pathChirp: A Light-Weight Available Bandwidth Estimation Tool for Network-Aware Applications

Vinay J. Ribeiro,[†] Rudolf H. Riedi,[†] Richard G. Baraniuk[†] Jiri Navratil,[‡] Les Cottrell[‡]
[†] Department of Electrical and Computer Engineering, Rice University
[‡] SLAC/SCS-Network Monitoring, Stanford University

Abstract— **Many grid computing applications require massive data transfers for distributed analysis. Knowledge of network properties like the available bandwidth along the communication paths between grid nodes can reduce data transfer times and improve scheduling of grid computing tasks thereby enhancing performance.**

This paper presents *pathChirp*, a light-weight active probing tool for estimating the available bandwidth on a communication network path. Based on the concept of "self-induced congestion," pathChirp features an exponential flight pattern of probes we call a chirp. By rapidly increasing the probing rate within each chirp, pathChirp obtains a rich set of information from which to dynamically estimate the available bandwidth. PathChirp does not require synchronous nor highly stable clocks at the sender and receiver. Simulations and Internet experiments show that pathChirp provides good estimates of the available bandwidth while using only a fraction of the number of probe bytes that current state-of-the-art techniques use.

Keywords: probing, networks, bandwidth, available bandwidth, chirp, end-to-end, inference

I. Introduction

Inferring the unused capacity or *available bandwidth* on network paths is of great importance for several performance critical applications. A classic example is *grid computing* for high-energy physics data analysis (see Figure 1). The sheer volume of data generated in particle accelerator experiments often renders the computing resources at a single research institution insufficient. Take the case of the Large Hadron Collider being constructed at CERN that is expected to generate 12–14 Peta bytes (1 Peta byte=10^{15} bytes) of data which will require the equivalent of 70,000 of today's fastest PC processors for analysis [1]! Grid computing alleviates the resource scarcity problem by harnessing the computing power of multiple institutions around the globe. Since the grid must transfer huge volumes of data across the network, the efficient use of network resources like the bandwidth becomes critical for performance. Other applications that could benefit from knowledge of the available bandwidth are rate-based streaming applications [2], end-to-end admission control [3], server selection [4], optimal route selection in overlay networks [5], congestion control [6], as well as service level agreement verification [7].

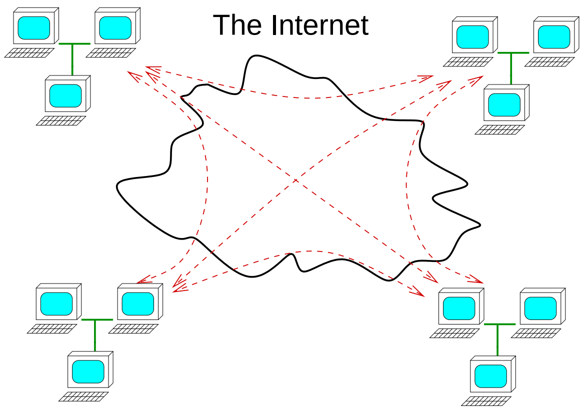


Fig. 1. *By harnessing computational resources across the global Internet, grid computing makes massive data processing tasks feasible. Knowledge of the available bandwidth between grid nodes helps optimize performance.*

Obtaining useful estimates of the available bandwidth from routers is often not possible due to various technical and privacy issues or due to an insufficient level of measurement resolution or accuracy. Thus, it becomes necessary to infer the required information from the network edge via an active or passive probing scheme. Ideally, a probing scheme should provide an accurate estimate of a path's available bandwidth in as short a time as possible, preferably over only a few round trip times (RTTs), while imposing as light a load as possible on the network.

Current schemes for available bandwidth estimation fall into two broad classes. The first class of schemes is based on statistical cross-traffic models, such as *Delphi* [8] and the methods proposed in [9]. While these schemes can potentially provide accurate estimates of

This work was supported by DoE SciDAC grant DE-FC02-01ER25462, NSF grant ANI–0099148, NSF grant ANI–9979465, and the Texas Instruments Leadership University Program. Email: {vinay, riedi, richb}@rice.edu, {jiri,cottrell}@slac.stanford.edu. Web: spin.rice.edu, www-iepm.slac.stanford.edu.

cross-traffic, to date they have been designed for *single hop* (bottleneck) scenarios and as such may not be robust in multi-hop networks which are more common in practice.

The second class of schemes is based on the concept of *self-induced congestion*, which relies on a simple heuristic: If the probing rate exceeds the available bandwidth over the path, then the probe packets become queued at some router, resulting in an increased transfer time. On the other hand, if the probing rate is below the available bandwidth, the packets face no queuing delay. The available bandwidth can then be estimated as the probing rate at the onset of congestion. Such schemes are equally suited to single and multiple hop paths, since they rely only on whether the probe packets make it across the path with an unusual delay or not.

Two examples of the self-induced congestion approach are *pathload* [10] and *TOPP* [11]. Pathload employs long constant bit-rate (CBR) packet trains and adaptively varies the rates of successive packet trains in an effort to converge to the available bandwidth rate. Because of its adaptive search, pathload can have long convergence times (on the order of 100s of RTTs) [10] and use several MB of probe traffic per estimate. TOPP [11] uses packet pairs of different spacings well-separated in time and estimates available bandwidth from the time-averaged spacing of packets at the receiver. As a result TOPP does not make use of delay *correlation* information obtainable from packet trains with closely spaced packets.

In this paper, we propose a new self-induced congestion available bandwidth estimation scheme we call *pathChirp*. Unique to pathChirp is an exponentially spaced *chirp probing train* (see Fig. 2). Chirp trains are highly efficient. First a chirp of N packets has $N-1$ packet spacings that would normally require $2N-2$ packets using packet pairs. Second by exponentially increasing the packet spacing, chirps probe the network over the range of rates $[G_1, G_2]$Mbps using just $\log(G_2) - \log(G_1)$ packets. One other advantage of chirps (or any other packet train) over packet pairs is that they capture critical delay correlation information that packet pairs do not. PathChirp exploits these advantageous properties of chirps to rapidly estimate available bandwidth using few packets.

To avoid confusion, we emphasize that the only commonality between pathChirp and our Delphi algorithm [8] is the chirping packet train. Delphi uses chirps to estimate the available bandwidth at a range of different time scales based on a multifractal tree model for the bandwidth over time. It does not use the self-induced congestion principle. Unlike Delphi, pathChirp does not use a statistical model.

