Efficient Techniques for End-to-end Bandwidth Estimation: Performance Evaluations and Scalable Deployment

Alok Shriram

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2009

Approved by:

Jasleen Kaur, Advisor
F. Donelson Smith, Reader
Kevin Jeffay, Reader
Ketan Mayer-Patel, Reader
Constantine Dovrolis, Reader

ABSTRACT

ALOK SHRIRAM: Efficient Techniques for End-to-end Bandwidth Estimation:
Performance Evaluations and Scalable Deployment.
(Under the direction of Jasleen Kaur)

Several applications, services, and protocols are conjectured to benefit from the knowledge of the end-to-end available bandwidth on a given Internet path. Unfortunately, despite the availability of several bandwidth estimation techniques, there has been only a limited adoption of these in contemporary applications. We identify two issues that contribute to this state of affairs. First, there is a lack of comprehensive evaluations that can help application developers in calibrating the relative performance of these tools—this is especially limiting since the performance of these tools depends on algorithmic, implementation, as well as temporal aspects of probing for available bandwidth. Second, most existing bandwidth estimation tools impose a large probing overhead on the paths over which bandwidth is measured. This can be a significant deterrent for deploying these tools in distributed infrastructures that need to measure bandwidth on several paths periodically.

In this dissertation, we address the two issues raised above by making the following contributions:

- We conduct the *first* comprehensive black-box evaluation of a large suite of prominent available bandwidth estimation tools on a high-speed network. In this evaluation, we also illustrate the impact that technological and implementation limitations can have on the performance of bandwidth-estimation tools.

- We conduct the *first* comprehensive evaluation of available bandwidth estimation algorithms, independent of systemic and implementation biases. In this evaluation, we also illustrate the impact temporal factor such as measurement timescales have on the observed relative performance of bandwidth-estimation tools.

- We demonstrate that temporal properties can significantly impact the AB estimation process. We redesign the interfaces of existing bandwidth-estimation tools to allow temporal parameters to be explicitly specified and controlled.

- We design AB inference schemes which can be used to scalably and collaboratively infer the available bandwidth for a large set of end-to-end paths. These schemes allow an operator to select the desired operating point in the trade-off between accuracy and overhead of AB estimation. We further demonstrate that in order to monitor the bandwidth on all paths of a network we do not need access to per-hop bandwidth estimates and can simply rely on end-to-end bandwidth estimates.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

1.1 Motivation

Why measure end-to-end available bandwidth? The Internet provides a best-effort service model—applications that transfer data over the Internet are provided no guarantees or even apriori knowledge about the end-to-end performance that their transfers should expect. Such knowledge, if available, can help applications in better aligning their configuration and data transmission with the current state of network resources. Consequently, many applications rely on mechanisms or protocols that *probe* network paths to estimate the level of end-to-end transfer performance that a given network path can provide.

Over the past two decades, there has been significant interest in designing mechanisms for specifically probing for the *packet delay* and *packet loss* characteristics of Internet paths, and in using these to help guide data transport by Internet applications [BOP94, AGKT98, AP99]. More recently, with the emergence of data-intensive applications and increasing deployment of broadband access networks, there is a growing need to focus on yet another characteristic that applications are likely to be interested in—that of the *transfer rate* that can be obtained on a given path. The *end-to-end available bandwidth (AB)*—which represents the maximum spare capacity available among all links of a given path—has emerged as a prime metric of choice for such applications. Several application domains are conjectured to benefit from the knowledge of this quantity:

- *Congestion-control Protocols:* A key objective of congestion-control protocols is to determine the maximum rate at which data can be transferred over a given path without overloading the network resources. It can be clearly seen that such a rate would be the maximum of the spare bandwidth available across all links on the path—consequently, efficient mechanisms for estimating the end-to-end AB can be quite useful for guiding such protocols.

- *Video Streaming Protocols:* Video-streaming applications often adapt the bit-rate of the video-stream by changing encoding parameters—a low-quality, low bit-rate stream is sent when a high bit-rate can not be supported on a given network path [BGMS04, FBB01]. Efficient mechanisms that continually estimate the end-to-end AB can help reconfigure the encoding parameters adaptively.

- *Audio Streaming:* Some applications require only a low, but consistent bit-rate—for instance, audio streaming generates data at a nearly constant rate of 50-100 Kbps and works well only when the path can support the rate [Mos08]. Even such applications can benefit from the knowledge of end-to-end AB in order to decide, for instance, if a new client request for an audio stream can be served (given characteristics of the AB on the Internet path from the server to the client).

- *Server Selection:* For content-based services, in which a desired content may be available at several servers, the knowledge of the end-to-end AB on the paths from each server to a given client can help the client select the best server to download from—this is especially useful for clients interested in downloading large files or streams.

- *Overlay Routing:* With the emergence of multi-homing and overlay infrastructures [ABKM01, AMS+03, RK03], applications can now choose to transfer data over alternate paths different from the default Internet path to a destination. The selection of an alternate path can be better informed using the knowledge of end-to-end AB on each candidate path.

Each of the above applications would benefit from efficient techniques that probe for the end-to-end AB on a given set of path(s).

The dilemma: *which probing technique?!* Several sophisticated techniques have been developed in recent literature for measuring end-to-end AB on a given network path [JD02b, Rib03, SKK03a, Nav03, HS03, CC96a, ipe, KV06]. Unfortunately, there has been only a limited adoption of these techniques in Internet protocols/services. Indeed, most applications continue to use legacy mechanisms available prior to the emergence of these techniques. We believe that there are two key reasons for this state of affairs. First, there is a lack of comprehensive evaluations of these techniques—consequently, it is not clear which techniques (if any) are efficient and well-suited for a given application. Second, many of the recently-developed techniques rely on sending large amounts of probe traffic for accurately estimating end-to-end AB. The associated overhead and latency of probing—especially when used in popular services/protocols—acts as a significant deterrent for protocol designers [SMH+05].

In this dissertation, our goal is to help alleviate these two issues. We precisely formulate our goals next.

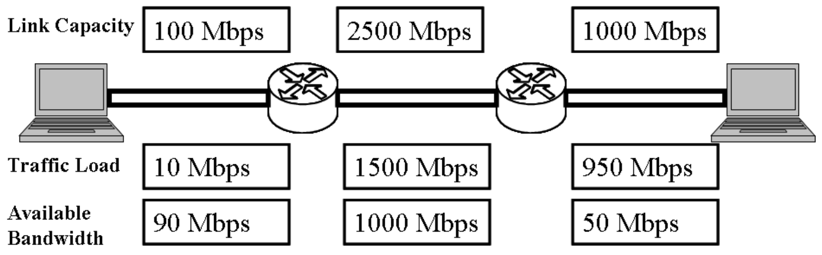


Figure 1.1: Illustration of end-to-end AB

1.2 Dissertation Goals

We begin by first formally defining the concept of end-to-end available bandwidth and identifying the requirements from tools used for estimating it.

1.2.1 Requirements from ABETs

End-to-end available bandwidth (AB) represents the maximum spare capacity available among all links of a given path. Formally, the *per-hop* bandwidth available on i_{th} link of a path, AB_i, is defined as:

$$AB_i[t_1, t_2] = C_i - \frac{B_i(t_1, t_2)}{(t_2 - t_1)} \tag{1.1}$$

where $AB_i[t_1, t_2]$ is the spare bandwidth available on the link i over the time interval $[t_1, t_2]$, C_i is the transmission capacity of link i, and $B_i(t_1, t_2)$ is the total traffic transmitted on the link during $[t_1, t_2]$. The *end-to-end* available bandwidth of a network path is defined as the minimum of the spare bandwidth available at each of the constituent links of the path:

$$AB_{e2e} = \min_{i \in [1,N]} \{AB_i\} \tag{1.2}$$

For instance, in Fig 1.1, the end-to-end AB is 50 Mbps.

AB estimation tools, henceforth also referred to as ABETs , are designed to estimate the end-to-end AB on a given network path. Some distributed applications—such as overlay routing and server selection—also attempt to simultaneously employ such tools on multiple paths. Several *key requirements* guide the design of ABETs :

- *High Estimation Accuracy:* Each of the example applications listed before would be able to make optimal application-level decisions only if provided with an *accurate* knowledge of AB. For instance, underestimation of end-to-end AB would prevent video-streaming applications and congestion-control protocols from, respectively, maximizing application quality and throughput—while overestimation of AB would drive network resources into a persistently overloaded state for the duration of the transfers.

- *Small Response Time:* AB on a given link can vary significantly over time—Fig 1.2, which plots the AB observed on a production Internet link during a 30-second interval, illustrates this. Consequently, many applications—such as video streaming and congestion-control protocols—would need to continuously estimate up-to-date values of AB and adapt their transmission behavior accordingly. An ABET with a *large* response time would deliver only stale (and possibly invalid) values of AB to the associated application, thus impairing application performance.

- *Low Probing Overhead:* In order to prevent interference with transmission of useful application data, ABETs should themselves rely on sending few probes into the network. Furthermore, a high probing overhead would be a significant deterrent to the widespread adoption of these techniques in popular Internet protocols and services.

1.2.2 (Limitations of the) State of the Art

Lack of Comprehensive Evaluations Several tools (ABETs) have been proposed in recent literature for actively probing for the end-to-end AB on a given network path [CC96a, HS03, JD02b, Rib03, SKK03a, Jin04, MBG00, Nav03]. These tools typically operate by injecting specially-designed streams of probe packets onto the path, observing the end-to-end delays experienced by the probe packets, and then estimating the end-to-end AB from the observations—details can be found in Chapter 2. Unfortunately, it is not clear how well these tools meet the above requirements. Specifically, existing evaluations of ABETs suffer from the following fundamental limitations.

1. While most ABET designers conduct evaluations and comparisons of their tools against other tools, these evaluations are neither comprehensive in the tools nor the settings evaluated. All previous evaluation studies evaluate only a small (and different) sub-set of ABETs, depending on which ones were popular when the corresponding tool was proposed. Furthermore, these evaluations include only simple network and traffic scenarios. For instance, most do not evaluate tool performance against responsive cross-traffic, in high-speed networks, or when the tight and narrow links are different. As a result of these practices, the

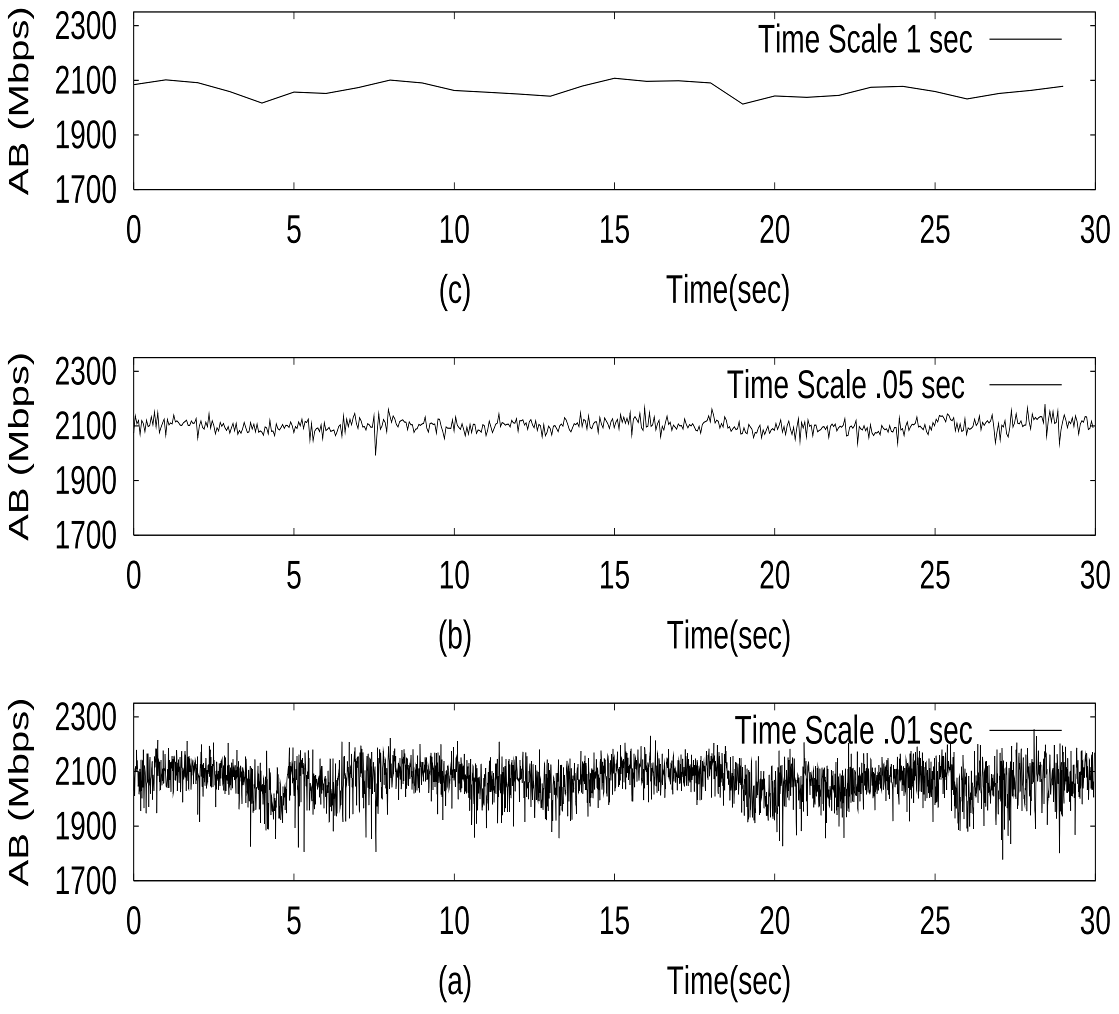


Figure 1.2: AB process observed on an Internet link over the same 30 sec interval at different timescales.

results are often not comprehensive and get inadvertently biased toward highlighting the salient features of the proposed tool. This state of affairs leads us to our first goal:

It is important to conduct comprehensive evaluations of prominent AB estimation tools under a common set of diverse network and traffic settings.

2. In current high-speed networks, variations in end-to-end delays may have an order of magnitude in the sub-millisecond range. Since most ABETs rely on measuring delay variations, high resolution and accuracy in time-stamping probe packets is crucial for ensuring the accuracy of the inferred AB. Current PC platforms, however, are incapable of guaranteeing high time resolution due to multi-tasking and the use of mechanisms such as interrupt coalescence [JD02b]. Some tool implementers work around this limitation by relying on statistical filtering and smoothing techniques, but others do not—the use of such techniques does impact tool performance significantly [AMPR03, JT03].

   It is important to note that the above implementation techniques are highly technology-specific. As technology improves [PV02a], the impact of these techniques on ABET performance is likely to diminish. It is, therefore, natural to ask the question: what is the extent to which *current* implementation technology limits tool performance? In particular, *how well would tool designs—including the design of their probe-streams and inference logic—perform if technology advances in the future?* This leads us to our second goal:

   It is important to evaluate, in an implementation-agnostic manner, the algorithmic aspects of prominent ABETs .

3. Existing ABET designs focus primarily on, and differ most significantly in, the construction of probe streams and in the logic used to estimate AB from the observed delays. Most tool designs, however, seem to ignore three central temporal quantities related to measurement of the AB process—that of the measurement time-scale, the sampling intensity and strategies, and the probing duration. In particular, most existing ABETs do not allow the choice of these quantities and little is known about the impact of these quantities on the performance characteristics of an ABET. This leads us to our third goal:

   It is important to study the impact of probing-related temporal quantities—including the measurement timescale, sampling intensity and strategy, and probing duration—on the estimation of AB.

In summary, our first set of objectives are concerned with evaluating ABETs under diverse settings of network, traffic, and temporal conditions.

Significant Overhead in Multi-path Application Domains Another fundamental issue with the state-of-the-art in ABET design is the large overhead when these tools are deployed in services such as overlay routing or server selection, that need to measure AB simultaneously on *multiple* paths. A naive approach for such services would be to run an instance of the tool on each of the N^2 paths in an N-node overlay infrastructure. However, such an approach suffers from two significant limitations:

1. *High Overhead:* Even with a low overhead tool, a single measurement of the AB may inject on the order of several megabytes of traffic into the network. Conducting N^2 measurements, would significantly overload the network path, especially as the number of nodes in the overlay infrastructure increases.

2. *High Response-Time:* ABETs can interfere with each other if run simultaneously on paths that share congested links. Hence, for the set of paths that interfere with each other, the measurements need to be run sequentially. In the worst case running N^2 measurements sequentially would fundamentally limit the frequency with which the AB information for all paths can be updated.

The above limitations lead to our final goal:

It is important to design a scalable AB monitoring scheme for distributed infrastructures, in which the number of measurements that need to be made scale well with the size of the infrastructure.

In this dissertation, we pursue the four goals identified above by: (i) conducting a systematic evaluation of *implementations* of prominent ABETs ; (ii) studying the impact of *temporal factors* on the AB estimation process; (iii) conducting an implementation-agnostic evaluation of the AB estimation techniques; and (iv) designing a scalable AB estimation scheme for simultaneous and distributed monitoring of multiple paths. In what follows, we briefly summarize the approach and main results for each of these—the details follow in subsequent chapters.

1.3 Goal 1: Black-box Evaluation of ABET Implementations

Our first goal is to evaluate prominent tools designed for estimating end-to-end AB. For this, we rely on the publicly-available implementations of these tools and understand the accuracy, overhead, and response-times of these implementations under diverse network and traffic conditions. We are especially interested in testing the network speeds to which these implementations can scale—this is because at high speeds, the lack of high-precision and accurate timers on current PC platforms could impair the performance of such tools. Our hope is to identify tool implementations that are suitable for contemporary application domains.

Approach With the above goals in mind, we conduct ABET evaluations under each of the following scenarios:

- On a high-speed network test-bed with commercial routers and switches, with constant bit-rate cross traffic.
- On a high-speed network test-bed with commercial routers and switches, with traces of representative traffic collected from production Internet links and replayed as cross-traffic on the test-bed.
- On an Internet2 gigabit network path between Sunnyvale and Atlanta against real cross-traffic flowing on the path—the actual AB was verified using SNMP counters on the network path.
- On the Internet path between San Diego Supercomputing center and Oak-Ridge National Laboratories, where actual AB was unavailable but the relative performance of the tools was studied.

Summary of Results Some highlights of the findings of our evaluation can be summarized as follows:

1. Tools utilizing packet-pair techniques like Abing and Spruce should be aware of delay quantization possibly present in the networks.
2. AB can not be measured reliably in gigabit high-speed networks using 1500 Byte MTUs and with only microsecond time-stamp resolution.
3. ABETs should also be able to detect, and perform well in, the presence of interrupt coalescence.
4. TCP-based bandwidth estimation schemes like Iperf perform well, but even an approximately 1% packet loss can severely affect AB estimates.

1.4 Goal 2: Impact of Temporal Factors on AB Estimation

Temporal Factors We identify three important temporal aspects of the process of sampling AB by any bandwidth estimation tool:

- *Measurement Timescale:* A critical parameter in the definition of AB in Equation (1.1) is the length, $(t_2 - t_1)$, of the time interval over which it is observed—we refer to this quantity as the *measurement timescale* (MT). In Fig 1.2, we plot the time-series of AB, observed at three different timescales of $10ms$, $50ms$, and $1s$, during the *same 30 s* observation period on an Internet link. We observe that the AB process can appear quite different depending on the timescale at which it is observed. In particular, it is likely that the MT impacts the accuracy as well as variability of the AB measured by a given ABET. Consequently, any application that relies on such a tool would want the tool to measure AB at an MT relevant to the application domain. For instance, while a large-file-transfer application is likely to be interested in only the average AB obtainable at super-second timescales, a media-streaming application is likely to also be interested in knowing the small-timescale variations in AB.

- *Measurement Duration (Run-time):* *Run-time* (RT) refers to the length of the time interval over which several samples of the AB process are collected, and used to infer properties of the AB process. In practical terms, the run-time is the total time taken by a tool from invocation to reporting an AB estimate. This includes the time taken for sending several probe streams (each of which potentially returns one sample of AB), and converging on an AB estimate.

  The most significant impact of run-time on AB measurement is in terms of its robustness and its variability. While a shorter run-time is likely to yield samples more consistent with each other, longer run-times are more likely to yield a *sufficient* number of samples for reliably estimating the mean as well as variability in the AB process.

- *Sampling Intensity and Strategy:* Given an observation timescale, the *AB process* consists of a series of back-to-back readings of AB observed within a given time interval. ABETs essentially only *sub*-sample this AB process. Existing tools differ in the fraction of the AB process—henceforth, referred to as the *sampling intensity* (SI)—that they sample during the tool run-time. Existing tools also differ in their *sampling strategy*—the manner in which AB samples are collected from within a given time interval [CPB93]. The sampling strategy and the fraction of the AB process sampled are likely to impact the accuracy of estimating the mean AB in a given time interval. For instance, larger is the sampling rate, better is likely to be the AB estimation accuracy; however, greater would be the network overhead.

In this dissertation, we study how the choice of MT, RT, SI, and sampling strategy by a tool impacts the accuracy, variability, and stability of the measured AB. We organize the relevant issues in the form of three main questions:

(i) How does the choice of sampling strategy, sampling intensity, MT and RT impact the *accuracy* of the estimated AB? (ii) How does the choice of MT and RT affect the *variability* of the measured AB? (iii) How *stable* is AB in the post-measurement periods? Answering these questions reveals the impact that the above mentioned temporal parameters have on the performance characteristics of the ABETs .

Evaluation Approach As mentioned before, existing ABETs differ significantly in the design of their probe streams and inference logic. In order to answer the questions raised above in a manner independent of these choices, we assume the existence of a perfect tool that can estimate the end-to-end AB perfectly. This assumption lets us study the impact of the above-identified (currently) design-agnostic quantities—namely, run-time, measurement timescale, and sampling intensity and strategy—while isolating the analysis from the impact of design-dependent factors. It also lets us adopt a *passive* trace-analysis based approach for answering the above questions. With such an approach, it is possible for us to compute the *ground truth* about the AB process.

We conduct passive analysis of link-level traces collected from several types of production Internet links.

Summary of Results The main findings of our evaluation can be summarized as follows:

1. The choice of the measurement timescale does not impact the accuracy as long as the sampling intensity is held constant.

2. Sampling more that 30% of the AB process does not yield a significant improvement in the accuracy of the AB estimate.

3. The AB process shows significant variability at timescales of less than 50 ms, which corresponds to sending a large train of packets rather than packet-pairs.

4. Back-to-back measurements of the AB do not change significantly, which can be exploited for predictive purposes in applications that need to continuously estimate AB.

1.5 Goal 3: Implementation-agnostic Evaluation of ABETs

Approach In order to evaluate the performance of prominent ABET designs in an implementation-agnostic manner, we adopt a two-pronged approach. First, we rely on a simulation-based evaluation approach using the NS-2[NS2] simulator. Adopting such an approach eliminates all sources of systemic biases, since the simulated environment gives us the ability to create a *technologically-perfect network* where (i) fine-grained clock granularity is achievable on end-systems, (ii) interrupt coalescence effects do not occur, (iii) packet losses do not occur and, (iv) buffer sizes are unlimited. Second, we design a common implementation framework for instantiating prominent ABETs in NS-2—a common framework helps us avoid any bias due to differences in implementation efficiencies. It also allows us to enhance existing interfaces for all ABETs to enable controlling the measurement timescale and sampling intensity. We extract implementation details from the publicly available versions of the tools and the publications that describe them, and implement these within our framework. We then evaluate the ABETs under diverse settings of traffic and network conditions, including: (i) single-hop topologies with constant-bit-rate as well as dynamic and representative cross-traffic, (ii) multi-hop topologies with multiple tight and/or narrow links, and (iii) on topologies with a representative mix of *responsive* cross-traffic. We study several performance characteristics (Table 1.1) of the ABETs .

Performance Parameter	**Description**
Accuracy	Difference between the actual and measured AB
Overhead	Traffic injected by ABET to make single AB measurement
Intrusiveness	Rate of traffic injected by ABET to make single AB measurement
Run-Time	Time taken to make a single AB measurement
Perturbation	Impact on response times of TCP flows

Table 1.1: Performance characteristics of an ABET

Summary of Results The main findings of our evaluation can be summarized as follows:

1. Increasing the MT improves the accuracy of the estimates. However, the gains in accuracy are negligible beyond a timescale of 50 ms.

2. Increasing the SI has *no* impact on the accuracy of the AB estimates.

3. As the MT increases the run-time of a tool also increases.

4. Pathload has the most overhead, while Pathchirp has the least. This is especially true when assessing the impact of these tools on responsive TCP cross-traffic.

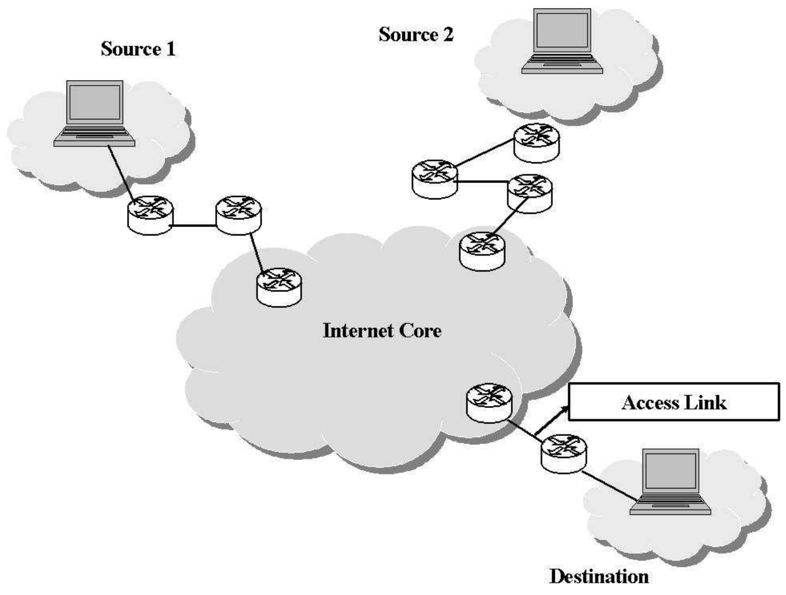


Figure 1.3: Internet Architecture

1.6 Goal 4: Scalable AB Inference for Overlays

In order to design a scheme, which can give us complete N^2 path AB without making N^2 measurements we rely on two key insights about the Internet

1. *Sharing of the access hops:* Most of the Internet architecture is structured as illustrated in Fig 1.3. End-nodes either lie within enterprise networks or metropolitan-area networks belonging to local Internet Service Providers (ISPs). These edge networks are then connected via access links to other ISPs that comprise the core of the Internet—these include major Tier-1 and Tier-2 ISPs. Let the *access segment* of an end-node refer to the sequence of edge hops and links that connect it to the Internet core. The connectivity structure of the Internet then implies that the paths from any two given source nodes, S_1 and S_2, to a common destination D are likely to share the access segment of D (see Fig 1.3).[1]

2. *Bottlenecks lie close to edges:* Recent Internet-wide measurement studies have reported that the bottleneck links of most end-to-end network paths lie close to the end-nodes, and are likely to be the access/peering links between customers and ISPs [Li05, RRB04]. This essentially implies that the bottleneck link of a path is likely to lie on the access segments of either end-nodes.

[1] If the source nodes are topologically close to each other, their paths to D would share a larger number of hops.

Let $AB(i, j)$ denote the end-to-end AB on the path from node i to j. We define AB_i^{in} to be the minimum of the AB on all links in the access segment of node i, in the direction of carrying traffic from the core to the node i, and AB_i^{out} to be the minimum AB on these links, in the direction of carrying traffic from node i to the Internet core (see Fig 1.3). The two observations made above then collectively imply that the end-to-end AB on the path between two given nodes, S_1 and D_1, can be approximated as:

$$AB(S_1, D_1) \quad \simeq \quad \min\{AB_{S_1}^{out}, AB_{D_1}^{in}\} \tag{1.3}$$

The above formulation can be used to infer the AB on the path between S_1 and D_1 without necessarily directly measuring $AB(S_1, D_1)$.

Since we know that access hops are shared, we can reduce the measurements that need to be made by clustering nodes together which share the same access segments to the rest of the nodes in the network. Once we have created these clusters, a *representative node* in a cluster can measure the $AB_{D_i}^{in}$ of all the nodes outside the cluster. The other nodes in the group can measure $AB_{S_j}^{out}$ to their *representative node*. We can then use Equation 1.3 to infer the end-to-end AB between any two nodes. We propose three variants of this basic approach that offer a trade-off between estimation accuracy and probing overhead—the most scalable variant incurs an overhead only linear in complexity to the size of the overlay.

We evaluate our approaches on the PlanetLab infrastructure [Pla] by relying on the S^3 monitoring service [YSB$^+$06]. Our evaluation shows that our approach can estimate AB with an estimation accuracy within 20% for a majority of the estimates—this significantly outperforms the accuracy of existing schemes.

1.7 Thesis Statement

In this dissertation, we demonstrate that:

1. It is possible to isolate and study the temporal and algorithmic aspects of AB estimation.

2. Contemporary implementations of system timers and interrupt coalescence can significantly impair the performance of an otherwise sound AB estimation logic.

3. The timescale at which AB is measured significantly impacts both the accuracy and variability of AB estimates.

4. ABETs based on the probe-rate model are more robust to traffic and path dynamics.

5. It is possible to design a scalable inference scheme, that helps achieve desirable operating points on the trade-off between accuracy and overhead, for collectively monitoring the AB on all paths in a distributed overlay network.

1.8 Summary of Contributions

The major contributions of this thesis can be summarized as follows:

- We conduct the *first* comprehensive black-box evaluation of a large suite of prominent AB estimation tools on a high-speed network. In this evaluation, we also illustrate the impact that technological and implementation limitations can have on the performance of ABETs .

- We conduct the *first* comprehensive evaluation of AB estimation algorithms, independent of systemic and implementation biases. In this evaluation, we also illustrate the impact temporal factor such as measurement timescales have on the observed relative performance of ABETs .

- We demonstrate that temporal properties can significantly impact the AB estimation process. We redesign the interfaces of existing ABETs to allow temporal parameters to be explicitly specified and controlled.

- We design AB inference schemes which can be used to scalably and collaboratively infer the AB for a large set of end-to-end paths. These schemes allow an operator to select the desired operating point in the trade-off between accuracy and overhead of AB estimation. We further demonstrate that in order to monitor the AB on all paths of a network we do not need access to per-hop AB estimates and can simply rely on end-to-end AB estimates.

1.9 Roadmap

The remainder of this thesis is organized as follows. In Chapter 2 we will discuss related work in the field of AB estimation. Chapter 3 will describe our black-box evaluation of ABET tool implementations on a high-speed network. Chapter 4 will describe our study of the impact of temporal factors on the process of AB estimation. Chapter 5 describes the algorithmic evaluation of prominent ABET designs. Chapter 6 describes the design of our scalable AB monitoring approach in an overlay network. We conclude with Chapter 7 by outlining future work and our conclusions.

1.10 Notations

In the rest of this dissertation, we will rely on the following notations.

