Timing is Everything: Accurate, Minimum Overhead, Available Bandwidth Estimation in High-speed Wired Networks

Han Wang
hwang@cs.cornell.edu

Chiun Lin Lim
cl377@cornell.edu

1Department of Computer Science
Cornell University
Ithaca, NY 14850

Ki Suh Lee
kslee@cs.cornell.edu

Ao Tang
atang@ece.cornell.edu

Erluo Li
 erluoli@cs.cornell.edu

2Department of Electrical and Computer Engineering
Cornell University
Ithaca, NY 14850

Hakim Weatherspoon
hweather@cs.cornell.edu

ABSTRACT

Active end-to-end available bandwidth estimation is intrusive, expensive, inaccurate, and does not work well with bursty cross traffic or on high capacity links. Yet, it is important for designing high performant networked systems, improving network protocols, building distributed systems, and improving application performance. In this paper, we present minProbe which addresses unsolved issues that have plagued available bandwidth estimation. As a middlebox, minProbe measures and estimates available bandwidth with high-fidelity, minimal-cost, and in userspace; thus, enabling cheaper (virtually no overhead) and more accurate available bandwidth estimation. MinProbe performs accurately on high capacity networks up to 10 Gbps and with bursty cross traffic. We evaluated the performance and accuracy of minProbe over a wide-area network, the National Lambda Rail (NLR), and within our own network testbed. Results indicate that minProbe can estimate available bandwidth with error typically no more than 0.4 Gbps in a 10 Gbps network.

1. INTRODUCTION

Active end-to-end available bandwidth estimation is intrusive, expensive, inaccurate, and does not work well with bursty cross traffic or on high capacity links [13, 33]. Yet, it is important [17, 43, 45]. It is necessary for designing high performant networked systems, improving network protocols, building distributed systems, and improving application performance.

The problems associated with available bandwidth estimation stem from a simple concept: Send a train of probe packets through a network path to momentarily congest the bottleneck link, then infer the available bandwidth at the receiving end from probe packet timestamps [20, 36, 40, 46]. There are three significant problems with current bandwidth estimation approaches. First, such approaches are intrusive since load is being added to the network. Current measurement methods require creating explicit probe packets that consume bandwidth and CPU cycles. Second, current approaches often do not have enough fidelity to accurately generate or timestamp probe packets. By operating in userspace to maximize programming flexibility, precise control and measurement of packet timings is sacrificed. As a result, the probe packet timings are often perturbed by operating systems (OS) level activities. Without precise timestamps, accurate estimation becomes very challenging, especially in high-speed networks. Finally, Internet traffic is known to be bursty at a range of time scales [22], but previous studies have shown that existing estimation approaches perform poorly under bursty traffic conditions [48].

In this paper, we present a middlebox approach that addresses the issues described above. MinProbe performs available bandwidth estimation in real-time with minimal-cost, high-fidelity for high-speed networks (10 Gigabit Ethernet), while maintaining the flexibility of userspace control and compatibility with existing bandwidth estimation algorithms. MinProbe minimizes explicit probing costs by using application packets as probe packets whenever possible, while generating explicit packets only when necessary. MinProbe greatly increases measurement fidelity via software access to the wire and maintains wire-time timestamping of packets before they enter the host. Finally, a userspace process controls the entire available bandwidth estimation process giving flexibility to implement new algorithms as well as reuse existing estimation algorithms. Importantly, minProbe operates in real-time, such that it is useful as a networked systems building block.

MinProbe was designed with commodity components and achieves results that advance the state of the art. It can be built from a commodity server and an FPGA (field programmable gate array) PCIe (peripheral component interconnect express) pluggable board, SoNIC [26]. As a result, any server can be turned into a minProbe middlebox. Indeed, we have turned several nodes and sites in the GENI (global environment for networking innovations) network into minProbe middleboxes. It has been evaluated on a
testbed that consists of multiple 10 Gigabit Ethernet (10 GbE) switches and on the wide-area Internet via the National Lambda Rail (NLR). It achieves high accuracy: Results illustrate available bandwidth estimation with errors typically no more than 0.4 Gbps in a 10 Gbps network. It estimates available bandwidth with minimal overhead: Available bandwidth estimates use existing network traffic as probes. Overall, minProbe allows software programs and end-hosts to accurately measure the available bandwidth of high speed, 10 gigabit per second (Gbps), network paths even with bursty cross traffic, which is a unique contribution.

Our contributions are as follows:

- **High performance**: We are the first to demonstrate a system that can accurately estimate available bandwidth in 10 Gbps network.
- **Minimal overhead**: Albeit not the first middlebox approach to available bandwidth estimation (e.g. MGRP [37]), our system performs accurate bandwidth estimation with minimal overhead.
- **Ease of use**: Our system provides flexible userspace programming interface and is compatible with existing algorithms.
- **Sensitivity study**: We conduct a sensitivity study on our estimation algorithm with respect to internal parameter selection and network condition.

### 2. BACKGROUND

Available bandwidth estimation is motivated by a simple problem: What is the maximum data rate that a user could send down a network path without going over capacity? This data rate is equal to the available bandwidth on the *tight* link, which has the minimum available bandwidth among all links on the network path. While the problem sounds simple, there are four main challenges to available bandwidth estimation—timeliness, accuracy, non-intrusiveness, and consistency. In particular, an available bandwidth estimation methodology and tool would ideally add as little overhead as possible and return timely and accurate estimation on the available bandwidth consistently across a wide range of Internet traffic conditions.

In this section, we first discuss many of the key ideas and assumptions underlying existing available bandwidth estimation methods and tools. Then, motivate the development of minProbe by illustrating the limitations faced by these current methods and tools.

#### 2.1 Methodology

Many existing available bandwidth estimation tools take an end-to-end approach. A typical setup sends probe packets with a predefined interval along the path under measurement, and observe change in certain packet characteristics at the receiver to infer the amount of cross traffic in the network. The key idea to such inferences: When the probing rate exceeds the available bandwidth, the observed packet characteristics undergo a significant change. The turning point where the change occurs is then the estimated available bandwidth. See Figure 9, for example, the turning point where queuing delay variance increases is the available bandwidth estimate. Most existing available bandwidth estimation tools can be classified according to the packet characteristics and metrics used to observing a turning point.