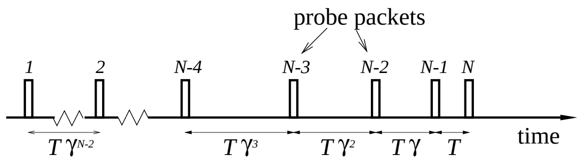


Fig. 2. *Chirp probe train, with exponential packet flight pattern.*

We present the pathChirp concept and algorithm in Section II and analyze its performance as a function of its parameters in Section III. After testing pathChirp in multi-hop scenarios in Section IV we compare it to TOPP and pathload in Sections V and VI respectively. Section VII overviews experiments on the real Internet. We conclude in Section IX with a discussion and directions for future research. The pathChirp tool is available as open-source freeware at spin.rice.edu/Software/pathChirp.

II. pathChirp

In this paper we are concerned with a single sender – single receiver path of a communication network. We explicitly permit multiple queues; to this end we model a path as a series of store-and-forward nodes each with its own constant service rate, equipped with FIFO queues. This is an accurate model for today's Internet. We focus on estimating the available bandwidth over the path based on queuing delays of probe packets transmitted from the sender to the receiver. We use information only on the *relative* delays between probe packets; this allows us to not require clock synchronization between the sender and receiver.

A. Available bandwidth

Denote the capacity of the output queue of router node i as C_i, and the total traffic (other than probes) entering it between times a and b as $A_i[a,b]$. Define the path's *available bandwidth* in time interval $[t-\tau, t]$ as

$$B[t-\tau,t] = \min_i \left(C_i - \frac{A_i[t-\tau+p_i, t+p_i]}{\tau} \right), \quad (1)$$

where p_i is the minimum time a packet sent from the sender could take to reach router i. The delay p_i includes the speed-of-light propagation delay and packet service times[1] at intermediate queues.

In reality probe packets suffer queuing delays in addition to the minimum delay p_i. Thus probes transmitted

[1]We assume a constant probe packet size which implies a constant packet service time.

during $[t-\tau, t]$ can arrive at router i outside time interval $[t-\tau+p_i, t+p_i]$ and do not exactly measure $B[t-\tau, t]$. For large τ ($\gg$ RTT), however, the effect of queuing delay becomes inconsequential.

B. pathChirp overview

PathChirp estimates the available bandwidth along a path by launching a number of packet chirps (numbered $m = 1, 2, \ldots$) from sender to receiver and then conducting a statistical analysis at the receiver.

First some notation for chirps (see Fig. 2). Consider chirp m consisting of N exponentially spaced packets, each of size P bytes. Define the ratio of successive packet inter-spacing times within a chirp as the *spread factor* γ, the queuing delay of packet k as $q_k^{(m)}$,[2] the sender transmission time of packet k as $t_k^{(m)}$, the inter-spacing time between packets k and $k+1$ as $\Delta_k^{(m)}$, and the instantaneous chirp rate at packet k as

$$R_k^{(m)} = P/\Delta_k^{(m)}. \tag{2}$$

Since $\Delta_k^{(m)}$ and $R_k^{(m)}$ are the same for all chirps, we drop their superscripts in the subsequent discussion.

In a CBR fluid cross-traffic scenario, we have

$$\begin{aligned} q_k^{(m)} &= 0, && \text{if } B\left[t_1^{(m)}, t_N^{(m)}\right] \geq R_k \\ q_k^{(m)} &> q_{k-1}^{(m)}, && \text{otherwise} \end{aligned} \tag{3}$$

which leads to a simple estimate: $\widehat{B}\left[t_1^{(m)}, t_N^{(m)}\right] = R_{k^*}$, where k^* is the packet at which the queuing delay begins increasing.

The assumption of CBR cross-traffic clearly oversimplifies reality. In particular, due to bursty traffic, queuing delays will typically not increase monotonically within a chirp, or any probing train for that matter. Fig. 3 depicts the queuing delays of a typical chirp train. We refer to such a plot as a queuing delay *signature*. Typically a signature consists of *excursions* from the zero axis ($q_k^{(m)} > 0$ for several consecutive packets) caused by bursts of cross-traffic. The first few excursions end with the queuing delays returning to zero. This is because the chirp rate R_k is less than the bottleneck link speed ($C_{\min} := \min\{C_i\}$) on the path, which allows the queues to relax in the absence of cross-traffic. The last excursion usually ends with increasing queuing delays because $R_k > C_{\min}$, which causes the chirp packets to fill up intermediate queues.

[2]If the clocks at the sender and receiver are unsynchronized but stable, then the difference between receiver and sender time stamps is the queuing delay plus a constant.

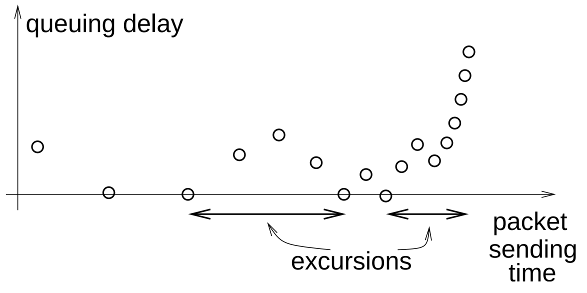


Fig. 3. *A typical chirp queuing delay signature.*

PathChirp uses the shape of the signature, to make an estimate $E_k^{(m)}$ of the *per-packet* available bandwidth $B\left[t_k^{(m)}, t_{k+1}^{(m)}\right]$. It then takes a weighted average of the $E_k^{(m)}$'s corresponding to each chirp m to obtain estimates $D^{(m)}$ of the *per-chirp* available bandwidth $B\left[t_1^{(m)}, t_N^{(m)}\right]$:

$$D^{(m)} = \frac{\sum_{k=1}^{N-1} E_k^{(m)} \Delta_k}{\sum_{k=1}^{N-1} \Delta_k}. \tag{4}$$

Finally it makes estimates $\rho[t-\tau, t]$ of the available bandwidth $B[t-\tau, t]$ by averaging the estimates $D^{(m)}$ obtained in the time interval $[t-\tau, t]$.

C. Excursion segmentation

In order to accurately compute $E_k^{(m)}$, pathChirp segments each signature into regions belonging to excursions and regions not belonging to excursions.

Based on the principle of self-induced congestion, we assume that increasing queuing delays signify less available bandwidth than the instantaneous chirp rate at that moment while decreasing delays signify the opposite, that is,

$$E_k^{(m)} \geq R_k, \qquad \text{if } \; q_k^{(m)} \geq q_{k+1}^{(m)} \tag{5}$$

$$E_k^{(m)} \leq R_k, \qquad \text{otherwise.} \tag{6}$$

In a single-hop scenario, (5) is exactly true while (6) need not always be true. For example if packets k and $k+1$ are spaced very far apart (say by 1 hour), then the fact that $q_k^{(m)} < q_{k+1}^{(m)}$ tells us little about $E_k^{(m)}$. This is because the packets k and $k+1$ cannot possibly induce congestion in the network and rather only provide independent samples of the path queuing delay.

