i \leq K$ ), assuming we converge to the available capacity within  $K$  trains
- $\Delta r_t$  = Fixed Inter pair gap for each train

Thus,

$$R = (m * \Delta r_i) + (m - 1) * \Delta r_t + 2 * m * packetSendTime$$

$m$  = number of packet pairs per train (used in TCP based TOPP;  $m = n_{packet}/2$ )

$packetSendTime$  = time to inject a packet of  $payloadSize$  bytes into the network (by the probe host). This is a function of the packet size being used for the probe and is determined anew every time LinkWidth is run.

Our implementation of LinkWidth needs to change the sending rate  $O$  depending upon the ratio  $R_m/R_o$ . Thus, we need to control the sending rate by controlling the intra-pair or inter-pair delays. We thus choose the inter-pair delay  $\Delta r_i$  to be computed anew for each train. The intra-pair delay  $\Delta r_t$  is accepted in as user input and chosen as a constant for each of the equations (it wouldn't have been possible to compute two unknowns  $\Delta r_i$  and  $\Delta r_t$  from a single equation).

### 3.5.1 Computation of Intra - Pair Gaps

From (Figure 6) above, the intra-pair gaps can be computed as follows:-

$trainLen$  = The number bytes from the end of the first packet to the end of the last packet

From the previous section we have the formula for end-to-end gap of the sending train as

$$R_o = (m * \Delta r_i) + (m - 1) * \Delta r_t + 2 * m * packetSendTime$$

or  $O = (trainLen * 8) / R_o$

Solving for  $\Delta r_i$ , given that the values of the other variables are known and with the approximation  $m - 1 \approx m$  we get

$$\Delta r_i = (trainLen * 8) / (O * m) - (\Delta r_t + 2 * packetSendTime)$$

Thus, for each train  $\Delta r_i$  value is computed anew.  $O$  varies between  $O_{MIN}(0)$  and  $O_{MAX}$  ( $C$  - the bottleneck capacity). The sending train rate is varied by varying this intra-pair gaps ( $\Delta r_i$ ) while the value of inter-pair gap  $\Delta r_t$  stays fixed (It is accepted as an input from the user program).

### Controlling the Inter-Pair and Intr-Pair Gaps in Real-Time

It is essential to introduce precise delays to implement the Inter-Pair and Intra-Pair gaps for sending pack-

ets in real-time (for accurately controlling the sending rates). For ensuring that LinkWidth sends out packets with precise inter-packet/intra-packet delays, we use the system call `nanosleep()`. The process scheduling is switched to real-time SCHED\_FIFO with highest scheduling priority. This essentially gives maximum CPU time slices to the process; not achievable with the conventional SCHED\_OTHER non-real-time Round Robin scheduling. Small delays of less than 2 microseconds are thus implemented as busy waiting loops. This technique works correctly in case of Linux 2.4. Linux 2.6 actually switches a task to sleep state and can cause a dispatch latency which can as high as 10 microseconds; hence failing to ensure real-time delays [25]. *The high-res timers feature (CONFIG\_HIGH\_RES\_TIMERS) enables POSIX timers and nanosleep() to be as accurate as the hardware allows (around 1 usec on typical hardware). This feature is transparent - if enabled it just makes these timers much more accurate than the current HZ resolution[26].*

Thus we have presented the entire picture of how LinkWidth measures installed and available capacity using a variant of RPT, used originally in Pathneck, and thereby uses this result and a single-end point controlled TCP variant of TOPP to estimate the available capacity.