Bandwidth estimation tools such as Spruce [46] and IGI [16] operate by observing the change in the output probing rate. The idea is that when the probing rate is below the available bandwidth, probe packets should emerge at the receiver with the same probing rate. However, once the probing rate exceeds the available bandwidth, the gap between packets will be stretched due to queuing and congestion, resulting in a lower output probing rate. On the other hand, bandwidth estimation tools such as Pathload [20], PathChirp [40], TOPP [36], and Yaz [42] operate by observing the change in one-way delay (OWD), i.e. the time required to send a packet from source to destination. OWD increases when the probing rate exceeds the available bandwidth, as packets have to spend extra time in buffers.

An alternative methodology to the end-to-end approach would be to query every network element (switch/router) along a network path. A network administrator could collect the statistical counters from all related ports, via protocols such as sFlow [5]; or infer from Openflow control messages such as PacketIn and FlowRemoved messages [50]. However, obtaining an accurate, consistent, and timely reading from multiple switches in an end-to-end manner can be very difficult [2]. Further, collecting counters from switches is done at per-second granularity and requires network administrative privileges, which makes the approach less timely and useful for building distributed systems, improving network protocols, or improving application performance. As a result, we assume and compare to other end-to-end approaches for the rest of this paper.

#### 2.2 Current Limitations

While useful, the end-to-end approach for available bandwidth estimation has inherent issues that limit its applicability, especially in high-capacity links. We outline a few of these challenges.

- **High-fidelity measurements require high-fidelity instruments**: Generating packet trains in userspace typically require a CPU-intensive busy-wait loop, which is prone to operating system level noise, such as OS scheduling, interrupt coalescing, and batching. As a result, generated interpacket gaps of a packet train rarely match the interpacket
gaps specified by the userspace program. Moreover, timestamping in user or kernel space adds considerable error to the available bandwidth estimation. Even timestamping in hardware adds noise when timestamping packet (albeit, less noise than kernel space, but still enough to make available bandwidth estimation unusable in many situations [27]). The takeaway: Without high-fidelity capabilities, available bandwidth estimation lacks accuracy.

- **Non-negligible overhead.** Bandwidth estimation tools add traffic to the network path under measurement. This may adversely affect application traffic and measurement accuracy [37]. The amount of probe traffic is proportional to the rate of sampling and the number of concurrent measurement sessions. As a result, the effect of probe packets on cross traffic exacerbates with traffic increase.

- **Traffic burstiness impacts measurement results.** Bandwidth estimation results are sensitive to traffic burstiness. However, existing tools typically ignore the burstiness of cross traffic in favor of simple estimation algorithms. Limitations of batching and instrumentation further mask burstiness to some degree to userspace tools. As a result, existing tools may not be able to reflect true available bandwidth with bursty traffic.

In this paper, we introduce a new tool that can be applied to existing available bandwidth estimation algorithms and tools while overcoming the limitations discussed above. Indeed, we demonstrate that we can accurately estimate available bandwidth in high-speed networks with minimal overhead with existing algorithms. In the next section, we discuss the details of the design and implementation of our methodology.

3. **MinProbe DESIGN**

MinProbe measures the available bandwidth with high-fidelity, minimal-cost and in userspace; thus, enabling cheaper (virtually no overhead) and more accurate available bandwidth estimation. It achieves these goals through three main features:

First, minProbe is a middlebox architecture that uses application network traffic as probe traffic to eliminate the need for explicit probe packets. When an application packet arrives at minProbe, minProbe decides whether the packet should be used as a probe; if so, minProbe modulates the timings between application packets chosen as probe packets before forwarding them to their destination. A programmable flow table accessible from userspace controls the selection of an application packet as a probe packet. Thus, by using application traffic implicitly as available bandwidth probes, we are able to remove all the traditional costs and overheads. A similar idea was proposed in MGRP [37], which has the same goal of lower overhead, but MGRP lacks high-fidelity measurement capability.

Second, minProbe is a high-fidelity network measurement substrate that is capable of modulating and capturing traffic timings with sub-nanosecond precision. The high-precision is achieved by enabling software access to the physical layer of the network protocol stack. When minProbe modulates probe packets (application packets), it adds and removes minute spacings between packets through direct access to the physical layer.

Finally, minProbe is accessible from userspace. From userspace, users can control sub-nanosecond modulations between packets to generate probes and obtain sub-nanosecond timings between received packets to estimate available bandwidth. Further, all probes can be generated and timing measurements can be received in real-time from userspace.

We envision minProbe to be deployed in an architecture where a separate control plane manages a rich set of middlebox functionalities with event trigger support(e.g [9]). In particular, we believe the separation of measurement hardware and production hardware enables high-fidelity measurement in high-speed networks (and in real-time) that is difficult to achieve otherwise.

3.1 **Precise Probe Control**

MinProbe offers enhanced network measurement capabilities: High-fidelity packet pacing to generate probe packets and high-fidelity packet timestamping to measure received packet times. We achieve high-fidelity capabilities via direct access to the physical layer from software and in real-time. To see how minProbe works, we first describe the physical layer of 10 Gigabit Ethernet (GbE). Then, we describe how minProbe takes advantage of the software access to the physical layer to achieve high-fidelity network measurement.

3.1.1 **10 GbE Physical Layer**

According to the IEEE 802.3 standard [3], when Ethernet frames are passed to the physical layer (PHY), they are reformatted before being sent across the physical medium. On the transmit path, the PHY encodes every 64bits of an Ethernet frame into a 66bit block, which consists of a two bit synchronization header (sync-header) and a 64-bit payload. As a result, a 10 GbE link actually operates at 10.3125 Gbaud (10G × 1/32). The PHY also scrambles each block before passing it down the network stack to be transmitted. The entire 66bit block is transmitted as a continuous stream of symbols, which a 10 GbE network transmits over a physical medium. As 10 GbE always sends 10.3125 gigabits per second (Gbps), each hit in the PHY is about 97 pico-seconds wide. On the receive path, the PHY layer removes the two-bit header and re-scrambles each 64bit block before decoding it.

Idle symbols (/I/) are special characters that fill the gaps between any two packets in the PHY. When there is no Ethernet frame to transmit, the PHY continuously inserts /I/ symbols until the next frame is available. The standard requires at least twelve /I/s after every packet. Depending on Ethernet frame and PHY alignment, an /I/ character can be 7 or 8 bits, thus it takes about 700~800 pico-seconds to transmit one /I/ character [3]. Importantly, controlling or simply counting the number of /I/s between packets is what enables high-fidelity. However, before the SoNIC [26] platform was created, /I/ characters were typically inaccessible from higher layers (L2 or above), because they are discarded by hardware, and definitely were not accessible in software.