To make correct use of (6), we segment each signature into excursion regions and apply (6) only to these regions. The basic idea behind pathChirp's excursion segmentation algorithm is quite simple. Intuitively if $q_k^{(m)}$

increases and remains larger than 0 for several consecutive packets, then it is likely that these packets are all part of the same busy period[3] at a congested queue along the path. In this case we expect $q_k^{(m)} < q_{k+1}^{(m)}$ to correspond to self-induced congestion, thus validating (6). We would thus like to find regions in the signature for which $q_k^{(m)} > 0$ for several consecutive packets.

In practice we do not necessarily know the clock offset between the end hosts running pathChirp. This combined with the machine added noise to the time stamps makes it infeasible to use $q_k^{(m)} > 0$ for excursion detection. PathChirp instead uses the *relative* queuing delay within a chirp to detect excursions. It also avoids using hard queuing delay thresholds, since the magnitude of queuing delay is heavily dependent on link speeds that vary from path to path. For example, from basic queuing theory a 10Mbps link loaded at 50% utilization by a Poisson traffic source (with constant packet size) will have a larger average queuing delay than a similarly utilized 100Mbps link fed with Poisson traffic.

The details of pathChirp's excursion segmentation algorithm are as follows. The goal is to identify potential starting and ending packet numbers i and j respectively for an excursion. Every packet i where $q_i^{(m)} < q_{i+1}^{(m)}$ is a potential starting point of an excursion. We define the end of the excursion j as the first packet where

$$q(j) - q(i) < \frac{\max_{i \le k \le j}[q(k) - q(i)]}{F}, \qquad (7)$$

where F is a parameter called the *decrease factor*. At j the queuing delay relative to $q(i)$ has decreased by a factor of F from the maximum queuing delay increase after i and up to j. If $j - i > L$, that is the signature region is long enough, then all packets between i and j form an excursion.

The last excursion of a signature usually does not terminate; that is, there is some packet l with $q_l^{(m)} < q_{l+1}^{(m)}$ such that there is no $j > l$ for which (7) holds (replacing i by l in (7)). This excursion is treated differently to the others while setting $E_k^{(m)}$ which we describe next.

D. Computing the per-packet estimates $E_k^{(m)}$

Now it only remains to compute the per-packet available bandwidth estimates $E_k^{(m)}$. Each chirp packet k falls into one of the following three categories that decide $E_k^{(m)}$.

Case (a): If k belongs to an excursion that terminates and $q_k^{(m)} \le q_{k+1}^{(m)}$, then set

$$E_k^{(m)} = R_k. \qquad (8)$$

This satisfies (6).

Case (b): If k belongs to an excursion that does not terminate, then set

$$E_k^{(m)} = R_l, \;\; \forall\, k > l, \qquad (9)$$

where l is the start of the excursion.

The reason that we do not use (8) for case (b) is that the chirp rate during this particular excursion can be much higher than $C_{\min}$. Since the available bandwidth is always less than $C_{\min}$, we must have $E_k^{(m)} < R_k$.

We note however that according to (5) we must have $E_k^{(m)} > R_k > R_l$ if $q_k^{(m)} > q_{k+1}^{(m)}, \; k > l$. Hence (9) leads to a conservative estimate of $E_k^{(m)}$ for such k.

Case (c): For all k not belonging to the above cases we set $E_k^{(m)} = R_l$. This includes all those k not belonging to excursions as well as those with decreasing queuing delay belonging to excursions. In case the last excursion of the signature does terminate, we choose $l = N - 1$.

For the pseudo-code of the pathChirp algorithm see Figs. 4 and 5. Since the pseudo code uses delay information of only a single chirp, we drop superscript (m) in all quantities.

E. Implementation details

PathChirp infers available bandwidth online using UDP chirp packet probes. PathChirp's parameters are the probe packet size P, the spread factor γ, the decrease factor F, the busy period threshold L, and the time interval τ over which the D_m instantaneous estimates are smoothed. The average probe load on the network and the range of instantaneous rates within each chirp are user specified options. PathChirp spaces the chirps apart in time to achieve the specified average probing rate. Each UDP packet carries a sender timestamp which the receiver uses along with its own local timestamp in the delay estimation process.

In pathChirp, probe packets travel one-way from sender to receiver, and the receiver performs the estimation. We prefer to not merely echo back information to the sender to avoid the problem of echo probe traffic interfering with the sender-to-receiver chirp probes. This can occur on links that are not full-duplex, for example those in shared LANs.

PathChirp addresses the practical problem of context switching. When a context switch takes place at a host receiving probe packets, the packets are temporarily

[3]A *busy period* is a time interval during which the queue is never idle.

pathChirp Algorithm

```
 procedure estimate_D($\mathbf{q}$){
    /* $\mathbf{q}$ denotes the vector of a single chirp train's
     queuing delays */
    for ($k = 1$ to $N - 1$) $E_k = 0$;  /*initialize*/
    $i = 1$; /* Denotes current packet number */
    $l = N - 1$; /* $N$=number of chirp packets*/
    while($i \leq N - 1$){
      if ($q_i < q_{i+1}$){
        $j =$ excursion($\mathbf{q}$,$i$,$F$,$L$)
        choose case($j$):
          Case(a): ($j > i$) and ($j \leq N$)
            for ($s = i$ to $j - 1$)
              if ($q_s < q_{s+1}$) $E_s = R_s$;
          Case(b): $j = N + 1$
            for ($s = i$ to $N - 1$) $E_s = R_i$;
            $l = i$;
        /* end choose case */
        if ($j = i$) $j = j + 1$;
        $i = j$;
        } /* end if */
      else
        $i = i + 1$;
    } /* end while*/
    $D = 0$;
    for ($i = 1$ to $N - 1$){  /*computing $D$*/
      if ($E_i == 0$)
        $D+ = R_l \Delta_i$; /* Case (c) */
      else
        $D+ = E_i \Delta_i$;
    }; /* end of for loop */
    $D = D / \sum_{1 \leq i \leq N-1} (\Delta_i)$;
    return $D$;
 }
```

Fig. 4. *The pathChirp algorithm*

```
 procedure excursion($\mathbf{q}$,$i$,$F$,$L$){
   $j = i + 1$;
   max_q$= 0$;
   while(($j \leq N$) and ($q(j) - q(i) >$ max_q$/F$))
     {
      max_q=maximum(max_q,$q(j) - q(i)$);
      $j = j + 1$;
     }
   if (($j \geq N$)) return $j$;
   if ($j - i \geq L$)
      return $j$;
   else
      return $i$;
 }
```

Fig. 5. *The excursion segmentation algorithm.*

buffered while the CPU handles other processes. This introduces delays between packets reaching the application layer just before the context switch and after it. In addition, the buffered packets rapidly reach the application layer after the context switch. These delays may be mistakenly construed as router queuing delays and thus corrupt pathChirp's network inference. When the difference between two consecutive receive time stamps is less than a threshold d, we detect a context switch and discard the concerned chirp. We note that a d value of 30μs is lower than the transmission time of a 1000 byte packet on an OC-3 link (50μs). Thus one would expect packet arrival times at the receiver to exceed 50μs if the last link is of OC-3 or lower speed. Currently d is hardcoded in the program. In future d will be adaptively chosen to suit the machine in question.