## 4. EXPERIMENTS AND RESULTS

LinkWidth has been designed to determine the installed and available capacity to any host connected from only one host in the network. The objective is to determine the installed and available capacity to a host, especially having low access link capacity (ideally an individual user connected to the Internet through a DSL / Cable ISP Link to the Internet). To emulate such a link, we set up an in-lab testbed (Figure 7). Here the dashed and double-headed arrow represents the traffic shaped link (used to emulate the slow access link). The probing host, H3, runs a copy of LinkWidth to determine the available capacity to the host which can be seen as a crude approximation of the end host. There is also a host, H1, which generates the cross traffic (viz. CBR-UDP and elastic HTTP cross traffic to H2). For measuring the Installed and Available Capacity of across hosts connected across a Wide Area Network we run LinkWidth and other known tools to probe for installed and available capacity to hosts connected to the Interest (Figure 8) and to the PlanetLab Network [1]. The next two subsections describe in detail the installed and available capacity estimation over an in-lab test-bed and across hosts connected to the Internet and to the PlanetLab Network.

### 4.1 Network Topology and Experimental Setup for Lab Experiments

The following topology (Figure 7) was used for the in-lab experiments. Our motivation is to measure the Installed and Available Capacity of the traffic shaped slow access link, shown using the dashed and double-headed arrow (connecting H2 to router R2), from the probe host H3, in the absence / presence of cross traffic generated by host H1 connected to the network using a fast 100Mbps Fast Ethernet Link.

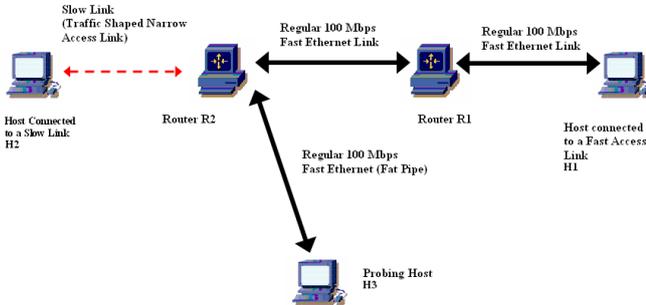


Figure 7: In Lab Test Bed : Used for measuring the installed and available capacity of the slow link connecting H2 to router R2

#### Test bed configuration

**CPU:** Intel Celeron 2.0 Ghz.

**OS:** Linux 2.4/Linux 2.6.17 (patched with Linux high resolution timer)

**Packages:** RedHat 9.0 / Fedora Core 5

**Network Interface Card:** Integrated 10/100 Ethernet Adaptors

**Network Link Emulation:** Nistnet [16] (for emulating the slow access link we used nistnet to traffic shape a 100 Mbps ethernet link to the required link capacity)

Figure 7 describes how to experimental testbed was setup. Host H1 generates cross traffic to the host H2 which is connected to the rest of the network through a traffic shaped slow link (shown through the dashed and double-headed arrow connecting H2 to the router R2). Router R2 emulates a slow access link using Nistnet [16] network link emulation program. The probe host H3 runs a copy of LinkWidth and various other installed / available capacity estimation tools. The operating system used on all the machines was Linux 2.6.15. The probe host ran both Linux 2.4 as well as Linux 2.6.17 patched with the *Linux High Resolution Timer* [26] .

#### 4.2 Network Topology and Experimental Setup for Hosts Connected Across the Internet

The topology (Figure 8) was used for the measuring the capacity available capacity of hosts connected across the Internet. Basically the measurement involved measuring the installed and available capacity of the end-to-end path connecting hosts across real Internet using LinkWidth. The idea is to validate the results we ob-

tain by running LinkWidth against what we get when we run other tools (IPERF and PATHCHAR).