3.1.2 **High-Fidelity Measurement**

The key insight and capability of how minProbe achieves high-fidelity is from its direct access to the /I/ characters in the physical layer. In particular, minProbe can measure (count) and generate (insert or remove) an exact number of /I/ characters between each subsequent probe packet to measure the relative time elapsed between packets or generate a desired interpacket gap. Further, if two subsequent probe packets are separated by packets from a different flow.
accessing the physical layer of 10 GbE in software in real-time, by inserting an exact spacing between packets. By inserting high-fidelity probe packets through high-fidelity accuracy) [27].

consequently, many existing bandwidth estimation tools report significant estimation error (large variation and low accuracy) [27].

In a similar fashion to measuring the space between packets, minProbe generates probe packets through high-fidelity pacing, by inserting an exact spacing between packets. By accessing the physical layer of 10 GbE in software in real-time, minProbe can insert or remove /I/ characters from application traffic as needed. In particular, two variables of the probe traffic are of interest: The gap between packets and the overall rate of packet trains. Users can program minProbe via command line calls or API calls to specify the number of /I/ characters (i.e. the number of 100s of pico-seconds) to be maintained between subsequent probe packets, allowing userspace programs to perform high-fidelity pacing.

Direct access to /I/ characters from software and in real-time differentiates minProbe from other measurement tools. Specifically, minProbe is able to characterize and generate an exact spacing (interpacket gap) between probe packets.

3.1.3 Probing without a Probe

Explicit probes are not necessary for minProbe. Similar to MGRP [37], minProbe can use application traffic as probe packets. It has a programmable flow table that performs flow matching on pass-through traffic. Users can insert entries into the flow table to specify which flows are probes. Traffic that have a match in the flow table are of interest: The gap between packets and the overall rate of packet trains. Users can program minProbe via command line calls or API calls to specify the number of /I/ characters (i.e. the number of 100s of pico-seconds) to be maintained between subsequent probe packets, allowing userspace programs to perform high-fidelity pacing.

Direct access to /I/ characters from software and in real-time differentiates minProbe from other measurement tools. Specifically, minProbe is able to characterize and generate an exact spacing (interpacket gap) between probe packets.

3.2 Generalized Probe Model

Figure 3 shows a generalized probe train model we developed to emulate a number of existing bandwidth estimation algorithms and used for the rest of the paper. The horizontal dimension is time. The vertical dimension is the corresponding (instantaneous) probe rate. Each pulse (we call it a train) contains multiple probe packets sent at a particular rate, as depicted by the height. In particular, the parameter N represents the number of packets in each train and parameter R represents the (instantaneous) probe rate of the train (i.e. we are able to change the interpacket gap between N packets to match a target probe rate R). Packet sizes are considered in computing the probe rate. For a mixed-size packet train, minProbe is able to adjust the space between all adjacent packets within the train to achieve the desired probe rate R. The gaps between successive probe trains are specified with parameter G (gap). Finally, each measurement sample consists of a set of probe trains with increasing probe rate. The distance in time between each measurement sample is identified with parameter D (distance). With these four parameters, we can emulate most probe traffic used in prior work, as shown in Table 2 and discussed in Section 2.1. For example, to emulate Spruce probes, we can set the parameters (N, R, G, D) to the values (2, 500Mbps, 1.2ns, 48us), which would generate a pair of probe packets every 48us with minimal inter-packet gap (12 /I/ characters or 1.2ns) at 500Mbps (5% of the capacity of a 10Gbps link), as in-
The difference in one-way delay tells us that

\[ t_B = t_A + (q_n - q_1) \quad (3) \]

Substitute the relationship for \( t_A \) and \( t_B \) and we can see that the difference in one-way delay is equivalent to the difference of interpacket gap.

### 3.3 Implementation

We built a prototype of minProbe on the software network interface, SoNIC, platform with two 10 GbE ports [26]. The detailed design of the SoNIC platform has been described in prior work [26]. Here, we briefly recount the main features of SoNIC. SoNIC consists of two components: a software stack that runs on commodity multi-core processors, and a hardware PCIe pluggable board. The software implements all of the functionality in the 10GbE physical layer that manipulates bits to enable access to /I/ in software. The hardware performs line-speed data transfer between the 10 GbE transceiver and the host. To enable the high-fidelity network measurements required by minProbe, we extended SoNIC’s capabilities with three new features:

**Packet Filtering and Forwarding:** We extended SoNIC to support packet forwarding and filtering at line rate. Packet forwarding preserves the timing characteristics of the pass-through traffic exactly: Data is copied from an incoming port to an outgoing port, including the exact number of /I/ characters in between each packet. To filter for probe packets, we use an in-kernel flow table that matches on the 5-tuples of a packet IP header. The packets that have a match are temporarily buffered in a queue, interpacket gap modulated, and sent once enough packets have been buffered for a single packet train (a measurement sample).

**Packet Gap Extraction and Manipulation:** MinProbe has two operational modes: a modulation mode for sending probes and an extraction mode for receiving probes. In the modulation mode, we extended SoNIC with additional packet queues for temporarily buffering of probe packets, and modified the amount of /I/ characters in front of the buffered packet to change the packet spacing. If /I/ characters are removed from the probe packet trains to increase the probe rate, we compensate for the loss of /I/ characters at the end of the probe packet train to conform to the 10 GbE standard, and vice versa. In the extraction mode, the number of /I/ characters are extracted from the matched probe packet trains and sent to the userspace programs via kernel upcalls.

### Additional Notes

Additionally, the difference in one-way delay tells us that

\[ t_B = t_A + (q_n - q_1) \quad (3) \]

Substitute the relationship for \( t_A \) and \( t_B \) and we can see that the difference in one-way delay is equivalent to the difference of interpacket gap.

### 3.3 Implementation

We built a prototype of minProbe on the software network interface, SoNIC, platform with two 10 GbE ports [26]. The detailed design of the SoNIC platform has been described in prior work [26]. Here, we briefly recount the main features of SoNIC. SoNIC consists of two components: a software stack that runs on commodity multi-core processors, and a hardware PCIe pluggable board. The software implements all of the functionality in the 10GbE physical layer that manipulates bits to enable access to /I/ in software. The hardware performs line-speed data transfer between the 10 GbE transceiver and the host. To enable the high-fidelity network measurements required by minProbe, we extended SoNIC’s capabilities with three new features:

**Packet Filtering and Forwarding:** We extended SoNIC to support packet forwarding and filtering at line rate. Packet forwarding preserves the timing characteristics of the pass-through traffic exactly: Data is copied from an incoming port to an outgoing port, including the exact number of /I/ characters in between each packet. To filter for probe packets, we use an in-kernel flow table that matches on the 5-tuples of a packet IP header. The packets that have a match are temporarily buffered in a queue, interpacket gap modulated, and sent once enough packets have been buffered for a single packet train (a measurement sample).