We are currently studying other ways of circumventing time stamp corruption due to context switching. One of them is to use time stamps generated by NIC cards rather than application layer ones.

PathChirp discards all chirps with dropped packets.

III. Performance and Parameter Choice

In this section, we use simulations to better understand the role of the various pathChirp parameters. In the experiments, we use a single queue with capacity 10Mbps fed with Poisson packet arrivals. The cross-traffic packet sizes were randomly chosen to be 1500 or 40 bytes with equal probability. Internet traffic has been shown to have weak correlations over time lags less than 100ms [12] in spite of stronger correlations (or long-range dependence (LRD)) at time lags greater than 1s. Since the duration of a chirp is typically less than 100ms a Poisson cross-traffic model which does not possess LRD suffices.

We varied the packet size P, spread factor γ, decrease factor F and busy period threshold L while keeping τ and the total probing load constant at 500kbps. Recall that we can maintain any average low probing rate by spacing the chirp trains far enough apart. Our choice for the performance metric is the mean squared error (MSE) of the estimate $\rho[0,\tau]$ normalized by the second moment of the true $B[0,\tau]$. All experiments report 90% confidence intervals.

Probe packet size P: First we assess the impact of probe packet size P on estimation performance. Obviously the number of bytes transmitted per chirp decreases with P. Thus by reducing P we can send more chirps for the same average probing rate, giving us more estimates $D^{(m)}$ per time interval τ. However from (2) we observe that for the same set of probing rates R_k, a small P results in a proportionately small Δ_k. Intuitively the cross-traffic arriving over a time interval Δ_k is more bursty for smaller Δ_k. For instance when $\Delta_k \rightarrow 0$ the cross-traffic process is far from smooth and to the contrary is a binary process: we either have one packet arriving or none at all. Thus shorter chirps will exhibit more erratic signatures and give less accurate estimates.

Fig. 6 demonstrates the effect of the probe packet size P on estimation performance. We set $\gamma = 1.2$ and vary the parameters F and L as well as the link utilization. Observe that in most cases larger P values give better performance. In a few cases in Fig. 6(a) the MSE increases slightly with P.

The results show that pathChirp generally performs better with larger packet sizes. In Internet experiments we thus use $P \geq 1000$ bytes.

Spread Factor γ: The spread factor γ controls the spectrum of probing rates in a chirp. A smaller γ leads to a dense spectrum of rates R_k, potentially increasing the accuracy of estimates $D^{(m)}$. It also leads to a finer sampling of network delay, thus potentially improving pathChirp's ability to identify excursions. However it also increases the number of packets per chirp and hence reduces the number of estimates $D^{(m)}$ per time interval τ, possibly degrading the estimate $\rho[t-\tau,t]$.

Fig. 7 demonstrates the effect of the spread factor γ on estimation performance. We observe that the MSE decreases (that is, improves) with decreasing γ. This experiment uses $P = 1300$ byte packets. Since $\gamma > 2$ can give errors as high as 100% even in CBR scenarios, we have excluded them in the experiments.

PathChirp uses $\gamma = 1.2$ by default.

Busy period threshold L and decrease factor F: The busy period threshold L and decrease factor F influence

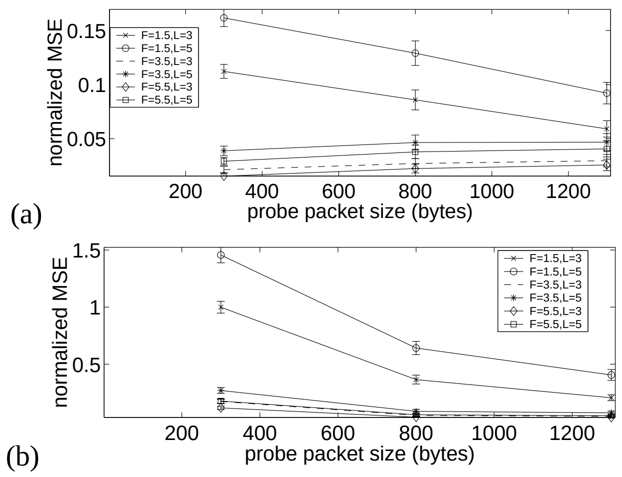


Fig. 6. *Normalized mean squared error vs. probe packet size P for two utilizations: (a) 30% and (b) 70%. In most cases the MSE decreases with increasing packet size. The experiment used $\gamma = 1.2$.*

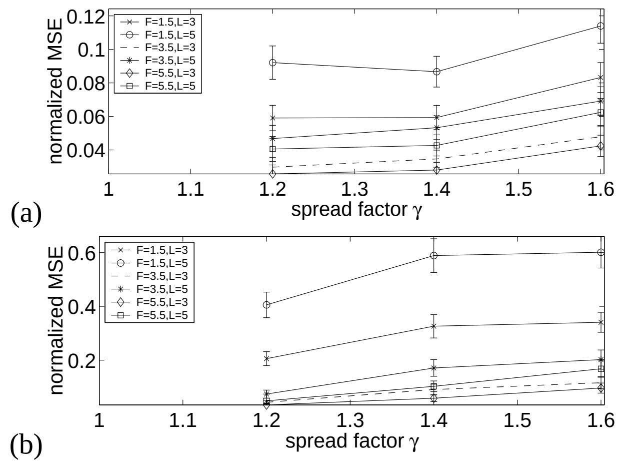
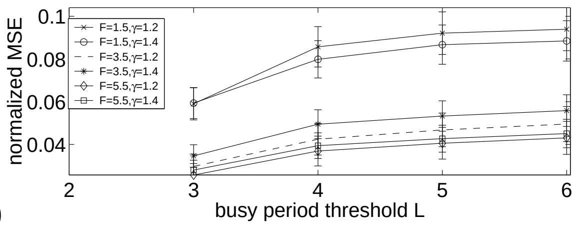


Fig. 7. *Normalized MSE vs. spread factor γ for two utilizations: (a) 30% and (b) 70%. The MSE decreases with decreasing γ.*

pathChirp's excursion segmentation algorithm. Recall that the $E_k^{(m)}$ estimates corresponding to an excursion region are always less than what they would be if the region was not marked as belonging to one (compare cases (a) and (c) in Section II-D). Increasing L or decreasing F makes it harder for bumps in signatures to qualify as valid excursions thus leading to over-estimates of the available bandwidth. Conversely, decreasing L or increasing F will lead to under-estimation of available bandwidth. The optimal choice for the busy period threshold L and decrease factor F will depend on the cross-traffic statistics at queues on the path.