Notation	Expansion
AB	Available Bandwidth
ABET	Available Bandwidth Estimation Tool
CT	Cross-traffic
CDF	Cumulative Distribution Function
MT	Measurement time scale
SI	Sampling Intensity
SNMP	Simple Network Management Protocol
RT	Run-Time
RTT	Round Trip Time
SABI	Scalable Available Bandwidth Inference
TCP	Transport Control Protocol
IAT	Inter-Arrival Time
GigE	Gigabit Ethernet
Mb/s or Mbps	Megabits per second
Gb/s or Gbps	Gigabit per second
OC-48	Optical Carrier (2.5 Gb/s) link
NIC	Network Interface Card

Table 1.2: Table of notations

CHAPTER 2
Design of Bandwidth estimation tools

The area of AB estimation has been an active area of research for the past few years. In this chapter we summarize related work that studies several aspects of the AB estimation problem. We first examine techniques that were developed to estimate the bottleneck capacity on an end-to-end path. We next discuss tools for estimating end-to-end AB. Next we examine past evaluations of AB estimation tools as well as formal analysis geared towards better understanding the performance parameters of AB estimation tools. Finally we examine methodologies that have been proposed to efficiently monitor large overlay networks.

2.1 Background

We start with some definitions and observations that will be used in this chapter.

End-to-end Bottleneck Capacity The transmission capacity of a link refers to the speed at which data can be transmitted on the link. If the transmission capacity of a link is 1 Mbps, then it follows that the maximum amount of data that can be transmitted on the link in a single second is 10^6 bits. Alternatively, the time taken to transmit a single bit is 1 μs.

Given an Internet path consisting of several links, the end-to-end *bottleneck capacity* refers to the minimum of the transmission capacities of all the constituent links of the path. Formally, if the path consists of n links and C_i is the transmission capacity of the i^{th} link, then the bottleneck capacity of the path is defined as:

$$\min_{1 \leq i \leq n} \{C_i\} \tag{2.1}$$

For example, in Figure 2.1, the bottleneck capacity is 100Mbps. The link with the least transmission capacity (the second link in Figure 2.1) is commonly referred to as the *narrow* link of the path [JD02a].

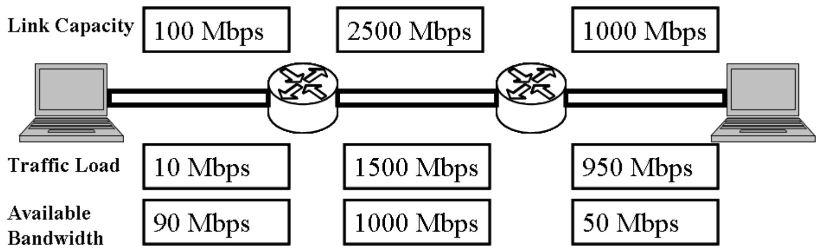


Figure 2.1: Illustration of an Internet path

End-to-end Available Bandwidth The end-to-end AB was formally defined in Section 1.2.1. Informally, it represents the minimum of the spare bandwidth available over all links of a path. The link with the least amount of spare bandwidth is commonly referred to as the *tight* link [JD02a].

In Figure 2.1, the end-to-end available bandwidth is 50 Mbps and the first link is the narrow link. In this example, the tight and narrow links are different.

Multi-homing in Access Networks At this point it would be prudent to make some observations about the structure of the Internet (Figure 2.2). The Internet is organized in a hierarchical structure of tiered Internet Service Providers (ISPs), that cooperate with each other based on either a customer-subscriber model or a peer-to-peer model, in order to transfer data between two end nodes. Tier-1 [tie] service providers operate large networks and provide long haul connectivity over large geographical areas—contemporary examples include Verizon, Sprint, and AT&T. Tier-1 service providers typically have peering relationships with one another which allows them to exchange data on a no-cost basis with each other. One layer below in the hierarchy are tier-2 service providers, which are predominantly regional in geographical scope. Such providers may have peering relationships with other tier-2 service providers, but also have customer-provider relationships with tier-1 ISPs in order to use the latter to send or receive customer data. Finally, tier-3 ISPs interact with other providers almost exclusively based on the customer-provider model. Since tier-3 networks are often used to provide Internet access to customers, we also refer to these as *access networks*.

Tier-3 ISPs can be multi-homed, which means that they can connect to more than one tier-2 or tier-1 ISPs. A recent analysis of Internet paths suggests that multi-homing is on the rise in the Internet [AMS+03]. As a result of multi-homing, there are several potential paths that can be taken in order to reach a single destination. Thus, it is becoming increasingly possible for an ISP to make a decision regarding a path that will be best suited for an application based on the applications requirements—information about end-to-end AB on each candidate path

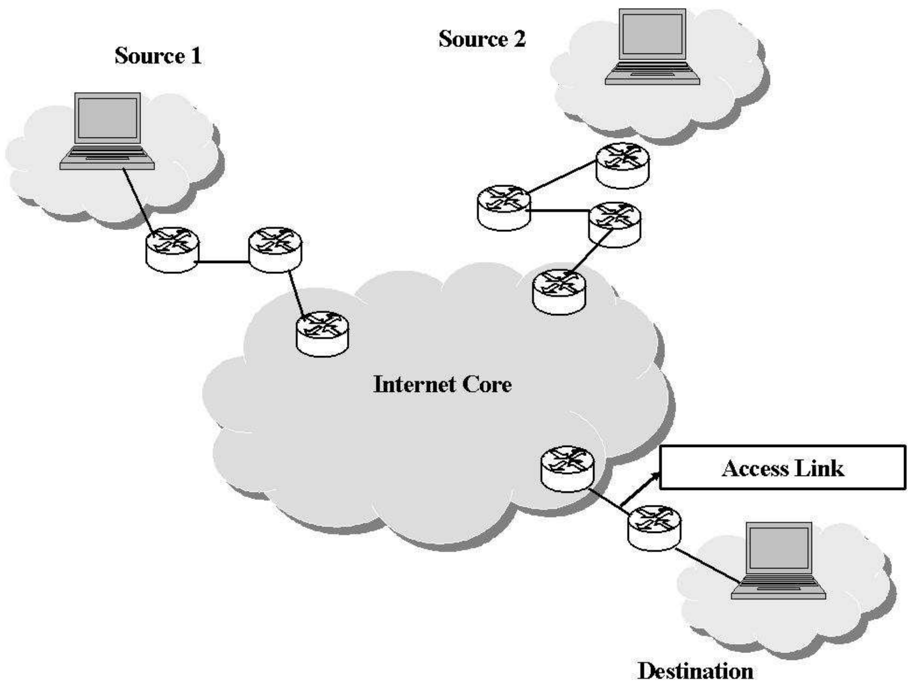


Figure 2.2: Internet Architecture

would be useful in making such decisions.

2.2 Capacity estimation tools

A seminal paper in the area of per-link capacity estimation was by Van Jacobson who proposed a tool called *Pathchar*[Jac]. Pathchar estimates per-hop capacity by using the Variable Packet Size (VPS) probing methodology, that relies on using probe packets of different sizes to estimate the capacity of every hop along a path. Let us consider a path with K hops, and assume that the capacity of the i_{th} link is C_i bits per second (bps)—thus, the time taken to transmit a packet of size L bits on the i_{th} link will be $\frac{L}{C_i}$ seconds. Now, the time taken for a L-bit packet to travel from the sender to the i_{th} hop and back can be expressed as:

$$T_i(L) = \sum_{k=1}^{i}(\frac{L}{C_i} + Queuing_{fwd} + Propagation_{fwd} + Queuing_{rev} + Propagation_{rev} + \frac{L}{C_i}) \qquad (2.2)$$

where $Queuing_{fwd}$ and $Queuing_{rev}$ are the delays experienced by waiting in the queue of the i_{th} link on the forward and reverse paths, respectively. $Propagation_{fwd}$ and $Propagation_{rev}$ is the propagation delay in either direction of link i. Finally, $\frac{L}{C_i}$ is the time taken to transmit the packet on the link.

Most per-hop capacity estimation tools rely on a time-to-live (TTL) limiting mechanism. Every packet sent on the Internet has a TTL field that is initialized by the sender. At every router that the packet traverses, this TTL field is decremented by one. If a router receives a packet where the TTL is zero, it discards the packet—most routers also send an Internet Control Message Protocol (ICMP) packet, which is typically 64 bytes, back to the sender. Thus if we limit the TTL of a packet to i, then we can force the i_{th} router to send back a constant size ICMP message to the sender—this mechanism can yield the information needed to compute $T_i(L)$. So the above equation now becomes:

$$T_i(L) = \sum_{k=1}^{i}(\frac{L}{C_i} + Queuing_{fwd} + Propagation_{fwd} + Queuing_{rev} + Propagation_{rev} + \frac{L_{icmp}}{C_i}) \quad (2.3)$$

Now if we send a large number of probe packets to hop i, we would expect that the packet with the lowest $T_i(L)$ would experience no queuing delay in the forward and the reverse path. From this, Equation 2.3 can be simplified as:

$$T_i(L) = \sum_{k=1}^{i}(\frac{L}{C_i} + Propagation_{fwd} + Propagation_{rev} + i\frac{L_{icmp}}{C_i}) \quad (2.4)$$

Now since the $Propagation_{fwd}$, $Propagation_{rev}$, and L_{icmp} are constants, Equation 2.4 reduces to:

$$T_i(L) = K + \sum_{k=1}^{i}(\frac{L}{C_i}) \quad (2.5)$$

where $K = Propagation_{fwd} + Propagation_{rev} + \frac{L_{icmp}}{C_i}$. The equation now has the form $F(L) = K + M_i * L$, where M_i is $\sum_{k=1}^{i} \frac{1}{C_i}$ and $F(L)$ is $T_i(L)$. Now if we were to measure M_i by using packets of different sizes, then $C_i = \frac{1}{M_{i+1} - M_i}$.

Using the above relation, the capacity of all the hops of a path can be measured. Mah et. al. [Mah00] designed an improvement for Pathchar, which used linear regression to better estimate the values of M_i. Downey[Dow99] designed a tool called Clink, which operates on the same principle as Pathchar, which is also robust to routing instability. Prasad et. al. [PDM02] analyzed the working of the VPS methodology and found that this technique is prone to under estimating the capacity of links which have Layer-2 store and forward devices as a part of a link. This error however is localized to the links which have the layer-2 devices and does not propagate causing errors in links after the layer-2 link. Dovrolis et. al. [Dov01] designed a tool which could measure the end-to-end capacity of an Internet path. Pathrate uses variable sized packets, which are sent back-to-back from the sender in order to estimate the capacity of the end-to-end path. The principle behind this approach is that two packets sent back-to-back will arrive at receiver with a spacing between them which is proportional to the capacity of the path between them. This spacing between the packets is referred to as the *dispersion* of the packet-train. Formally specified, the dispersion is defined as follows $T_{disp} = \frac{L}{C_{end-to-end}}$. Pathrate uses a system of creating histogram and studying the most frequently occurring modes in the histogram in order to estimate the value of T_{disp} and then compute the end-to-end capacity.

We next discuss tool that were designed for estimating the available bandwidth (versus bottleneck capacity) on an Internet path or link.

Tool	Probe Stream	Inference Metric
Pathload [JD02b]	Equi-Spaced Train	One-way Delay
Pathchirp [Rib03]	Exponential spacing	Dispersion
Spruce [SKK03a]	Packet-Pair	Dispersion
Abing [Nav03]	Packet-Pair	Dispersion
IGI [HS03]	Packet Train	Dispersion
Iperf [ipe]	TCP-Stream	Throughput
Cprobe [CC96a]	Packet Train	Receiving Rate

Table 2.1: Available Bandwidth Estimation Tools

2.3 AB Estimation Tools

2.3.1 End-to-End AB estimation tools

Table 2.1 lists some prominent AB estimation tools. All AB estimation tools work under the same underlying principle which involves sending probe packets at well defined and known intervals on an Internet path, such that they temporarily induce a load on the path. This load will cause the existing cross traffic on the path to be interspersed by the probe packets. When these probe packets reach the other end of the path (receiver), the receiver studies how the intervals between the probe packets has changed and compares these intervals to the known intervals between the probe packets when the probe packets were initially sent. Using empirical or analytic techniques, the AB of the end-to-end path can then be computed. We will discuss the details of some of the prominent tools next.

- **Pathload**[JD02b] reports the AB using two thresholds, the $upper_{range}$ and $lower_{range}$ AB, which represents the range of AB values observed during the tool run. Pathload starts by initializing the $lower_{range}$

to be 0 and the $upper_{range}$ to be the capacity of the end-to-end path. It then sends out a stream at an initial guess of the AB. The one-way delay in the stream is analyzed to study if this stream rate was greater than or less than the AB. If the AB is exceeded the $upper_{range}$ is set to the current sending rate of the stream. If the AB is not exceeded, the $lower_{range}$ is set to the current sending rate. The rate of the next stream is then determined by using the relation $\frac{upper_{range}+lower_{range}}{2}$. This process continues iteratively, till the difference between the $upper_{range}$ and the $lower_{range}$ becomes less than a certain threshold. At this point the current $upper_{range}$ and $lower_{range}$ are reported as the AB.

- **PathChirp**[RRB$^+$03] uses an exponentially spaced packet train to estimate the AB. It uses a parameter called the spread factor, which defines the exponent of the Chirp train. For instance if the spread factor is 2, a 5 packet Chirp would send the first two packet spaced at 1 Mbps, the third at 2 Mbps, the fourth at 4 Mbps and the fifth at 8 Mbps. The Chirp is analyzed at the receiver end to study at which rates in the Chirp the sending rate was greater than the receiving rate. The principle being that since a Chirp covers a wide range of rates, the rates after which the AB is exceeded will be characterized by the sending rate being greater that the receiving rate. Thus we can infer the AB of an end-to-end path by observing these points of change. Multiple chirps are used in order to improve the confidence of the estimates.

- **IGI/PTR**[HS03] sends a train of back-to-back packets to the receiver which is used to estimate the path capacity. It then uses the path capacity as its first sending rate to the receiver. The gaps at the source and the gaps at the destination are compared to infer if the current sending rate was greater than the AB . If so the current sending rate is reduced by a constant decrease factor that can be specified during the tool run, and this process is continued till the point where the receiver reports that the source gaps and the destination gaps are equal. It is at this point that IGI and PTR differ. IGI estimates the cross-traffic rate using the difference in the gap values. It then subtracts this estimate of the cross-traffic rate from the bottleneck link capacity to infer the end-to-end available bandwidth. PTR on the other hand compute the average receiving rate of the stream and infers that as the AB of the path.

- **Abing[Nav03] and Spruce[SKK03b]** use a packet-pair based technique to estimate the AB. The sender sends two packet spaced $\delta_{in} = \frac{S}{C}$ seconds apart; where S is the packet size and C is the link capacity. The receiver then observes the spacing between the packets when it arrives at the receiver and computes the dispersion in order to compute the AB by using the following relation $AB = C \times (1 - \frac{\delta_{out}-\delta_{in}}{\delta_{in}})$. Where δ_{out} is the spacing in the packet-pair when it reaches the receiver. While Spruce spaces its packet-pairs using a Poisson process, Abing sends packet-pairs periodically.

- **Iperf** [ipe] is a standard benchmarking and monitoring tool. It can be used to compute the throughput of a path by actually running a TCP connection on the path. While it has been shown that the TCP throughput of an end-to-end path is not the same as the AB [DJ03], Iperf continues to be a popular choice among network operators for measuring the AB.

- **Abget[DMA$^+$06] and QuickProbe[KV06]** are variants of the Pathload tool. Abget uses a TCP connection to measure the AB down-stream AB on a path by controlling the acknowledgments that the client receives in order to control the rate at which the server sends data. This method of controlling the rate of an incoming packet stream is then used to implement the Pathload logic and the inference is carried out in the same manner as described above. QuickProbe another variant of Pathload reduces the time-taken by Pathload to make an inference by reducing the number of packets that are required to make an inference.

- **Cprobe**[CC96a] first finds the end-to-end capacity of a given path using another tool called Bprobe[CC96a]. It then sends a packet train at that rate to the destination host. The rate at which this stream is received at the other end of the path is inferred to be the AB.

Most ABET designs as observed above focus on the construction of the probe-streams and the logic used to estimate the AB from the observed delays. However, these tool designs ignore two central temporal quantities related to the measurement of the AB process: the MT and the SI . Thus it is important to redesign the ABET interfaces to allow the choice of MT and SI, and study the impact of these on ABET performance.

In this thesis in order to study the impact that the observation time-scale and the sampling frequency have on the accuracy of the tool, we redesign the tool interfaces such that we can set the observation time-scale and sampling frequency. We can then evaluate the tools under the same conditions and compare their performance.

2.3.2 Per-hop AB estimation tools

In the recent past another class of AB estimation tools have been proposed which measure the AB on every hop of a path. Two prominent tools in this domain are Pathneck and Stab.

Stab[RRB04] works on a principle similar to PathChirp. However here every packet in PathChirp is represented as a two packet combination in Stab which we will refer to as a Measurement Pair(MP). The first packet is a TTL limited *load packet* which has a large size and the second packet is a small packet which is considered to be a *measurement packet*. In order to measure the AB up-to the i^{th} hop on a path, the load packets have in every packet-pair have their TTL set to *i*. The MP train is sent in exactly the same manner as PathChirp. The intuition behind this approach is that the load packets will create the excursion patterns at any given hop, which will also be reflected in the small-sized measurement packet, since the measurement packets will always queue up behind the the load packets. When a MP reaches the i^{th} hop, all the load packets will be dropped (because they are TTL limited), and the measurement packet which reflect the spacing of the load packets when they were dropped will carry on to the receiver. The receiver can infer the AB of the i_{th} hop using the same inference logic as PathChirp. This procedure is done for every hop, to get a per-hop estimate of the AB .

Pathneck[Li05] uses a stream construction called a Recursive Packet Train (RPT). A recursive packet train consists of two components (i) A Load Train and (ii) A Measurement Train. A single RPT is constructed as follows the first *k* packets are TTL limited measurement packets with small size and are sent back-to-back. The next *L* packets are large load packets, which are sent at a rate specified by the IGI/PTR algorithm. The final *k* packets are also TTL limited load packets. The first and last measurement packet have a TTL of 1, the second and second last measurement packet have a TTL of 2 and so on for all the k measurement packets. In the case of Pathneck is set to 30 and L is set to 70, though these are parameters and can be varied. The intuition behind Pathneck working is as follows. As the RPT traverses the path, at each successive hop the first and the last packet of the RPT will be dropped and a TTL expired message will be sent back to the receiver. By observing the dispersions of the TTL limited packets, the amount of time the load packets were queued at a particular hop can be computed, which using IGI/PTR inference logic can give us information about the AB on a given hop.

In this thesis we show that scalable approaches to monitoring the AB in a network can be done using end-to-end AB estimation tools. We show that other approaches[HS05] which require per-hop AB information are less accurate than our method and also do not perform as well as existing end-to-end AB estimation tools.

2.3.3 Implementation techniques to achieve high time-stamping accuracy

In current high-speed networks, variations in end-to-end delays may have an order of magnitude in the sub-millisecond range. Since most ABETs rely on measuring delay variations, high resolution and accuracy in time-stamping probe packets is crucial for ensuring the accuracy of the inferred AB. Current PC platforms, however, are incapable of guaranteeing high time resolution due to multi-tasking and the use of mechanisms such as interrupt coalescence [JD02b]. This issue has also been studied in [Pax97] where the authors derive that if a system has a clock granularity of C_r and the system sends packet of size B bytes, then this system will be

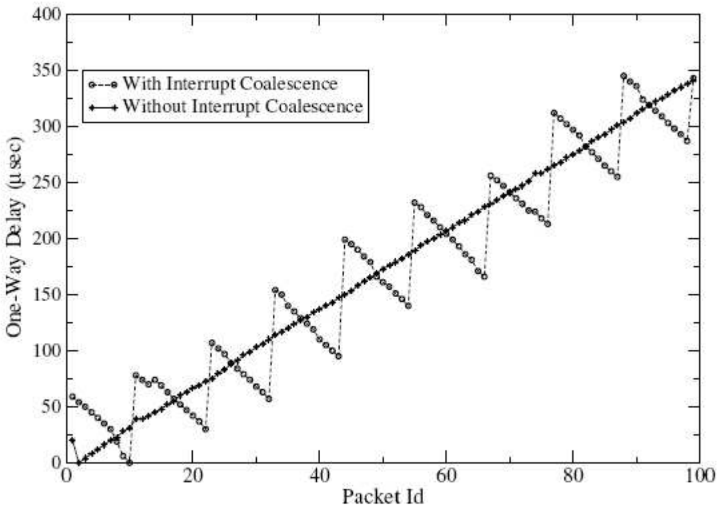


Figure 2.3: Impact of Interrupt Coalescence. (Graph from [PJD04])

unable to make a distinction between packet-pairs sent at rate $\frac{B}{C_r}$ and ∞. Most tool implementers work around this limitation using two techniques [JD02b, Rib03, SKK03a]: (i) they rely on OS support for detecting and discarding probe streams that appear to not have been time-stamped accurately; and (ii) they collect observations from several probe streams before converging on a robust estimate of AB. While such implementation techniques do not differ much across current ABETs, these do impact tool performance significantly [AMPR03, JT03]. It is important to note that the above techniques are highly technology-specific. As technology improves [PV02a], the impact of these techniques on ABET performance is likely to diminish.

Existing tools differ in the efficiency with which such systemic biases are handled. For instance in [PJD04] the authors propose a technique which can be used to detect interrupt coalescence. They propose that the graph of the one-way delays of a large stream of packets being affected by interrupt coalescence would look like an increasing saw-tooth function when it was received at the destination machine. Figure 2.3 taken from [PJD04] illustrates this effect. This is because packets will be buffered at the card till an interrupt timer expires and all the buffered packets will be delivered back-to-back to the kernel and then the application. In such a scenario, only the last packet of every burst should be considered since it has the most current timing information and has suffered from the least buffering at the NIC.

The above technique requires multiple packets to be sent in order to reliably detect interrupt coalescence. However, not all existing ABET tools have been designed with this consideration in mind and as a result many tools suffer from biases introduced by interrupt coalescence. *In this thesis, we evaluate some prominent AB estimation tools in an environment where systemic effects like interrupt coalescence could come into play. We then study how the ABETs could be impacted by these effects and correlate our findings to other studies which have focused specifically on these issues.*

2.4 Tool Evaluation

Several tool proponents compare the performance of their tools against that of others under controlled lab settings as well as in Internet-wide experiments [HS03, Rib03, SKK03a]. Ribeiro et. al. [Rib03] compare the performance of PathChirp to that of Pathload [JD02b] and TOPP [MBG00] in an emulated lab setting. They find that PathChirp is more accurate than TOPP and less intrusive than Pathload. Hu et. al. [HS03] compare the performance of IGI/PTR to that of Pathload and Iperf on 13 Internet paths of capacity within 100 Mbps. They observe that while the readings of the three tools match on some paths, they fluctuate on other. Since the actual AB of these paths were not known, tool accuracy was not verified. Strauss et. al. [SKK03a] compare the performance of Spruce to that of Pathload and IGI. They use SNMP data collected at five minute intervals to evaluate the accuracy of these tools on two 100 Mbps paths. They also compare the sensitivity of the tools to changes in AB by performing several experiments on the RON testbed. They find that IGI is inaccurate at high loads, and Spruce is more accurate and less intrusive than Pathload. Coccetti et. al. [CP02] evaluate early ABETs, including Iperf and Pathload, on a low speed (less than 4 Mbps) 4-5 hop topology with and without cross-traffic. They conclude that tool results strongly depend on configuration of the router queues and that a considerable amount of care would need to be taken while interpreting the results from any ABET, especially if QoS features were present in the network. In [LRL04], the authors analyze at large time-scales, the performance of several bandwidth estimators that can be represented mathematically. Unfortunately, their evaluation considers only low-bandwidth paths with a single bottleneck link. Furthermore, several iterative estimators cannot be represented using their formulation.

Unfortunately, such studies are not comprehensive in either the tools or the settings evaluated. All of the studies described above evaluate only a small (and different) sub-set of ABETs, depending on which ones were popular when the corresponding tool was proposed. Furthermore, these evaluations include only simple network and traffic scenarios—for instance, none of the above evaluate tool performance against responsive cross-traffic, in high-speed networks, or when the tight and narrow links are different.[1] As a result of these practices, the results are often not comprehensive and get inadvertently biased toward highlighting the salient features of the proposed tool.

As listed in Section 1, there are a wide variety of factors that could impact the accuracy and performance parameters of an ABET. Unfortunately there are *no evaluation studies* that systematically *evaluate all the parameters* that can influence the performance of a given ABET. *Furthermore no evaluations take into consideration the temporal aspect of the process of AB estimation and the impact that it could have on the tools accuracy*. To summarize, existing ABET evaluations are either biased by limitations of current implementation technology and/or are not comprehensive in evaluating tools against diverse network,probing, systemic and temporal conditions. *In this thesis we study the factors that can affect the performance of an ABET and systematically quantify the impact that they have on the performance parameters of ABETs.*

2.5 Formal analysis of AB estimation tools

In order to better understand the performance bounds of AB estimation tools, there have also been several efforts to quantify the biases and the errors that can be observed by the various AB tools and techniques.

In [LRLL04] the authors analyze the performance of AB estimation tools on a single-hop path. They assume a simple FIFO model of queuing and derive the ”Response Curve” of an input probe that is used to estimate the AB . The response curve is the function that relates the input gaps to the output gap on a single-hop path as a function of the cross-traffic intensity and link capacity that is present on that path. The authors derive an expression which given an input gap,cross-traffic intensity, probe packet size and link capacity can define bounds for the expected gap values. The difference between the actual output gap response curve and the theoretical lower bound of the response curve is defined as the probing bias, which directly relates to the amount by which the cross-traffic process is being changed because of the external probes that we are introducing into the network. The authors also conclude that using a longer probing packet train or larger packet sizes, would reduce the probing bias.

In [LRL05] the authors extend the single-hop model into the more general case model of a multi-hop path and formulate a ”Response Curve” for a multi-hop path using a fluid cross-traffic model. They conclude, that the for a multi-hop path, that the relation between the input and output gap (Response Curve) is a continuous

[1]The *tight* link of a path is one with the least amount of available bandwidth, while the *narrow* link is the one with the least transmission capacity [JD02b]. The tight link of a path may not be the same as the narrow link if it carries significant amount of traffic load.

piecewise linear function. The first point of slope change is analytically shown to be the point at which the AB of the path is obtained. They then extend the fluid-cross traffic model to a more realistic cross-traffic model and show that the response curve for the realistic cross-traffic model is lower bounded by the its fluid counterpart. They also analytically show that as the packet size or the packet train length approaches infinity, the bias terms approaches 0. That is theoretically we can obtain perfect knowledge of the AB if we can send arbitrarily large packets or arbitrarily long packet trains.

In [LFV07b] the authors model an end-to-end path and an AB estimator on that path as a min-plus system, in the context of network calculus. The primary assertion behind this approach is that it is possible to completely describe a min-plus system by using only its impulse response. An impulse response of a system is the response of a system when the input to the system is the burst function, which in the context of AB estimation is a probe stream which has an instantaneous rate of infinity Mbps. This impulse response is also defined as the service curve of the system. Thus the objective of the Min-plus analysis is that given an arrival function and a departure function which can be obtained by observations, can we obtain the service curve of the end-to-end path (Min-plus system). The authors show analytically that it is not possible to get the exact service curve, and therefore derive an expression to find the service curve which maximally lower bounds the actual service curve. The authors then show how this formulation can be used in passive monitoring, rate scanning and Chirps to obtain a lower bound on the service curve and hence infer the AB on that path.

In [LDS06] the authors analytically study the performance of AB estimation tools, specifically the performance of the packet-pair techniques. They show that even in the case where there is a single bottleneck link and the situation where the narrow link and the tight-link are the same there are situations where the packet-pair technique will result in an underestimation of the AB. In the case of path-persistent (Refer to Figure 2.5) cross-traffic the estimator that is used in the packet-pair method can accurately measure the AB. However in situations where the cross-traffic is not path-persistent, the packet-pair estimator will always underestimate the AB when the sending rate is greater than the AB.

In this thesis we empirically study the issues of bias that are introduced by varying network conditions and cross-traffic interaction models in ABETs and verify many of the observations made in the theoretical models proposed above.

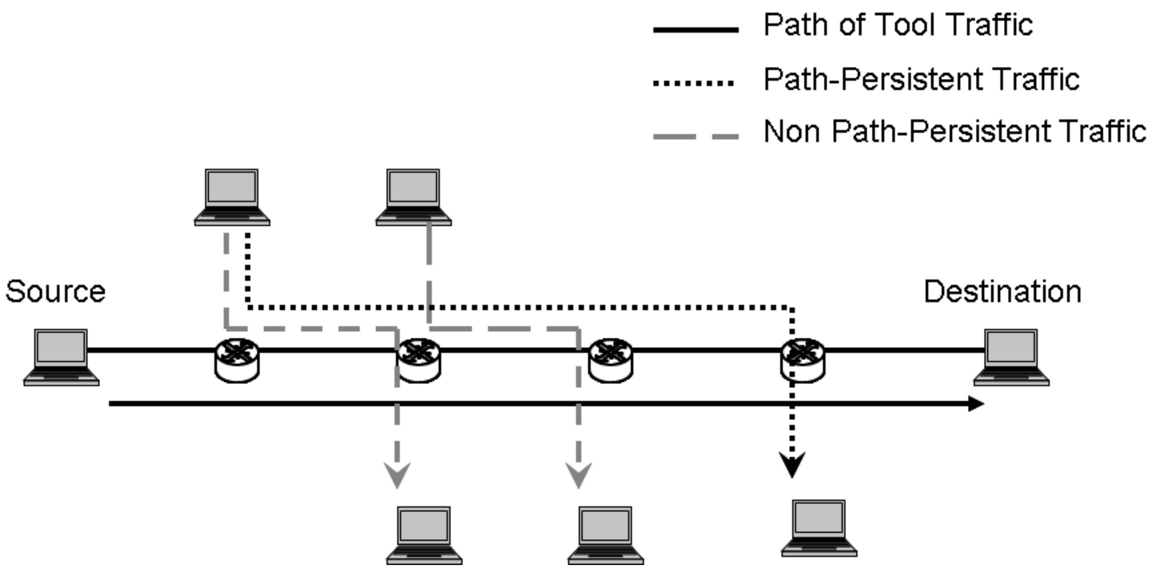


Figure 2.4: Illustration of Path-Persistent and Non-Path persistent traffic patterns

2.6 Efficient Network Monitoring

Recent work has addressed the issue of monitoring of all links of a given network in a scalable manner. Most approaches rely on the observation that many end-to-end paths in a network share several links—this redundancy can be exploited to drastically reduce the number of end-nodes used for probing links as well as the number of probes sent by these. The reduction problem has been modeled by many as a vertex-cover problem. [BCG$^+$01] optimizes the number of SNMP probes that need to be sent to NetFlow-enabled routers to query for the link utilization and the latency. The authors provide a generic heuristic algorithm, which produces a near optimal set of links that need to be monitored. [KK06] optimizes the problem of beacon placement in the presence of dynamic IP routes, such that each link of a give network can be deterministically monitored for link failures and delays, while placing a minimum number of probing beacons. [CKC05] measure the delay and loss efficiently by using a linear-algebraic approach for selecting a subset of links, and then inferring the network properties from this subset. [SQZ06] uses a Bayesian network based approach to monitor the delay, capacity and loss in an efficient manner.