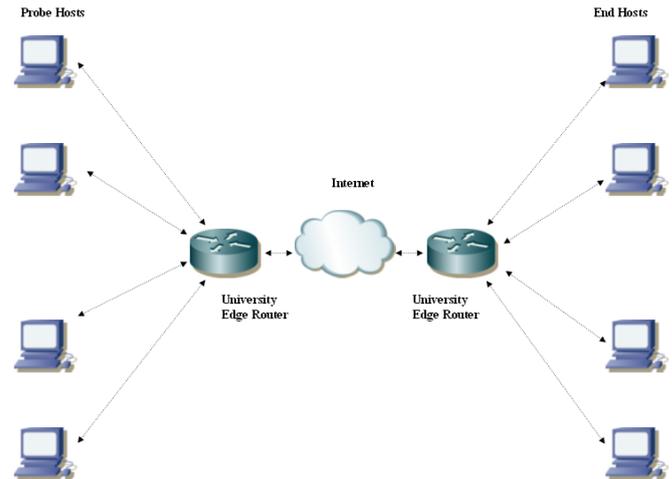


Figure 8: Measuring Capacity of Hosts Connected to the Internet

#### 4.3 Experiments and Results

The experiments for measuring installed and available capacity were performed both in an in-lab setup (where we could control parameters such as bottleneck link capacity and cross traffic rate) as well as across hosts connected to the Internet (where such parameters couldn't be controlled).

##### Results of Measuring Installed and Available Capacity

For the in-lab experiments we used the topology as shown in Figure 7 as described in previous subsection (subsection 4.1). The measurement of installed and available capacity involved generation of non-elastic Constant Bit-Rate(CBR) UDP traffic and elastic HTTP traffic.

**Note: All measurements units used in this paper are Mega Bits Per Second (Mbps) where 1 Mbps =  $1 \times 10^6$  bits.**

<sup>1</sup>I=Installed Capacity/A=Available Capacity

<sup>2</sup>The first result is obtained with Linux 2.4 while the second is obtained with Linux 2.6.17 patched with the high precision timer patched. Very evidently, using the high precision timer patch, we can send packets back-to-back with least possible delays; while in case of measurement of available capacity, we cannot generate the precise inter-packet delays less than 1 usec, which prevents us from sending the packet at the full capacity by decreasing the inter-packet gaps to small non-zero values. Evidently, this is an upper bound of the smallest possible delays which prevents us from controlling sending rates faster than about 78 Mbps (even in the absence of cross traffic)

<sup>4</sup>U=Available capacity for uploading to server/D=Available capacity for downloading from server

CBR/UDP Cross Traffic Rate(Mbps)	0	20	40	50
LinkWidth	88(I)/87(A) <sup>1</sup> 90(I)/78(A) <sup>2</sup>	66(I)/63(A)	48(I)/45(A)	43(I)/40(A)
Iperf	82	0.50-40	5-38	6-15
Bing	83	88	90	70
Pchar	82	68	45	38
Pathchar	80	62	45	32
Clink	78	48	47	43
Pathrate	98	N/A	N/A	N/A <sup>3</sup>
PathLoad	95	87-90	64.20 - 92.40	0-3.3 <sup>5</sup>
PathChirp	73	65	66.5	66.5
abget	30-40U/50-60D <sup>4</sup>	20-30U/0-20D	10-20U/20-40D	40-50U/20-50D

Table 1: Installed and Available Capacity for CBR/UDP Cross Traffic (Units : Mbps)

The CBR-UDP traffic was generated using **Real-time UDP Data Emitter (RUDE)** and **Collector of RUDE (CRUDE)** [12]. To generate TCP workloads, we used **HTTPERF** [5]. The slow link (shown as the dashed and double-headed link in Figure 7) was emulated using Nistnet. To measure the installed and available capacity to H2 from R2 (over the slow access link) in the presence of cross-traffic, we ran LinkWidth and some other tools viz. IPERF, pathchar, pathload, between the probing host H3 and H2.

We thus present in table 4.3 the results obtained from measurement of installed and available capacity of the slow link using the various tools and by varying the CBR/UDP cross-traffic flowing over the slow access link connecting R2 to host H2. As evident from the results, what we measure as the installed capacity is the actually the un-utilized capacity of the link. Unlike reactive TCP traffic, CBR/UDP is aggressive and doesn't slow down in the presence of cross-traffic. Thus we are never able to achieve the full capacity of the link in case of aggressive CBR/UDP cross traffic. Thus what we measure as the installed capacity is actually the un-utilized link capacity (available capacity) and hence what we measure as the available capacity is almost the installed capacity.