From Figs. 8 and 9 observe that for our single queue Poisson cross-traffic scenario small values of L and large values of F give better performance.



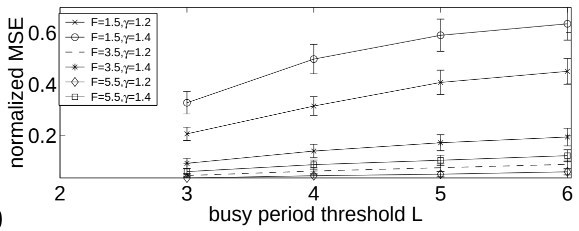
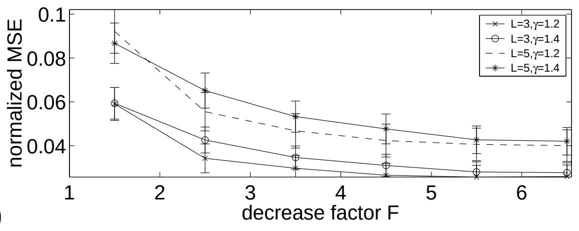


Fig. 8. *Normalized MSE vs. busy period threshold L for two utilizations: (a) 30% and (b) 70%. The error improves with decreasing L.*



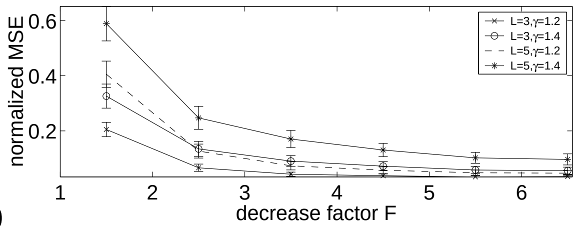


Fig. 9. *Normalized MSE vs. decrease factor F for two utilizations: (a) 30% and (b) 70%. The error improves with increasing F.*

Internet experiments indicate that the optimum values of $L = 3$ and $F = 6$ obtained from the above experiments provide overly conservative estimates of available bandwidth. This could possibly be due to the noise present in real experiments that is absent in simulations. The pathChirp tool instead uses $L = 5$ and $F = 1.5$ as default.

IV. Multi-Hop scenarios

Real Internet paths almost always are multi-hop. Although we are unaware of any rigorous study of the number of congested queues on typical Internet paths, we hypothesize that congestion largely occurs at the edge of the network close to the source or receiver. Thus data packets might likely encounter two congested queues, one on each end of their paths. One fact that supports the argument that congestion occurs at the edge is that backbone ISPs have reported very low packet loss and queuing delay on their networks [13]. While it is possible for paths to have no congested queues or possibly one, it is important for tools like pathChirp to be robust to the presence of at least two congested queues along the end-to-end path.

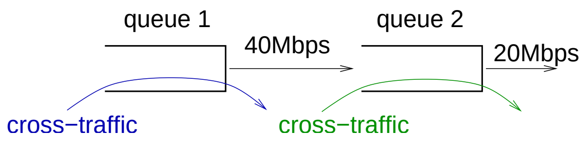


Fig. 10. *Multi-hop experiment.*

This section tests pathChirp in a two-hop scenario as depicted in Fig. 10. As before, competing cross-traffic packet arrivals are Poisson and the packet sizes are chosen at random to be 1500 or 40 bytes with equal probability. The parameters we use are $\gamma = 1.2$, $L = 5$, $F = 2$, $P = 1500$ and $\tau = 3$s.

Each experiment consists of two scenarios. In the first, we load both queues with cross-traffic such that one queue has less available bandwidth (the *tight* queue) than the other (the *slack* queue). The slack queue essentially adds noise to the chirp packet delays. In the second, we set the cross-traffic rate at the slack queue to zero. An error of comparable magnitude in the two scenarios implies that pathChirp is robust to the noise of the slack queue in the first case.

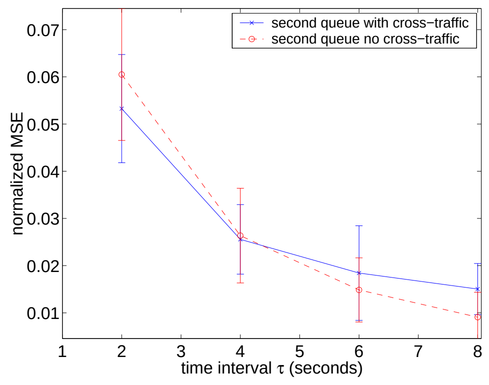
In the first experiment the cross-traffic rate at the first queue is 30Mbps and that at the second queue 5Mbps. This sets the available bandwidth at the first queue (the tight one) to 10Mbps and that at the second (the slack one) to 15Mbps. From Fig. 11(a) we observe that the MSE is practically indistinguishable between the cases where the slack queue has 5Mbps cross-traffic and no cross-traffic at all.

In the second experiment both queues are fed with 10Mbps cross-traffic which sets the available bandwidth at the first queue to 30Mbps and that at the second to 10Mbps. From Fig. 11(b) to our surprise we observe that the MSE is marginally smaller when the slack queue is loaded that when it is not.

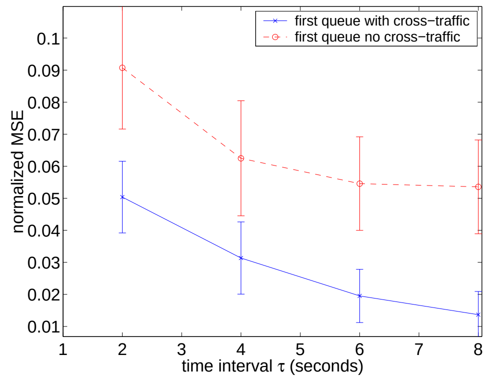
The results show that pathChirp is robust in multi-hop scenarios.

V. Comparison with TOPP

This section compares pathChirp with TOPP [11] using simulations when both use the same probing bit rate and probe packet spacings.