Unfortunately, the problem of scalable monitoring of end-to-end AB in an overlay has not been well-addressed. The only existing scheme, Broute [HS05] uses the intuition of shared access hops and the fact that bottlenecks lie close to the edges uses special nodes referred to as landmarks as well as a per-hop AB estimation tool in order to monitor the all-pairs AB in a scalable manner. The algorithm relies on a series of landmark nodes to which Pathneck[HLM$^+$04]; a tool to locate the bottleneck on an Internet path; is run. Pathneck also provides lower bounds on the per-hop AB. Every node monitors the per-hop AB on its ingress and egress links to a series of landmark nodes. When the AB between any two nodes is required the system will take the minimum of the AB at the sources egress links and the destinations ingress link and infer the end-to-end AB as the minimum of those two ABs. Evaluation of this scheme demonstrated that the AB estimation accuracy is less than 50% for 80% of the cases. *In this thesis we design a scalable AB monitoring infrastructure, which reduces the number of measurements and shows better inference accuracy than previously proposed schemes*.

2.7 Network aware application designs

There are many applications that make network aware decisions. For instance the Resilient Overlay Network (RON) [ABKM01] monitors the delay and bandwidth on its networks and is able to provide delay or loss optimized paths as required. Multimedia applications like [NN98] also rely on the delay information in order to construct their multi-cast routing tree. The knowledge of the delay is also useful in web caching [WY00] where the latency is used a metric to decide where a some content should be cached. The objective of this system is to cache data at geographically proximate locations, such that requests can be responded to with minimum delay. Peer-to-Peer like Bit-torrent [Bit] can also make intelligent choices about peers as studied in [Qur04]. Despite there being many potential AB aware applications, it is not clear what the performance gains would be on using an AB aware application.

There has also been some focus on designing applications which use the knowledge of AB. For instance SOBAS[DPJ04] relies on using the knowledge of the AB in order to adaptively set its socket buffer size, so as to optimize the performance of a transport protocol. This scheme works by probing for the AB and setting the socket buffer to the corresponding size. This scheme is particularly useful in high bandwidth networks, where congestion windows can become very large and exceeding the congestion window could cause a large number of packet drops. The scheme proposed helps maintain the congestion window to a level where there are very few packet drops. Dovrolis et al [ZDA06] also studied the performance of a scheme which could do bandwidth adaptive routing. The authors study the performance difference between doing proactive and reactive routing using the knowledge of the AB. However the authors do not address how they would obtain complete bandwidth information to implement this scheme. [TUAK04] utilizes the information about the AB and the capacity to determine what the output rate of the streaming protocol should be. As can be observed there are several potential applications that could benefit from the knowledge of the AB. However there is no clear quantification as to what performance improvements could be obtained by using an AB aware application. *In this thesis we design an infrastructure that can monitor the AB of a network, which would give the above listed applications access to the AB information aiding them in making better network aware decisions.*

CHAPTER 3
Evaluation of ABET implementations

We begin our series of evaluations by first evaluating publicly-available implementations of ABETs—in this first study, we treat each of these tools as a black-box and evaluate the default design choices along the algorithmic, implementation-related, and sampling-related temporal dimensions. A major focus of our analysis in this chapter is to understand the extent to which systemic issues and implementation efficiencies impact tool performance—the impact of temporal and algorithmic factors is studied in subsequent chapters.

ABETs face increasingly difficult measurement challenges as link speeds increase. Consider the issue of time precision: on faster links, time-gaps between packets decrease, rendering packet probe measurements more sensitive to timing errors. Available bandwidth measurements on high-speed links stress the limits of clock precision especially since additional timing errors may arise due to the NIC itself, the operating system, or the Network Time Protocol (designed to synchronize clocks of computers over a network) [PV02b]. Additionally, mechanisms such as interrupt coalescence that are used to improve network packet processing efficiency, mislead end-to-end tools that assume uniform per-packet processing and timing [PJD04].

On the other hand, ABETs are being increasingly deployed in high-speed network settings such as the Network Weather Service [Wol98] and the TeraGrid [Ter] infrastructure. Since the systemic issues mentioned above play a greater role in high-speed networks, it is critical to develop an understanding of the performance of prominent ABETs in such environments. Unfortunately, as described in Chapter 2, most past evaluations of ABETs have been restricted to paths with capacities of 100 Mbps or less—furthermore, these evaluations are often not comprehensive in the set of tools evaluated.

In this chapter we describe a comprehensive evaluation of ABETs in a high-speed network testbeds—to the best of our knowledge, this is the *first* such evaluation presented in the literature. We show that a comparative performance evaluation of various ABET implementations is inadvertently biased against certain tools since their performance is adversely impacted by interrupt coalescence and buffer size limitations (systemic biases) along with the inability of certain tool designs to detect these effects (implementation biases). We also run these tools on Internet paths and observe that the tools display similar performance characteristics to what we observed in the test-bed setting. This implies that the results we obtain here are generalizable to the performance of the ABETs in actual deployments. It also presents strong motivations for designing and testing ABETs in high-speed testbeds similar to the one that we will describe in this chapter before deploying them on live systems.

3.1 Background: Interrupt Coalescence and network measurements

We first discuss how interrupt coalescence can impact network measurement and specifically AB estimation. Jain et. al. [PJD04] have conducted an in-depth analysis of this issue. Recall from Chapter 2 that AB estimation tools send a series of probe packets with a controlled and pre-determined spacing between them on the path of interest, and the receiver studies changes in the inter-packet gaps in order to make an inference about the AB . Clearly, ensuring high precision time-stamping and accurate spacing between the packets is critical for obtaining an accurate estimate of the AB.

To put this in context, in order to achieve a data rate of 1 Gbps, a 1500 byte packet would have to be transmitted (and received) every 12 μs. On general-purpose machines, packet transmissions and receipt by a network interface card (NIC) is handled by means of triggering interrupts—for instance, when a packet is received, the current process running on the CPU is interrupted, context is saved, and the packet is processed before returning the context to the original process. This processing and the associated context switch can be fairly costly operations to perform. Modern systems, consequently, reduce the overhead by grouping together packets that arrive close in time, and use a single interrupt to trigger their processing. This process is known as Interrupt coalescence (IC).

Unfortunately, since IC buffers packets arriving close together and uses a single interrupt to process all the packets, any timing information between the packets is lost—all such packets appear to have arrived back-to-back at the system. This can seriously limit the accuracy of AB estimation tools. There are no universal agreed upon parameters to setup IC. The network card that we used in our experiments had three parameters, which could be set. First, there is the maximum rate at which interrupts can be generated. Second, we can set the minimum time between a packet arriving and an interrupt being generated, which controls the maximum time a packet is queued at a device before it is acknowledged by the system. Finally, we can set the time between a packet arriving and a new interrupt being generated, which controls the minimum time a packet will spend in the queue. Using these parameters to control the IC can cause one of two effects in a pair of packets. First, if both the packets arrive between the interrupt intervals the two packets will be queued and acknowledged back-to-back and will appear to have arrived with no spacing between them—this could cause *compression* in the spacing between the packets. The second situation is if the pairs of packets arrive with some spacing between them and the first packet is acknowledged by an interrupt before the second packet arrives. This would cause the second packet to stay in the queue for at least the minimum interrupt generation delay, which will *increase* the delay between the two packets. Thus, IC can distort the spacing between packets by increasing or decreasing it.

In [PJD04], the authors have proposed a scheme which can be used to detect IC and conduct AB estimation even in the presence of IC. This technique relies on the fact that a train of packet impacted by IC will show a saw-tooth like arrival process (as illustrated in Figure 2.3). The authors propose a heuristic to identify the spikes and use only the first observation from each spike to make the estimates of the AB.

The above discussion highlights the fact that IC can have a significant impact on the process of AB estimation. A part of our evaluation in this chapter, we illustrate the impact that IC can have on some AB estimation tools and discuss why certain techniques are more sensitive to the effects of IC on contemporary computers.

3.2 Methodology

From Table 2.1 we selected the following tools for this study: *abing*, *pathchirp*, *pathload*, and *Spruce*. For comparison we also included *Iperf* [ipe] which measures achievable TCP throughput. *Iperf* is widely used for end-to-end performance measurements and has become an unofficial standard [CL02] in the research networking community.

We were unable to test *cprobe* [CC96a] because it only runs on an SGI Irix platform and we do not have one in our testbed. We did not include *netest* [Jin04] in this study since in our initial tests this tool inconsistently reported different metrics on different runs and different loads. We excluded *pipechar* [JYCA01] after tests on 100 Mb/s paths and *IGI* [HS03] after tests on 1 Gbps paths since they were unresponsive to variations in cross-traffic.

3.2.1 The high-speed testbed

In collaboration with the CalNGI Network Performance Reference Lab [San04], CAIDA researchers developed an isolated high-speed testbed that can be used as a reference center for testing bandwidth estimation tools. This resource allows us to test bandwidth estimation tools against known and reproducible cross-traffic scenarios and to look deeply into internal details of tools operation.

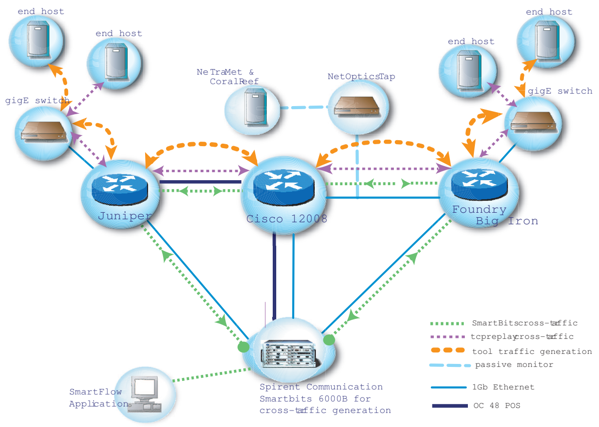


Figure 3.1: Bandwidth Estimation Testbed.

In the testbed configuration (Figure 3.2.1), all end hosts are connected to switches capable of handling jumbo MTUs (9000 B). Three routers in the testbed end-to-end path are each from a different manufacturer. Routers were configured with two separate network domains (both within private RFC1918 space) that route all packets across a single backbone. An OC48 link connects a Juniper M20 router with a Cisco GSR 12008 router, and a GigE link connects the Cisco with a Foundry BigIron 10 router. We use jumbo MTUs (9000 B) throughout our OC48/GigE configuration in order to support traffic flow at full line speed [Jor04].

Bandwidth estimation tools run on two designated end hosts each equipped with a 1.8 GHz Xeon processor, 512 MB memory, and an Intel PRO/1000 GigE NIC card installed on a 64b PCI-X 133 MHz bus. The operating system is the CAIDA reference FreeBSD version 4.8.

The laboratory setup also includes dedicated hosts that run *CoralReef* [KMK$^+$01] and *NeTraMet* [Bro04] passive monitor software for independent verification of tool and cross-traffic levels and characteristics. Endace DAG 4.3 network monitoring interface cards on these hosts tap the OC-48 and GigE links under load. *CoralReef* can either analyze flow characteristics and packet IATs in real time or capture header data for subsequent analysis. The *NeTraMet* passive RTFM meter can collect packet size and IAT distributions in real time, separating tool traffic from cross-traffic.

3.2.2 Methods of generating Cross-traffic

The algorithms used by bandwidth estimating tools make assumptions about characteristics of the underlying cross-traffic. When these assumptions do not apply, tools cannot perform correctly. Therefore, test traffic must be as realistic as possible with respect to its packet IAT and size distributions.

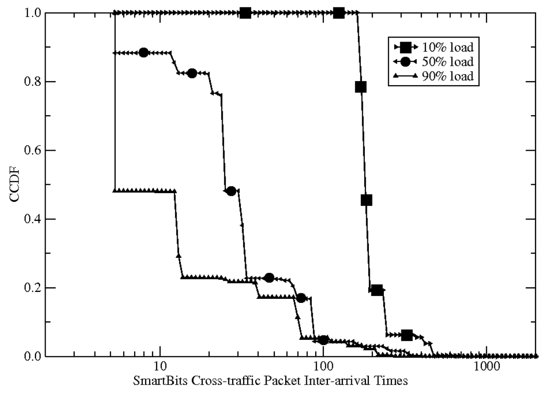
In our study we conducted two series of laboratory tool tests using two different methods of cross-traffic generation. These methods are described below.

Synthetic Cross-traffic Spirent Communications SmartBits 6000B [Spi04a] is a hardware system for testing, simulating and troubleshooting network infrastructure and performance. It uses the Spirent *SmartFlow* [Spi04b] application that enables controlled traffic generation for L2/L3 and QoS laboratory testing. Using SmartBits and *SmartFlow* we can generate pseudo-random, yet reproducible traffic with accurately controlled load levels and packet size distributions. This traffic generator models pseudo-random traffic flows where the user sets the number of flows in the overall load and the number of bytes to send to a given port/flow before moving on to the next one (burst size). The software also allows the user to define the L2 frame size for each component flow. The resulting synthetic traffic emulates realistic protocol headers. However, it does not imitate TCP congestion control and is not congestion-aware. In our experiments we varied traffic load level from 100 to 900 Mb/s which corresponds to 10-90% of the narrow GigE link capacity. At each load level, *SmartFlow* generated nineteen different flows. Each flow had a burst size of 1 and consisted of either 64, 576, 1510 or 8192 byte L2 frames. The first three sizes correspond to the most common L2 frame sizes observed in real network traffic [NLA04]. We added the jumbo packet component because high-speed links must employ jumbo MTUs in order to push traffic levels to line saturation. While [NLA04] data suggest a tri-modal distribution of small/medium/large frames in approximately 60/20/20% proportions, we are not aware of equivalent published packet-size data for links where jumbo MTUs are enabled. We mixed the frames of four sizes in equal proportions. Packet IATs (Figure 3.2a) ranged from 4 to more than 400 μs. We used passive monitors *CoralReef* and *NeTraMet* to verify the actual load level of generated traffic and found that it matched the requirements within 1-2%.

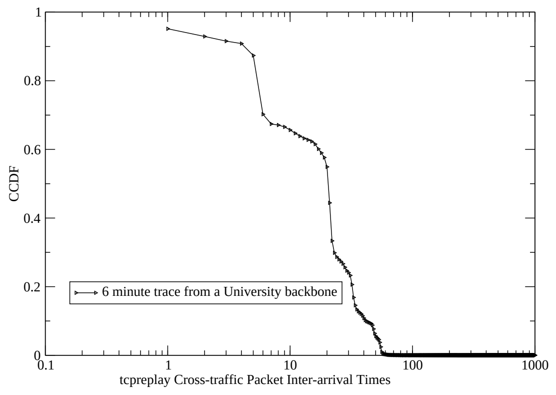
Playing Back Traces of Real Traffic We replayed previously captured and anonymized traffic traces on our laboratory end-to-end path using a tool *tcpreplay* [Tur04]. This method of cross-traffic generation reproduces actual IAT and packet size distributions but is not congestion-aware. The playback tool operated on two additional end hosts (separate from the end hosts running bandwidth estimation tools) and injected the cross-traffic into the main end-to-end path via GigE switches.

We tested bandwidth estimation tools using two different traces as background cross-traffic:

- a 6-minute trace collected from a 1 Gbps backbone link of a large university with approximately 300-345



(a) Smartbits IAT distribution



(b) Trace-replay IAT distribution

Figure 3.2: CCDF of packet IAT distribution.

Mb/s of cross-traffic load;

- a 6-minute trace collected from a 2.5 Gbps backbone link of a major ISP showing approximately 100-200 Mb/s of cross-traffic load.

Neither trace contained any jumbo frames. Packet sizes exhibited a tri-modal distribution as in [NLA04]. Packet IATs (Figure 3.2b) ranged from 1 to 60 μs. We used CoralReef to continuously measure *tcpreplay* cross-traffic on the laboratory end-to-end path and recorded timestamps of packet arrivals and packet sizes. We converted this information into timestamped bandwidth readings and compared them to concurrent tool estimates. Both traces exhibited burstiness on microsecond time scales, but loads were fairly stable when aggregated over one-second time intervals.

3.3 Evaluation Results

In this section we present tool measurements in laboratory tests using synthetic, non-congestion-aware cross-traffic with controlled traffic load (*SmartFlow*) and captured traffic traces with realistic workload characteristics (*tcpreplay*). In Section 3.4 we show results of experiments on real high-speed networks.

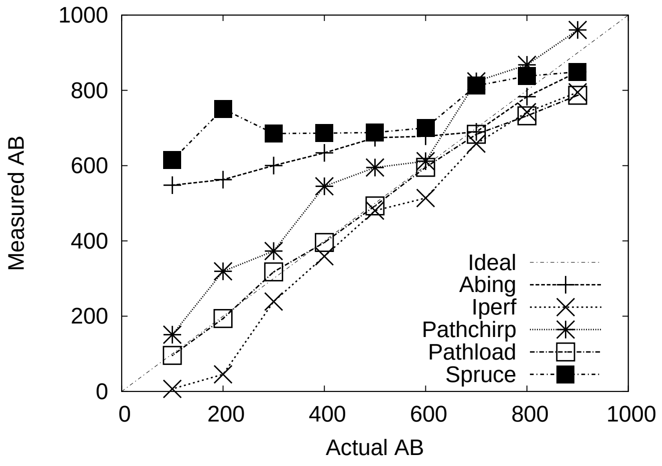


Figure 3.3: Comparison of /ab measurements on a 4-hop OC48/GigE with synthesized cross-traffic

Experiments with Synthesized Cross-traffic We used the SmartBits 6000B device with the *SmartFlow* application to generate bi-directional traffic loads, varying from 10% to 90% of the 1 Gbps end-to-end path capacity in 10% steps. We tested one tool at a time. In each experiment, the synthetic traffic load ran for six minutes. To avoid any edge effects, we delayed starting the tool for several seconds after initiating cross-traffic and ran the tool continuously for five minutes. Figure 3.3 shows the average and standard deviation of all available bandwidth values obtained during these 5 minute intervals for each tool at each given load.

Our end-to-end path includes three different routers with different settings. To check whether the sequence of routers in the path affects the tool measurements, we ran tests with synthesized cross-traffic in both directions. We observed only minor differences between directions. The variations are within the accuracy range of the tools and we suspect are due to different router buffer sizes.

We found that *abing* (Figure 3.3) reports highly inaccurate results when available bandwidth drops below 600 Mb/s (60% on a GigE link). Note that this tool is currently deployed on the Internet End-to-End Performance Monitoring (IEPM) measurement infrastructure [SLA04] where the MTU size is 1500 B, while our high-speed test lab uses a jumbo 9000 B MTU. We attempted to change *abing* settings to work with its maximum 8160 B probe packet size, but this change did not improve its accuracy.

We looked into further details of *abing* operating on an empty GigE path. The tool continuously sends back-to-back pairs of 1478 byte UDP packets with a 50 ms waiting interval between pairs. *abing* derives estimates of available bandwidth from the amount of delay introduced by the ”network” between the paired packets. *abing* puts a time stamp into each packet, and the returned packet carries a receiver time stamp. Computing the packet IAT does not require clock synchronization since it is calculated as a difference between timestamps on the same host. Since these timestamps have a μs granularity, the IAT computed from them is also an integer number of μs. For back-to-back 1500 B packets on an empty 1 Gbps link (12 Kbits transmitted at 1 ns per bit) the IAT is between 11 and 13 μs, depending on rounding error. However, we observed that every 20-30 packets the IAT becomes 244 μs. This jump may be a consequence of interrupt coalescence or a delay in some intermediate device such as a switch. The average IAT then changes to more than 20 μs yielding a bit rate of less than 600 Mb/s. This observation explains *abing* results: on an empty 1 Gbps tight link it reports two discrete values of available bandwidth, the more frequent one of 890-960 Mb/s and occasional drops to 490-550 Mb/s. This oscillating behavior is clearly observed in time series of *abing* measurements (Figure 3.4) described below.

Another tool, *Spruce* (Figure 3.3), uses a similar technique and, unsurprisingly, its results are impeded by the same phenomenon. *Spruce* sends 14 back-to-back 1500 B UDP packet pairs with a waiting interval randomly chosen between 160-1400 ms between pairs probes. In *Spruce* measurements, 244 μs gaps between packet pairs occur randomly between normal 12 μs gaps. Since the waiting time between pairs varies without pattern, the reported available bandwidth also varies without pattern in the 300-990 Mb/s range.

Results of our experiments with *abing* and *Spruce* on high-speed links caution that tools utilizing packet pair techniques must be aware of delay quantization possibly present in the studied network. Also, 1500-byte frames and microsecond time stamp resolution are simply not sensitive enough for probing high-speed paths.

In SmartBits tests, estimates of available bandwidth by *pathchirp* are 10-20% higher than the actual value determined from SmartBits settings (Figure 3.3). This consistent overestimation persists even when there is no cross-traffic. On an empty 1 Gbps path this tool yields values up to 1100 Mb/s. The reason for this overestimation is PathChirp's exponentially spaced packet trains, which detects the AB only after it exceeds the existing AB on the path.

We found that results of *pathload* were the most accurate (Figure 3.3). The discrepancy between its readings and actual available bandwidth was $<$10% in most cases.

The last tested tool, *Iperf*, estimates not the available bandwidth, but the achievable TCP throughput. We ran *Iperf* with the maximum buffer size of 227 KB and found it to be accurate within 15% or better (Figure 4e). Note that a smaller buffer size setting significantly reduces the *Iperf* throughput. This observation appears to contradict the usual rule of thumb that the optimal buffer size is the product of bandwidth and delay, which in our case would be (10^9 b/s) x (10^{-4} s) $\sim$ 12.5 KB. Dovrolis *et. al.* discuss this phenomenon in [DPJ04].

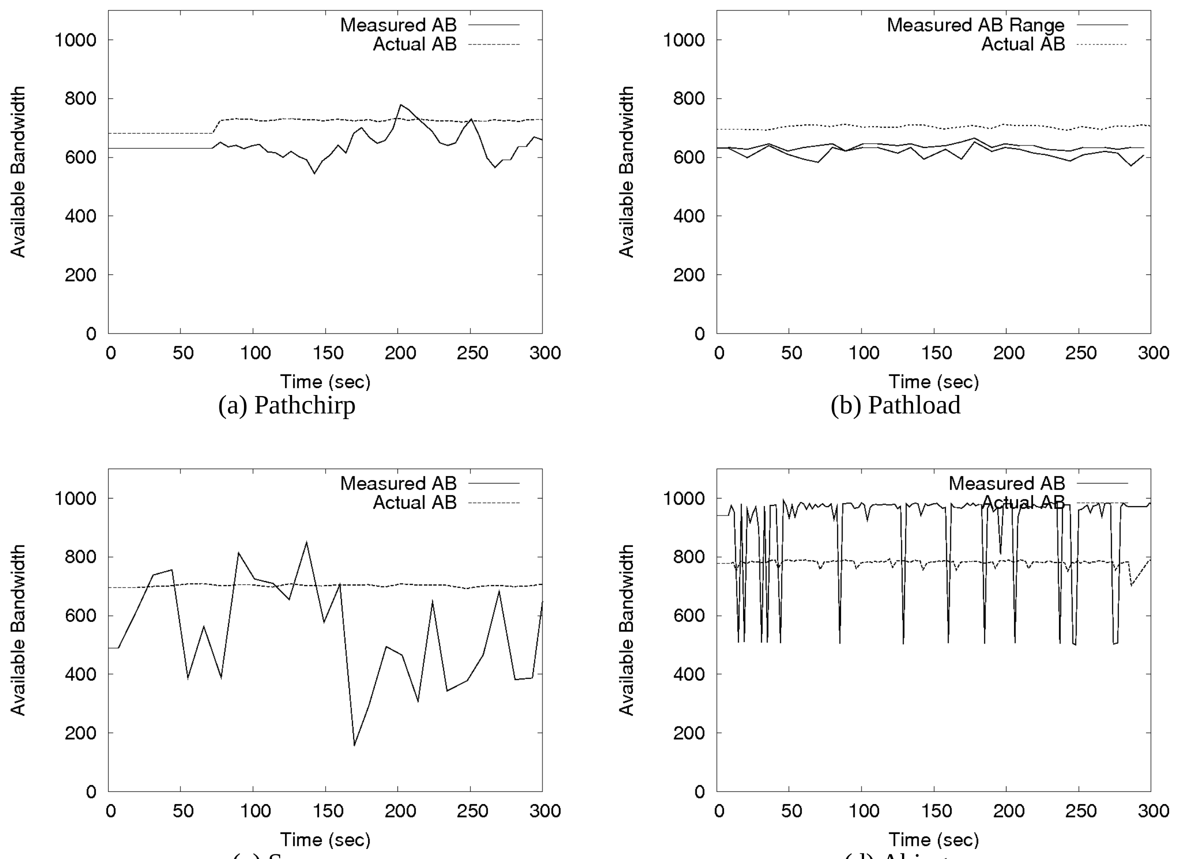
Performance of Packet-pairs Spruce and Abing, both of which operate on the packet-pair methodology suffer from the same problem of Interrupt Coalescence. In both cases, we observe that most packet-pairs show a queuing delay of 11-13 microseconds regardless of the AB. This is indicative of the fact that any spacing between the packet-pair that was introduced by cross-traffic is being lost when the packets are being buffered and acknowledged using IC. At the other end of the spectrum we observe some packet pairs arriving with a dispersion of approximately 250 microseconds which is indicative of the fact that some packet pairs are being split across the boundaries of interrupts. [PJD04] have proposed a methodology which can be used to detect and perform measurements AB in the presence of IC. However this methodology requires a long train of packets to be able to reliably gain estimates around IC, which limits is applicability to packet-pair techniques. The difference in the performance of abing and spruce are due to the fact that spruce uses a randomized algorithm to pick intervals between sending its packet pairs. Furthermore high-speed networks stress the limits of timer resolution and hence AB measurements techniques that rely on being able to precisely measure the delay between two closely spaced packets are likely to yield inaccurate estimates.

Experiments with Trace Playbacks The second series of laboratory tests used previously recorded traces of real traffic. For these experiments we extracted six-minute samples from longer traces to use as a *tcpreplay* source. As in SmartBits experiments, in order to avoid edge effects we delayed the tool start for a few seconds after starting *tcpreplay* and ran each tool continuously for five minutes.

Figure 3.4 plots a time series of the actual available bandwidth, obtained by computing the throughput of the trace at a one-second aggregation interval and subtracting that from the link capacity of 1 Gbps. Time is measured from the start of the trace. We then plot every value obtained by a given tool at the time it was returned.

As described in Section 3.2.2, we performed *tcpreplay* experiments with two different traces. We present tool measurements of the University backbone trace, which produced the load of about 300 Mb/s leaving about 700 Mb/s of available bandwidth. The tool behavior when using the ISP trace with a load of about 100 Mb/s was similar and is not shown here.

In tests with playback of real traces, *abing* and *Spruce* exhibit the same problems that plagued their performance in experiments with synthetic cross-traffic. Figure 3.4d shows that *abing* returned one of two values, neither of which was close to the expected available bandwidth. *Spruce* results (Figure 3.4c) continued to vary without pattern. One difference in the performance of Spruce, is that spruce now on an average tends to underestimate rather than overestimate the AB . This points to the fact that more spruce packets are being dispersed by



(a) Pathchirp (b) Pathload

(c) Spruce (d) Abing

Figure 3.4: Comparison of ABET measurements on a 4-hop OC48/GigE path played back real traffic.

larger values, which could point to a switch on our network path, introducing an additional delay between our probe packets.

pathchirp measurements (Figure 3.4a) had a startup period of about 70 s when the tool returned only a constant value. The length of this period is related to the tool's measurement algorithm and depends on the number of chirps and chirp packet size selected for the given tool run. After the startup phase, *pathchirp*'s values alternate within 15-20% of the actual available bandwidth.

The range reported by *pathload* (Figure 3.4d) slightly underestimates the available bandwidth by $<$16%.

Iperf reports surprisingly low results when run against *tcpreplay* traffic (Figure 3.5e). Two factors are causing this gross underestimation: packet drops requiring retransmission and a too long retransmission timeout of 1.2 s (default value). In the experiment shown, the host running *Iperf* and the host running *tcpreplay* were connected

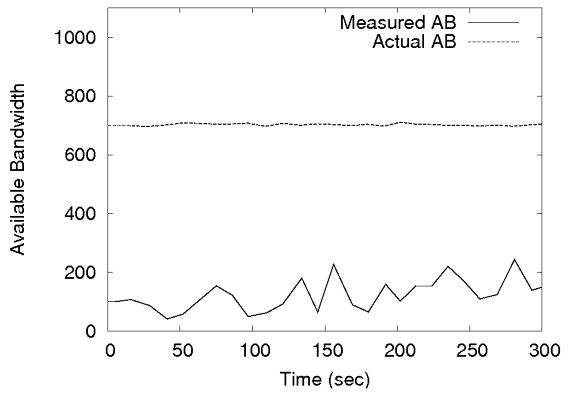


Figure 3.5: Performance of iperf on a 4-hop OC48/GigE path with played back real traffic.

to the main end-to-end path via a switch. We checked the switch's MIB for discarded packets and discovered a packet loss of about 1% when the tool and cross-traffic streams merge. Although the loss appears small, it causes *Iperf* to halve its congestion window and triggers a significant number of retransmissions. The default retransmission timeout is so large that it consumes up to 75% of the *Iperf* running time. Decreasing the retransmission timeout to 20 ms and/or connecting the *tcpreplay* host directly to the path bypassing the switch considerably improves *Iperf*'s performance. Note that we were able to reproduce the degraded *Iperf* performance in experiments with synthetic SmartBits traffic when we flooded the path with a large number of small (64 B) packets. These experiments confirm that ultimately the TCP performance in the face of packet loss strongly depends on the OS retransmission timer.

3.3.1 Comparison of Tool Operational Characteristics

We considered several parameters that may potentially affect a user's decision regarding which tool to use: measurement time, intrusiveness, and overhead. We measured all these characteristics in experiments with SmartBits synthetic traffic where we can stabilize and control the load. We define tool measurement time to be the average measurement time of all runs at a particular load level. On our 4-hop OC-48/GigE topology, the observed measurement durations were: 1.3 s for *abing*, 11 s for *Spruce*, 5.5 s for *pathchirp*, and 10 s for *Iperf* independent of load. The *pathload* measurement time generally increased when the available bandwidth decreased, and ranged between 7 and 22 s.