The same experiment is repeated with decreasing the link capacity of the slow access link connecting H2 to R2 to 10 Mbps. The table 2 presents the results from the experiment. In both cases, part of the link capacity being used up by CBR/UDP cross traffic can never be used (the CBR/UDP cross-traffic is as aggressive and opportunistic as the probe traffic).

Next, we present results obtained from generating elastic TCP cross-traffic. For this we generate two different kinds of workloads, one using wget (this results in single, long lived TCP connections for large files) and the using HTTPERF (which gives multiple, simultaneous short lived ones). In each case the web server

is run on the host H2 while the client programs (wget and httperf) are run on the host H1. The probe host H3 is used for measuring the installed and available capacity of the traffic shaped slow link connecting H2 to the router R2. The following table 4.3 presents results obtained from measuring installed and available capacity in the presence of elastic long lived connections (achieved by running many single-threaded wget from H1 to H2 over the traffic shaped bottleneck link connecting R2 to H2).

The TCP connection setup by wget tries to achieve the maximum capacity. However, the presence of probing cross traffic from the tools mentioned in table 4.3, would cause the packets to be queued (further may also cause them to be dropped); thereby causing TCP to reduce its sending rate. Thus, this lower sending rate would make room for the probe traffic and hence the available capacity achieved is the one available to the probe traffic in the presence of elastic wget traffic. TCP traffic being non-aggressive and elastic, makes room for the aggressive probe traffic (in case of tools like LinkWidth which doesn't set up a true end-to-end TCP connection) or shares the bandwidth equally with the end-to-end TCP probe traffic of tools like IPERF, Pathchirp etc. LinkWidth does take into account the fact that the received dispersion  $R_m$  is more than the sending dispersion  $R_o$  (in case the aggressive sender is sending more than the end-to-end available capacity), thereby halving the sending rate  $O_i$ . *Even then, the criteria we use to decrease the sending rate in LinkWidth is more aggressive and greedy than slow start and multiplicative decrease used by plain vanilla TCP.* This cause LinkWidth to grab a larger share of the end-to-end installed capacity (resulting in a slight over-estimation of the available capacity). However in later subsections, we see that the LinkWidth in fact under-estimates the available capacity in the presence of asymmetric and lossy link(s) (just to avoid over-estimation

CBR/UDP Cross Traffic(Mbps)	0	2	5
LinkWidth	9.6(I)/9.5(A)	9.3(I)/7.4(A)	4.6(I)/3.2(A)
Iperf	8.5.1	6.2	3.53
Bing	10	7.5	5.8
Pchar	10	N/A	N/A <sup>3</sup>
Pathchar	10	9.1	6.7
Clink	10	7.2	5.9
Pathrate	9.6	9.4-9.6	9.1-9.5
PathLoad	9.2	6.2	0-3.4
PathChirp	10	7-10	9.3-9.5
abget	10-90 U/0-10 D	10-90 U/0-10 D	10-90 U/0-10 D

**Table 2: Installed and Available Capacity for CBR/UDP Cross Traffic Rate for Lower Bottleneck Capacity (10Mbps) (Units : Mbps)**

Bottleneck Link Capacity(Mbps)	10	30	70	100
LinkWidth	9.6(I)/5.5(A)	30(I)/24(A)	70(I)/58(A)	82(I)/75(A)
Iperf	6.2	26	29-57	38-67
Bing	10	67	95	97
Pathchar	9.8	56	66	74
Clink	8.2	65	78	82
Pathrate	9.2	N/A	N/A	N/A <sup>3</sup>
Pathload	7.2	22	50-80	85
Pathchirp	7-12	30-60	54	62.5
abget	10-90 U/0-10 D	0-10 U/60-70 D	0-10 U/10-70 D	0-20 U/30-70 D

**Table 3: Installed and Available Capacity Estimation in the Presence of Long-Lived WGET Workloads (Units : Mbps)**

resulting from the measured  $R_m$  lower than the actual value of  $R_m$ .