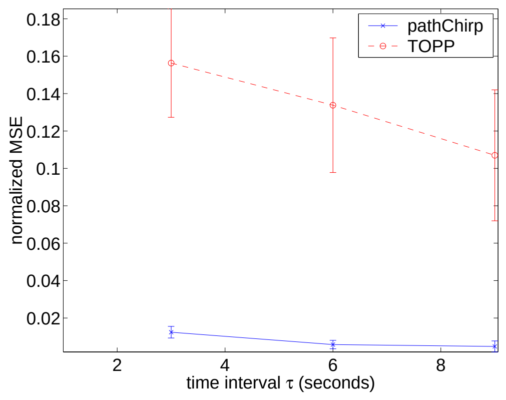


(a)

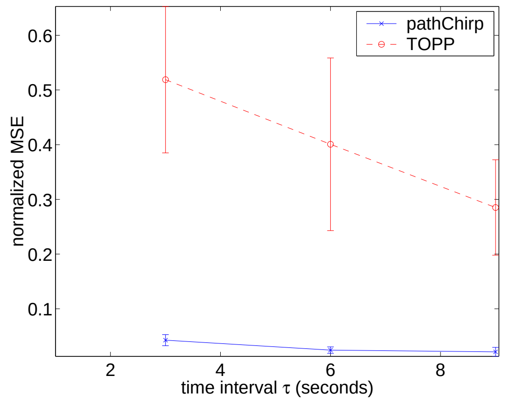


(b)

Fig. 11. *Performance in multi-hop experiments. The MSE in the case of both queues being loaded is comparable to that when only one is loaded implying that pathChirp is robust to multi-hop paths.*



(a)



(b)

Fig. 12. *Comparison of pathChirp and TOPP in a single-hop scenario for two utilizations: (a) 30% and (b) 70%. Observe that pathChirp performs far better than TOPP.*

A. *TOPP*

TOPP sends out several packet pairs well-separated in time [11]. Denote the set of unique packet-pair spacings at the sender arranged in decreasing order as δ_k, $k = 1, \ldots, N-1$ and the corresponding *average* spacings at the receiver as η_k, $k = 1, \ldots, N-1$. Then under the assumption of *proportional sharing* (see [11] for details) of bandwidth at all queues on the path, the plot of η_k/δ_k vs. P/δ_k is piecewise linear with increasing slope. The very first linear segment equals 1 for $P/\delta_k \in (0, B[-\infty,\infty])$, implying that the first breakpoint gives the available bandwidth $B[-\infty,\infty]$. In practice the measured values of η_k/δ_k will be noisy, making a statistical estimation of available bandwidth necessary. We employ the regression-based statistical estimation described in [14].

To compare pathChirp with TOPP, we keep probing loads the same and compute the MSE of the available bandwidth estimates over time intervals of length τ seconds, that is, $B[n\tau,(n+1)\tau]$, $n = 0, 1, \ldots, \infty$. We obtain TOPP's estimate of $B[n\tau,(n+1)\tau]$ using only the probes transmitted during $[n\tau,(n+1)\tau]$. PathChirp's estimates $\rho[n\tau,(n+1)\tau]$ are obtained as described in Section II-B.

For pathChirp we fix the spread factor γ and separate the chirps in time to maintain the desired average probing rate. For TOPP we use packet-pairs with the same inter-spacing times as the chirp packets, that is $\delta_k = \Delta_k$. The separation times between consecutive packet-pairs are chosen as independent exponentially distributed random variables.

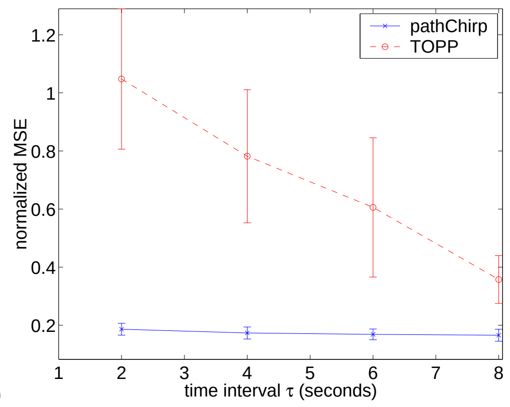
B. *Single-hop scenarios*

This experiment uses a single queue with link speed 20Mbps fed with Poisson cross-traffic. The probe rate is 1Mbps and the pathChirp parameters are set to $P = 1500$ bytes, $\gamma = 1.2$, $F = 5$, and $L = 3$.

Fig. 12 displays the MSE for experiments with two different utilizations. Observe that the pathChirp outperforms TOPP by about an order of magnitude.

C. *Multi-hop scenarios*

We next compare pathChirp and TOPP in the multi-hop scenario depicted in Fig. 10 with average probing rate 500kbps. In the first experiment we set the Poisson cross-traffic rates so that the first queue has 10Mbps available while the second has 15Mbps available. The

(a)


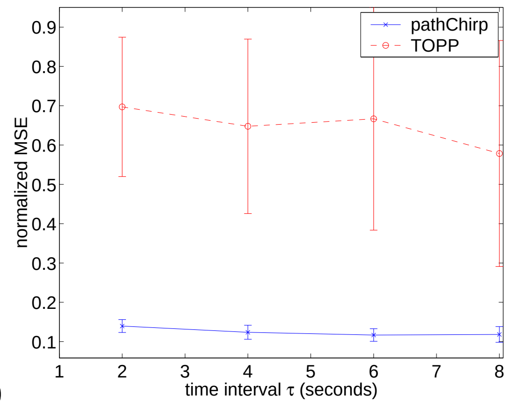
(b)


Fig. 13. *Comparison of pathChirp and TOPP in multi-hop scenarios. In (a) the first queue has less available bandwidth than the second while in (b) the second has the least available bandwidth. Observe that pathChirp performs far better than TOPP.*

first queue is thus the tight one. In the second experiment the rates are set so the the first queue has 20Mbps available and the second has 10Mbps available, thus making the second queue the tight one.

From Fig. 13 observe that again pathChirp outperforms TOPP like in the single-hop scenarios.

Since pathChirp uses queuing delay *correlation* information present in signatures and not just the *average* delay increase between packet pairs, the above results are not surprising. A theoretical analysis supporting these empirical findings is part of our future research.

VI. Comparison with pathload

We now compare pathChirp with pathload (version pathload_1.0.2) [10] using a simple test bed at Rice University depicted in Fig. 14. The goal is to compare their efficiency in terms of number of bytes used to obtain available bandwidth estimates of equal accuracy.

PathChirp and pathload differ in their measurement methodology as well as their output quantities. PathChirp provides a single estimate of available bandwidth per specified time interval τ. Pathload instead

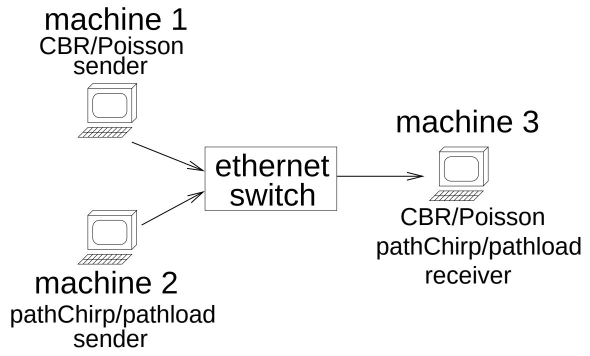


Fig. 14. *Testbed setup at Rice University.*

provides minimum and maximum bounds on the available bandwidth while taking a variable amount of time to make the estimate.