We define tool intrusiveness as the ratio of the average tool traffic rate to the available bandwidth, and tool overhead as the ratio of tool traffic rate to cross-traffic rate (Figure 3.3.1). *pathchirp*, *abing*, and *Spruce* have low

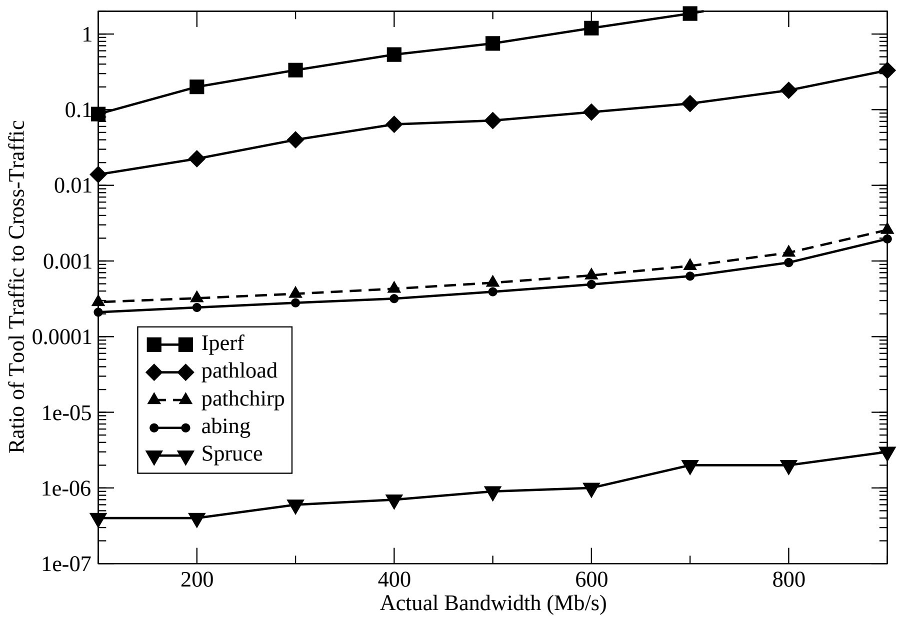


Figure 3.6: Tool overhead vs. available bandwidth.

overhead, each consuming less than 0.2% of the available bandwidth on the GigE link and introducing practically no additional traffic into the network as they measure. *pathload* intrusiveness is between 3 and 7%. Its overhead slightly increases with the available bandwidth (that is, when the cross-traffic actually decreases) and reaches 30% for the 10% load. As expected, *Iperf* is the most expensive tool both in terms of its intrusiveness (74-79%) and overhead costs. Since it attempts to occupy all available bandwidth, its traffic can easily exceed the existing cross-traffic.

3.4 Real World Validation

Comparisons of bandwidth estimation tools have been criticized for their lack of validation in the real world. Many factors impede if not prohibit comprehensive testing of tools on production networks. First, network conditions and traffic levels are variable and usually beyond the experimenters' control. This uncertainty prevents unambiguous interpretation of experimental results and renders measurements unreproducible. Second, a danger that tests may perturb or even disrupt the normal course of network operations makes network operators reluctant to participate in any experiments. Only close cooperation between experimenters and operators can overcome both obstacles.

We were able to complement our laboratory tests with two series of experiments in the real world. In both setups, the paths we measured traversed exclusively academic, research and government networks.

Experiments on the Abilene Network We carried out the available bandwidth measurements on a 6 hop end-to-end path from Sunnyvale to Atlanta on the Abilene Network. Both end machines had a 1 Gbps connection to the network and sourced no traffic except from running our tools. The rest of links in the path had either 2.5 or 10 Gbps capacities. We ran *pathload*, *pathchirp*, *abing*, and *Iperf* for 5 min each, in that order, back-to-back. We concurrently polled the SNMP 64-bit InOctect counters for all routers along the path every 10 s and hence knew the per-link utilization with 10 s resolution. We calculated the per-link available bandwidth as the difference between link capacity and utilization. The end-to-end available bandwidth is the minimum of per-link available bandwidths. During our experiments, the Abilene network did not have enough traffic on the backbone links to bring their available bandwidth below 1 Gbps. Therefore, the end machines' 1Gbps connections were both narrow and tight links in our topology. Due to some logistical constraints we could not run Spruce during the same experiment. We therefore present the results of the Spruce run separately in Figure 3.4 [1].

Figure 3.4 shows our tool measurements and SNMP-derived available bandwidth. Measurements with *pathload*, *pathchirp*, and *Iperf* are reasonably accurate, while *abing* and *spruce* readings wildly fluctuate in the whole range between 0 and 1000 Mb/s.

The discrepancy between *Iperf* measurements and SNMP-derived values reflects tool design: *Iperf* generates large overhead ($>$70%) because it intentionally attempts to fill the tight link. Consequent readings of SNMP counters indicate how many bytes traversed an interface of a router during that time interval. They report total number of bytes without distinguishing tool traffic from cross-traffic. If a tool's overhead is high, then available bandwidth derived from SNMP data during this tool run is low. At the same time, since tools attempt to measure available bandwidth ignoring their own traffic, a high-overhead tool will report more available bandwidth than SNMP. Therefore, *Iperf* shows a correct value of achievable TCP throughput of $\sim$950 Mb/s while concurrent SNMP counters account for *Iperf*'s own generated traffic, and thus yield less than 200 Mb/s of available bandwidth. A smaller discrepancy between *pathload* and SNMP results reflects *pathload*'s overhead ($\sim$10% per our lab tests).

Experiments on the SDSC-ORNL Path In the second series of real-world experiments we tested *abing*, *pathchirp*, *pathload*, and *Spruce* between our host at SDSC (running CAIDA reference FreeBSD version 4.8)

[1]We also tested *Spruce* in the other series of real network experiments, see subsection on SDSC-ORNL paths below

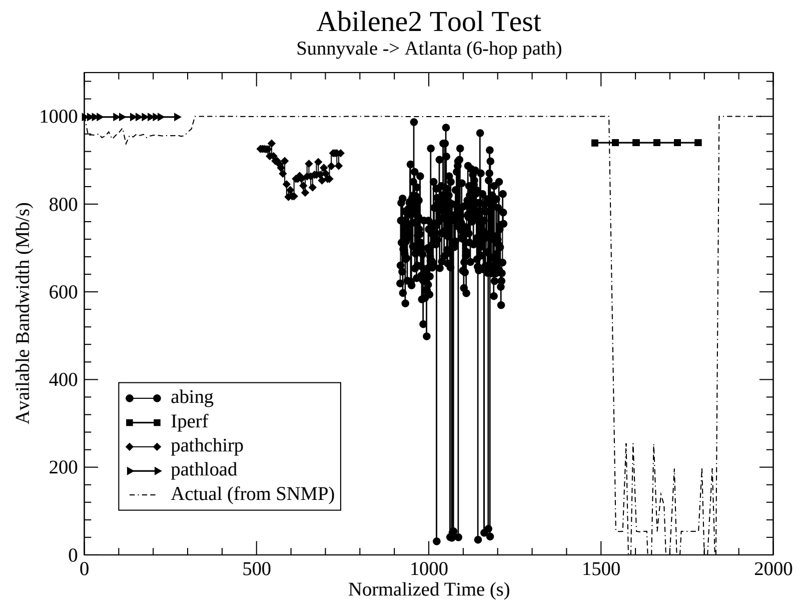


Figure 3.7: Real world experiment conducted on the Abilene network

.

and a host at Oak Ridge National Lab (running Red Hat Linux release 9 with a 2.4.23 kernel and Web100 patch [Mat03]). These experiments are of limited value since we did not have concurrent SNMP data for comparison with our results. However, we had complete information about link capacities along the paths which at least allows us to distinguish plausible results from impossible ones. We include these experiments since they present first insights into the interplay between the probing packet size and the path MTU.

The two paths we measured are highly asymmetric. The SDSC-ORNL path crosses CENIC and ESNet, has a narrow link capacity of 622 Mb/s (OC12) and MTU of 1500 bytes. The ORNL-SDSC path crosses Abilene and CENIC, has a narrow link capacity of 1 Gbps and supports 9000-byte packets end-to-end. Both paths remained stable over the course of our experiments and included OC12, GigE, 10 GigE, and OC192 links. Under most traffic scenarios, it seems highly unlikely for the 10 Gbps links to have less than 1 Gbps of available bandwidth. Lacking true values of available bandwidth from SNMP counters for absolute calibration of tool results, we assume that the narrow link is also the tight link in both our paths.

We ran each tool using either 1500 or 9000 byte packets. *abing*, *pathchirp*, and *pathload* support large probe packet size as an option[2]. *Spruce* uses a hard-coded packet size of 1500 bytes; we had to trivially modify the code

[2]The *abing* reflector has a hard-coded packet size of 1478 bytes.

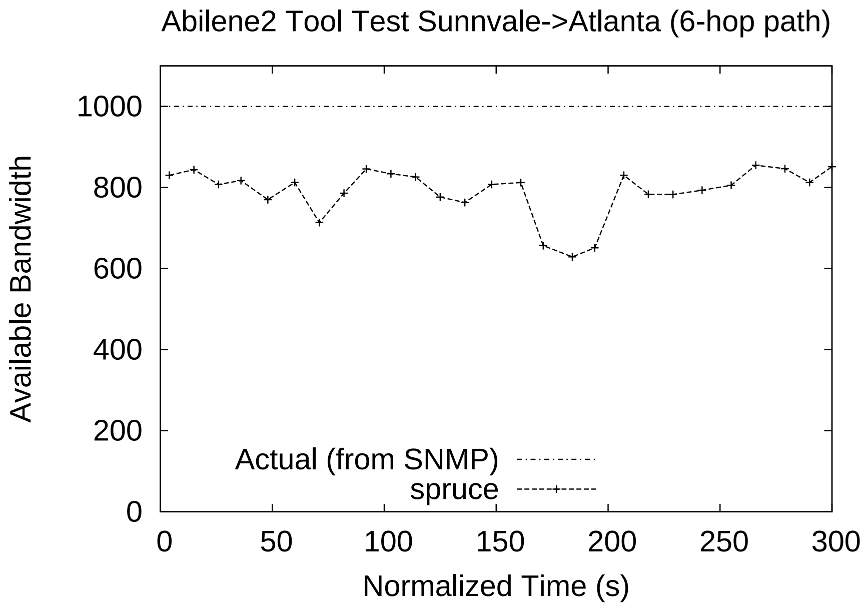


Figure 3.8: Real world experiment conducted on the Abilene network for spruce.

Table 3.1: Summary of wide-area bandwidth measurements ("f"= produced no data).

Direction	**Path Capacity, MTU**	**Probe Packet Size**	**Tool readings (Mb/s)**			
			abing[a]	**pathchirp**	**pathload**	**Spruce**
SDSC to ORNL	622 Mb/s (OC12), 1500	1500	178 / 241	543	>324	296
		9000	f / 664	f	409 – 424	0
ORNL to SDSC	1000 Mb/s (GigE), 9000	1500	727 / 286	807	>600	516
		9000	f / 778	816	846	807

[a]Sender at SDSC for 1st value and at ORNL for 2nd value.

to increase the packet size to 9000 B. Table 3.1 summarizes our results while a detailed description is available in [Hyu04].

abing has a sender module on one host and a reflector module on the other host and measures available bandwidth in both directions at once. We found that its behavior changed when we switched the locations of sender and reflector. *abing* with 9000 B packets did not return results from SDSC to ORNL ("f" in Table 3.1). We could see that the ORNL host was receiving fragmented packets, but the *abing* reflector was not echoing packets. In the opposite direction, from ORNL to SDSC, *abing* with 9000 B packets overestimates the available bandwidth for the OC12 path (reports 664 Mb/s on 622 Mb/s capacity). Note that almost the factor of 3 difference in GigE path measurements with 1500 B packets (727 and 286 Mb/s) may be due to different network conditions since these tests occurred on different days.

pathchirp produced results on both paths when run with 1500 B packets and on the GigE path with 9000 B packets, but failed on the OC12 path with large packets. There does not appear to be any significant advantage to using large packets over small ones. Variations between consequent measurements with the same packet size are sometimes greater than the difference between using large and small packets.

In tests with 1500 B packets, on both paths *pathload* reports that results are limited by the maximum host sending rate. With 9000 B packets, this tool yielded available bandwidth estimates for both paths, but issued a warning "actual probing rate [does not equal] desired probing rate" for the OC12 path.

Spruce performed poorly in experiments with small packets from SDSC to ORNL, reporting wildly fluctuating values of available bandwidth. Tests with 9000 B packets in this direction always produced 0 Mb/s. However, in the ORNL to SDSC direction, its readings were more consistent and on par with other tools.

We suspect that fragmentation is responsible for most of the problems when probing packet size and path MTU mismatch. While using large packets to measure high-speed links is beneficial, more work is necessary to consistently support large packets and to reduce failures and inaccuracies stemming from fragmentation.

3.5 Conclusion

We find from the results of our experiments with *abing* and *Spruce* that tools utilizing packet pair techniques must be aware of delay quantization possibly present in the studied network. Also, 1500 byte frames and microsecond time stamp resolution are not sensitive enough for probing high-speed paths. *Pathload* and *Pathchirp* perform the best since they do have the facility to use greater than 1500 packet size and both the tools can also detect interrupt coalescence. *Iperf* performs well on high-speed links if run with its maximum buffer window size. However even small ($\sim$1%) but persistent amounts of packet loss seriously degrade its performance. Conservative settings of the OS retransmission timer further exacerbate this problem. Thus we observe that performance of several ABETs are affected by systemic and implementation biases. In order to better understand the performance of the various ABETs, we need to isolate the impact that these systemic and implementation biases have on the performance of ABETs and re-evaluate the ABETs.

CHAPTER 4
Impact of Temporal Parameters

This dissertation argues that the performance of ABETs is not only influenced by algorithmic and implementation-related design choices, but also by the sampling-related temporal parameters that they operate with. Unfortunately, these has been only a limited study of the latter. In this chapter, we attempt to study the impact of temporal aspects of sampling on the AB estimation process. Our approach is especially designed to isolate and ignore the impact of algorithmic and implementation-related aspects of ABET design. We begin by identifying the parameters of interest to us.

4.1 Temporal Parameters of Interest

AB is a time-varying process and any bandwidth-estimation tool must necessarily sub-sample the process. We focus on the following fundamental dimensions of sampling the AB process:[1].

Measurement Timescale: A critical parameter in the definition of AB in Equation (1.1) is the length, $(t_2 - t_1)$, of the time interval over which it is observed—we refer to this quantity as the *measurement timescale* (MT). In Fig 1.2, we plot the time-series of AB, observed at three different timescales of $10ms$, $50ms$, and $1s$, during the *same 30 s* observation period on an Internet link. We observe that the AB process can appear quite different depending on the timescale at which it is observed. In particular, it is likely that the MT impacts the accuracy as well as variability of the AB measured by a given ABET. Consequently, any application that relies on such a tool would want the tool to measure AB at an MT relevant to the application domain. For instance, while a large-file-transfer application is likely to be interested in only the average AB obtainable at super-second timescales, a media-streaming application is likely to also be interested in knowing the small-timescale variations in AB.

[1]The importance of considering measurement timescales and durations has also been mentioned in [JD04]. However, the impact of these parameters on AB measurement has not been quantified.

Unfortunately, most existing ABETs do not explicitly select (or report) the MT used in AB estimation. Furthermore, the implicit choices of MT made by these tools can only be roughly estimated, and are a function of the path transmission capacity and tool configuration parameters. Tools such as Spruce [SKK03a] and Abing [Nav03], that rely on using a packet-pair as a probe stream, have a MT on the order of $12\mu s$, on a $1Gbps$ path—this corresponds to the separation between two back-to-back $1500B$ packets.[2] Tools such as Cprobe [CC96b], PathChirp [Rib03], and Pathload [JD02b], that instead rely on using longer packet trains as probe streams, have a much larger MT—ranging from $10ms$ to several hundreds of ms on a $1Gbps$ path. The exact value of the MT for a probe stream depends on the size of the packet train and the rate at which it is sent—both of these factors are adaptive in Pathload and PathChirp. Iperf [ipe], which is a tool used primarily for diagnostic purposes, measures the maximum throughput that a TCP connection can attain[3]—the MT is the same as the total tool run-time.

In this chapter, we study how the choice of MT by a tool impacts the accuracy, variability, and stability of the measured AB. We use four different values of MT, representative of existing tools, that differ by more than an order of magnitude: $10ms$, $50ms$, $100ms$, and $500ms$.

Measurement Duration (Run-time): *Run-time* (RT) refers to the length of the time interval over which several samples of the AB process are collected, and used to infer properties of the AB process. In practical terms, the run-time is the total time taken by a tool from invocation to reporting an AB estimate. This includes the time taken to send several probe streams (each of which potentially returns one sample of AB), and converge on an AB estimate.

The most significant impact of run-time on AB measurement is in terms of its variability. For a given MT and SI, the longer is the tool run-time, the more variable are likely to be the different AB samples collected. On the other hand, longer run-times are more likely to yield a *sufficient* number of samples for reliably estimating the mean as well as variability in the AB process.

RT (as well as MT) is also likely to affect the stability of the measured AB in the post-measurement periods. A longer run-time is likely to yield more reliable AB estimates, that are not subject to short-term traffic-load fluctuations, and are indicative of the AB that can be expected for some time.

[2]Tools that rely on packet-pairs have been shown to be inaccurate, especially on high-speed paths [SMH$^+$05]. This is conjectured to be so primarily because of the small MT—at such timescales, the AB process appears quite bursty. As a result, it is difficult to get reliable and stable AB estimates. We exclude such timescales from our analysis in this study.

[3]It has been shown in [DJ03] that TCP throughput is not an accurate measure of AB.

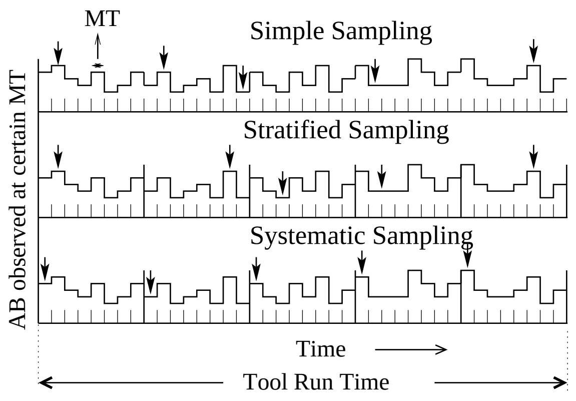


Figure 4.1: Factors Affecting the AB process

Existing AB tools vary widely in their typical run-times—an recent evaluation study of AB tools reports the typical run-times of Abing, Spruce, Pathchirp, Iperf, and Pathload to be: 1, 2, 5, 10, and 20 seconds, respectively [SMH+05]. We use these values to study the impact of tool run-time on the variability as well as stability of the AB process during measurement and post-measurement periods, respectively.

Sampling Strategy and Intensity: Given an observation timescale, the *AB process* consists of a series of back-to-back readings of AB observed within a given time interval. ABETs essentially only *sub*-sample this AB process—the sampling strategy and the fraction of the AB process sampled are likely to impact the accuracy of estimating the mean AB in a given time interval. For instance, larger is the sampling rate, better is likely to be the AB estimation accuracy; however, greater would be the network overhead.

Existing tools differ in the fraction of the AB process—henceforth, referred to as the *sampling intensity* (SI)—that they sample during the tool run-time. In our analysis, we vary this fraction from 0.1 to 0.9 ($10-90\%$ of the AB process gets sampled). We vary SI by simultaneously controlling the MT and the sampling rate (number of AB samples collected per second). SI is given by the product of MT and the sampling rate divided by the RT .

Given a sampling rate, existing tools also differ in their *sampling strategy*—the manner in which AB samples are selected from within a given time interval. We use the framework described in [CPB93] to study three kinds of sampling strategies (see Fig 4.1): (i) *Simple sampling*, in which AB samples are selected randomly from within the given time interval; (ii) *Stratified sampling*, in which the time interval is divide into equi-sized units, and one sample is selected randomly from each unit; and (iii) *Systematic* *sampling*, in which the time interval is divided into equi-sized units and the first AB reading from each unit is used as a sample. Spruce uses simple sampling; while Pathchirp, Pathload, and Abing use systematic sampling.

We organize the issues raised above in the form of three main questions: (i) How does the choice of sampling strategy, sampling intensity, MT and RT impact the *accuracy* of the estimated AB? (ii) How does the choice of MT and RT affect the *variability* of the measured AB? (iii) How *stable* is AB in the post-measurement periods? Answering these questions reveals the impact that the above mentioned temporal parameters have on the performance characteristics of the ABETs .

Our work represents, to the best of our knowledge, the *first* investigation of AB measurement along these dimensions.

4.2 Analysis Methodology

As mentioned before, existing ABETs differ significantly in the algorithmic design of their probe streams and inference logic, as well as in their implementation efficiencies. In order to answer the questions raised above in a tool-independent manner, hence, we assume the existence of a perfect probing stream—referred to as an *Istream*—and a corresponding perfect inference logic, that can infer the sampled AB perfectly by analyzing the performance of an *Istream*. This assumption lets us study the impact of (currently) design-agnostic quantities—namely, run-time, measurement timescale, and sampling intensity and strategies—while isolating the analysis from the impact of design-dependent algorithmic and implementation factors. Formally, we assume that when run over a link i of transmission capacity C_i, the Istream-based AB estimate would be precisely equal to the quantity formalized in Equation 1.1.

Unfortunately, a perfect tool does not exist in practice to be used for our study. Instead, the assumption about the tool simply lets us adopt a *passive* trace-analysis based approach for answering the above questions, in which it is possible for us to compute the *ground truth*. As illustrated in the discussion before, the availability of a link-level packet trace gives us the ability sample the link with 100% sampling intensity and lets us compute perfectly the AB process on the corresponding link at different timescales. We rely on the Coralreef [KMK$^+$01] package for doing this. A perfect trace of the AB process can also be sub-sampled according to a given sampling intensity, sampling strategy, timescale, as well as run-time in order to answer the questions raised above.

It is important to note that the use of link-level packet traces gives us access to the AB process of only a *single link*, and not the *end-to-end* AB process of a network path. Computing the latter passively would require access to the link-level packet traces of *all* the constituent links of a path—given the limited number of publicly-available packet traces, that is currently infeasible. Note, however, that in practice, analyzing just the link-level AB process may not be a significant limitation. This is because most end-to-end paths are expected to have at most a single bottleneck link [JD02b, JD04, SKK03b, Nav03, Rib03], and the AB process on such a bottleneck governs the end-to-end AB process [SMH$^+$05].

4.3 Data Sets

In order to perform our analysis we use link-level packet traces collected from 8 different locations (15 different bidirectional links). Table 4.1 lists these traces. As can be observed all links have Gigabit or higher capacities. Our traces are diverse in the link-locations, traffic loads, and user-communities represented. The UNC and Leip traces were collected, respectively, at the edges of the University of North Carolina and the University of Leipzig. The Abilene, MFN, Cesca, Paix, and San Diego traces were obtained from CAIDA and NLANR.

		Average Load (Mbps)	
Trace	Link Capacity	Forward Direction	Reverse Direction
UNC	1 Gbps	328.8	88.2
Leip	1 Gbps	13.1	35.8
Cesca	1 Gbps	228.2	245.9
SanDiego	1 Gbps	68.0	39.3
Abilene IC	2.5 Gbps	421.6	518.4
Abilene IK	2.5 Gbps	320.7	585.8
MFN	2.5 Gbps	349.1	608.1
Paix	2.5 Gbps	107.2	n/a

Table 4.1: Data sets used

Our results therefore should be applicable to high-speed networks on which ABETs are expected to be increasingly deployed. One caveat to extrapolating these results is that all the traces that we have are for links that are heavily multiplexed (large number of connections), some of these conclusions could potentially change if the bottleneck link on the path were the first or last link (access links) or links with only a few connections on them.

4.4 Putting things into perspective

The analysis of temporal properties of Internet traffic has been an area of intense research. One of the seminal works in this area was by Leland et. al. [LTWW93], which uncovered the self similar nature of Internet traffic. This work suggested that Internet traffic observed at different timescales can be characterized by similar distributions. This was a fairly significant result since it challenged the popularly held notion that the aggregated traffic on Internet links could be modeled as a Poisson process, just as was done for telephone networks. This work illustrated that Internet traffic patterns are significantly more complicated and can lead to large queue build-up in routers, even when the average traffic load is moderate. This has some serious implications for router manufacturers, since this implies that router buffers now need to be much larger than what was earlier thought using conventional Poisson modeling.

A counter view point presented by Cleveland et. al. [CCLS01] asserts that with increasing multiplexing and load the long range dependence property that was observed in Internet traffic tends toward an independent process. The arguments on either side are fairly exhaustive but beyond the scope of this discussion. A good discussion and further resources in the area of traffic modeling can be found in [KMF04].

Another area of work has focused on understanding the causes for long-range dependence in Internet traffic—heavy-tailed distributions of object sizes has been identified as a leading cause [PKC96]. Some also argue that the TCP protocol mechanisms also contribute to this behavior [GCM00]. However, as observed in [JD05b], the time-scales of this long range dependence is limited to larger, super-RTT timescales. Hao et. al. [JD05b] investigate the role of TCP in causing burstiness at sub-RTT time-scales. They illustrate that most of the variability at these time-scales can be explained by TCP self-clocking in the dominant flows whose delay-bandwidth product significantly exceeds its window size—the resultant burstiness extends up to the typical RTT of the dominant TCP flows. Given that we are interested in understanding the AB process at timescales that are typically in the sub-RTT time-scale, the above result could be useful in putting our observations into context.

We relate our observations to those made in the literature throughout this chapter. This is especially important because our analysis of temporal aspects of bandwidth estimation is limited to only a finite number of Internet traces. Thus, even though our observations are consistent across all traces that we analyze, we present them in the light of other evidence that exists in the area of traffic modeling.

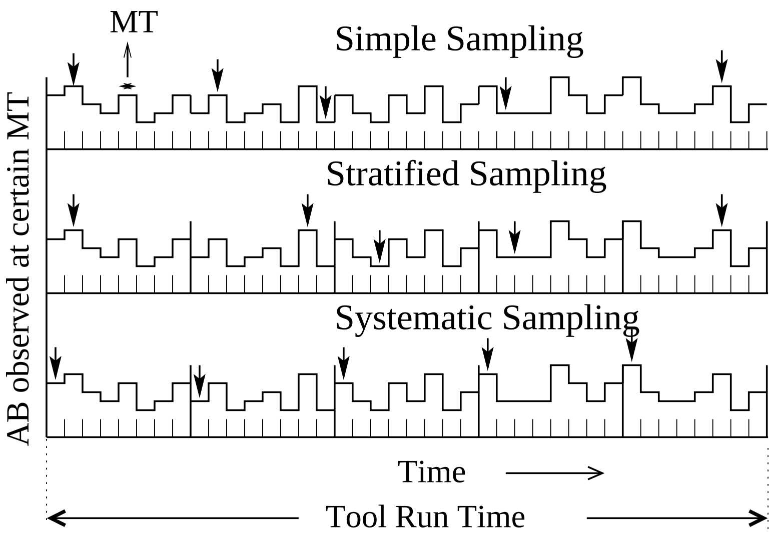


Figure 4.2: Sampling strategies.

4.5 How does the way the AB is sampled affect the accuracy?

AB estimation tools necessarily *sub*-sample the AB process during their run-time. In what follows, we evaluate the impact of sampling strategies, intensity, timescale, and duration on the accuracy of the sampled AB. It is worth noting that a recent experimental study has shown that the accuracy of existing ABETs is no better than 10% on high-speed paths [SMH$^+$05]. In this section, consequently, we consider any inaccuracy smaller than this value as insignificant.

4.5.1 Does the choice of sampling strategy impact accuracy of the sampled AB?

We consider the three kinds of sampling strategies—simple, stratified, and systematic—as described in Chapter 1 (refer Figure 4.2). For a given choice of MT, SI, and RT, we analyze each packet-trace as follows: (i) we translate the trace into a corresponding AB process observed at the timescale MT; (ii) we divide the AB process into segments of time-length RT each (see Fig 4.2); (iii) for each segment, we compute the average, AB_{avg}, of the AB process observed within that segment; (iv) within each segment, we sub-sample the AB process according to the three sampling strategies, and compute the averages of the samples as: AB_{sim}, AB_{stra}, and AB_{sys}, respectively, for simple, stratified, and systematic sampling (see Fig 4.2); (v) we compute the *sampling inaccuracies* for each segment as: $|AB_{avg} - AB_{sim}|$, $|AB_{avg} - AB_{stra}|$, and $|AB_{avg} - AB_{sys}|$, respectively; (vi) we compute the cumulative distribution (CDF) of these three inaccuracy metrics, over all segments in the trace.

Strategy	UNC-0			Abilene-IC1			Abilene-IK1		
	5 %	50 %	95 %	5 %	50 %	95 %	5 %	50 %	95 %
Simple	0.04	0.38	1.10	0.12	1.3	3.91	0.13	1.45	4.64
Systematic	0.08	0.9	3.14	0.62	6.21	19.0	0.64	6.84	29.38
Stratified	0.79	3.12	5.57	0.59	6.74	19.23	0.72	7.49	29.78

Figure 4.3: Sampling Strategy vs. Accuracy (Mbps)

Fig 4.3 lists the 5%, 50%, and 95% of the inaccuracies for the three sampling strategies, observed within the UNC-0, Abilene-IC1, and Abilene-IK1 traces, with MT = 10 ms, SI = 0.7, and RT = 10s. We observe that the median inaccuracy in measuring AB is smaller with simple sampling (within 1.5 Mbps and 0.4 Mbps for the Abilene and UNC traces, respectively) than with systematic or stratified sampling (7 Mbps and 3 Mbps for the Abilene and UNC traces, respectively). A similar trend is visible for the 95% values of the computed inaccuracies. However, for *all* traces analyzed, we find that even the 95% values of the inaccuracies lie within 10% of the link AB—this is close to the resolution accuracy of existing ABETs as shown in Figure 4.4. Thus, it may be fair to conclude that *although simple strategy is likely to yield better sampling accuracy, the inaccuracies of systematic and stratified sampling are not significant for current tools.* Since most existing ABETs rely on systematic sampling, we use it in all of our subsequent analysis.