These results however don't clearly show how much is the actual available/used capacity of the path is and presence of both probe traffic and cross traffic. It only shows how much the probe traffic is able to achieve, with no concrete evidence of the of our hypothesis that what is achieved is actually a share of the bandwidth (where one of the share is for our probe traffic while the other share is due to the cross traffic).

Experiments where the cross-traffic is due to multiple simultaneous TCP connections, where the capacity is equally shared equally (best effort) amongst all multiple (simultaneous connections) and where the probability of the probe traffic to be successfully sent (without transmission errors and re-ordering) is equally likely as that of the TCP cross-traffic, would give an accurate measure of the available capacity. Moreover, we are trying to achieve a situation where our probe trains are aggressive; while at the same time reactive enough to slow down the sending rate upon reaching a rate where the end-to-end dispersion of the receiver is not more than the sending dispersion by a factor of  $\epsilon$ . In such a situation the end-to-end dispersion of the packet train

(or train of packet pairs) should be same with which it is sent out from the sender ( $R_m/R_o \approx 1 \pm \epsilon$ ). Thus we used HTTPERF to generate multiple simultaneous connections over the (traffic shaped slow link) path from the H1 to H2 via R2. We probed the link from H3 to H2, using LinkWidth, IPERF and abget. The results of the experiment can be observed in the table 4.

As evident from table 4 in most of the cases what we achieve is only a share of the entire capacity. HTTPERF supports simultaneous connections. The number of connections at any time is controlled by the connection rate (expressed as the number of connections per second) parameter. The number of calls per connection controls the number of HTTP requests per connection (we emulate session oriented HTTP workloads). The longer the number of calls per connections, the longer the lifetime of the connections and the more aggressive they are. For N connections per second, sharing the link with capacity C, the achieved capacity of our probe should be approximately  $C/(N+1)$ . The results presented in table 4 are thus in accordance with our hypothesis. This is in fact what we achieve, and had been verified IPERF. In the process we have also evaluated the effectiveness of abget. As evident abget is not able to correctly deter-

<b>Conn./sec</b>	<b>2</b>	<b>3</b>	<b>10</b>
<b>LinkWidth</b>	2.9(I)/2.0(A)	3(I)/1.08(A)	2.8(I)/0.368(A)
<b>Iperf</b>	2.57	0.991	0.406
<b>abget</b>	0-100 U/0-10 D	0-100 U/90-100 D	0-100 U/0-100 D
<b>Conn./sec</b>	<b>2</b>	<b>3</b>	<b>6</b>
<b>LinkWidth</b>	4.8(I)/2.7(A)	5.1(I)/2.3(A)	4.95/0.864
<b>Iperf</b>	2.75	2.86	0.991
<b>abget</b>	0-100 U/70-90 D	0-100 U/70-90 D	0-100 U/10-30 D
<b>Conn./sec</b>	<b>2</b>	<b>5</b>	<b>10</b>
<b>LinkWidth</b>	8(I)/4.8(A)	9(I)/3(A)	8/0.97(A)
<b>Iperf</b>	5.49	2.3	0.862
<b>abget</b>	10-90 U/0-10 D	10-90 U/0-10 D	0-10 U/0-10 D
<b>Conn./sec</b>	<b>2</b>	<b>5</b>	<b>25</b> <sup>6</sup>
<b>LinkWidth</b>	48.6(I)/36(A)	47(I)/12(A)	37(I)/3(A)
<b>Iperf</b>	27	5.2	1.2
<b>abget</b>	0-100 U/0-100 D	0-100 U/0-100 D	0-100 U/10-90 D
<b>Conn./sec</b>	<b>1</b>	<b>4</b>	<b>10</b>
<b>LinkWidth</b>	95(I)/80(A)	94(I)/21(A)	94(I)/13(A)
<b>Iperf</b>	74	22	10
<b>abget</b>	0-100 U/0-100 D	0-100 U/0-100 D	0-100 U/0-100 D

**Table 4: Installed and Available Capacity for Short Lived Elastic HTTP Cross Traffic (Units : Mbps)**

mine the available capacity for very small link capacities which is congested with cross traffic. The installed and available capacity, as what we observed is close to what is achievable by an end-to-end TCP connection (which IPERF does).