**Packet Gap Extraction and Manipulation:** MinProbe has two operational modes: a modulation mode for sending probes and an extraction mode for receiving probes. In the modulation mode, we extended SoNIC with additional packet queues for temporarily buffering of probe packets, and modified the amount of /I/ characters in front of the buffered packet to change the packet spacing. If /I/ characters are removed from the probe packet trains to increase the probe rate, we compensate for the loss of /I/ characters at the end of the probe packet train to conform to the 10 GbE standard, and vice versa. In the extraction mode, the number of /I/ characters are extracted from the matched probe packet trains and sent to the userspace programs via kernel upcalls.
**Function** | **Description** | **Example**
---|---|---
`set_mode` | set middlebox mode | `set_mode`
`set_gap` | set gap between probes | `set_gap(120,120,120)`
`flow_add` | add flow used as probes | `flow_add(srcip,dstip)`

**Table 3: Application Programming Interface**

**Application Programming Interface:** The last feature we added was the userspace accessibility. Users can insert/remove flow entries, dynamically adjust probe packet gaps, and retrieve probe packet gap counts from minProbe. It is required to implement a flexible user and kernel space communication channel for passing measurement data and control messages between user and kernel space. We used the `netlink [4]` protocol as the underlying mechanism and implemented the exchange of information using `netlink` messages. For example, we encapsulate the interpacket gaps in a special `netlink` message, and transmitted it from kernel to userspace, and vice versa.

Userspace daemons control minProbe. Userspace daemons provide a programmable interface for the kernel-to-user data path. Table 3 presents a few API calls that are supported by minProbe. For example, as shown in Figure 4, users can program the flow table through the daemon or directly by `flow_add` commands. In addition, users can retrieve information from the kernel through the userspace daemon. In particular, a kernel module captures the interpacket gap /1/ counts from the probe traffic and passes the information to userspace through the userspace daemon. All communication consumes a reasonably small amount of bandwidth. For instance, the most expensive action is the transfer of interpacket gap /1/ counts from kernel to userspace, which requires a few hundred mega-bytes per second of bandwidth for minimum sized 64 byte packets at 10 Gbps.

## 4. EVALUATION

MinProbe can accurately estimate the available bandwidth in high-speed networks with minimal overhead. To demonstrate its capabilities and accuracy, we evaluated minProbe over three different physical networks: Two topologies in our controlled environment and the National Lambda Rail (NLR). We used a controlled environment to study the sensitivity of estimation accuracy to various network conditions. We also evaluated minProbe on NLR to study its performance in wide area networks. To highlight the accuracy minProbe is capable of, Figure 8 in Section 4.2 shows that minProbe can accurately estimate the available bandwidth within 1% of the actual available bandwidth in a 10Gbps network with topology shown in Figure 6a.

In the remainder of this section, we evaluate minProbe in depth and seek to answer the following questions:

- How sensitive is minProbe to application and cross traffic characteristics (Section 4.3 and 4.4)?
- Does minProbe work in the wild (Section 4.5)?
- Can other software middleboxes accurately measure available bandwidth (Section 4.6)?

### 4.1 Experimental setup

Our evaluation consists of two parts: Controlled environment experiments and wide area network (WAN) experiments. For the WAN experiments, we evaluate minProbe over the National Lambda Rail (NLR). NLR is a transcontinental production 10Gbps Ethernet network shared by research universities and laboratories with no restriction on usage or bandwidth. As a result, traffic in the NLR is typically bursty, yet persistent, and similar to the Internet. We provisioned a dedicated virtual route that spans nine routing hops over 2500 miles, as shown in Figure 5. Since the underlying physical network is also shared with other provisioned routes, we observed persistent cross traffic between 1 Gbps to 3.5 Gbps.

For the controlled environment experiments, we set up two different network topologies in a 10 Gbps network testbed: Dumb-bell (DB) and parking-lot (PL), as illustrated in Figure 6 and described in [49]. We used one minProbe middlebox (MP0) to modulate the application traffic generated from a server directly connected to MP0. We used the other minProbe middlebox (MP1) as the receiver of the modulated probes to capture the timing information from the application traffic. A network of one or more hops separated the two middleboxes, where one link was a bottleneck link (i.e. a tight link with the least available bandwidth). Our experiments attempted to measure and estimate the available bandwidth of this bottleneck (tight) link. The testbeds were built with commercial 10 GbE switches, represented by circles in Figure 6. MinProbe middleboxes and hosts that generated cross traffic were separately connected to the switches using 10 GbE links, illustrated with the solid line. In the DB topology, the link between the two switches is the bottleneck (tight) link. In the PL topology, the bottleneck (tight) link depends on the pattern of the cross traffic.

MP0 and MP1 were Dell PowerEdge T710 servers. Each server had two Intel Xeon Westmere [19] X5670 2.93GHz processors, with six cores in each CPU and a total of 24 GB RAM. The Westmere architecture of the processor is well-known for its capability of processing packets in a multi-threaded environment [11, 14, 35]. The switches used in the environment consisted of an IBM G8264 RackSwitch and were Dell PowerEdge T710 servers. Each server had two Intel Xeon Westmere [19] X5670 2.93GHz processors, with six cores in each CPU and a total of 24 GB RAM. The Westmere architecture of the processor is well-known for its capability of processing packets in a multi-threaded environment [11, 14, 35]. The switches used in the environment consisted of an IBM G8264 RackSwitch and

![Figure 5: National Lambda Rail Experiment](image)

![Figure 6: Controlled Experiment Topologies](image)
The second class of cross traffic was generated from stationary bursty distributions, CBR2. CBR2 is created by varying both the packet size and packet chain length distributions with two parameters $D_{\text{size}}$ and $D_{\text{len}}$. Both parameters range from 0.0 (zero variance) to 1.0 (high variance). We draw the packet size from a log-normal distribution which has been demonstrated to closely resemble the distribution of Internet traffic in [28]. The mean of the log-normal distribution is fixed, while the variance is controlled by the parameter $D_{\text{size}}$ to be $D_{\text{size}} \cdot V_{\text{size}}$, where $V_{\text{size}}$ is a fixed high variance value. A packet chain is a series of consecutive packets with minimal interpacket gap in between, and from [12], we know its length takes on a geometric distribution. The parameter of the geometric distribution is taken to be $1 - D_{\text{len}}$. Thus, CBR1 is a special case of CBR2 where both $D_{\text{size}}$ and $D_{\text{len}}$ equals 0.0.