Note: *Claffy et. al. [CPB93] conducted a study on the impact of varying sampling strategies on sampling network traffic in order to get a representative sample of traffic flowing on a link. This study found that there was a significant improvement in performance when the sampling was event-based rather than time-based. However for the process of AB estimation since we have no direct access to events like increase in the load on the network (except in the extreme cases of congestion), we are bound to sample the AB using a timer-driven approach. Here Claffy et. al. found that the relative difference between adopting different sampling strategies was not significant, as is illustrated in our results as well.*

4.5.2 How does probe-stream duration impact the accuracy of estimated AB?

The duration of individual probe streams transmitted within the run-time of a tool determine its MT—indeed, each probe stream samples the AB process for this amount of time. In order to assess the impact of this MT on a tool's accuracy, we analyze each trace as follows. Using systematic sampling, for a given RT, SI, and MT , (i) we compute the AB process at MT, and divide it into segments of time-length RT each, as described in Section 4.5.1;

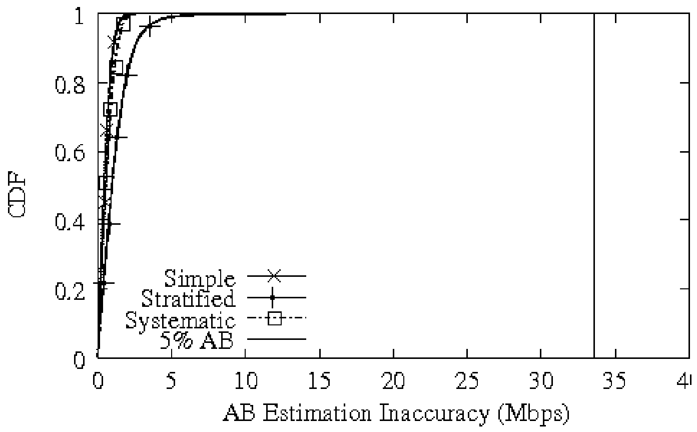


Figure 4.4: CDF of Inaccuracies with different sampling strategies

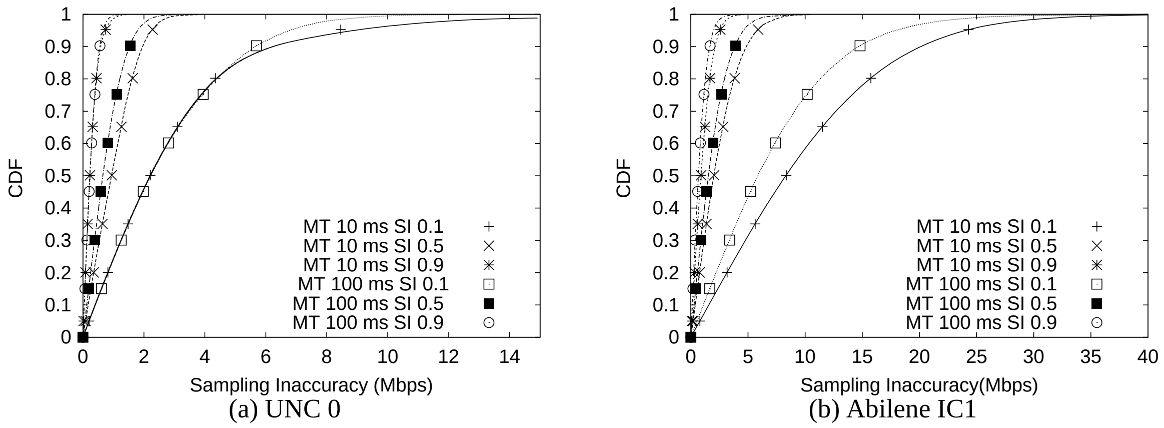


Figure 4.5: Impact of MT on accuracy

(ii) for each segment, we then compute AB_{avg} and AB_{sys}; (iii) we compute the CDF of the sampling inaccuracy $|AB_{avg} - AB_{sys}|$ observed over all segments within the trace.

We use an RT of 10s and compute the above CDFs for MT of 10 ms and 100 ms, and SI of 0.1, 0.5, and 0.9. Figs 4.5(a) and 4.5(b), plot these CDFs for the Abilene-IC1 and UNC-0 traces, respectively. We observe that for a given MT, and as expected, increasing the SI improves the accuracy of the sampled AB. We also observe that for a given SI, MTs that differ by even an order of magnitude have a negligible impact on the sampling accuracy. Thus, while SI impacts the measurement accuracy significantly, MT does not.

The above observations have the following implications for ABET design: (i) *The same sampling accuracy* *may be attained by a tool by either using a few long probe-streams, or several short probe-streams (as long as both result in the same SI)*. The latter may be useful for applications that benefit from the timely-availability of an initial AB estimate, even if its only roughly accurate. The first few probe streams are likely to yield such a rough estimate quickly, while the later probes make the estimate robust. This flexibility may not be available if longer, fewer probe streams are used. (ii) *Any application-specific MT may be used for sampling, without impacting the measurement accuracy significantly, as long as an inversely proportional number of samples are collected at that timescale* (thus, maintaining the same SI).

4.5.3 What is the marginal cost of increasing sampling intensity?

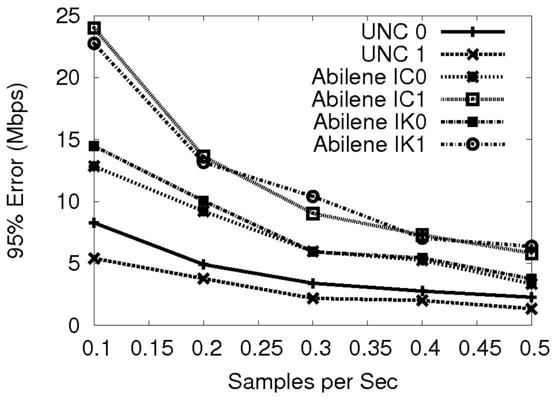


Figure 4.6: Sampling Accuracy vs. SI

The observations made above indicate that the sampling intensity has a significant impact on the accuracy of the sampled AB; we next examine this impact quantitatively. For this, for each trace, we compute the CDF of the sampling inaccuracy, $|AB_{avg} - AB_{sys}|$ for a given choice of MT, SI, and RT, exactly as described in Section 4.5.2. Fig 4.6 plots the 95% of sampling inaccuracy observed with different values of SI, with an MT of 10 ms and an RT of 10 s, for several traces. As expected, we find that increasing the SI decreases the sampling inaccuracy—however, the marginal improvement in sampling accuracy decreases with increasing SI. In particular, *an ABET is unlikely to improve its sampling accuracy significantly beyond a sampling intensity of* 30%—maintaining a low SI can help the ABET reduce the network overhead of AB estimation.

4.5.4 How does RT impact accuracy?

Finally, we evaluate the impact of the tool run-time on its sampling accuracy. For each trace, we compute CDFs of the sampling inaccuracy $|AB_{avg} - AB_{sys}|$ as described before. For MT = 10 ms and SI = 0.5, Fig 4.8 plots the 95% value from the CDFs, as a function of RT. We find that as RT increases, the sampling inaccuracy decreases. This is to be expected, as a larger RT yields a larger number of AB samples for a given SI—we find, however, that the marginal improvement in sampling accuracy reduces with increasing RT. In particular, *an ABET is unlikely to improve its sampling accuracy significantly beyond an RT of 5 s.*

RT	SI	5 %	50 %	95 %	5 %	50 %	95 %	5 %	50 %	95 %
(s)		UNC-0			Abilene-IC1			Cesca-0		
1	0.5	0.154	1.660	4.7	0.321	3.4	10.29	0.159	1.725	5.06
2	0.4	0.1457	1.579	4.62	0.291	3.16	9.50	0.144	1.57	4.613
10	0.2	0.155	1.66	4.91	0.25	2.714	9.215	0.133	1.467	4.18
20	0.1	0.186	1.895	8.2	0.291	3.12	9.97	0.148	1.64	4.7

Figure 4.7: RT vs. Inaccuracy (Mbps)

Observe that increasing RT or SI has a positive impact on the sampling accuracy. However, increasing either of these also results in a proportional increase in the total probe traffic introduced into the network. We next ask: *does any one of these two parameters represent a better tradeoff between the sampling accuracy and network overhead?* Fig 4.7 lists the 5%, 50%, and 95% values of the inaccuracy CDFs, computed with an MT of 10 ms, for several combinations of (RT, SI): (20s, 0.1), (10s, 0.2), (2s, 0.4), and (1s, 0.5). We find that for a given trace, the sampling inaccuracies are similar for the combinations of: (1s, 0.5) and (20s, 0.1), as well as for: (2s, 0.4) and (10s, 0.2). This observation has two implications. First, it implies that an ABET can achieve similar AB estimation accuracy by sampling more intensely within a shorter run-time. In particular, this contradicts a claim made in [JD04] that a tool with a longer run-time is likely to yield more accurate AB estimates—our analysis indicates that this is only true if the sampling intensity is held constant. In reality, *it is thus possible to design a faster tool without sacrificing estimation accuracy, by simply increasing the sampling intensity of the tool*. Second, note that in order to maintain the same accuracy, the relative increase in SI is larger than the relative reduction in run-time. Thus, *a single invocation of a faster tool that achieves similar accuracy, is likely to insert more probe-traffic into the network.*

In the next two sections, we evaluate the impact of MT and RT on the variability and stability of the AB process. For the analysis in the rest of this chapter, we assume a sampling intensity of 1 (the AB process is

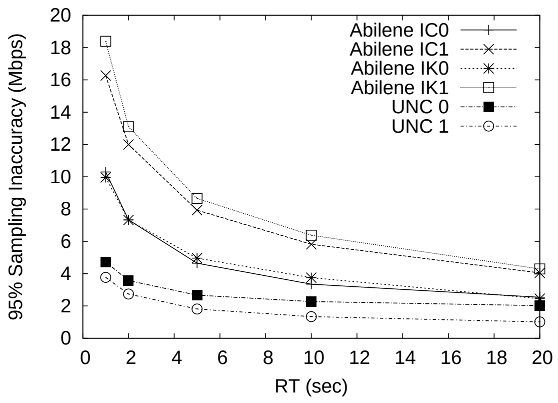


Figure 4.8: Run-Time vs. Accuracy

observed completely by an ABET).

4.6 How does the MT and RT affect variability?

Recall from Fig 4.9 that the AB process can exhibit low-to-high variability, depending on the timescale at which it is observed. Furthermore, the longer is the tool run-time, the greater is the opportunity to witness variability in the corresponding AB process. To quantify these effects, we next evaluate the impact of MT and RT on AB variability. Our objective is to find the set of timescales and durations that characterize an AB process with low variability.

The importance of reporting the variability in AB, in addition to its average, has been recognized recently [JD05a]. In fact, a variant of a popular AB estimation tool called Pathload reports variability in the form of the maximum and minimum AB observed during the tool's run [JD05a]. In this section, we first address the issue of *what metric is appropriate for characterizing AB variability as a function of MT and RT?* In particular, we investigate whether for a given value of MT and RT, the *standard-deviation*—which is likely to be more robust to outliers—is a more predictable metric than the *range* metric described above.

What is a predictable measure of variability? We analyze each trace as follows: (i) we compute the AB process at MT, and divide it into segments of time-length RT each, as described in Section 4.5.1; (ii) for each segment, we compute the range, AB_{range}, as the difference between the maximum and minimum AB observed in that segment; we also compute the standard deviation, AB_{std}, of the AB values observed in that segment; (iii)

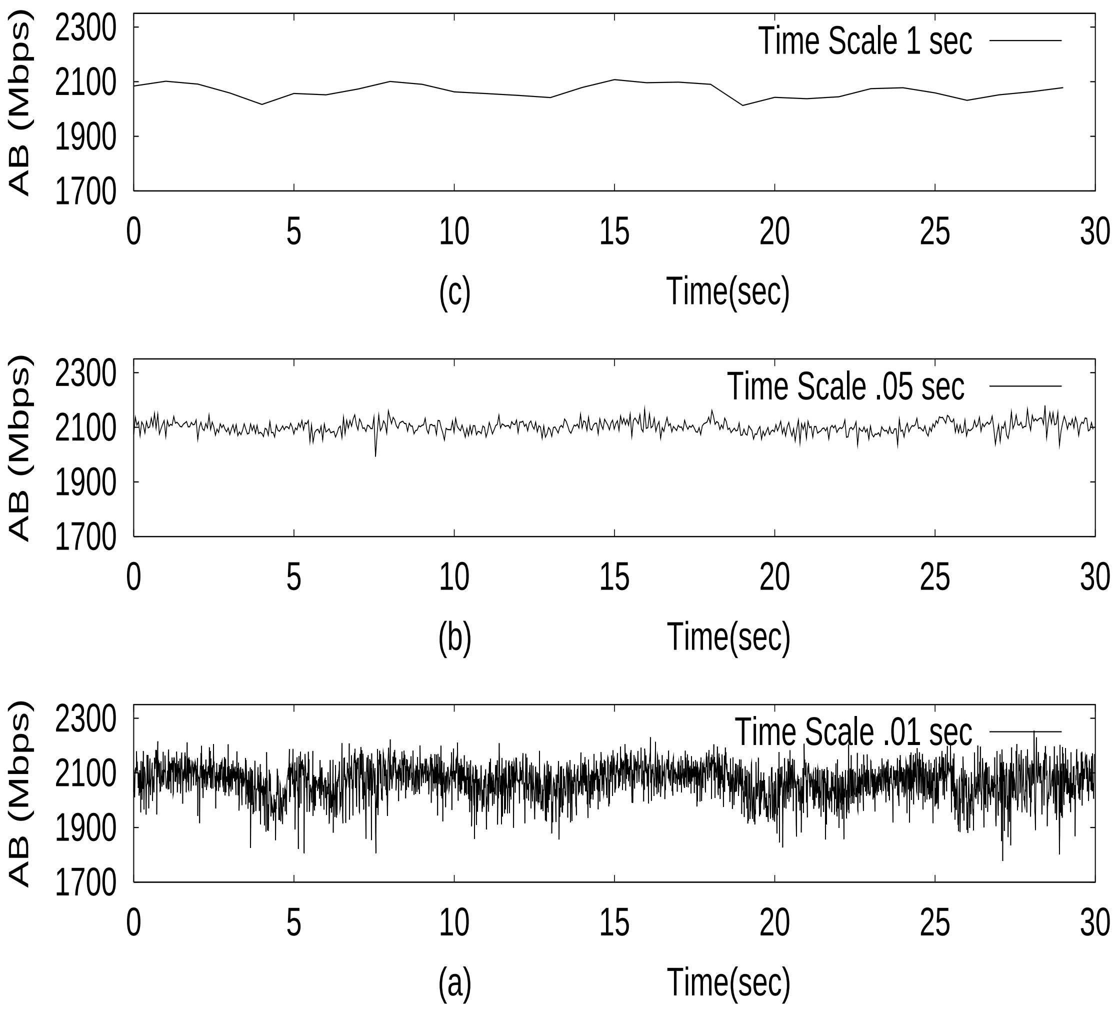


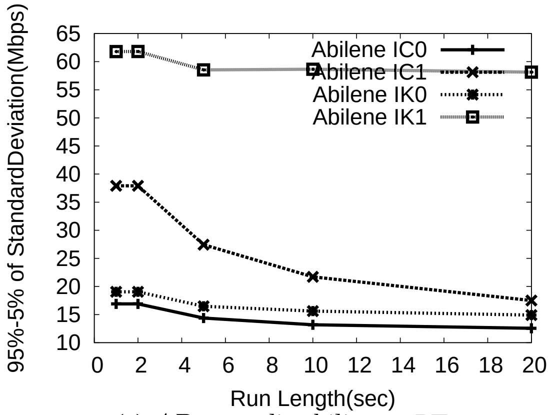
Figure 4.9: AB process observed at link during same 30 s window at different MT .

RT (s)	MT (ms)	AB_{range} (Mbps)			AB_{std} (Mbps)		
		5 %	50 %	95 %	5 %	50 %	95 %
1	10	215.01	288.7	390.1	44.6	56.9	73.8
	50	54.4	81.3	126.6	14.4	20.7	31.3
	100	31.6	56.7	96.8	9.5	16.9	28.8
5	10	293.5	367.2	479.4	47.7	58.7	73.1
	50	88.5	117.6	176.8	17.1	22	32.6
	100	69.7	103.5	154.3	13.7	21.2	33.3
20	10	355.5	436.7	571.4	51.1	60.4	73
	50	117.5	153.4	238.8	19.1	23.6	37.4
	100	107	144.5	212.9	17.8	25.2	36.7

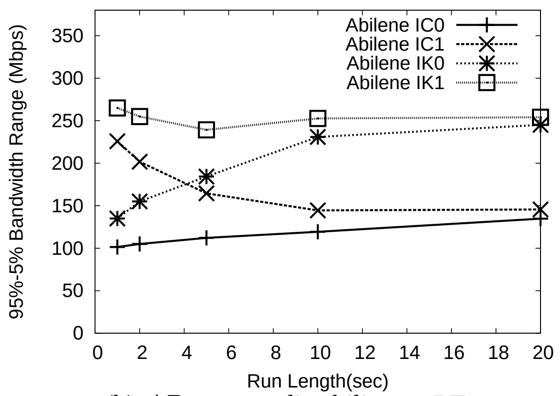
Table 4.2: Abilene: AB variability metrics

RT (s)	MT (ms)	AB_{range} (Mbps)			AB_{std} (Mbps)		
		5 %	50 %	95 %	5 %	50 %	95 %
1	10	118.6	150.9	193.3	22.3	26.5	31.1
	50	34.5	50.7	71.3	9.1	12.7	17.1
	100	17.6	29.2	45.6	5.4	8.8	13.1
5	10	160.3	188.1	231.7	23.9	26.9	30.0
	50	57.6	71.3	93.2	11.1	13.4	16.4
	100	34.6	45.7	64.5	7.5	9.7	12.8
20	10	190.5	218.8	267.3	24.8	27.1	29.5
	50	74.2	87.6	113.9	12.3	13.8	16.3
	100	48.4	59	84.8	8.7	10.2	13

Table 4.3: UNC: AB variability metrics



(a) AB_{std} predictability vs. RT



(b) AB_{range} predictability vs. RT

Figure 4.10: Impact of RT on range and standard deviation of AB

we compute the CDFs of the AB_{range} and AB_{std}, as observed over all segments within the trace.

Tables 4.2 and 4.3 list the 5%, 50%, and 95% values of the AB_{range} and AB_{std} CDFs, for the Abilene-IC1 and UNC-0 traces, respectively. We observe that for any given MT and RT, the difference between the 95% and 5% values of AB_{range} is much larger than that of AB_{std}. This implies that for a given combination of MT and RT, the latter is a more predictable metric of variability as compared to the former.

Furthermore, we find that the predictability of AB_{std} improves with increase in RT, whereas the predictability of AB_{range} does not. This is illustrated in Figs 4.10 (a) and 4.10 (b), that plot the difference between the 95% and 5% values of these two metrics respectively, as a function of RT and when MT = 10ms. We observe that the difference decreases with RT for the AB_{std} metric, but exhibits no such trend for the AB_{range} metric. This implies that the *standard-deviation is a better choice to use for characterizing AB variability*. Furthermore, *tools with longer run-times are likely to report more robust variability estimates.*

How does RT impact AB variability? From Tables 4.2 and 4.3, we also observe that for a given MT, as the RT increases, the median value (as well as other percentiles) of AB_{std} also increases. The relative increase in the variability, however, is small. This suggests that *tools with longer run-times are likely to report only slightly higher values of AB variability*.

How does MT impact AB variability? For any given RT, Tables 4.2 and 4.3 indicate that as MT increases, AB_{std} reduces. The reduction in variability is most significant at smaller timescales. For instance, at an MT of

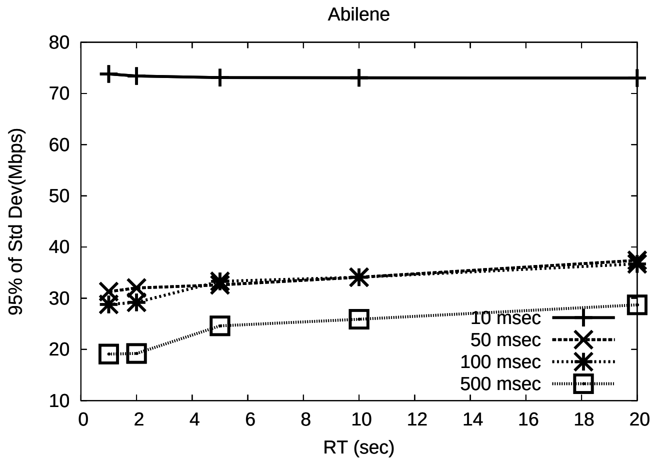


Figure 4.11: Impact of MT and RT on variability

$10ms$, AB_{std} can be as high as $100Mbps$ (maximum observed value) for the Abilene-IC1 trace. At an MT of $50ms$ or higher, AB_{std} lies within $40Mbps$ for all traces (including Abilene-IC1). This latter value corresponds to less than 2% of the link capacity, which is within the resolution accuracy of all existing ABETs [SMH$^+$05]. This implies that *in order to sample an AB process that does not exhibit significant variability, ABETs should sample it at timescales of* $50ms$ *or higher*. In particular, *the results of ABETs that rely on using* packet-pairs *instead of longer* packet-trains *are likely to be significantly impacted by AB variability*. Figure 4.11 illustrates the impact that MT and RT have on the variability of the AB process.

Note: *It is observed in [JD05b] that the range of timescales at which Internet traffic exhibits high variability depends on traffic specific characteristics such as the RTTof the dominant flows. We expect that in the traces we analyze (all collected from links well within the North America continent), such flows have a typical RTTof around 50 ms. We also expect that if we repeat our analysis on traces collected off of trans-continental links, the timescales beyond which variability in the sampled AB is low, would be larger. A detailed analysis of these traffic characteristics is beyond the scope of this dissertation.*

4.7 How does the RT impact the stability of estimates?

Applications that rely on using the knowledge of the AB can do so only after the measurement is made. The implicit assumption here is that the AB does not change from the time it was being measured to the time after the measurement was made. In order to study the validity of this assumption we study the study the stability of

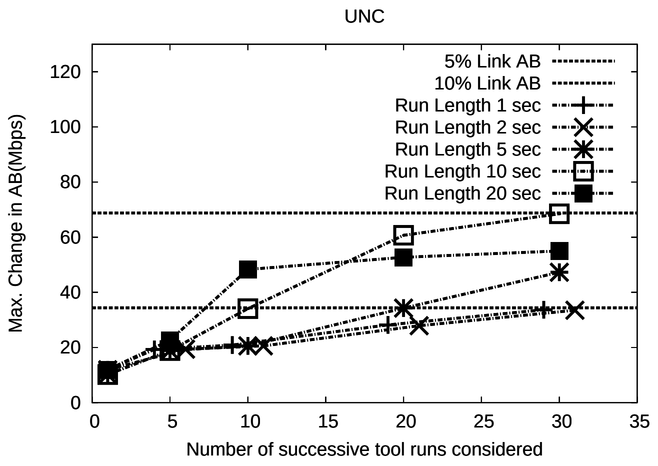


Figure 4.12: Stability of AB

the AB in post-run periods. For a given RT and N (the number of successive tool runs examined) we analyze each trace as follows: (i) we compute the AB process with MT equal to RT, and divide it into segments of time-length RT each, as described previously; we denote the AB reading of the i^{th} segment by AB^i_{avg}; (ii) for the i^{th} segment, we compute the *post-run deviation* metric as: $PRD_N = \max_{j\in[1,N+1]}\{|AB^i_{avg} - AB^{i+j}_{avg}|\}$; (iii) we compute the CDF of the PRD_N observed over all segments within the trace.

Figure 4.12 plots the 95% of the UNC-0 trace against different values of N. Each line represents, a different value of RT and the two horizontal lines represent 5 and 10 % of the link AB .

As expected, we observe that as the value on N increases, PRD_N increases. However an interesting data point at N=1, shows that in back-to-back tool runs the AB does not change by more than a factor of 4%. If we know the error range of a tool,*(i) we can use that knowledge in conjunction with the fact that back-to-back AB measurements do not change significantly to reduce the number of measurements we make*. For instance if we knew that the accuracy of an ABET was at most 5%, then on the basis of Figure 4.12, we could make one estimate in an RT and that estimate would be valid for at least 5 RT after the measurement.

Fig 4.4 lists the 5%, 50%, and 95% of the observed PRD_N, for RT = $\{$1s, 5s, 20s$\}$ and N = $\{$1, 5, 30$\}$, for the Abilene-IC1 and UNC-0 traces.

Here again we find that in any pair of neighboring tool-runs, the AB does not change by more than $12Mbps$ or $40Mbps$ for the UNC-0 and Abilene-IC1 traces, respectively. In fact, we find that for *all* of the traces analyzed, the AB measured in a pair of back-to-back tool runs does not differ by more than 4%, most of the time.

RT (s)	N	Abilene (Mbps)			UNC (Mbps)		
		5 %	50 %	95 %	5 %	50 %	95 %
1	1	1.3	13.1	41.2	0.3	3.7	11.9
	5	15.4	36.6	75.4	3.9	9.2	19.3
	30	37.4	63.3	128.6	9.8	16.1	33.7
5	1	0.8	11	39.5	0.3	3	10.2
	5	10.8	30.6	95.1	3.5	8	18.9
	30	37.4	81	162.1	8.9	14.5	47.3
20	1	1.0	11.6	59.1	0.2	2.6	11.9
	5	16.1	43.4	120.8	2.3	7.7	22.6
	30	56.2	111	204.4	11.1	19.2	55

Table 4.4: Stability in AB

The above observation is relevant for the design of ABETs for applications that need to continuously monitor the AB on a path by running an ABET repeatedly. In particular, consider the case when such applications use Pathload-like ABETs, that spent a considerable portion of their run-time in arriving at a *coarse* estimate of AB , and then work on fine-tuning that estimate. Such ABETs could exploit the fact that *AB does not change significantly between neighboring tool runs*, and use the result of the *last* tool-run as the *coarse* estimate of the current AB —this should result in the next tool-run completing much faster, while also introducing much less probe traffic into the network path.

4.8 Conclusion

From the above study we have gain insights into the impact that temporal parameters have on the AB process and their impact on ABET designs. We summarize our findings below:

- **Accuracy Related:** (i) A higher sampling intensity results in better sampling accuracy, although the gains are insignificant beyond a SI of 30%. (ii) The choice of MT does not impact sampling accuracy significantly, as long as SI is maintained. In particular, the same sampling accuracy may be attained by a tool by either using a few long probe-streams, or several short probe-streams (both with the same SI).

- **Variability Related:** (i) The AB process exhibits significant variability at MTs smaller than 50 ms. This corresponds to sending several packets within each probe stream (unlike ABETs that use packet-pairs). (ii) As RT increase the variability also increases.

- **Stability Related** (i)The average AB does not change significantly across neighboring back-to-back tool runs. This observations can be exploited for applications that need to continuously monitor the AB on a path by running an ABET repeatedly.

CHAPTER 5
Impact of Probe-Stream design and Inference Logic

We next study the impact of replacing a perfect probe-stream and inference logic, with realistic probe-stream and inference logic combinations. We select several prominent ABETs from Table 2.1 —namely, Pathload [JD02b], PathChirp [Rib03], Spruce [SKK03a], IGI [HS03], Fast-IGI [HS03], and Cprobe [CC96c]—that represent existing diversity in the algorithmic techniques used for inferring end-to-end AB. We implement each of these tools in the NS-2 [NS2] network simulation environment. We rely on published literature as well as publicly-available implementations (whenever available) to extract details of each tool.

5.1 Setting the MT and SI in AB estimation methodologies

Besides studying the impact of the probe-stream design and inference logic on the performance parameters, we are also interested studying the impact of the MT and SI. Unfortunately even though most ABET techniques have many algorithm specific parameters that could be set none of the tools allow these two parameters to be set. In order to be able to set the MT and the RT we redesign our tool interfaces to accommodate this change. We will briefly describe the changes below.

5.1.1 Incorporating MT

As mentioned in Section 1 the MT is the length of the probe-stream that is injected into the network. We control the length of the probe stream in different ways in each of the ABET methodologies.

- Cprobe/IGI/Fast-IGI All three of these tools send several probe streams, each at a uniform rate, in order to estimate the AB—while Cprobe sends all streams at a high bit-rate, IGI/Fast-IGI iteratively change the bit-rate of each stream in order to converge on the AB. The number of packets sent within each probe stream is typically fixed. In order to incorporate MT, we make the number of packets (N) a *configurable* value that is set such that when the stream is sent at the desired bit-rate (R) and default packet size (P) (of 1500 Bytes), the stream duration is equal to MT: $N = \lceil R \times MT/P \rceil$.

- Pathload Like IGI, Pathload also iteratively sends several probe streams at different bit-rates. However, the Pathload AB inference logic requires that the number of packets sent in each probe stream be a perfect square. In order to incorporate MT, we first compute a rough estimate of N using the relation: $N = \lceil R \times MT/P \rceil$. If this is not a perfect square, we decrease P by the least amount required to ensure that it is.[1]

- Spruce Spruce relies on a packet-pair based AB inference technique, which sends two packets back-to-back in each probe stream. Unfortunately, this fixes the MT to a small value. However, due to the small probes and open-loop nature of Spruce, its run-time is fairly small; often smaller than the values of MT that may be of interest to us. We exploit this property to redesign Spruce's interface—given a desired MT, we aggregate and compute the average of all AB estimations made within a duration of MT, in order to estimate the AB at that MT.

- PathChirp In PathChirp, each probe-stream—also referred to as a *chirp*—is an exponentially-spaced stream of N packets. The inter-packet spacing and N are determined using three parameters: the lower rate, L, the upper rate, U, and the spread-factor, S. Specifically, the spacing between packet i and $i+1$ is given by: $P/(L \times S^{i-1})$, and N is computed using the relation: $U = L \times S^N$. In order to incorporate MT, we first compute the length of a chirp as the following sum of a geometric series: $P/L \times (1 - \frac{1}{S^n})/(1 - \frac{1}{S})$. Given L and U, we then select the pair (S, N) such that the above chirp length is close to the desired MT.

5.1.2 Incorporating the SI

The SI is the fraction of time that is occupied by a tool in probing for the AB during its total RT . If G is the gap between successive probe-streams, SI is given by: $\frac{MT}{MT+G}$. This relation can be used to control SI in open-loop tools such as Cprobe, Spruce,and PathChirp—specifically, given an SI, the gap is set to: $G = MT \times \frac{1-SI}{SI}$.

In *closed-loop* tools such as Pathload/IGI/Fast-IGI, however, the construction of a probe-stream is determined by the delays experienced by the previous probe-stream—these tools, therefore, can not send more than one probe-stream per RTT. Thus, the SI can not be set to a value higher than $MT/MT + RTT$, which is a fairly low value for typical Internet paths. Thus, for all practical purposes, SI can not be controlled in closed-loop tools. It is interesting to note, however, that the path RTT is likely to impact the feedback-loop and, hence, the performance (especially the run-time) of such tools.

[1]This convoluted way of controlling the MT in Pathload (and in PathChirp, as described later) highlights the limitation of existing ABET designs.

5.2 Performance Metrics

We characterize the performance of each ABET using two types of metrics:

- *Accuracy-related:* Each run of an ABET should yield a good estimate of the end-to-end AB. In order to quantify the accuracy of an ABET estimate, we compute its *AB estimation error* as the difference between the estimated AB and the actual AB. The actual AB of a link is computed as the ratio of the number of bits that traverse the link during the tool run, to the tool run-time.

- *Cost-related:* We quantify the cost of using an ABET with several metrics. The *run-time* is defined as the time taken by a tool to return an estimate. The faster a tool runs, the better and valid its AB estimates are. Since we are relying on a simulation environment, this time is primarily governed by the number and sizes of probe-streams and the convergence logic used to estimate AB. For closed-loop tools, the run-time is also affected by the path RTT. The *probing overhead* is defined as the total amount of network probe traffic sent by the tool in order to arrive at a single estimate of AB. For large-scale deployment and use of ABETs, it is important that they use low amount of probe traffic. The *intrusiveness* is defined as the average bit-rate of a tool—this is given by the ratio of the overhead to the run-time. Since the run-times of ABETs can differ by orders of magnitude, it is important to compare the rate at which they inject probe traffic.

  In addition, we study the impact of probe traffic on the response time of ongoing TCP connections.