#### Measuring Capacity and Available Capacity of Hosts Connected to the Internet and PlanetLab

There was no way to control the cross-traffic across hosts connected to the Internet. That is exactly why we tried to measure the installed and available capacity of hosts connected to the Internet using LinkWidth and verified them using IPERF (to verify available end-to-end available capacity) and pathchar (to verify the end-to-end bottleneck capacity)<sup>7</sup>. The following table 5 gives a comparison of LinkWidth, Pathchar and Iperf when measuring the installed and available capacity to three separate destinations. The first two are two hosts in two different universities connected to the Internet. The third is a privately owned computer connected to the Internet through a local ISP. All these hosts were probed from a host within our university’s Local Area Network.

These experiments were repeated by probing various geographically dispersed hosts, connected to the PlanetLab network. The results are presented in table 6.

Our results are quite close to those obtained from using IPERF (which creates an end-to-end TCP connection). Pathload, on the other hand, apparently overestimates the results in the presence of uncontrolled

<sup>9</sup>UL=Uplink Capacity DL=Downlink Capacity

<b>Tool Used</b>	<b>LinkWidth</b>	<b>Pathchar</b>	<b>Iperf</b>
Host 1	62(I)/0.3(A)	42	0.5
Host 2	56(I)/1.03(A)	35	0.6
Host 3	5(I)/3.4(A)	6	5.2

**Table 5: Measuring Installed and Available Capacity of Geographically Dispersed Hosts Connected to the Internet (Units : Mbps)**

cross-traffic. When probing using LinkWidth in debug mode (with verbose output), we observed lot of packet losses and re-ordering. This is due to the fact that the access link in this case is asymmetric (allowing faster downloads than uploads, which causes the inter-packet delay of the reply probe packets to be skewed, resulting in incorrect estimation of the received dispersion). Our measurement technique cannot account for error thus introduced in our estimate of installed and available capacity. Moreover, other unknown/unpredictable factors such as host state/activity and network cross-traffic patterns, makes it difficult to pin point if this variation in the inter-packet delay is due to the asymmetric properties of the access link/network or due to the host/network being probed. This variation of the inter-packet delays results in incorrect estimation of the received dispersion  $R_m$ .