### Table 4: IPD and IPG of uniformly spaced packet streams.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>792</td>
<td>1</td>
<td>156250</td>
<td>6400</td>
<td>7200</td>
</tr>
<tr>
<td>792</td>
<td>3</td>
<td>460814</td>
<td>2137</td>
<td>1572</td>
</tr>
<tr>
<td>792</td>
<td>8</td>
<td>950628</td>
<td>1068</td>
<td>636</td>
</tr>
<tr>
<td>792</td>
<td>8</td>
<td>1250000</td>
<td>800</td>
<td>200</td>
</tr>
</tbody>
</table>

**CAIDA:** In the third scheme, we generated cross traffic from a real Internet trace. In particular, we used the CAIDA OC-192 [6] trace which was recorded using a DAG card with nano-second scale timestamps.

While there is not one commonly accepted definition for traffic burstiness in the literature, burstiness generally refers to the statistical variability of the traffic. In other words, high variability in traffic data rates implies very bursty traffic. Variability, and by extension, traffic burstiness could be captured by wavelet-based energy plots [47]. Specifically, the energy of the traffic represents the level of burstiness at a particular timescale (e.g. at the microsecond timescale). The higher the energy, the burstier is the traffic. Figure 7b and 7d illustrate this wavelet-based energy plot when applied to the byte arrivals for CAIDA Internet traces and synthetically generated traffic, respectively. The x-axis represents different time scale, ranging from nano-second to second in log scale. The y-axis represents the abstracted energy level at a particular time scale. We were mostly interested in the micro-second timescale, which correspond to x-axis values around 15. Notice that the synthetic traffic behave very much like CAIDA internet traces for energy levels for these values.

For most experiments, we used 792 bytes as the application traffic packet size, which according to [6], is close to the average packet size observed in the Internet. We adjusted the traffic data rate by varying interpacket gaps (IPG). In particular, we insert a specific number of /1/’s between packets to generate a specific data rate. We controlled the exact data rate of CBR1 and CBR2 since we control the exact number of /1/’s inserted between packets. However, for the CAIDA traces, we selected portions of the trace recording times to obtain different average data rates. In addition, when we replay a trace, we only need to preserve the timing information of the trace, not the actual payload. We used SoNIC to replay the CAIDA trace. As described in [26], SoNIC can regenerate the precise timing characteristics of a trace with no deviation from the original.

### 4.2 Baseline Estimation

How does available bandwidth estimation perform using minProbe in a base case: A simple topology with a couple to several routing hops (Figure 6a and b) and uniform cross traffic (uniform packet size and interpacket gaps)? We illustrate the result with the following experiments.

In the first experimental network setup, we use a dumbbell topology with two routing hops and a single bottleneck link (Figure 6a). Cross traffic is generated with a constant bit rate that has uniform packet sizes and interpacket gaps (CBR1). Node $N_0$ sends cross traffic to node $N_1$ according to the CBR1 uniform distribution.

The second experimental setup is similar to the first except that we use a parking-lot topology (Figure 6b). Further, cross traffic is generated with the CBR1 uniform distribution between neighboring nodes: $N_{\text{even}}$ to $N_{\text{odd}}$ (e.g. $N_0$ to...
Introducing any new packets. Further, we configured MP samples. Note that there is no overhead since we are not introducing any new packets. Further, we configured MP to capture the timing information of the probe packet samples (i.e., application traffic destined to APP from source APP). The probe packets modulated by MP were parameterized using the model introduced in Section 3.2. We used the parameters \((N, R, G, D) = (20, [0.1 : 0.1 : 9.6 \text{ Gbps}, 10 \text{ us}, 4 \text{ ms})\) where MP sends a probe packet sample every 4 ms enabling us to collect 250 samples per second. Each probe sample consists of around 100 or more packets [20,40]. A probe train (i.e., the number of packets in a train) to the accuracy of available bandwidth estimation: the length of a probe train and the distribution of probe packet sizes. The length of a probe train affects the delay experienced by application traffic (See the description of probe train length in Section 3.2 and equation 1). The longer the probe train, the longer the application traffic has to be buffered in the middlebox, resulting in longer delays the application traffic experiences. Typically, a probe train consists of around 100 or more packets [20, 40]. A probe train of two packets is essentially similar to the packet pair scheme described by Spruce [46].

Creating probe packet trains from variable sized packets is difficult because packets with different sizes suffer different delays through a network. Because minProbe does not have control over the application traffic that is used as packet probes, it is important to understand how the accuracy of minProbe changes with different distributions of probe packet sizes. Another consideration is that many estimation algorithms prefer large probe packets to small packets [20, 40].

We evaluate the effect of the length of a probe packet train (i.e., the number of packets in a train) to the accuracy of available bandwidth estimation in Section 4.3.1 and the effect of the distributions of probe packet sizes in Section 4.3.2 and 4.3.3. For these experiments, we always used CBR1 as cross traffic.

### 4.3 Impact of Application Traffic

Next, we study whether minProbe is impacted by characteristics of application traffic. In particular, we focus on two characteristics that matter the most for available bandwidth estimation: the length of a probe train and the distribution of probe packet sizes. The length of a probe train affects the delay experienced by application traffic (See the description of probe train length in Section 3.2 and equation 1). The longer the probe train, the longer the application traffic has to be buffered in the middlebox, resulting in longer delays the application traffic experiences. Typically, a probe train consists of around 100 or more packets [20, 40]. A probe train of two packets is essentially similar to the packet pair scheme described by Spruce [46].

Creating probe packet trains from variable sized packets is difficult because packets with different sizes suffer different delays through a network. Because minProbe does not have control over the application traffic that is used as packet probes, it is important to understand how the accuracy of minProbe changes with different distributions of probe packet sizes. Another consideration is that many estimation algorithms prefer large probe packets to small packets [20, 40].

We evaluate the effect of the length of a probe packet train (i.e., the number of packets in a train) to the accuracy of available bandwidth estimation in Section 4.3.1 and the effect of the distributions of probe packet sizes in Section 4.3.2 and 4.3.3. For these experiments, we always used CBR1 as cross traffic.

#### 4.3.1 Impact of Varying Probe Train Length

Figure 8: Available bandwidth estimation in a dumb-bell topology under CBR traffic. Both cross traffic and probe traffic share one bottleneck with the capacity of 10Gbps. The x-axis represents the actual available bandwidth of the bottleneck link. The y-axis represents the estimation by minProbe. This evaluation demonstrates minProbe’s ability to accurately measure the available bandwidth and achieve the estimation with minimal probing overhead.