We conduct several types of experiments in order to study the above metrics—we describe these next.

5.3 Validation

The accuracy of most ABETs is typically established by their proponents by running them on links shared by cross-traffic with a *constant bit-rate* (CBR). We validate our NS-2 implementations of the selected ABETs by using the network topology depicted in Figure 5.1.[2] We run CBR cross-traffic between nodes N2 and N3, and instances of ABETs between nodes N0 and N1. We vary the cross-traffic load from 100 Mbps to 900 Mbps and for each load, we record the AB estimates from several back-to-back runs of each tool. It is important to note

[2]Unless stated otherwise, all link capacities and link delays in all of our topologies are set to $1Gbps$ and $10ms$, respectively, and sufficient buffers are provisioned to avoid packet losses.

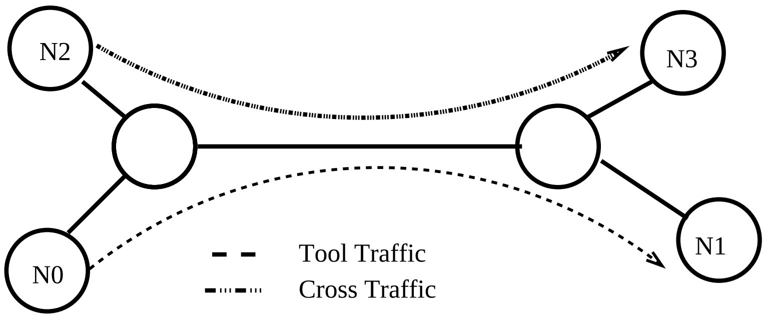


Figure 5.1: Topology with a Single Bottleneck Link

that all of our evaluations are conducted in high-speed gigabit networks—most ABET designs have not been evaluated in such a setting previously.

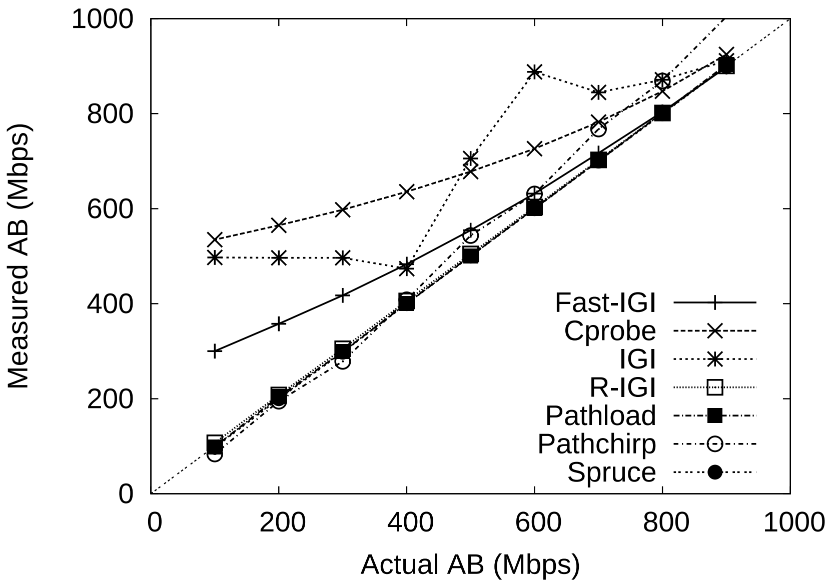


Figure 5.2: Validation of ABET Implementations

Figure 5.2 plots the average of the estimated AB against the actual AB. We find that Pathload and Spruce are quite accurate in reporting the AB. PathChirp estimates deviate slightly at higher values of AB—we run the same set of experiments using a publicly-available NS-2 implementation of PathChirp and find that the AB estimates are quite similar to our implementation. CProbe, IGI, and Fast-IGI do a poor job of estimating the AB in the high-speed setting simulated. Cprobe works on the simple logic of sending a stream of packets at a fairly high rate (given by the bottleneck capacity)—the rate at which the probe-stream arrives at the receiver is taken as the estimate of the end-to-end AB. It has been shown in [JD04] that the receiving rate in such cases is not a good estimate of AB. Due to its inaccuracy, we do not use Cprobe for our subsequent evaluations.

We next investigated the reasons for the poor performance of IGI. Note that IGI always over-estimates the AB. On careful examination of the IGI design and implementation, we discovered a key design factor that was leading to over-estimation on high-speed network paths:

The equation used for estimating the cross-traffic load (Equation 3 in [HS03]) uses the link capacity as the multiplier—in our understanding, it should be using the current sending rate as the multiplier.

We change our IGI implementation accordingly to create a new version, henceforth referred to as R-IGI. Figure 5.2 also plots the results of R-IGI validation—we find that R-IGI performs quite well. In our subsequent experiments, we use R-IGI.

Our implementation of Fast-IGI (validated in Fig 5.2) also incorporates the above-mentioned changes. However, it still leads to high estimation errors when the traffic load is higher than 500 Mbps. Since most of our subsequent evaluations are not conducted at such high loads, we include Fast-IGI in our subsequent evaluations.

5.4 Evaluating the Accuracy of ABETs in Dynamic Traffic Conditions

The validation experiments presented in Section 5.3 also confirm the high accuracy of several prominent ABETs when the network traffic load does not change. In reality, this is seldom the case with loaded Internet links. In order to reproduce in our simulations, the dynamic traffic conditions that characterize real Internet links, we rely on *replaying* packet-level traces collected from several Internet links. Specifically, we collect five 1-hour packet traces (from four different Internet links) which are summarized in Table 5.1—the traffic load of these traces ranges from $160Mbps$ to $530Mbps$. We then use the *replay trace* module in NS-2 for creating an exact replica of the link-level packet-arrival process (and consequently, the AB process) for each trace. In this section, we evaluate ABET accuracy against this type of cross-traffic

Trace	Traffic Type	Average Load
Ibiblio	Web server access link	160 Mbps
UNC05	University access link	230 Mbps
UNC28	University access link	358 Mbps
IPLS-CLEV	Internet2 backbone link	410 Mbps
IPLS-KSCY	Internet2 backbone link	530 Mbps

Table 5.1: Traces used for evaluations

5.4.1 Single Bottleneck Scenario

We first evaluate the tools using the topology of Fig 5.1, but with the traces replayed (instead of CBR traffic) as cross-traffic between nodes N2 and N3. This topology represents paths on which an ABET is likely to encounter only a single congested link. We use this setup to study the impact of traffic load, MT, SI, and RTT on the AB estimation accuracy of different tools.

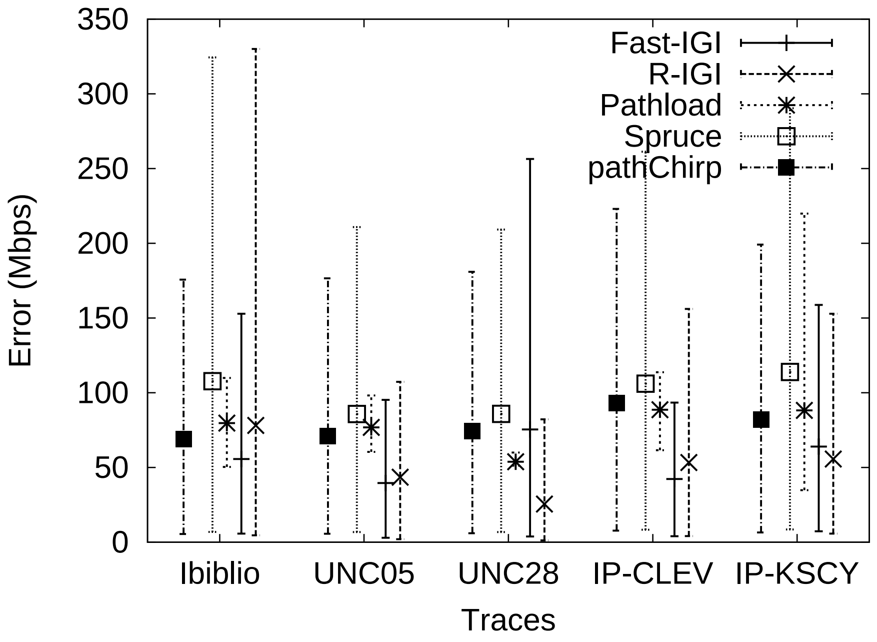
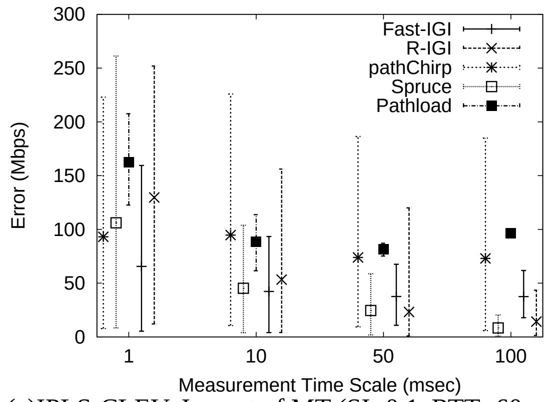


Figure 5.3: Tool errors with default parameters

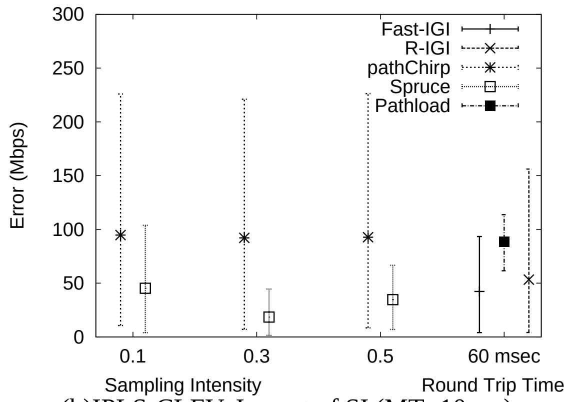
Default Tool Configuration

We first run each ABET against all five traces, using the default configuration of tool parameters, which dictate the implicit choices of MT and SI—the default MT for Pathload, PathChirp, R-IGI, Fast-IGI, and Spruce are roughly: 10 ms, 10 ms, 1 ms, 1 ms, 0.5 ms, respectively, and the default SI for both Spruce and PathChirp is 0.1. Each tool is run back-to-back for 300 seconds and the AB estimation error of each run is computed. Fig 5.3 plots the average, and the 5- and 95-percentiles (as error bars), of this estimation error for each tool and trace used. We observe that:

- The average estimation errors of ABETs are higher with dynamic cross-traffic than with CBR cross-traffic



(a)IPLS-CLEV: Impact of MT (SI=0.1, RTT=60 ms)



(b)IPLS-CLEV: Impact of SI (MT=10 ms)

Figure 5.4: Impact of MT, SI, and RTT

(Section 5.3), and range from 20 - 120 Mbps. Pathload, PathChirp, and Fast-IGI have similar average estimation errors, while R-IGI has lower and Spruce has higher errors.

- The estimation errors vary widely around the average. The variability is least for Pathload and quite high for Spruce and PathChirp—estimation errors sometimes exceed 300 Mbps.
- For each tool, the AB estimation errors are similar across the five traces, even though the traffic load in these traces are quite different. However, it is important to remember that the highest link utilization represented by these traces is only 53%—it is not clear if higher loads would impact estimation errors.

Impact of MT, SI, and RTT

The default choices of MT and SI vary widely across existing ABETs [SK06]. In order to compare tool performance in an unbiased manner, we next systematically control MT, SI, and RTT, and study the impact on the AB estimation error of each ABET. Specifically, we select MT from (1, 10, 50, 100 ms), SI from (0.1, 0.3, 0.5) for open-loop tools, and RTT from 60-300 ms for closed-loop tools—these values are representative of the diversity found in existing ABETs and Internet paths [SK06, AKSJ03].

Fig 5.4 plots the average and 5- and 95- percentiles of the AB estimation error with the IPLS-CLEV trace—the trends are quite similar for the other traces. We observe that:

- Increasing the MT improves the accuracy of all ABETs. This is to be expected—larger MTs imply that a larger number of probe packets interact with the cross-traffic and are able to better sample the AB process.

However, the gain in accuracy is most significant at fine time-scales. The gains are negligible beyond an MT of 50 ms.

The impact of MT on PathChirp is lower than on the other tools. This is due to the exponential inter-packet spacing in the probe streams—the number of probes sent does not increase proportionally with MT.

- More importantly, by keeping the MT the same across different tools, the relative performance difference between the tools changes! Most significantly, Spruce now is the most accurate, while it was the least accurate with the default settings of MT.
- SI has a negligible impact on the AB estimation accuracy of the open-loop tools, Spruce and PathChirp. This result may seem contrary to the observations made in [SK06] that high values of SI lead to better sampling accuracy—it is important to note, however, that the AB estimation accuracy is also limited by the accuracy of the inference logic used by the respective tools. Our observations indicate that increasing the rate of probing the AB process is not likely to help improve the accuracy of current tools.
- Similar to SI, RTT has no impact on the AB estimation accuracy of the closed-loop tools, Pathload and R-IGI.

5.4.2 Multiple Bottlenecks

Bottleneck Location—Different Tight and Narrow Links

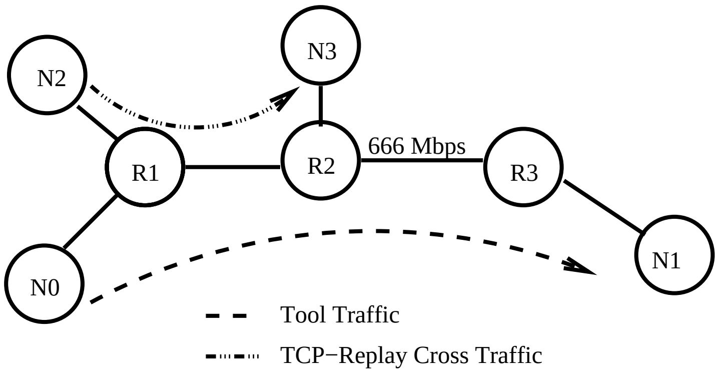
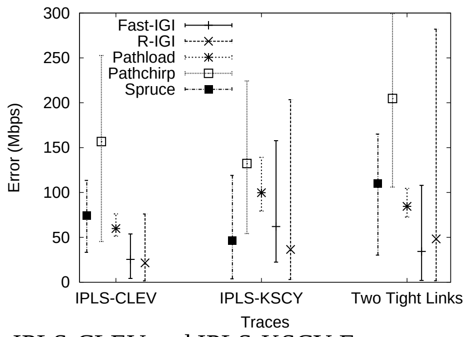


Figure 5.5: Different tight and narrow links

The inference logic of several ABETs—including IGI and Spruce—is based on the premise that on the path for which AB is to be estimated, the tight as well as narrow link are the same. In practice, this may not be the



IPLS-CLEV and IPLS-KSCY Error ranges

Figure 5.6: Performance with multiple bottleneck links (MT=50ms, SI=0.1)

case with many Internet paths—indeed, an ISP access link that is shared among a large user population may have a lower AB than the last-mile narrow link for many broadband users. In order to study ABET performance on such paths, we simulate the topology of Fig 5.5. The 666 Mbps link between the routers R2 and R3 is the narrow link (all other links have a 1 Gbps capacity). The ABETs run between the nodes N0 and N1. We replay traces between nodes N2 and N3 in order to ensure that link R1-R2 is the tight link for the tool traffic—for this, we use two traces: IPLS-CLEV (410 Mbps) and IPLS-KSCY (530 Mbps). We compute the actual end-to-end AB in any given time interval as the *minimum* of the AB on links R1-R2 and R2-R3. We use this to compute the AB estimation error for each tool run. Fig 5.6 plots the average and the 5- and 95-percentiles of the AB estimation errors observed from several back-to-back tool runs, with MT of 50 ms, and SI of 0.1. We observe that the error of PathChirp and Spruce increases by a factor of 2-3, compared to the scenario of Fig 5.1, while the performance of other ABETs is not impacted much. With this change, the relative rankings of Spruce and PathChirp changes and these now have the highest estimation errors.

Multiple Bottleneck—Two Potential Tight Links

Most ABET designers implicitly (and often, explicitly) assume the existence of only a *single* congested (bottleneck) link on the concerned path. It is conjectured that current ABETs might underestimate end-to-end AB in the presence of multiple bottleneck links [JD02a]. In order to study this scenario, we simulate the topology of Fig 5.7. We replay the IPLS-CLEV trace between N2 and N3, and the IPLS-KSCY trace between nodes N4 and N5. The ABETs are run between nodes N0 and N1. With this setup, the tools encounter one narrow link (R2-R3), and two potential tight links (R1-R2 and R3-R4)—on an average, the latter is the "tighter" link;

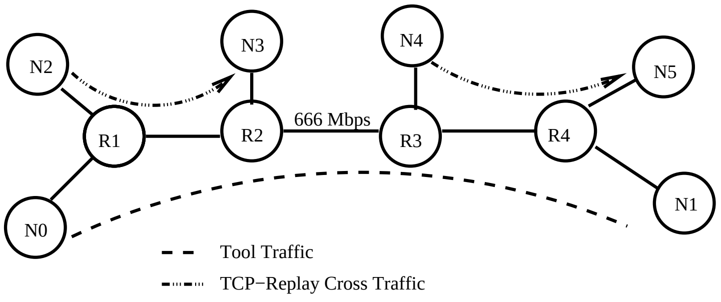


Figure 5.7: Single narrow link; two tight links

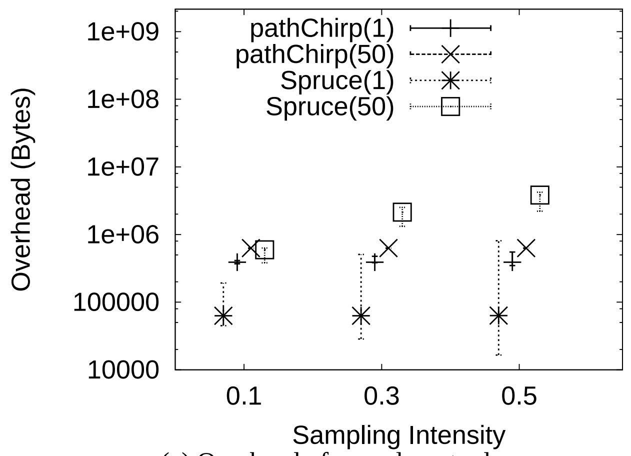
however, the tool traffic experiences queuing at both links.

We compute the actual end-to-end AB as the *minimum* of the AB on links R1-R2 and R3-R4. We run each ABET several times with MT of 50 ms and SI of 0.1. Fig 5.6 plots the average and the 5- and 95-percentiles of the AB estimation errors of different tools. On comparison to the other plots on the same figure, we observe that the accuracy of Pathload, R-IGI, and Fast-IGI is not significantly impacted by the presence of multiple tight links. However, the accuracy of PathChirp and Spruce further degrades and these are the most inaccurate.

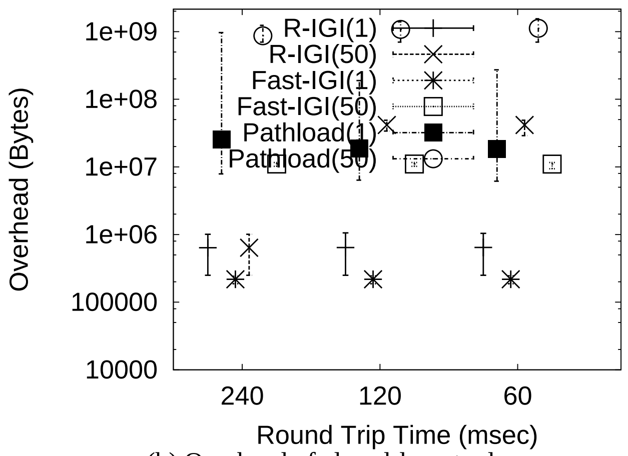
5.5 Evaluating the Costs of ABETs

Overhead Fig 5.8 (a), (b) plot the average and the 5- and 95-percentiles of the overhead for each tool at different SI and with MT of 1 and 50 ms. We observe that:

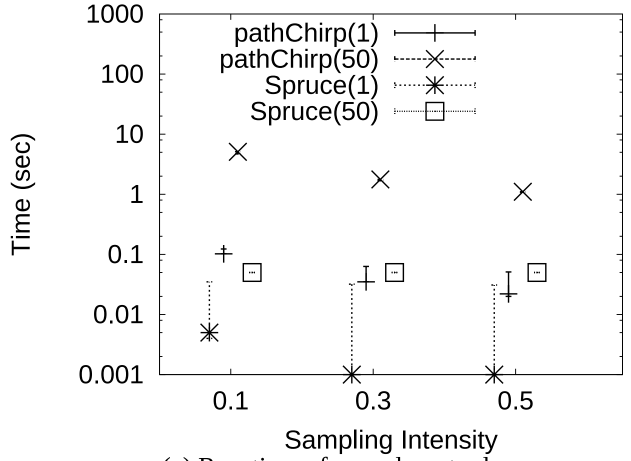
- For any given MT, PathChirp, R-IGI, and Fast-IGI have the least overhead. The overhead of each run of Pathload is larger by more than an order of magnitude and can be as high as a giga-byte.
- Tool overhead increases with MT. While the increase is linear for most tools, it is not for PathChirp. This is because PathChirp uses an exponentially-spaced packet stream—increasing the stream duration, therefore, increases the number of packets only sub-linearly. Consequently, while the overhead of Pathload increases from 50 MB to 2.5 GB as the MT increases from 1 to 50 ms, the overhead of PathChirp increases from 0.5 MB to only 0.75 MB.
- SI and RTT have no impact on the overhead of most tools. The overhead is dictated by the size and number of probe streams sent—the same number of probe-streams are needed to arrive at an AB estimate, irrespective of these quantities. For Spruce, however, the overhead increases with SI—this is an artifact of



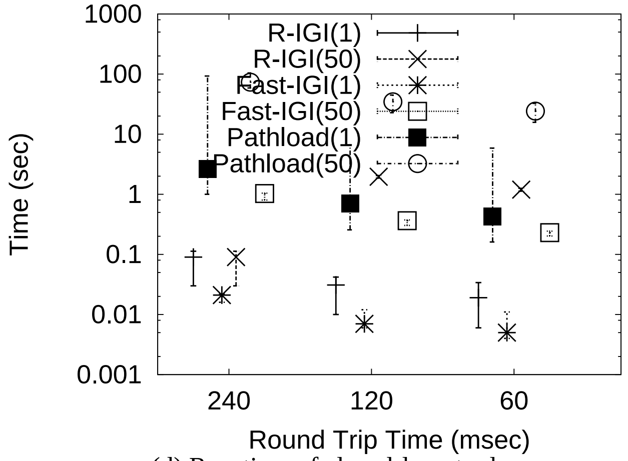
(a) Overhead of open-loop tools



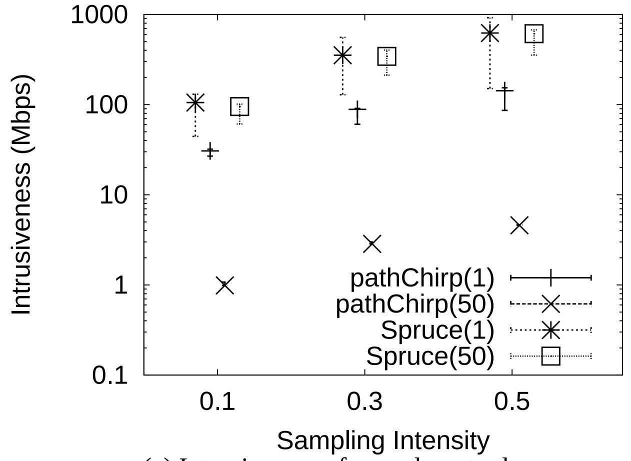
(b) Overhead of closed-loop tools



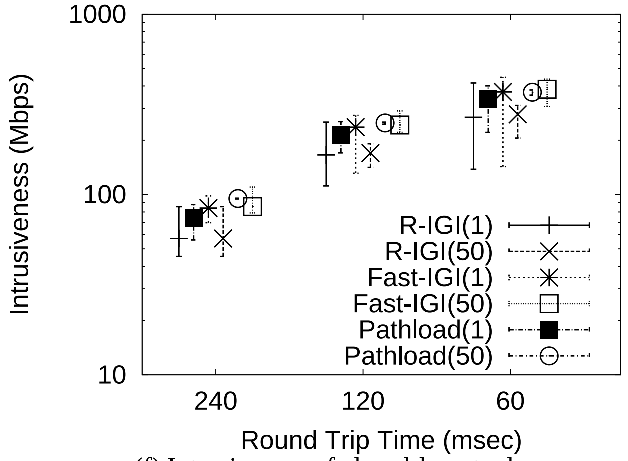
(c) Run-time of open-loop tools



(d) Run-time of closed-loop tools



(e) Intrusiveness of open-loop tools



(f) Intrusiveness of closed-loop tools

Figure 5.8: Costs of ABETs with the Ibiblio trace (numbers in parenthesis indicate the MT in ms)

the fact that we are using a large number of tool runs in order to incorporate SI into its AB estimates.

Run-time Fig 5.8 (c), (d) plot the average and the 5- and 95-percentiles of the run-time for each tool at different SI and with MT of 1 and 50 ms. We observe that:

- Spruce is the fastest tool; this is true in spite of the fact that we aggregate several tool-runs in order to get an estimate at the desired MT.

  Pathload is the slowest tool, taking 10-100 seconds to return an AB estimate. R-IGI takes 1-10 seconds and PathChirp takes a few seconds (with its typically configured MT). Fast-IGI is roughly 5 times faster than R-IGI.

- Increasing the MT results in a proportional increase in the run-time of all tools. For Spruce, however, this is an artifact of the way we are incorporating MT into the AB estimates yielded by it.

- The run-time of closed-loop tools is proportional to the path RTT, which characterizes the feedback delay of these tools.

- As SI increases, the run-time for PathChirp decreases. This is because open-loop tools such as PathChirp rely on sending and observing a fixed number of probe streams—larger is the SI, faster would these streams be sent.

  SI has no impact on the run-time for Spruce—this is, however, an artifact of the way we incorporate MT into the Spruce estimates.

- The run-times of most tools are predictable—they do not vary significantly around the average.

Intrusiveness Fig 5.8 (e), (f) plot the average and the 5- and 95-percentiles of the intrusiveness for each tool at different SI and with MT of 1 and 50 ms. Recall that intrusiveness is given by the ratio of the overhead to run-time of a tool. We observe that:

- All closed-loop tools are quite intrusive and can temporarily congested high-speed links. The run-times suggest that tools like Pathload can induce such congestion for several seconds.

  Spruce is also quite intrusive—it sends back-to-back probe packets at the line rate. However, since its run-time is small, it is unlikely to induce congestion for long durations (unless it is run several times). PathChirp is the most non-intrusive tool.

- The closed-loop tools—Pathload, R-IGI, and Fast-IGI—have very similar intrusiveness. This may seem surprising given that both the run-times and overheads of these tools vary by nearly two orders of magnitude. However, it is important to note that all of these tools rely on a feedback loop and iteratively search for the AB—consequently, these operate at time-units that are RTT long. Both R-IGI and Pathload use the concept of self-loading streams and, consequently, their per-RTT overhead (which is the intrusiveness) is similar.

- As MT increases, the intrusiveness of closed-loop tools increases proportionally. This is to be expected; the per-RTT overhead of these tools is given by the size of each probe-stream, which is proportional to the MT.

  Increasing MT decreases the intrusiveness of PathChirp. As mentioned before, while the run-time of PathChirp increases linearly with MT , its overhead increases only sub-linearly due to the exponential nature of the probe stream. The intrusiveness, consequently, decreases.

  MT has no impact on Spruce—however, this is also an artifact of the way we incorporate MT into its AB estimates.

- Increasing SI increases the intrusiveness of open-loop tools. This is to be expected, as a larger number of probe streams are sent per unit time as a result of increasing SI.

- Intrusiveness of closed-loop tools increases as the RTT decreases. This is to be expected as the per-RTT overhead remains the same.

We conclude that in terms of the cost metrics, the tool that is likely to run quickly, while not perturbing ongoing traffic much seems to be PathChirp; the cost of Pathload, R-IGI, and Fast-IGI seems to be the highest.

5.5.1 Impact on Responsive Cross-Traffic

TCP is the dominant transport protocol used by most Internet applications [ea07]. TCP uses congestion-control mechanisms to reduce the data sending rate on detecting network congestion. A key issue in the wide-scale deployment of ABETs is that of how adversely do these tools impact the performance of applications that rely on such *responsive* transport protocols. In this section, we study this issue.

Unfortunately, the trace replay methodology used in Sections 5.4 and 5.5 is not suitable for studying this issue—it recreates only the link-level packet-arrival process and does not incorporate TCP behavior. In particular, it does not model the impact of queuing delays and losses on the subsequent packet transmission behavior of a TCP connection. Recent efforts have focused on developing traffic-generation tools that also incorporate the responsive behavior of TCP—Tmix [HCSJ04] is one such tool. It takes as input a link-level packet trace (such as those summarized in Table 5.1), and for each TCP connections that appears in the trace, it derives the RTT and the application-level data generation behavior (including user think times). Recently, an NS-2 version of Tmix has been developed [WAHC$^+$06], which takes this derived connection descriptor as input and emulates per-connection application bots with similar RTTs and data-generation behaviors.

We use this version of Tmix in the topology of Fig 5.1, in which the nodes N2 and N3 now each emulate a cloud of servers and clients that instantiate connections between these two nodes. Different per-connection RTTs are simulated using the delay-box environment [del] of NS-2 and the router buffer sizes are limited to 100 MSS-sized packets to help emulate packet losses (even without the ABETs). The connections to be simulated are derived from a real Internet access link and have an average traffic load of 300 Mbps.