We perform two sets of experiments to compare the tools. To measure the efficiency of the tools, in each experiment we compute the average number of bytes over 25 runs that each tool takes to provide estimates accurate to 10Mbps.

To obtain the bytes used by pathload, we set its bandwidth resolution parameter to 10Mbps and take the average number of bytes used to make 25 estimates.

To count the bytes used by pathChirp, we employ the following procedure. Denoting the start of the experiment as time 0, we compute the estimate $\rho[0,\tau]$ for different values of τ. We define τ^* as that value of τ for which the difference between the 90 and 10 percentiles of $\rho[0,\tau]$ (obtained from 25 experiments) is less than 10Mbps. We then compute the number of probing bytes that pathChirp sends in a time interval of length τ^*.

In this experiment pathChirp used default parameter values: $\gamma = 1.2$, $P = 1000$ bytes, $F = 1.5$, and $L = 5$ packets.

In the first set of experiments, we set the available bandwidth to a constant value using *iperf* CBR UDP traffic [15] while in the second set of experiments we employ Poisson UDP traffic [16]. The iperf packet size is 1470 bytes while that of the Poisson traffic is 1000 bytes. The results in Tables I and II indicate that pathChirp needs less than 10% of the bytes that pathload uses. In addition to the average number of bytes the two tools use to achieve the desired accuracy, Tables I and II provide the 10%-90% values of pathChirp estimates and the *average* of pathload's minimum and maximum bounds of available bandwidth. Observe that pathChirp's estimates have a consistent negative bias, implying that its measurements are conservative.

These results demonstrate pathChirp's utility, especially for applications requiring rapid estimates of the

TABLE I

Efficiency comparison of pathChirp and pathload with iperf CBR cross-traffic.

Available Bandwidth	Efficiency		Accuracy	
	pathChirp	pathload	pathChirp (10-90%)	pathload avg. of min-max bounds
30Mbps	0.41MB	4.3MB	14-25Mbps	16-34Mbps
50Mbps	0.32MB	5.5MB	49-56Mbps	40-49Mbps
70Mbps	0.26MB	9.9MB	59-68Mbps	63-70Mbps

TABLE II

Efficiency comparison of pathChirp and pathload with Poisson cross-traffic.

Available Bandwidth	Efficiency		Accuracy	
	pathChirp	pathload	pathChirp (10-90%)	pathload avg. of min-max bounds
30Mbps	0.35MB	3.9MB	19-29Mbps	16-31Mbps
50Mbps	0.75MB	5.6MB	39-48Mbps	39-52Mbps
70Mbps	0.6MB	8.6MB	54-63Mbps	63-74Mbps

available bandwidth using only a light probing load.

VII. Internet experiments

This section describes Internet experiments with pathChirp. The experiments use the Y topology depicted in Fig. 15(a). PathChirp is employed over a path from the Stanford Linear Accelerator Center (SLAC) to Rice University. To provide some control on the estimated bandwidth, we introduce Poisson traffic along a path from either Caltech or StarLight (Chicago) to Rice. At the time of the experiments, the Caltech-Rice path consisted of 14 layer-3 hops (from traceroute), the SLAC-Rice path consisted of 12 hops, and 4 of the links were shared. The StarLight-Rice path consisted of 9 hops of which 3 were common to the SLAC-Rice path.

For this experiment, the pathChirp parameters are set as follows: $P = 1000$ bytes, $\gamma = 1.2$, $L = 5$ and $F = 1.5$. We choose τ to correspond to 11 chirp transmissions. The Poisson traffic packets are of size 1000 bytes.

In the experiment we sent bursts of Poisson traffic at different rates to study pathChirp's ability to track the resulting changes in available bandwidth. The success is demonstrated in Figs. 15(b) and 15(c). Observe that the estimates decrease in proportion to the rate of the Poisson traffic, with stronger dips corresponding to larger Poisson rates. Note that we have subtracted the Poisson rates from an indicated arbitrary reference value for the sake of clarity. This is because available bandwidth is negatively proportional to the introduced cross-traffic.

We have not compared pathChirp with pathload over the real Internet since we have not obtained enough router data to know the true available bandwidth to evaluate their accuracy.

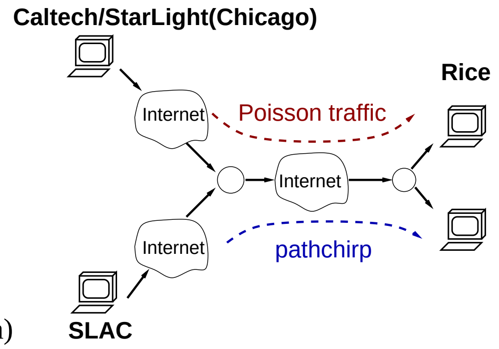
VIII. Gigabit Network Experiments

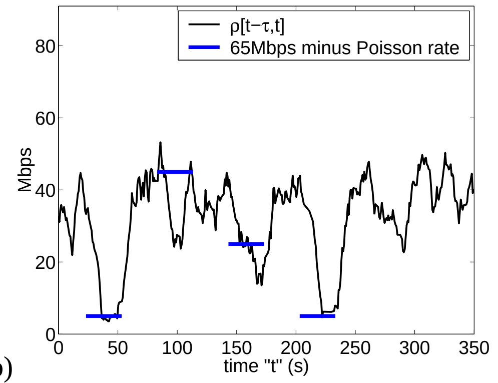
We finally present experiments on a Gigabit network testbed (see Figure 16(a)). These results are particularly relevant to the high-energy physics community that uses high-speed networks to transfer data.

High-speed networks presend several challenges to packet delay based bandwidth measurement that are usually not encountered at lower speeds. First, network interface cards tend to coalesce packet arrival interrupts before forwarding packets to the kernel. This adds to the perceived packet delay thus interfering with measurements. Second packet transmission times are smaller and hence packet delay measurements can tolerate less noise than on networks of 100Mbps or lower speeds.

In our Gigabit experiments we use 5000 byte probe packets. The larger packet size increases the packet transmission time thereby reducing the effect of interrupt coalescence and clock noise on the available bandwidth estimates.