Figure 8 shows the result. It shows the actual available bandwidth on the x-axis and estimated available bandwidth on the y-axis for the the dumb-bell and parking-lot topologies. We estimated the available bandwidth with the mean of 10 measurement samples. The estimations in the dumb-bell topology were within 0.1 Gbps of the actual available bandwidth. The estimations in parking-lot topology tended to underestimate the available bandwidth, due to the multiple bottleneck links in the network path. Higher available bandwidth scenarios were more difficult to measure as accurately in a multiple-hop network path. Alternatively, the maximum value of a measurement could be used to estimate available bandwidth if the estimation was always overly conservative; this would actually increase the estimation accuracy.

Figure 9 shows the raw probe sample data that was used to estimate the available bandwidth of different cross traffic rates in the dumb-bell topology. The x-axis is the rate (R) at which the probe trains were sent. The y-axis is the variance of the queuing delay, which was computed from the one-way delay (OWD) experienced by the packets within the same probe train. We kept the cross traffic rate constant at 1Gbps, 3Gbps, 6Gbps and 8Gbps. The available bandwidth was estimated to be the turning point where the delay variance shows a significant trend of increasing.

![Figure 8: Available bandwidth estimation in a dumb-bell topology under CBR traffic.](image)

![Figure 9: Scatter-plot showing the queuing delay variance of probe packets versus the probe rate.](image)
In order to understand how the number of packets in a probe packet train affects the accuracy of minProbe, we varied the number of packets in each train while using the same parameters as the baseline experiment (Section 4.2). In particular, we used \((N, [0.1 \times 0.1 \times 9.6] \text{ Gbps, 10 us, 4 ms})\) while increasing \(N\) from 5 to 100. Table 5 illustrates the estimated available bandwidth when different train lengths were used. The actual available bandwidth is shown in the row marked “Actual” and the estimated available bandwidth is in the rows below “Length” in Table 5. As shown in the table, increasing the number of probe packets per flow yields diminishing returns. A probe train with five packets is not as accurate as the baseline experiment with 20 packets regardless of the actual available bandwidth. However, trains with larger than 20 packets result in similar estimation accuracy. The take-away: Increasing the number of packets in a probe packet train does not necessarily result in more accurate estimation. The minimum number of packets to obtain accurate estimation results was about 20 packets. More packets in a probe train elongate the delay of applications traffic due to buffering, but do not improve estimation accuracy.

What happens when there is not enough packets between a source and destination pair? This highlights the fundamental trade-off between application latency and probing overhead. MinProbe has a timeout mechanism to deal with this corner case. If the oldest intercepted packet has exceeded a pre-defined timeout value, minProbe will generate dummy probe packets to fulfill the need for probe packets and send out the probe train. Since minProbe requires much shorter probe trains compared to existing tools, opportunistically inserting additional dummy packets add minimal overhead.

### 4.3.3 Mixed Probe Packet Sizes

In a real network, the distribution of packet sizes is usually mixed with large and small packets. Figure 10 shows the distribution of packet sizes in a recorded Internet trace. We repeat the previous experiment using the baseline setting, except that in this case, we use APP0 to replay the Internet trace. We let MP0 modulate the probe train in presence of mixed-sized probe traffic. For each rate \(R \in [0.1, 9.6]\), MP0 ensures that the interpacket gaps between packets of the same flow are uniform. Figure 11 shows the result for CBR1 cross traffic. As can be seen, even though a probe train has mixed packet sizes, the estimation result is still accurate. The reason is that small packets only take a small fraction of total probe train length with respect to large packets. The estimation accuracy is largely determined by the large packets, thus remains accurate.

### 4.3.2 Impact of Varying Probe Size

Next, we evaluated minProbe with different packet sizes for probes. All the parameters were the same as in the baseline experiment except that we varied the size of packets for probes from 64 bytes to 1518 bytes (the minimum to maximum sized packets allowed by Ethernet). In particular, we used \((20, [0.1 \times 0.1 \times 9.6] \text{ Gbps, 10 us, 4 ms})\) while increasing probe packet sizes from 64 to 1518 bytes. As can be seen from Table 6, larger packet sizes (more than 512 bytes) perform better than smaller packet sizes in general. Note that we used 792 byte packets in our baseline experiment, which resulted in similar available bandwidth estimate accuracy as 1518 byte probe packets. On the other hand, smaller probe packets, such as 64 byte packets resulted in poor accuracy. This result and observation is consistent with the previous results from literature such as [40]. The take-away is that probe trains with medium to large packet sizes perform better than trains of smaller packet sizes.

### 4.4 Impact of Cross Traffic Characteristics

So far, we have evaluated minProbe under the assumption of constant bit-rate cross traffic. In this section, we no longer make this assumption and instead examine minProbe under bursty cross traffic. In particular, we evaluate minProbe using modeled synthetic bursty traffic (CBR2) and realistic Internet traces.

#### 4.4.1 Cross Traffic Burstiness

In order to understand how the number of packets in a probe packet train affects the accuracy of minProbe, we varied the number of packets in each train while using the same parameters as the baseline experiment (Section 4.2). In particular, we used \((N, [0.1 \times 0.1 \times 9.6] \text{ Gbps, 10 us, 4 ms})\) while increasing \(N\) from 5 to 100. Table 5 illustrates the estimated available bandwidth when different train lengths were used. The actual available bandwidth is shown in the row marked “Actual” and the estimated available bandwidth is in the rows below “Length” in Table 5. As shown in the table, increasing the number of probe packets per flow yields diminishing returns. A probe train with five packets is not as accurate as the baseline experiment with 20 packets regardless of the actual available bandwidth. However, trains with larger than 20 packets result in similar estimation accuracy. The take-away: Increasing the number of packets in a probe packet train does not necessarily result in more accurate estimation. The minimum number of packets to obtain accurate estimation results was about 20 packets. More packets in a probe train elongate the delay of applications traffic due to buffering, but do not improve estimation accuracy.