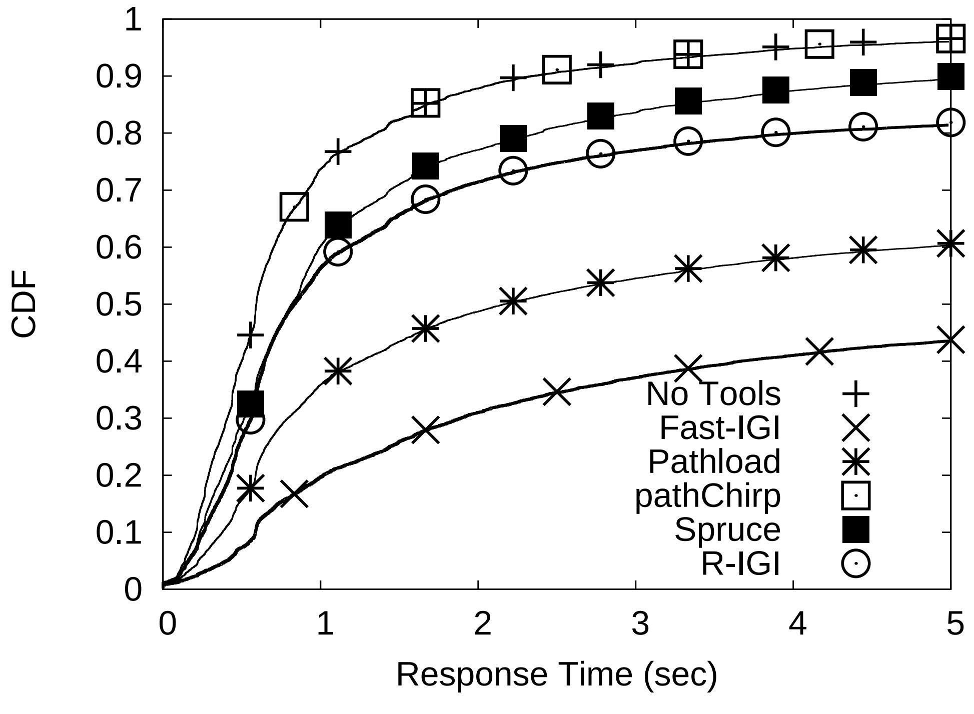


Figure 5.9: CDF of response times with default parameters

Tmix measures and reports the per-connection response time—the time taken for the connection to transfer all data between the two end-points. In order to assess the impact of ABETs on the simulated TCP connections, we run the tools continuously between nodes N0 and N1 and observe the impact on the distribution of connection response times. Each tool has an SI of 0.1, an MT of 50ms, and an RTT of 240 ms. Fig 5.9 plots this distribution for experiments conducted with each tool and without any tools. We find that:

- PathChirp has no noticeable impact on connection response times. As seen in Section 5.5, PathChirp has a fairly low intrusiveness—it does not cause much queue build-up on the bottleneck link.
- All of the other tools can significantly impact the response times of TCP connections. Of these, Spruce and R-IGI increase the connection response times for long connections by a factor of 2-3. For instance, the 75-percentile response time without any tool is a little over 1 s, while with Spruce and R-IGI , the 75-percentile response times are 1.7 s and 2.5 s, respectively.

  Note that while Spruce had one of the highest measures of intrusiveness, it does not fare among the worst in impacting connection response times. Its packet steams are too small (two packets) to sustain congestion long enough to inflict packet losses.
- Pathload and Fast-IGI can significantly impact the response times of all connections. While 75% of connections have a response time less than 1 s in the absence of any tool, nearly 65% and 80% of connections have a response time larger than 1 s in the presence of Pathload and Fast-IGI, respectively.

We conclude that if an application needs to run an ABET repeatedly on a given Internet path, it should use PathChirp. Such an application should never use Pathload or Fast-IGI as these are likely to significantly impact connection response times.

5.6 Notes on related work

5.6.1 Spruce and packet-pair techniques

Dovrolis et. al. [LDS06] show that Spruce has the tendency to underestimate end-to-end AB when the traffic is non-path persistent. However, when the traffic is persistent, the bias or error in Spruce is negligible. A similar analytical result has also been established in the paper by X. Liu [LRL05], which concludes that the bias inherent in packet-pair methods tends to under-estimate the end-to-end AB in multi-hop scenarios with one-hop persistent cross-traffic. They find that the overall under estimation can be quantified as the sum of two bias terms. The first bias term tends to zero as the number of packets in the packet-train used by spruce grows very large. The second bias term however never reduces to zero—this suggests that Spruce will always underestimate the AB on a multi-hop path with non path-persistent traffic. From our evaluations, we also find that Spruce performs well when the traffic is path persistent (as in the single bottleneck scenario). In the multiple bottleneck scenario, however, the traffic we instrument is one-hop persistent and we observe that the performance of spruce degrades quite significantly. Thus our evaluation corroborates past results that have been obtained on the performance of Spruce.

5.6.2 Pathload and rate based techniques

Liebeherr et. al. [LFV07a] model the network as a min-plus linear system and use this model to study the performance characteristics of rate-based techniques like Pathload and IGI. They observe that the rate limit that is used in rate-scanning based techniques significantly impacts the accuracy of the tools. They suggest that the rate limit should be at least greater than the current AB. Most rate-based AB estimation tools—including Pathload and IGI—already adopt this suggestion by estimating the end-to-end link capacity and using the capacity measure as the rate limit.

Liu et. al. [LRL05] find that rate based techniques also tend to underestimate the AB. Furthermore, they find that in cases where there are multiple tight links on the path, rate-based techniques also underestimate the AB. They also find that using longer packet trains will reduce this bias. From our evaluations, we find that using a larger MT (which is equivalent to increasing the length of the packet-train) does improve the performance of Pathload. We also find that the performance of Pathload degrades as we go from the simple single-bottleneck scenario into more complex ones. The tradeoff is that the use of longer packet-trains comes at the expense of an increasing tool overhead, which will limit the length to which we can increase the length of the packet-train.

5.6.3 Pathchirp and rate chirps

Liebeherr et. al. [LFV07a] using their min-plus model for AB estimation also found that decreasing the spread factor—for instance, by sending more packets over a longer time—could help improve accuracy. In our experiments we varied the MT by controlling the spread factor and the number of packets per-chirp, and hence it is not possible to extrapolate the results directly. However, we found that there is a slight improvement in the average performance of Pathchirp when the MT is increased—that is, when more packets are sent over a longer interval of time.

5.7 Conclusion

After evaluating the different AB estimation methodologies we make the following observations.

- **Regarding Accuracy:** (i) We observe that different CBR loads does not cause the accuracy of the different ABET technique to change. Only Fast-IGI shows a decrease in accuracy at loads greater than 600 Mbps. (ii) The accuracy of different AB methodologies across all the traces is comparable, even though the traces represents different average loads. (ii) The errors are higher with the trace-traffic than with CBR traffic. (iv)Increasing the MT improves the accuracy of all the ABETs. (v) Beyond a MT of 50 ms the gains in accuracy are negligible. (vi) Increasing the SI however does *not* have an impact on the accuracy. This is because of the inherent error that is present in every sample of AB inference. (vii) Similar to SI the RTT has no impact on the accuracy of an ABET.

- **Regarding Overhead:** (i) Pathchirp, Fast-IGI and IGI have the least overhead and Pathload has the most overhead (ii) As the MT increases the overhead increases except for Pathchirp where the overhead does not increase as a linear function of the MT. (iii) As the MT increases, the run time increases. (iv) As the SI increases or the RTT decreases, the run-time decreases. (v) Pathload takes between 30-100 seconds to run at MT of 50 ms. (vi) Spruce is the fastest tool since it could produce an AB estimate with every packet-pair it sends out. (vii) Increasing the MT increases the intrusiveness, except for Pathchirp in which case the intrusiveness reduces with larger MT. (vii) Increasing the SI also increases the intrusiveness. (ix) Pathchirp shows the lowest intrusiveness. (x) Spruce and other closed loop tools show similar intrusiveness characteristics.

CHAPTER 6
Scalable Monitoring of the AB

With an understanding of the issues that impact the performance of the various AB techniques and tools, we now visit the problem of designing a scalable AB inference scheme for multi-path monitoring applications. We first use the knowledge of the performance characteristics of ABETs from the previous studies to pick and calibrate our AB estimation methodology. We then describe the design of our scalable AB estimation scheme, along with the assumptions it is based on. Finally we present validation results of our scheme, that we obtained by evaluating our scheme on PlanetLab [Pla].

6.1 Selection of ABET estimation methodology

As argued in Chapter 1, large-scale multi-path AB monitoring requires that the per-path ABET used should: (i) impose low overhead on the path, (ii) have a fast response time, and (iii) should yield accurate estimates.

Some of these goals conflict with each other; for instance in the current spectrum of design choices, tools that are accurate are either overly intrusive (IGI, Figure 5.8(f)) or have long RT (Pathload, Figure 5.8(d)). In order to pick a tool and configure its parameters we utilize the insights that we have obtained from our previous evaluation studies (Chapters 3, 4 and 5). Three tools that meet the requirements of being able to report the AB fast and accurately are Fast-IGI, Spruce and Pathchirp. Since IGI and Pathload rely on a convergence logic and thus have a non-deterministic run-time they would not be suitable for this application (Figure 5.8(d)). Though Fast-IGI would be a much better choice in terms of accuracy, it exhibits high intrusiveness (Figure 5.8(f)). Furthermore, as was observed in Figure 5.8, at high loads Fast-IGI tends to overestimate the AB. Between Spruce and Pathchirp, Pathchirp has a lower intrusiveness (Figure 5.8(e)) and also has a short run-time, which makes it a good candidate for a large-scale monitoring application.

Next, we use insights from previous studies to set the Pathchirp parameters so that we can maximize the accuracy, while minimizing the intrusiveness and time to report a measurement. From Figure 4.6, we expect that a SI of at least 30% would give us good accuracy. Due to the exponential design of its probe-streams, Pathchirp already adopts a high sampling intensity (due to the low-intrusiveness of its probe-streams, it can afford to send successive probe-streams within a short gap). Another way to improve the accuracy would be to increase the MT (Figure 5.4(a)) to 50 ms. Since increasing the MT increases the RT of the tool (Figure 5.8(c)) we can offset this increase in RT by increasing the SI (Figure 5.8). While increasing the accuracy, increasing the MT also reduces the intrusiveness of Pathchirp (Figure 5.8(e)) and reduces the variability of the AB estimates made (Figure 4.11). Finally, we configure Pathchirp to run with 8192 byte packet size, which would reduce requirements on timer granularity and reduce the potential for interrupt coalescence effects (Figure 3.4).

6.2 Design of the AB Monitoring Scheme

Recently a network monitoring architecture—referred to as Broute—has been proposed in [HS05] that focuses instead on explicitly measuring $AB^{out}_{S_1}$ and $AB^{in}_{D_1}$, and uses Equation (1.3) to infer $AB(S_1, D_1)$. Unfortunately, while several tools have been designed and evaluated for accurately measuring *end-to-end* AB [JD02b, Rib03, SKK03a, SK07, SMH$^+$05], currently available tools for measuring AB^{out} or AB^{in} are fairly inaccurate. Indeed, in this chapter we show that use of the Broute approach can lead to AB inferences that are inaccurate by more than 50%.

We next address the question: *How can a monitoring service estimate in a scalable manner, the AB on the N^2 paths of an overlay of size N using only an end-to-end ABET?*

6.2.1 Approach

In this chapter, we eliminate the dependency on per-hop AB estimation tools and instead design a scalable AB monitoring architecture that relies only on tools that measure end-to-end AB. We refer to our architecture as *SABI (Scalable Available Bandwidth Inference)*. SABI exploits the existence of end nodes (or overlay nodes) that are connected by well-provisioned access segments. As shown below, such nodes can be used for inferring AB between other node pairs in a scalable manner. We present three different approaches—that differ in their accuracy and overhead—for doing so.

SABI Algo 1: Consider the additional node S_2 in Fig 2.2. Our previous observations on access-segments imply that:

$$AB(S_2, D_1) \quad \simeq \quad \min\{AB^{out}_{S_2}, AB^{in}_{D_1}\}$$

$$AB(S_1, S_2) \simeq \min\{AB^{out}_{S_1}, AB^{in}_{S_2}\}$$

Furthermore,

$$\begin{aligned}
&\min\{AB(S_1, S_2), AB(S_2, D_1)\} \\
&\simeq \min\{AB^{out}_{S_1}, AB^{in}_{S_2}, AB^{out}_{S_2}, AB^{in}_{D_1}\} \\
&= \min\{AB^{out}_{S_1}, AB^{in}_{D_1}\}, \\
&\qquad\qquad\qquad \text{if } AB^{in}_{S_2}, AB^{out}_{S_2} \geq AB^{out}_{S_1}, AB^{in}_{D_1} \\
&\simeq AB(S_1, D_1) \qquad (6.1)
\end{aligned}$$

Thus, if the access segment of S_2 consists of well-provisioned high-capacity links, then $AB(S_1, D_1)$ can be inferred as the minimum of $AB(S_1, S_2)$ and $AB(S_2, D_1)$.

Our first approach for scalable AB inference, therefore, assigns a well-provisioned *head-node* (such as S_2) to each potential source node (such as S_1) in the overlay. Head-nodes measure the AB on paths to all potential destination nodes (such as D_1). Source nodes only measure the AB on the path to their head-node. The end-to-end AB between any given source and destination is then inferred as the minimum of these two quantities. Thus, in an overlay of size N, if K head-nodes are assigned, the total number of AB measurements that need to be made in order to infer all N^2 AB values is given by: $O(K * N)$.

SABI Algo 2: In order to further reduce the number of AB measurements, consider the node D_2 in Fig 2.2. Then,

$$\begin{aligned}
AB(D_2, D_1) &\simeq \min\{AB^{out}_{D_2}, AB^{in}_{D_1}\} \\
AB(S_2, D_2) &\simeq \min\{AB^{out}_{S_2}, AB^{in}_{D_2}\}
\end{aligned}$$

Furthermore,

$$\begin{aligned}
&\min\{AB(S_1, S_2), AB(S_2, D_2), AB(D_2, D_1)\} \\
&\simeq \min\{AB^{out}_{S_1}, AB^{in}_{S_2}, AB^{out}_{S_2}, AB^{in}_{D_2}, AB^{out}_{D_2}, AB^{in}_{D_1}\} \\
&= \min\{AB^{out}_{S_1}, AB^{in}_{D_1}\},
\end{aligned}$$

$$\begin{aligned} & \text{if } AB^{in}_{S_2}, AB^{out}_{S_2}, AB^{in}_{D_2}, AB^{out}_{D_2} \geq AB^{out}_{S_1}, AB^{in}_{D_1} \\ \simeq\ & AB(S_1, D_1) \end{aligned} \qquad (6.2)$$

Thus, if the access segments of both S_2 and D_2 consist of well-provisioned high-capacity links, then $AB(S_1, D_1)$ can be inferred as the minimum of $AB(S_1, S_2)$, $AB(S_2, D_2)$, and $AB(D_2, D_1)$.

Our second approach for scalable AB inference, therefore, assigns a well-provisioned *head-nodes* (such as S_2 and D_2) to each potential node (such as S_1 and D_1) in the overlay. All head-nodes measure the AB on the paths to each other. Additionally, each node measures the AB on the path to/from its corresponding head-node. The end-to-end AB between any given source and destination is then inferred as the minimum of the three quantities mentioned above. Thus, in an overlay of size N, if K head-nodes are assigned, the total number of AB measurements that need to be made in order to infer all N^2 AB values is given by: $O(N + K^2)$.

SABI Algo 3: Observe that if both S_2 and D_2 have well-provisioned access segments, then it is likely that $AB(S_2, D_2)$ will be larger than $AB(S_1, S_2)$ and $AB(D_2, D_1)$. Thus, Equation (6.2) can be re-written as:

$$\min\{AB(S_1, S_2), AB(D_2, D_1)\} \quad \simeq \quad AB(S_1, D_1), \qquad (6.3)$$

if $AB^{in}_{S_2}, AB^{out}_{S_2}, AB^{in}_{D_2}, AB^{out}_{D_2} \geq AB^{out}_{S_1}, AB^{in}_{D_1}$. Thus, $AB(S_1, D_1)$ can be inferred as the minimum of $AB(S_1, S_2)$ and $AB(D_2, D_1)$.

Based on this idea, our third approach for scalable AB inference, just like the second approach above, assigns a well-provisioned *head-node* (such as S_2 and D_2) to each potential node (such as S_1 and D_1) in the overlay.

Also, like the second approach, each node measures the AB on the path to/from its corresponding head-node. The difference is that none of the head-nodes measure the AB on paths to each other. The end-to-end AB between any given source and destination is then inferred as the simply minimum of the two quantities mentioned above. Thus, in an overlay of size N, if K head-nodes are assigned, the total number of AB measurements that need to be made in order to infer all N^2 AB values is given by: $O(N)$.

Note that this third approach is similar in spirit to that of Broute [HS05]—the difference is that Broute relies on tools that explicitly measure the AB on access segments of S_1 and D_1, whereas our approach implicitly measures it by measuring the end-to-end AB between these nodes and their corresponding head-nodes. Tools for measuring end-to-end AB are significantly more accurate.

Need for Path Similarity Note that each of the above three algorithms rely on the assumption that the access segments are shared among the paths involved—for instance, the paths from S_1 to D_1 and from S_2 to D_1 should share the same incoming access segment of D_1. This assumption may not hold if any of the nodes S_1, S_2, or D_1 (or their local ISPs) are multi-homed.[1] Since multi-homing is on the rise in the Internet [AMS$^+$03], the SABI inferences can be applied in practice only if the routes from overlay nodes and their head-nodes to most destinations indeed share their last few hops. In order to ensure this, we adopt the following general approach for monitoring the all-pairs AB of an overlay in a scalable manner:

1. form groups of nodes that share similar underlying IP paths to other nodes in the overlay;
2. within each group, select the node with the highest-capacity access segment as the "head" node of the group;
3. measure AB from all nodes within a group to the head-node;
4. for Algo 2, additionally measure AB between all head-nodes in the overlay;
5. for Algo 3, additionally measure the AB from each head-node to all nodes outside the group; and
6. infer AB between any two nodes i and j, using one of Equations (6.1), (6.2), and (6.3).

6.2.2 Path-based Clustering

In order to group together nodes with similar routes to other destinations, we first define a *path difference* (PD) metric for any pair of overlay nodes. Let $H^m_{i,k}$ be an ordered vector representing the IP addresses of the last m IP hops that appear on the path from node i to node k. Then,

$$PD^m(i,j) \quad = \quad \frac{1}{N-2} \sum_{k \in \mathcal{S}-\{i,j\}} \mathcal{D}(H^m_{i,k}, H^m_{j,k})$$

where $\mathcal{S}$ is the set of all overlay nodes, and the function $\mathcal{D}$ returns the number of positions at which the values of the specified vectors differ—for instance, if $X = [1.2.3.4, 5.6.7.8, 9.10.11.12]$, and $Y = [13.14.15.16, 1.2.3.4, 9.10.11.12]$,

[1]A node or network is said to be multi-homed if there are more than one ISPs or access links connecting it to the rest of the Internet.

then $\mathcal{D}(X,Y) = 2$. Intuitively, the smaller the value of the $PD^m(i,j)$ metric, the more similar are the paths from nodes i and j to the other nodes in the overlay. We use this metric as the "distance" between nodes i and j—note that both $\mathcal{D}$ and PD satisfy the triangular inequality. A standard clustering algorithm (such as the K-means [kme]) can then be used to group the overlay nodes into clusters of "similar" nodes, as needed in step 1 of our approach.

Note that measuring the paths for N^2 node-pairs constitutes lower overhead than the corresponding AB measurements. Further, these measurements could be done at much larger intervals to form the node groupings, as routes are not expected to vary as rapidly as AB measurements.

6.2.3 Head Selection

Let $C_{i,k}$ be the end-to-end capacity from node i to k. Define $C_{j,j} = \max_{k \in \mathcal{S}-\{j\}} C_{j,k}$. For each cluster formed above, we select as the head-node the node i that has the highest value of: $C_i^{max} = \max_j \{C_{i,j}\}$.[2] This would ensure that the node with the highest access-capacity gets selected as the head node. Further, this also ensures that the head node is well-provisioned to be able to perform AB monitoring to all other nodes, on behalf of its cluster-members.

In Appendix A, we show that if $\sqrt{N}$ uniformly-sized clusters are used for an overlay of size N, then the SABI Algorithm 1 reduces the number of AB measurements of an all-pairs monitoring service by a factor of $\sqrt{N}$. Non-uniform clusters, however, achieve less reduction in overhead. It is also easy to see that with uniform clustering, SABI Algorithms 2 and 3 reduce the number of AB measurements of an all-pairs monitoring service by a factor of N.

Both the overhead and accuracy of our approaches is impacted by two parameters: the number of clusters, K, and the number of hops, m, considered for path-based clustering. We next empirically study this impact and evaluate our approach through wide-area Internet experiments. We next present a large scale performance evaluation of the SABI Algo 1. We also present a smaller scale evaluation of all three SABI algorithms and compare them to the Broute algorithm in the same scenario.

[2]Note that our definitions also imply that $C_i^{max} = C_{i,i}$.

6.3 Experimental Methodology

In this section, we evaluate our approach in an Internet-wide experimental setting. Below, we describe our evaluation methodology.

The Experimental Environment

We rely on nodes from the PlanetLab testbed [Pla] for our experimental evaluation. This testbed gives us access to a large and diverse set of geographically distributed nodes that can be used to form an experimental overlay. Use of the PlanetLab testbed, however, poses additional challenges. First, the PlanetLab operating environment poses software rate limits on the amount of traffic sent by any application—this is likely to adversely impact the accuracy of some capacity and AB estimation tools like Pathload [abt]. For our evaluation, we instead rely on PathChirp [Rib03], an available bandwidth estimation tool that does not send large probe streams at high rates and, consequently, does not get impacted by the PlanetLab rate-limiting.

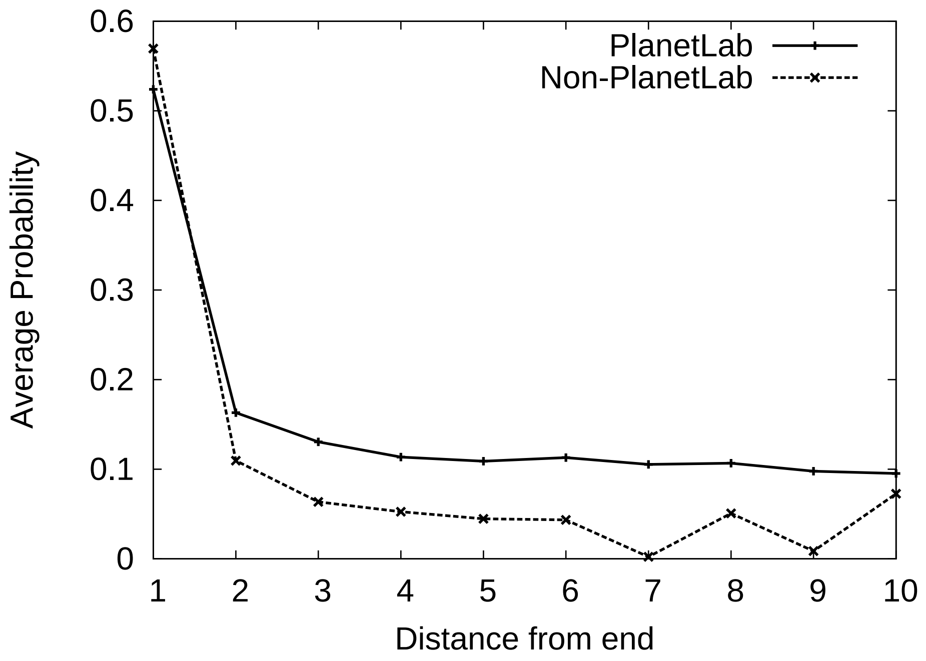


Figure 6.1: Location of Bottleneck Links

The rate-limiting may also imply that the bandwidth bottleneck is the first hop on each PlanetLab path, which may not be true in the general Internet. In order to assess the impact of the PlanetLab operating environment on the location of the bottleneck link, we select 50 PlanetLab nodes and estimate the location of the bottleneck link on all paths between these nodes using *Stab* [RRB04]. For each link on a path, Stab returns a probability that the link is a bottleneck link—we run Stab several times on each path and for each link, compute the average probability of it being a bottleneck link. Fig 6.1 plots this average probability as a function of the distance of the link from the closest end-point of the path—for instance, both the first and last hop appear at a distance of 1 from the end-point. We observe that the bottleneck is likely to lie within 1-2 hops from either end of a path.

For comparison, we also run Stab on all paths between seven non-PlanetLab nodes located across 4 universities and the commercial Internet. Fig 6.1 also plots the per-hop bottleneck probability for these paths—we find that the bottleneck locations are fairly similar to the PlanetLab environment. This suggests that the use of PlanetLab does not unrealistically bias our experimental environment.

Recall that our approach needs several types of measurements from each overlay path: the end-to-end available bandwidth measurements, per-hop IP route information for clustering, and end-to-end capacity information for choosing cluster heads. In order to obtain these measures, we leverage the Scalable Sensing Service infrastructure (S^3) [YSB$^+$06]. The S^3 infrastructure collects measurements of the end-to-end capacity, AB, loss rates, and traceroute data between all pairs of nodes in the PlanetLab testbed network (http://www.planet-lab.org) and presents snapshots of the data at the following website: http://networking.hpl.hyperplane.com/s-cube.

Selecting the Overlay Nodes

At the time of our evaluation, PlanetLab had grown to 711 nodes spread across 338 sites. Unfortunately, in an uncontrolled environment such as PlanetLab, it is difficult to consistently obtain all-pairs data on the end-to-end capacity and IP routes. In particular, due to several issues such as nodes being down, high-CPU loads on certain nodes, bandwidth limits, and tool errors, we do not have complete information for all pairs of PlanetLab end-nodes. Since our objective is to evaluate the SABI architecture, it is important for us to know the ground-truth about the overlay network on which we evaluate it. Our first challenge, consequently, is to find the largest PlanetLab overlay for which S^3 does provide all of the required evaluation measures for all node-pairs.

Note that both capacity and route information are relatively static quantities. We first systematically search the S^3 measurement logs to extract all capacity estimates made over the past two months. We then model our overlay-selection problem by constructing a graph G with 711 vertices, one corresponding to each PlanetLab node. We add an edge between a node pair in G if we find a successful capacity estimate in the S^3 logs for the path between the corresponding PlanetLab nodes. Our maximal-overlay-selection problem is then analogous to finding the largest clique in this graph. Using a heuristic algorithm, we find a large clique that consists of 144 nodes.[3] After eliminating nodes belonging to the same PlanetLab member-site, we were left with a set of 95 distinct sites.

For all paths between each pair of these 95 nodes, we then use web requests to S^3 to run traceroute and obtain the current IP route information. Paths involving 12 of these nodes did not return route information either because the nodes were down or the system was unable to perform the traceroute operation. For evaluating SABI, consequently, we rely only on the remaining 83 nodes—Appendix B lists the corresponding PlanetLab sites

Clustering

We use the IP route information to compute the similarity metric PD^m for each node-pair in our selected overlay. We then populate a matrix of distances between all node-pairs using the similarity metric. This matrix is then fed to the K-means clustering implementation in Matlab [Inc92]. We experiment with different values of K, the number of clusters generated. Within each cluster, the head-node is selected as the node with the largest value of C_i^{max}.

As mentioned earlier, our AB inference approach results in the maximal reduction in probing overhead when the node-clusters are uniform in size; each of the K clusters would have a size of N/K in this case. In practice, though, the geographical distribution of the overlay nodes may be skewed—in this case, the sizes of the resultant clusters are also likely to be skewed. To reduce the probing overhead, we randomly partition clusters of size larger than N/K into two or more sub-clusters, such that none of the resultant sub-clusters is larger than N/K. Each sub-cluster behaves as an independent cluster and selects its own head-node. This approach not only helps reduce the probing overhead, but also distributes the probing load uniformly across different head-nodes.

Measurements and metrics

Let H_i denote the head node of the cluster to which node i belongs. We use our overlay testbed and S^3 to evaluate our approach using the following algorithm:

```
For each node $i$ that is not a head-node
    For each node $j$ that lies outside the cluster containing $i$
        Measure $AB(i, H_i)$.
```

[3]The fact that a regular monitoring service such as S^3 could provide a capacity-clique for only around 25% of the PlanetLab nodes further highlights the challenge of wide-area experimentation on uncontrolled environments such as PlanetLab.

Measure $AB(H_i, j)$.

Measure $AB(H_i, H_j)$.

Measure $AB(H_j, j)$.

Measure $AB(i, j)$.

(SABI Algo 1) Estimate:

$AB^{inf1}(i, j) = \min\{AB(i, H_i), AB(H_i, j)\}$.

(SABI Algo 2) Estimate:

$AB^{inf2}(i, j) = \min\{AB(i, H_i), AB(H_i, H_j), AB(H_j, j)\}$.

(SABI Algo 3) Estimate:

$AB^{inf3}(i, j) = \min\{AB(i, H_i), AB(H_j, j)\}$.

This procedure helps us obtain nearly-simultaneously the values of $AB(i, j)$, the actual AB between nodes i and j, as well as $AB^{inf}(i, j)$, the AB inferred using SABI's algorithms. We then use the following two metrics to characterize the performance of our approach:

- *Accuracy:* We compute the accuracy of our inferences using two quantities: (i) the absolute inference error: $Ae(i, j) = \left|AB^{inf}(i, j) - AB(i, j)\right|$, and (ii) the relative inference error: $Re(i, j) = \frac{\left|AB^{inf}(i,j) - AB(i,j)\right|}{AB(i,j)}$.

  It is important to mention that experimental evaluations of existing ABETs have shown that most of these tools, including PathChirp, have typical AB estimation error of around 10% of the actual AB [SMH$^+$05]. Thus, inferences errors of within 10 Mbps would lie within the resolution of existing ABETs—such errors would be present even in an all-pairs AB monitoring infrastructure.

- *Overhead:* We compute the probing overhead as the total number of actual AB measurements that would need to be made in order to obtain a snapshot of AB for the complete overlay. Given $\mathcal{S}_\mathcal{C}$ is the set of clusters, and n_k is the number of nodes in cluster k, the probing overhead for different SABI algorithms will be:

  SABI Algo 1 : $\sum_{k \in \mathcal{S}_\mathcal{C}} (n_k * (n_k - 1) + (N - n_k))$

  SABI Algo 2 : $\sum_{k \in \mathcal{S}_\mathcal{C}} n_k * (n_k - 1) + (N - |S_C|)$

  SABI Algo 3 : $\sum_{k \in \mathcal{S}_\mathcal{C}} n_k * (n_k - 1)$

  The first term corresponds to the all-pairs measurements performed within a cluster, while the second term corresponds to the measurements made in order to infer the AB to nodes outside the cluster, which is not needed in the case of the third algorithm.

Parameters

Two main parameters can affect the performance of our AB inference approach:

K**, the number of clusters:** As formulated above, the number and size of clusters impact the probing overhead of our approach. K also impact the AB inference accuracy—the larger is the number of clusters, the more likely is it that all nodes in a cluster are actually similar in their PD metric. We study both the accuracy and probing overhead of our approach using cluster-sizes of 7, 9, 10, and 12. Note that with N of 83, $\sqrt{N}$ is around 9, which is optimal for reducing the probing overhead in SABI Algo 1 (as shown in Appendix A). Our choices for K evaluate values both less than and greater than this value.

m**, the path matching parameter:** In addition to K, path-based clustering is characterized by m, the number of hops on the path to a destination that are compared to quantify node similarity—we refer to this as the *path matching parameter*. Recent Internet measurements [Li05] suggest that most bottleneck links lie within 4-5 hops from the end-nodes. On the other hand, larger the value of m, the less likely is it that even “similar” nodes would share all m hops to a destination. We study the sensitivity of our results on the parameter m by using $m = 3, 5, 7$.

6.4 Evaluation Results

Our detailed evaluation mainly focuses on the SABI Algorithm 1. At the end of this section, we also provide a preliminary smaller scale comparative evaluation of all three SABI algorithms and also compare it to the Broute algorithm [HS05].

6.4.1 Path Based Clustering

We run K-means to generate $K = 7, 9, 10, 12$ clusters. We also evaluate three different settings of $m = 3, 5, 7$. S^3 could return all three measurements needed in the evaluation algorithm for only up to 44% of the node-pairs involved, despite repeated measurement attempts.