With the help of a Smartbit traffic generator we decrease the utilization on the link between the two routers in steps of 10%. Observe from Figure 16(b) that pathChirp tracks the changes in available bandwidth well. This result augurs well for pathChirp's use in high-speed grid computing and other network-aware applica-

(a)


(b)


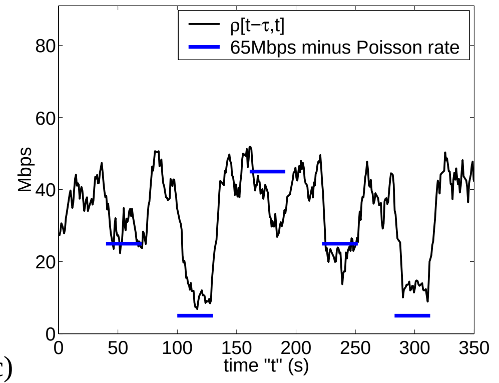
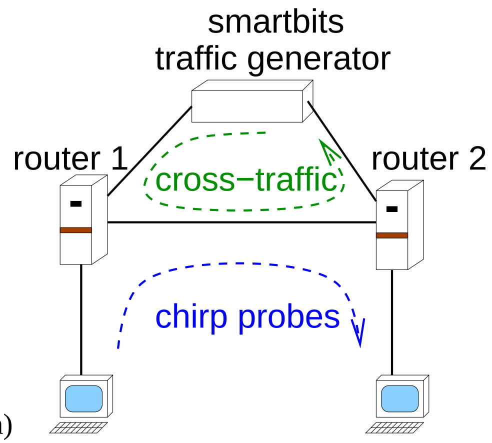
(c)


Fig. 15. *(a) Setup for the Internet experiment. (b) Available bandwidth estimates when Poisson traffic originates at Caltech. (c) Available bandwidth estimates when Poisson traffic originates at StarLight (Chicago). Observe that the pathChirp estimates fall in proportion to the introduced Poisson traffic.*

tions.

IX. Discussion and Conclusions

We have presented pathChirp, an active probing scheme that uses a novel "packet chirp" strategy to dynamically estimate the available bandwidth along an end-to-end network path. Internet and testbed experiments as well as simulations reveal that pathChirp provides accurate, though somewhat conservative, estimates of the available bandwidth. In addition, pathChirp outperforms existing tools in terms of estimation accuracy and efficiency.

(a)


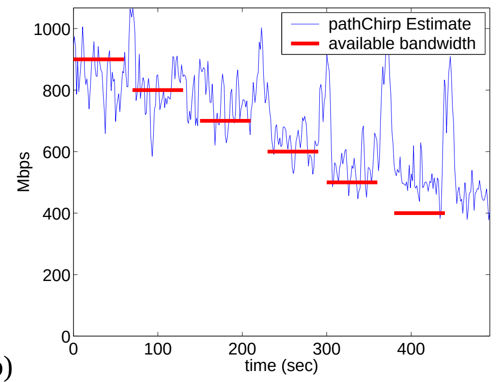
(b)


Fig. 16. *(a) Gigabit testbed network. All links have 1Gbps bandwidth. (b) As cross-traffic increases in steps of 10%, pathChirp accurately tracks the decrease in available bandwidth.*

The current algorithm of pathChirp for available bandwidth estimation mainly uses information about whether probe packet delays are increasing or decreasing. Currently we are investigating algorithms that more fully exploit the rich information contained in chirp delay signatures.

Future enhanced versions of pathChirp will also take the following several practical issues into account.

NIC interrupt coalescence: To reduce the load on the CPU, several high-speed NICs do not generate interrupts with every packet arrival but instead coalesce interrupts. Thus kernel packet timestamps do not reflect the correct packet arrival times on the NIC cards. This affects any probing scheme requiring accurate packet delay information including pathChirp.

Security: A malicious attacker could use deployed probing tools to generate UDP storms. PathChirp will exclude this possibility using inbuilt security mechanisms.

Automatic probing rate adjustment: Currently pathChirp requires the user to specify the range of feasible available bandwidth rates. Future versions of pathChirp will be more user friendly by eliminating this requirement.

Acknowledgments We wish to thank CAIDA for providing their Gigabit bandwidth estimation testbed for experiments, Ryan King and Niels Hoven (Rice) for their assistance with packet timing issues, and the NetDyn creators [17] whose code we modified. We are especially grateful to Allen Mcintosh (Telcordia) for his experiments with pathChirp and insightful comments that improved the tool.

References

[1] "Grid computing info centre." http://www.gridcomputing.com/.

[2] P. Chou and Z. Miao, "Rate-distortion optimized streaming of packetized media," in *Microsoft Research Technical Report MSR-TR-2001-35, February 2001.*

[3] L. Breslau, E. Knightly, S. Shenker, and I. Stoica, "Endpoint admission control: Architectural issues and performance," *In Proc. ACM SIGCOMM*, 2000.

[4] K. M. Hanna, N. Natarajan, and B. N. Levine, "Evaluation of a novel two-step server selection metric," *In Proc. IEEE Conference on Network Protocols (ICNP)*, Oct. 2001.

[5] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks," *In Proc. ACM SOSP*, 2001.

[6] T. Tuan and K. Park, "Multiple time scale congestion control for self-similar network traffic," in *Performance Evaluation*, vol. 36, pp. 359–386, 1999.

[7] "Real-time SLA monitoring tools." http://www.nwfusion.com/news/tech/2001/0115tech.html.

[8] V. Ribeiro, M. Coates, R. Riedi, S. Sarvotham, B. Hendricks, and R. Baraniuk, "Multifractal cross-traffic estimation," *Proc. of ITC Specialist Seminar on IP Traffic Measurement*, Sept. 2000.

[9] G. He and J. C. Hou, "On exploiting long-range dependency of network traffic in measuring cross-traffic on an end-to-end basis," *IEEE INFOCOM*, 2003.

[10] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput," *Proc. ACM SIGCOMM*, 2002.

[11] B. Melander, M. Björkman, and P. Gunningberg, "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks," *Global Internet Symposium*, 2000.

[12] Z.-L. Zhang, V. J. Ribeiro, S. B. Moon, and C. Diot, "Small-time scaling behaviors of Internet backbone traffic: an empirical study," *INFOCOM*, 2003.

[13] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, F. Tobagi, and C. Diot, "Analysis of measured single-hop delay from an operational backbone network," *IEEE INFOCOM*, 2002.

[14] B. Melander, M. Björkman, and P. Gunningberg, "Regression-based available bandwidth measurements," *preprint*.

[15] "Iperf." http://dast.nlanr.net/Projects/Iperf/.

[16] "Poisson traffic generator." http://spin.rice.edu/Software/poisson_gen/.

[17] "Netdyn: Network measurements tool." For more information, see http://www.cs.umd.edu/~suman/netdyn/.