What happens when there is not enough packets between a source and destination pair? This highlights the fundamental trade-off between application latency and probing overhead. MinProbe has a timeout mechanism to deal with this corner case. If the oldest intercepted packet has exceeded a pre-defined timeout value, minProbe will generate dummy probe packets to fulfill the need for probe packets and send out the probe train. Since minProbe requires much shorter probe trains compared to existing tools, opportunistically inserting additional dummy packets add minimal overhead.
Figure 13a shows an example of the raw data captured dataset [6], and replayed by SoNIC as a traffic generator. Real Internet Trace: cross traffic is bursty. The traffic was routed through the path shown in Figure 5 across over 2500 miles. Since it was difficult to obtain accurate readings of cross traffic at each hop, we relied on the router port statistics collected every 30 seconds. Specifically, we found using the EMA smoothed the measurement data and improved the estimation accuracy. As a result, we observed that cross traffic burstiness had limited impact on minProbe.

4.5 MinProbe In The Wild

In this section, we demonstrate that minProbe works in a wide area network, the National Lambda Rail (NLR). Figure 5 illustrates the experimental network topology. We setup a dedicated route on the NLR with both ends terminating at Cornell University. Node APP0 generated application traffic, which was modulated by MP0 using parameter \((N, R, G, D) = (20, [0.1 : 0.1 : 0.6])\) Gbps, 10 us, 4 ms). The traffic was routed through the path shown in Figure 5 across over 2500 miles. Since it was difficult to obtain accurate readings of cross traffic at each hop, we relied on the router port statistics to obtain a 30-second average of cross traffic. We observed that the link between Cleveland and NYC experienced the most cross traffic compared to other links, and was therefore the bottleneck (tight) link of the path. The amount of cross traffic was 1.87 Gbps on average, with a maximum of 3.58 Gbps during the times of our experiments.

Figure 14 shows the result of minProbe in the wild. We highlight two important takeaways: First, the average of our estimation is close to 2 Gbps, which was consistent with the router port statistics collected every 30 seconds. Second, we observed that the readings were very bursty, which means the actual cross traffic on NLR exhibited a good deal of burstiness. In summary, minProbe performed well in the wild, in the wide area. The available bandwidth estimation by minProbe agreed with the link utilization computed from switch port statistical counters: The mean of the minProbe estimation was 1.7Gbps, which was close to the 30-second average 1.8Gbps computed from switch port statistical coun-
baseline setup, with the exception that software routers were
L2 forwarding from one port to another. Again, we used the
driver [18]. We configured the software routers to perform
network datapath, and another based on the Intel DPDK
performed two software routers, one based on the Linux kernel
rated with data rates ranging according to (N, R, G, D) = (20,
1Gbps virtual network, if the probe train was less than 1200
packets, then there was no observable packet loss. As has
been shown in Section 4.3, minProbe only uses probe trains
with less than 100 probe packets. As a result, and interest-
ingly, rate limiters do not impact the applicability of min-
Probe in federate testbed environments.

4.6 Software Router and Other Hardware

Finally, we evaluate whether other middleboxes, such as
software routers, can be used to perform high-fidelity net-
work measurements required to accurately estimate avail-
able bandwidth without any overhead. In particular, we eval-
uated two software routers, one based on the Linux kernel
network datapath, and another based on the Intel DPDK
driver [18]. We configured the software routers to perform
L2 forwarding from one port to another. Again, we used the
baseline setup, with the exception that software routers were
placed between minProbe and the switch. The network path
started at the application. After minProbe modulated the
application traffic to create probe packet trains, the probe
trains were passed through a software router before being
sent to a 10 GbE switches.

The experiment was generated from probe packet trains
with data rates ranging according to (N, R, G, D) = (20,
[0.5 : 0.5 : 9.5] Gbps, 10 us, 4 ms). We show two figures
from this experiment to illustrate the result. First, Figu-
re 15(a) shows the CDF of the measured instantaneous data
rate of probe packet pairs passing through a software router
(i.e. we measured the interpacket gap of probe packet pairs
after passing through a software router). We configured each
software router to forward each packet as soon as it received
the packet to minimize any batching. Since we generated an
equal number of probe packets (20) for a probe train and
increased the probe packet train data rate by 0.5 Gbps, the

Figure 15: Software Routers do not exhibit the same fidelity
as minProbe.

ideal CDF curve should increase along a diagonal frp, 0.5
Gbps to 9.5 Gbps at 0.5 Gbps increments (the red line).
However, as seen in the figure, over 25% of the probe packet
pairs in the Intel DPDK-based software router were batched
with minimum interpacket gap: Batching can be seen via the
large increase in the CDF near the highest data rate close
to 10 Gbps. Worse, the vanilla kernel datapath (with ixgbe
kernel driver) batched over 40% of packets. These results
were consistent with the existing research findings that show
that network interface cards (NICs) generate bursty traffic
at sub-100 microsecond timescale. [25]

Second, Figure 15(b) highlights the same result but slightly
differently. The x-axis is the packet number and the y-axis
is the measured instantaneous probe packet pair rate (in
Gbps). The figure shows that minProbe was able to modu-
late interpacket gaps to maintain the required probe data
rate (all 20 packets in a probe train exhibited the target
data rate), while the software routers were not able to con-
trol the probe rate at all. To summarize, due to batching,
software routers are not suitable for performing the required
high-fidelity available bandwidth estimation on high-speed
networks.

Next, we investigated the question if prior work such as
MGRP [37] could perform on high-speed networks? To an-
swer this, we setup MGRP to perform the same baseline es-
timation experiment in Figure 8. Unfortunately, the result
was consistent with our findings above and in Figures 15a
and 15b. In particular, we found that MGRP could not rep-
port valid results when it was used in high-speed networks
(10Gbps Ethernet). The accuracy of MGRP was limited
by its capability to precisely timestamp the received probe
packets and its ability to control the probe packet gaps. Both
of these capabilities were performed at kernel level in MGRP
and were prone to operating system level noise. The ac-
curacy of MGRP was better than Pathload [20], it eliminated
the overhead of extra probing traffic, but the fundamental accuracy was still constrained by the measurement platform.

Finally, we studied the sensitivity of bandwidth estimation with respect to network hardware. In particular, we experimented with three different 10 GbE switches: IBM G8264T, Dell S4810, and NetFPGA-10G reference switch design [31]. We used the baseline setup for all experiments and configured all three switches to perform L2 forwarding. Figure 16 illustrates an example of captured queuing delay after each hardware. As can be seen from the figure, the three switches can be used nearly interchangeably since there was no significant difference in queuing delay. As a result, we have found that the underlying hardware itself may not contribute to the available bandwidth estimation. Instead, as earlier results in the paper indicate, the fidelity of the available bandwidth estimation depend more on the ability to control generated interpacket gaps and measure received interpacket gaps.

5. DISCUSSION

In this section, we discuss the issues regarding the deployment of minProbe, the applications of available bandwidth estimation, and whether minProbe can be generalized to measure bandwidth in 1GbE and 40GbE.