<sup>5</sup>Interrupt Coalescence Detected in Receiver NIC

<sup>3</sup>Goes on measuring without converging

<sup>6</sup>Number of calls per connection = 2

<sup>7</sup>IPERF gives an accurate estimate of the available capac-

Tool Used	Iperf	Pathload	Linkwidth	Bottleneck Link
planetlab-1.cs.princeton.edu(US)	36.5(UL)/19.5(DL) <sup>9</sup>	40	92(I)/18(A)	216.27.100.53
lefthand.eecs.harvard.edu(US)	5.45(UL)/4.94(DL)	84	94(I)/7(A)	140.247.2.62
planet1.pittsburgh.intel-research.net(US)	11(UL)/0.728(DL)	42	47(I)/12(A)	128.59.255.89
planetlab2.cis.upenn.edu(US)	23(UL)/30(DL)	97	80(I)/18(A)	199.109.4.13
planet3.berkeley.intel-research.net(US)	2.6(UL)/0.707(DL)	19	10(I)/3(A)	128.59.255.14
planet2.cc.gt.atl.ga.us(US)	9.5(UL)/9.5(DL)	95	76(I)/14(A)	143.215.193.9
planetlab2.cs.dartmouth.ed(US)	0.25(UL)/0.2(DL)	92	85(I)/0.15(A)	192.5.89.218
planetlab2.xeno.cl.cam.ac.uk(EUR)	2.59(UL)/0.12(DL)	96	93(I)/19(A)	128.232.103.202
planetlab-1.fokus.fraunhofer.de(EUR)	0.2(UL)/1.98(DL)	0.8	39(I)/0.172(A)	199.109.4.13
onelab1.inria.fr(EUR)	1.94(UL)/1.92(DL)	86-97	20(I)/3(A)	138.96.250.190
supernova.ani.univie.ac.at(EUR)	2.13(UL)/0.7(DL)	0	38(I)/1.1(A)	131.130.32.152
planetlab2.tmit.bme.hu(EUR)	2.20(UL)/2.18(DL)	>94 <sup>5</sup>	84(I)/2(A)	152.66.244.49
planetlab-1.man.poznan.pl(EUR)	2.15(UL)/2.15(DL)	98	6.5(I)/2.1(A)	150.254.210.61
csplanetlab1.kaist.ac.kr(ASIA)	1.4(UL)/1.2(DL)	6	68(I)/1(A)	199.109.7.13
ds-pl3.technion.ac.il(ASIA)	1.4(UL)/1.47(DL)	97	70(I)/2.8(A)	128.139.233.2
sjtu2.6planetlab.edu.cn(ASIA)	8(UL)/2(DL)	>17.38 <sup>5</sup>	75.5(I)/15(A)	202.112.61.13

**Table 6: Measuring Available Capacity of Geographically Dispersed Hosts Connected to the Planet-Lab Network (Units : Mbps)**

## 5. CONCLUSION

In this paper we described LinkWidth, a single-end controlled tool to measure the installed/bottleneck capacity and the available/un-utilized bandwidth. We give implementation details of how we extended the two existing techniques (Recursive Packet Train and Train of Packet Pair), to employ TCP RST packets sandwiched between TCP SYN packets. In addition, we show how to use a binary search approach to estimate installed/bottleneck and available/un-utilized capacity through a single tool.

Our experimental results for both installed and available capacity in various scenarios indicate that LinkWidth gives a good estimate of the installed and available capacity when compared to well accepted tools like PATHCHAR and IPERF. However, we cannot accurately measure links that exhibit packet loss or are assymmetric which seem to require a two-ended measurement tool. In lab experiments using CISCO routers, the RPT packet train method reports very accurate measurements of bottleneck capacity. Small cross-traffic packets do not introduce significant change to the end-to-end dispersion of the train. This decreases our error due to capacity underestimation or overestimation (otherwise prevalent in the older packet pair methods). Moreover, use of symmetric links/paths ensures that the perceived value of received dispersion is not very different from the correct value.

ity available for a TCP as it sets up a client-server connection and tries the send at best effort capacity. Pathchar, although takes long to converge, closely estimates the installed/bottleneck capacity under various scenarios of link capacities and cross-traffic rate

When we deploy our tool in the Internet from a single vantage point, we have no knowledge of what cross-traffic/link conditions to expect at the other end of the path. The only way our variant of Train of Packet Pair tries to react to congestion due to cross-traffic, is by observing the sending and “perceived” reception rate of the train of packet pairs. Although not as accurate as IPERF a two-end controlled tool, we are able to achieve a good estimate of available capacity/bandwidth. To optimize our estimates and to converge faster, we can use knowledge of the link cross-traffic and available bandwidth to “tune” some of our tool. However, given the fact that our tool is not using any congestion control, we appear to be more aggressive in measuring available capacity and thus get higher values when compared with tools that use regular two-end TCP connections to perform the same measurements.

## 6. REFERENCES

- [1] “planetlab”. <http://www.planet-lab.org/>.
- [2] B.Melander, M.Bjorkman, and P.Gunningberg. New end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Proceedings of GLOBECOM 2000*, November 2000.
- [3] C.Dovrolis, P.Ramanathan, and D.Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE Transactions on Networking*, December 2004.
- [4] D.Antoniades, M.Athanatos, A.Papadogiannakis, E.P.Markatos, and C.Dovrolis. Available bandwidth measurement as simple as running wget. In *Proceedings of Passive and Active Measurements*, March 2006.