5.1 Deployment

Middleboxes are widely deployed in datacenter networks to perform a wide spectrum of advanced network processing functions ranging from security (firewalls, intrusion detection), traffic shaping (rate limiters, load balancers), to improving application performance (accelerators, proxies, caches), to name a few. Similarly, minProbe deployment scenarios include middleboxes with enhanced measurement capabilities potentially located in the same rack as other servers, or even integrated into each server directly, possibly using hypervisor-based network virtualization, provided that there exists proper support for accessing the physical layer of the network in software. In the future, we will explore the possibility of deploying minProbe as a platform for providing high-fidelity network measurement as a service (NMaaS).

5.2 Virtualization and Availability Bandwidth Estimation

Because of virtualization, datacenter tenants no longer have direct access to the underlying physical network. The extra layer of indirection makes it hard to optimize application performance. MinProbe could potentially bridge the gap. It can measure the underlying network and provide performance metrics to the virtualized applications to optimize tasks such as VM placement and route selection.

5.3 Generalization to 1GbE and 40GbE

At the time of writing, minProbe operated only in 10 GbE networks. However, the high-fidelity capability of minProbe can be extended to support 1GbE and 40GbE networks as well. The architecture of the PHY for 40GbE networks is very similar to the PHY in a 10GbE network. As a result, as future work, we intend to extend minProbe to support 40GbE. Support for 40GbE networks will require parallelizing the processing the PHY layer functionality using modern multi-core architecture. Moreover, as future work, we intend to extend minProbe to support 1GbE networks. The PHY for 1GbE networks are less complicated than 10GbE PHY, which makes it possible to port minProbe to 1GbE. For example, the encoding and decoding schemes at physical coding layer in 1GbE is simpler than the 10GbE physical coding layer.

5.4 GENI

MinProbe is available on ExoGENI [10]. Users can provision compute nodes that have SoNIC cards installed and use it as a minProbe to run network measurement experiments. Currently, at the time of writing, minProbe was enabled at two ExoGENI sites: RENCI (Renaissance Computing Institute) and University of California, Davis. We are in the process of expanding to more sites in the near future.

6. RELATED WORK

Prior work in MAD [43], MGRP [37] and Periscope [15] provide a thin measurement layer (via userspace daemon and kernel module) for network measurement in a shared environment. These works complement minProbe. MinProbe is applicable in a shared environment. We also advance the conversation of available bandwidth estimation to 10Gbps, unlike most of the prior work, which were only demonstrated to operate in 1Gbps or less.

We use an algorithm similar to Pathload [20] and Pathchirp [40] to estimate available bandwidth. There are many related works on both theoretical and practical aspect of available bandwidth estimation [7, 20, 23, 29, 30, 32, 38, 40, 42, 44]. Our work contributes more on the practice of available bandwidth estimation in high speed network (10Gbps). We found that existing probing parameterizations such as probe trains are sound and still applicable in high-speed networks when supported by the appropriate instrumentation. Furthermore, [25,48] mentioned that burstiness of the cross traffic can negatively affect the accuracy of estimation. While we found that bursty traffic introduces noise into the measurement data, the noise can be filtered out using simple statistical processing, such as the moving average.

Middleboxes are popular building blocks for current network architecture [41]. MinProbe, when operating as a middlebox, is similar to software routers in a sense that minProbe consists of a data forwarding path and programmable flow table. Unlike software routers, which typically focus on high throughput, minProbe has the capability to precisely control the distance between packets; this capability is absent in most existing software routers. From the system design perspective, ICIM [34] has proposed a similar inline network measurement mechanism as our middlebox approach.
However, ICIM has only simulated their proposal, while we implemented a prototype and performed experiment in real network. Others have investigated the effect of interrupt coalescence [39] and memory I/O subsystem [24], but they are orthogonal to our efforts. MinProbe, albeit a software tool, avoids the noise of OS and network stack completely by operating at the physical layer of the network stack.

Another area of related work is precise traffic pacing and precise packet timestamping. They are useful system building blocks for designing a high-fidelity network measurement platform. Traditionally, traffic pacing [8] is used to smooth out the burstiness of the traffic to improve system performance. In minProbe, we use traffic pacing in the opposite way to generate micro-burst of traffic to serve as probe packets. Precise timestamping has been used widely in passive network monitoring. Typically, this is achieved by dedicated hardware platform, such as Endace Data Acquisition and Generation (DAG) card [1]. With minProbe, we achieve the same nanosecond timestamping precision, and we are able to use the precise timestamp for active network measurement, which the traditional hardware platform are incapable of.

7. CONCLUSION

Available bandwidth estimation as a research topic has been studied extensively, but there still lacks an accurate and low cost solution in high-speed networks. We present minProbe, which is a high-fidelity, minimal-cost and userspace accessible network measurement approach that is able to accurately measure available bandwidth in 10 gigabit-per-second (Gbps) networks. We provided a systematic study of minProbe’s performance with respect to various cross traffic and probe traffic conditions and patterns. We found that minProbe performs well even under bursty traffic, and the estimation is typically within 0.4 Gbps of the true value in a 10 Gbps network.

8. AVAILABILITY

The minProbe and SoNIC source code is published under a BSD license and is freely available for download at http://sonic.cs.cornell.edu. SoNIC nodes are available for experimentation as bare-metal nodes at the following ExoGENI sites: RENCI and UCDavis.

9. ACKNOWLEDGMENTS

This work was partially funded and supported by an Intel Early Career and IBM Faculty Award received by Hakim Weatherspoon, DARPA (No. D11AP00266), DARPA MRC (No. FA8750-11-2-0256), NSF CAREER (No. 1053757), NSF TRUST (No. 0424422), NSF FIA (No. 1040689), NSF CiC (No. 1047540), and NSF EAGER (No. 1151268). We would like to thank our shepherd, Oliver Spatsch, and the anonymous reviewers for their comments. We further recognize the people who helped establish and maintain the network infrastructure on which we performed our experiments: Eric Cronise and Laurie Collinsworth (Cornell Information Technologies); Scott Yoest, and his team (Cornell Computing and Information Science Technical Staff). We also thank the people who provided support for us to perform experiments on GENI and ExoGENI: Ilya Baldin (RENCI) and his team (RENCI Network Research and Infrastructure Program), Jonathan Mills (RENCI), and Ken Gribble (UC Davis).

10. REFERENCES

http://www.endace.com/
endace-dag-high-speed-packet-capture-cards.html.