- [5] D.Mosberger and J.Tai. HTTPERF. <http://www.hpl.hp.com/research/linux/httpperf/>, 1998.
- [6] A. Downey. CLINK. <http://www.kitchenlab.org/www/bmah/Software/pchar>, 1999.
- [7] A. Downey. Using pathchar to estimate internet link characteristics. In *Proceedings of ACM SIGCOMM 1999*, August 1999.
- [8] N. Hu, L. E. Li, Z.M.Mao, P. Steenkiste, and J. Wang. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput. In *Proceedings of ACM SIGCOMM 2004*, August 2004.
- [9] N. Hu, L. E. Li, Z.M.Mao, P. Steenkiste, and J. Wang. PATHNECK. <http://allendowney.com/research/clink/clink.1.0.tar.gz>, 2004.
- [10] V. Jacobson. PATHCHAR. <http://www.caida.org/tools/utilities/others/pathchar/>, 1997.
- [11] M. Jain and C. Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput, 2002.
- [12] J.Laine, S.Saaristo, and R.Prior. RUDE and CRUDE. <http://rude.sourceforge.net/>, 1999.
- [13] R. Kapoor, L. Chen, L. Lao, M. Gerla, and M. Sanadidi. Capprobe: A simple technique to measure path capacity. In *Proceedings of ACM SIGMETRICS 2004*, August 2004.
- [14] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proceedings of ACM SIGCOMM*, pages 61–72, August 2002.
- [15] B. Mah. PCHAR. <http://www.kitchenlab.org/www/bmah/Software/pchar>, 1997.
- [16] M.Carson and D.Santay. NISTNet-A Linux-based Network Emulation Tool. <http://www-x.antd.nist.gov/nistnet/nistnet.pdf>, 2003.
- [17] M.Jain and C.Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput. In *Proceedings of ACM SIGCOMM 2002*, August 2002.
- [18] M.Jain and C.Dovrolis. “pathload a measurement tool for end-to-end available bandwidth”. <http://www-static.cc.gatech.edu/~jain/talk/pam02.ppt>, 2002.
- [19] N.Hu and P.Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, August 2003.
- [20] A. Pasztor and D. Veitch. The packet size dependence of packet pair like methods. In *In IEEE/IFIP International Workshop on Quality of Service (IWQoS)*, 2002.
- [21] P.Beyssac. BING. [http://fgouget.free.fr/bing/bing\\_src-1.3.5.tar.gz](http://fgouget.free.fr/bing/bing_src-1.3.5.tar.gz).
- [22] R. Prasad, M.Murray, C. Dovrolis, and K. Claffy. Bandwidth estimation:metrics, measurement techniques, and tools. In *Proceedings of IEEE Network 2003*, August 2002.
- [23] S.Keshav. “congestion control in computer networks”. UC Berkely Technical Report TR-654, September 1991.
- [24] A. Stavrou and A. Keromytis. Countering DoS Attacks With Stateless Multipath Overlays. In *Proceedings of the 12<sup>th</sup> ACM Conference on Computer and Communications Security (CCS)*, pages 249–259, November 2005.
- [25] T.Gleixner and I.Molnar. Linux Programmer’s Manual. Linux Man Page Section 2 : System Call nanosleep().
- [26] T.Gleixner and I.Molnar. Linux High Resolution and Tickless Timers. <http://kerneltrap.org/node/6750/>, 2006.
- [27] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. IPERF. <http://dast.nlanr.net/projects/Iperf/>, 1997.
- [28] V.Ribeiro, R.Riedi, R.Baraniuk, J.Navratil, and L.Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Passive and Active Measurements Workshop 2003*, 2003.