Overhead Fig 6.2 plots as a function of K, the probe overhead—the number of probes that would be required for obtaining a single snapshot of all-pairs AB—for the clusters formed using the PD metric with $m = 5$. For

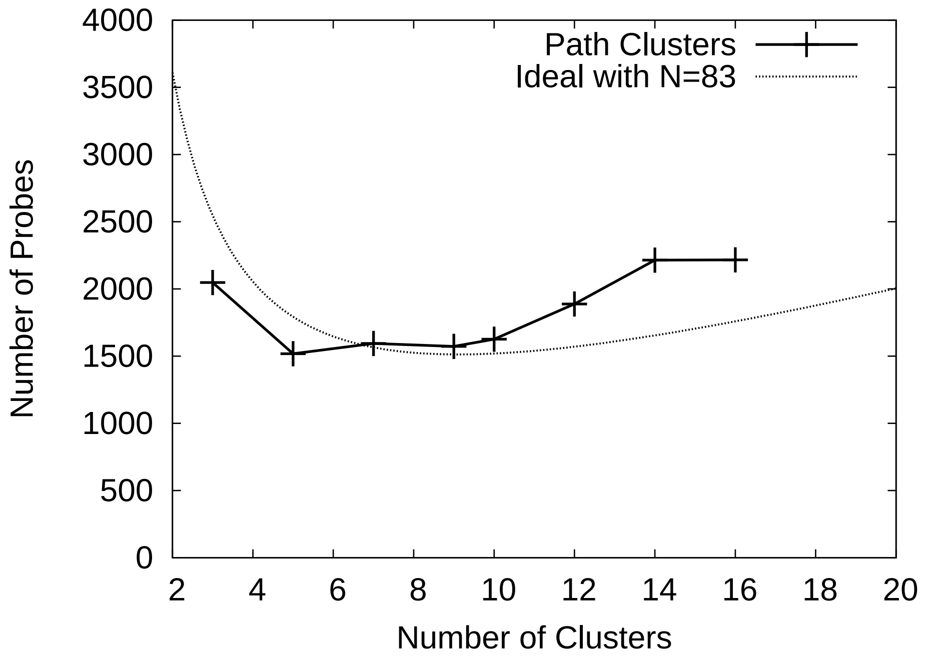


Figure 6.2: Probe Overhead

comparison, we also plot the best-case probe overhead for SABI Algorithm 1 if all clusters are of the same size. Note that in this case, a naive approach of all-pair AB measurement would need 6806 probes with $N = 83$.

We find that our approach can reduce the all-pairs measurements by a factor of up to 4. Also, as expected, we find that a cluster size close to $\sqrt{N}$ (9, in this case) is likely to yield the least probe overhead. The probe overhead with 9 clusters closely matches the ideal best-case (for uniform clusters).

Accuracy Figure 6.3 plots the values of $AB^{inf1}(i, j)$ as a function of $AB(i, j)$, for all pairs of nodes i and j, for which AB is inferred using $K = 12, m = 5$. We find that:

1. Most actual values of AB are clustered around three points: 10 Mbps, 40 Mbps, and 80 Mbps.

2. The inferred value of AB matches closely the actual AB for most points.

3. The inference error is higher for larger values of actual AB.

In order to quantify the inference errors, we use $m = 5$ and plot in Figures 6.4(a) and 6.4(b) for all node-pairs, the distribution of the absolute inference error and the relative inference error, respectively. We find that:

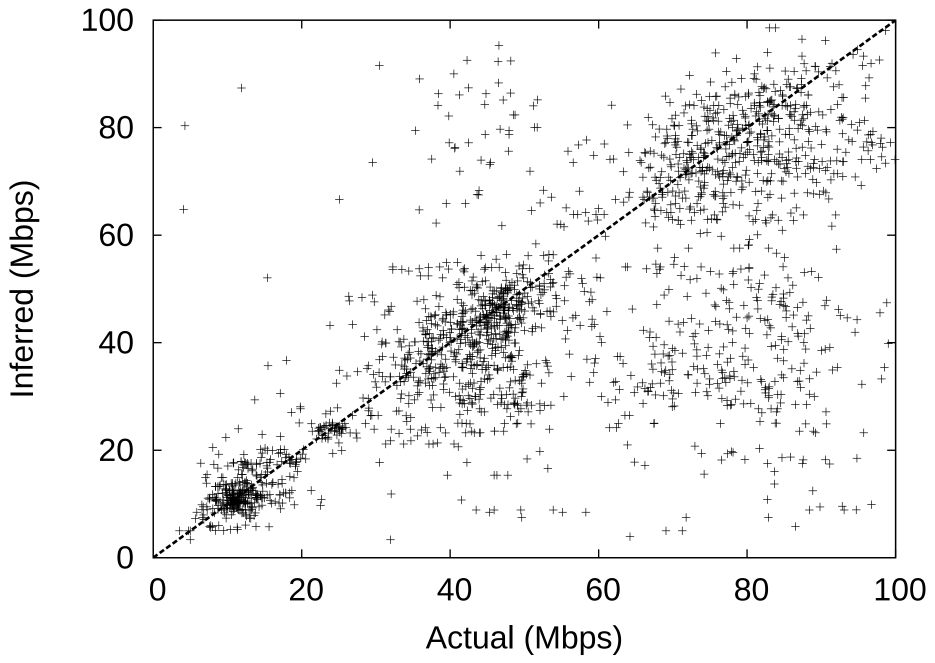


Figure 6.3: Path-based clustering: Actual vs. Inferred AB

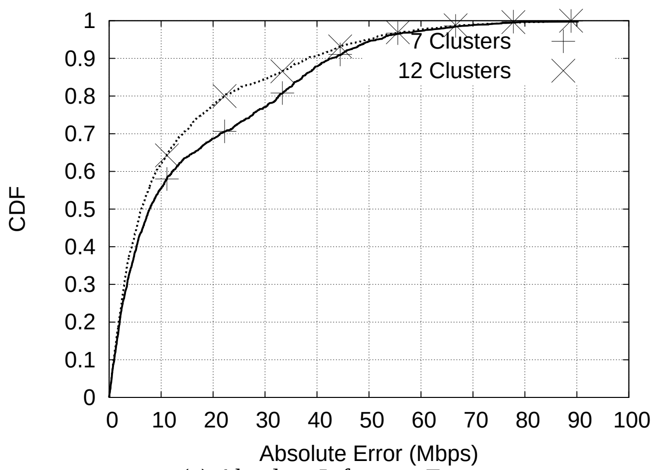
1. With 12 clusters, the median AB inference error is less than 20%, while 80% of the inferences lie within 40% of the actual.

2. 80% of the inferred AB lie within 20 Mbps of the actual AB. As discussed before, this lies within the estimation accuracy of existing ABETs.

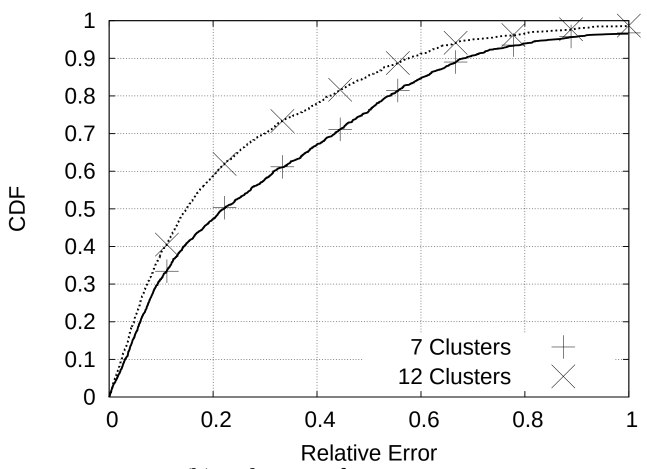
3. The larger is the number of clusters, the better is the inference accuracy.

Impact of Path Matching Parameter We next study the impact of varying m the path matching parameter on the accuracy of inferring AB. We regenerate clusters using three different similarity metrics corresponding to $M = 3, 5, 7$ and evaluate their AB inference accuracy. Figure 6.5 plots the cumulative distribution of the relative inferences errors for different values of m. We observe that a path matching parameter of 5 works the best. While $m = 3$ does not perform much worse, selecting an m of 7 noticeably degrades the performance of path-based clustering.

There are two reasons for preferring a smaller value of m. First, it is difficult to obtain route information corresponding to a longer access segment (such as 7)—in this case, we would require 7 routers on each path to respond to traceroute queries. Second, it is unlikely that nodes would share a larger suffix of their routes to a



(a) Absolute Inference Error



(b) Relative Inference Error

Figure 6.4: Distribution of Inference Errors

given destination. Thus, even "similar" source nodes would be characterized by a small value of the similarity metric, PD^m. Clustering, consequently, may not be able to form good node clusters.

6.4.2 Improving the Availability of AB Snapshots

Note that we were able to evaluate our clustering approach for no more than 25-40% of the node-pairs involved. As mentioned before, several factors related to the uncontrolled environment in PlanetLab—including node failure, tool failure, high loads, and rate limits—contribute to our inability to generate more data points. It is important to understand that such issues are likely to come up in most overlay infrastructures that wish to deploy an overlay monitoring service. We next consider an approach to improving the availability of AB estimates. Specifically, we assume that if an infrastructure is unable to measure the AB between two overly nodes, it would use the end-to-end capacity between the nodes as an approximation of the AB. Note that the capacity is a relatively static quantity and is already measured to facilitate node-clustering and head-selection.

In order to evaluate the impact of the above approximation on the inference accuracy of our approach, we re-consider our evaluation Algorithm, and for all cases in which the actual AB could be measured between nodes i and j, but one of the other two measurements was missing, we substitute the missing values with the capacities between the corresponding nodes. Doing so, helps us get 50-60% of the total data points. Figure 6.6 plots the cumulative distribution of the relative error. We find that this approximation does not adversely impact the AB inference accuracy of our approach—most of the new data points added follow a similar distribution of inference

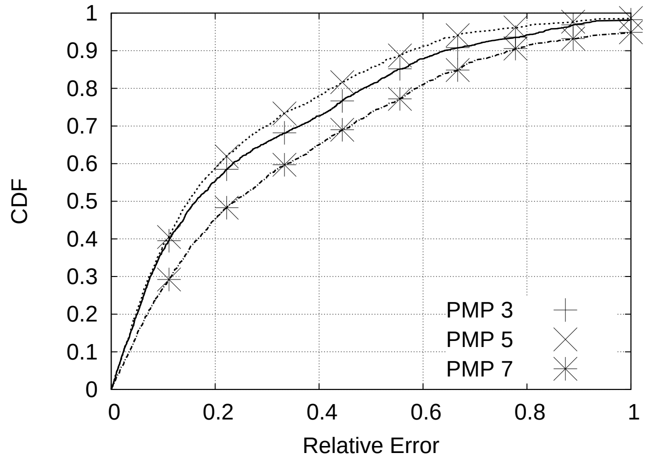


Figure 6.5: Impact of m on Inference Accuracy

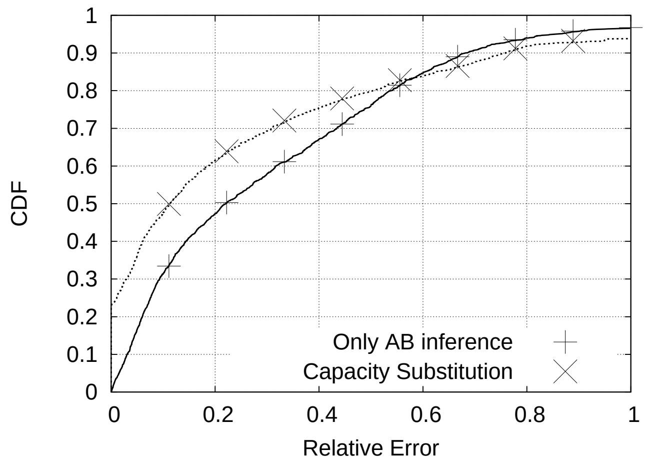


Figure 6.6: Distribution of Inference Errors with Capacity Substitution

error as the previous data. In fact, the overall distribution of estimation errors improve by observing these extra data points.

We conclude that to improve the availability of AB information, it is reasonable to use the capacity values when AB measures are missing.

6.4.3 Differential Rate Limiting on PlanetLab

PlanetLab implements a differential rate-limiting policy [rat]—traffic flowing on Internet-2 is allowed higher rate limits than traffic flowing in the general Internet. This differential treatment could cause inference errors

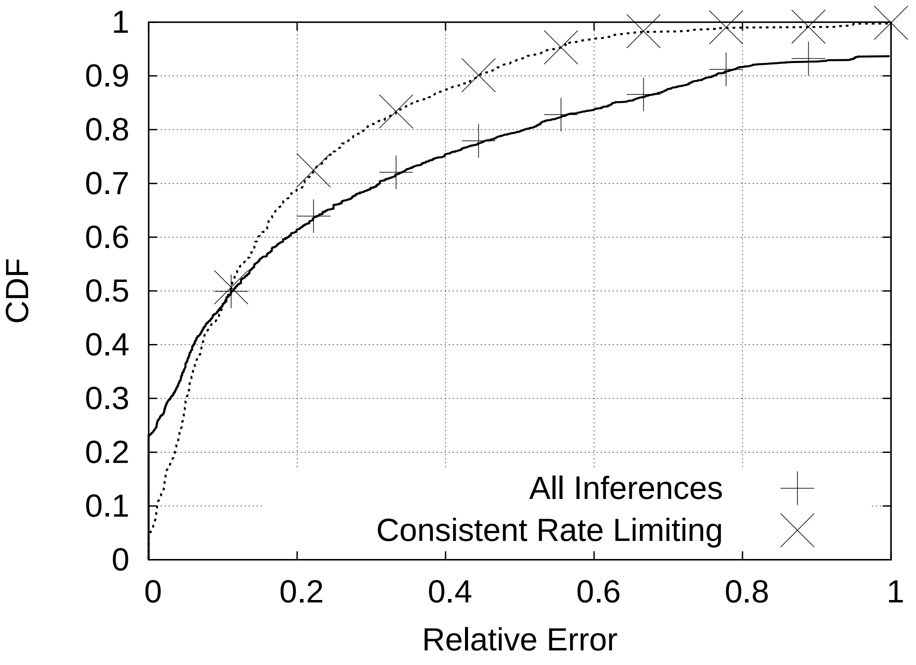


Figure 6.7: Performance of Consistently Rate Limited Paths

when an overlay node and its head-node do not belong to the same type of network. For instance, assume that the rate-limits for Internet-2 traffic is 100 Mbps, and for other traffic is 10 Mbps. In this case, if a source node belongs to Internet-2 and its head-node does not, then the inferred AB to a destination in Internet-2 may be much lower than the actual AB. Such inference errors could occur when overlay nodes are multi-homed—in this case, the paths to the head-node and a destination node may not share the access segment and, consequently, the bottleneck links.

Note that the differential rate-limiting would adversely impact the inference of only those node pairs in which exactly two nodes out of the three involved (source, destination, and head) belong to Internet-2. In order to assess the impact of this PlanetLab feature on our inference errors, we remove from our data all inferences that involve such node pairs.[4] Surprisingly, roughly 50% of our data points were of this type.

Figure 6.7 plots the distribution of the relative inference error when such node-pairs are removed from our data-set using $K = 7$ and $m = 5$. We find that the inference accuracy improves significantly—80% of the inferences have an error of 35% or less.

[4]In order to identify Internet-2 nodes, we first look for the ".edu" suffix in their IP names. Note that not all such nodes belong to Internet-2. We then examine the end-to-end capacity measurements from these nodes to other nodes in the system to identify instances of differential rate-limiting.

While this result is encouraging, it also suggests that our approach would need to select clusters carefully when overly nodes are multi-homed. We discuss one way of doing so next.

Identifying Ill-formed Clusters

In the presence of multi-homing or differential traffic management as in PlanetLab, we would need extra mechanisms to identify and modify ill-formed clusters. We rely on the observation that if a cluster is ill-formed for bandwidth-inference, it is likely to be ill-formed for even capacity-inference. In particular, for every 3-tuple of $\{i, H_i, j\}$, we compute the capacity inference error as: $\frac{|C_{i,j}^{inf} - C_{i,j}|}{C_{i,j}}$, where $C_{i,j}^{inf} = \min\{C_{i,H_i}, C_{H_i,j}\}$. If the estimation error is greater than 50%, we assume that the 3-tuple is ill-formed. After removing such 3-tuples from our data set, Figure 6.8 plots the AB inference errors of the remaining data points with $K = 7$ and $m = 5$. We observe that such filtering improves the inference accuracy. The inference error is now less than 35% for 80% of the inferences.

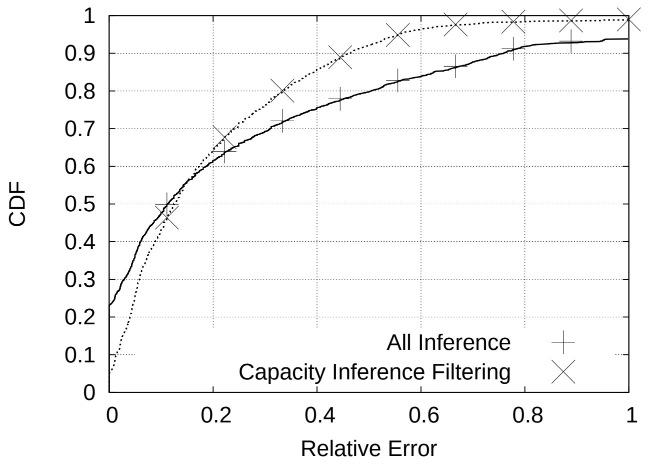


Figure 6.8: Using Capacity-inference to Filter Out Ill-formed Tuples

6.5 Sources of Error

Our approach has been able to achieve inference errors of less than 30-40% for 80% of the inferences, and median inference errors of less than 20%—we believe these numbers are promising due to several possible sources of errors. First, AB estimation tools themselves have an accuracy of about 20%—this is likely to impact any AB monitoring infrastructure. Furthermore, AB is a dynamic quantity that could change between successive measurements between a source and a head, and the head and a destination. Second, most ABETs require high time-stamping accuracy [PJD04] and are sensitive to heavy CPU load conditions—unfortunately, this is a frequent occurrence in PlanetLab. Finally, as we have shown, PlanetLab itself can introduce inference errors because of its differential rate-limiting policy. In light of these factors, and the fact that the only other AB monitoring approach reports an average error of 50%[HS05] (see Section 2), we believe that our approach is quite promising.

6.6 Comparative Evaluation of SABI algorithms

In this section, we provide a preliminary comparative evaluation of the three SABI algorithms and the Broute algorithm proposed in previous literature. This preliminary small scale evaluation on PlanetLab is promising and demonstrates the trade-offs between measurement overhead and inference accuracy. The methodology is similar to that described earlier with a few additional facets. A complication was that this evaluation coincided with the recent PlanetLab upgrade, which caused several nodes to be unavailable for several weeks. Starting with a new clique of 139 nodes, we computed 10 SABI clusters. Using CoMon [PP] data we identified 20 nodes that were lightly loaded and were able to return measurement data consistently. For these 20 nodes, which were spread across 8 clusters, we ran the measurements as described in the methodology.

To compare with Broute, we implement the peer-to-peer variant of Broute and use IP source and sink trees as opposed to AS trees. We consider the first four and last four hops in the trees to determine common segments. We consider IP source and sink trees primarily to keep the probing overheads to be of the same order as the SABI algorithms.

The overheads that would be incurred for the set of 139 nodes for the various schemes are given below. As a baseline, the all-pair measurements would have been 19,182 bandwidth probes. We need 4,027 (20.99%), 550 (2.87%) and 278 (1.45%) probes respectively for the three SABI algorithms with 10 clusters. For the corresponding Broute implementation, we would require 1,882 (9.8%) measurements.

After running these measurements, we had sufficient data to evaluate a common set of 48 paths for each of the 4 algorithms. Figure 6.9 plots the cumulative distribution function of the relative inference error for the three SABI algorithms and the Broute (peer-to-peer IP trees implementation described above). Though this is a small scale study, the results are very promising. Essentially, the three SABI algorithms perform well and approximately the same while Broute has poorer accuracy. Even SABI Algo 3, which uses less than 2% probes has a median error of about 15%. We ran this study for two different snapshots with similar results.

It is interesting to recall that SABI Algo 3 is similar in concept to the Broute approach—both rely on using the AB on the access segments of either end-points to infer the end-to-end AB. The main difference is that Broute relies on a per-hop AB estimation tool such as Pathneck [HLM+04], which can be inaccurate; whereas SABI drafts well-provisioned head-nodes and uses tools for end-to-end AB, which have shown to be reasonably accurate [SK07, SMH+05].

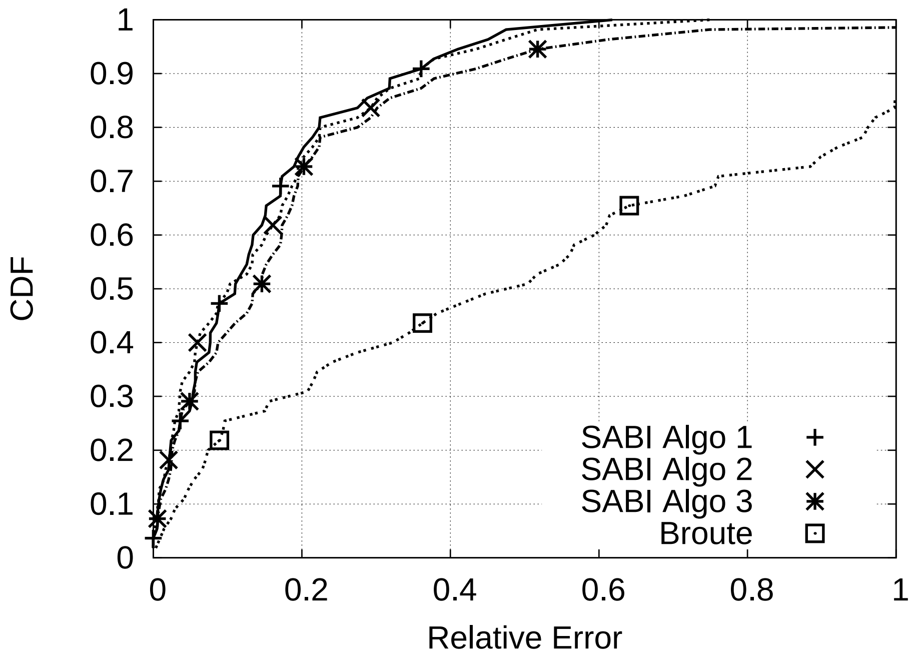


Figure 6.9: CDF of relative inference error for the SABI algorithms and Broute AB

6.7 Conclusion

In this chapter we describe SABI, a scalable method for end-to-end available bandwidth inference for large networked systems. SABI groups nodes into clusters such that the nodes in each cluster have a *similar* view of the nodes outside the cluster regarding the available bandwidth metric. Using the key insight that most bandwidth bottlenecks are close to the edges of the network, we propose a path similarity metric for measuring the similarity of a pair of nodes. Nodes are then grouped together according to this similarity metric, and a head representative node is chosen for each cluster that has highest capacity access link than others in the cluster. We propose three different inference algorithms with decreasing measurement overheads that define which measurements are performed within a cluster and which are performed across clusters. Extensive experimentation on the PlanetLab testbed have demonstrated that the SABI algorithms reduce the probing overhead significantly while maintaining the mean estimation error around 20%.

We plan to provide this bandwidth estimation as a service on PlanetLab for other researchers to use and experiment with.

CHAPTER 7
Concluding Remarks

In this dissertation, we systematically study the impact of each of the four factors (algorithmic, temporal,network and systemic) on ABET performance and find that:

- *With regards to temporal factors* we find that the (i) MT does not impact the inaccuracy as long as the SI is kept constant. (ii) As the SI increases the accuracy increase, however beyond a SI of 30% the gain in accuracy is not significant. Thus we can control the overhead that is introduced by a tool by limiting the SI to not greater than 0.5. (iii) We also found that the variability of the AB reduces at MT of greater than 50 ms. (iv) Longer the RT the higher the measure of variability. (v) Back-to-back measurements of the AB do not vary by more than 4%.

- *With regards to algorithmic factors and network factors*(i) We find that Pathload, and Spruce are the most accurate at high MT. (ii) By increasing the SI we cannot improve the accuracy of the AB techniques, since there is an inherent error associated with the inference logic. (iii) Pathchirp has the lowest overhead, and intrusiveness characteristics. (iv) When we move away from the single bottleneck scenario, we observe that the accuracy of Spruce and Pathchirp reduces.(v) Fast-IGI, IGI and Pathload are not affected by changing topologies.

- *With regards to systemic factors* (i) We find that tools using packet pair techniques must be aware of delay quantization. (ii) Furthermore a packet size of 1500 bytes is not sensitive enough for probing on high-speed paths. (iii) The accuracy of Pathload increases from what was observed in the NS-2 simulations. (iv) We find that Pathload and Pathchirp are the most accurate of the ABETs and have a performance of about 15% and 20% respectively.(v) Iperf is the most accurate, however a loss rate of even around 1% and conservative RTO values, could cause huge under-estimation errors.

From all of the above studies, we obtained insights into the impact that the different factors have on the performance of the ABET tools. We then show how these insights can be used to choose and calibrate an ABET for an AB monitoring infrastructure. We then use an end-to-end ABET to design three algorithms to scalably estimate the AB on a large network. We show that we can obtain information about the AB of n^2 paths of a network by making only O(n) measurements of the AB . These measurements are also accurate to about 30%, which is a significant improvement over previous schemes. Finally we compare the performance of our scheme to another scheme Broute, which requires per-hop AB information and show that our schemes consistently outperform Broute.

In this thesis we demonstrated the viability of building a system to monitor the AB on a large network. From a purely monitoring stand point this scheme would be useful for network operators to study the location of potential hot-spots in their network, without imposing a significant load. From the perspective of an application we have currently deployed a scaled down version of our AB monitoring scheme on PlanetLab and we hope to be able to scale this up to a full scale deployment in the near future.

We started off by observing that the adoption of AB estimation techniques in contemporary applications is fairly limited. We then identified and addressed two issues the seemed like hurdles for adoption: *which technique to use*, and *how to make AB estimation scalable in multi-path services*. However, there is at least one additional question that application/service designers are likely to be interested in: *what performance improvement can an application achieve by relying on the knowledge of end-to-end AB, versus metrics such as delay and loss rates, that represent a simpler characterization of paths?* As part of future work, we hope to investigate this issue by instrumenting protocols and services with AB estimation techniques and evaluating the performance benefits from doing so.

Appendix A
Minimizing the probe overhead

Let us consider an N node system in which we create K clusters. Let us furthur assume that all the clusters are of the same size. This implies that: $C = \frac{N}{K}$, where C is the number of nodes per cluster.

Our AB inference approach implies that the head node of a cluster will measure the AB to all the other nodes not in the cluster. Thus the number of probes sent from one cluster out to the other clusters is:

$$Probe_{out} \quad = \quad N - C \tag{A.1}$$

Each node within a cluster measures the AB to all the other nodes within the cluster. Thus the number of probes within a cluster is:

$$Probe_{in} \quad = \quad C \times (C - 1) \tag{A.2}$$

Summing (A.1) and (A.2), we get that the total number of probes sent out by a cluster is:

$$Probe_{clus} = C \times (C - 1) + (N - C) \tag{A.3}$$

And total number of probes across all clusters is:

$$Probe_{tot} = K \times (C \times (C - 1) + (N - C))eqn.iv) \tag{A.4}$$

The number of probes as a function of K is represented as:

$$F(K) = \frac{N^2}{K} + N \times K \tag{A.5}$$

Differentiating $F(K)$ w.r.t K, we get the minima at $K = \sqrt{N}$. Note that this does not include the probe overhead that is incurred to form the clusters, since this process will be carried out infrequently.

Appendix B
Sites used in Planet-Lab experiment

.ece.uprm.edu	.upc.es	.iralab.uni-karlsruhe.de
.cs.uga.edu	.umkc.edu	.ittc.ku.edu
.epfl.ch	.cs.brown.edu	.planetlab.uprr.pr
.mcgillplanetlab.org	.planetlab.cs.umd.edu	.pl.utsa.edu
.cs.utk.edu	.hip.fi	.ucs.indiana.edu
.unm.edu	.eece.ksu.edu	.uni-klu.ac.at
.berkeley.intel-research.net	.colbud.hu	.ottawa.canet4.nodes.planet-lab.org
.scs.cs.nyu.edu	.att.nodes.planet-lab.org	.learninglab.uni-hannover.de
.prakinf.tu-ilmenau.de	.cc.gt.atl.ga.us	.bu.edu
.cnds.unibe.ch	.cs.tcd.ie	.ethz.ch
.cs.stir.ac.uk	.cs.princeton.edu	.arizona-gigapop.net
.cesnet.cz	.csail.mit.edu	.cs.cornell.edu
.cs.dartmouth.edu	.cse.msu.edu	.csg.unizh.ch
.cs.purdue.edu	.csres.utexas.edu	.cs.uchicago.edu
.cs.umass.edu	.cs.unc.edu	.dcs.st-and.ac.uk
.diku.dk	.eecs.umich.edu	.hiit.fi
.informatik.uni-erlangen.de	.informatik.uni-goettingen.de	.info.ucl.ac.be
.iscte.pt	.isi.jhu.edu	.it.uu.se
.itwm.fhg.de	.ls.fi.upm.es	.netlab.uky.edu
.net-research.org.uk	.nrl.dcs.qmul.ac.uk	.pc.cis.udel.edu
.ssvl.kth.se	.uta.edu	.utep.edu
.win.trlabs.ca	.cs.columbia.edu	.csee.usf.edu
.cs.vu.nl	.cs.wayne.edu	.cs.wisc.edu
.elet.polimi.it	.engr.uconn.edu	.flux.utah.edu
.mini.pw.edu.pl	.poly.edu	.sics.se
.cs.duke.edu	.cs.uiuc.edu	.ece.iastate.edu
.ite.gmu.edu	.hpl.hp.com	.cs.rice.edu
.eecs.harvard.edu	vn1.cs.wustl.edu	

Table B.1: Sites used in experiment

BIBLIOGRAPHY

[ABKM01] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Symposium on Operating Systems Principles*, pages 131–145, 2001.

[abt] Planet-lab email list. http://lists.planet-lab.org/pipermail/users/2006-February/001782.html.

[AGKT98] George Apostolopoulos, Roch Guérin, Sanjay Kamat, and Satish K. Tripathi. Quality of service based routing: a performance perspective. *SIGCOMM Comput. Commun. Rev.*, 28(4):17–28, 1998.

[AKSJ03] J. Aikat, J. Kaur, D. Smith, and K. Jeffay. Variability in TCP round-trip times. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference*, October 2003.

[AMPR03] J.L. Alberi, A. McIntosh, M. Pucci, and T. Raleigh. Overcoming precision limitations in adaptive bandwidth measurements. In *3rd New York Metro Area Networking Workshop (NYMAN)*, September 2003.

[AMS$^+$03] Aditya Akella, Bruce Maggs, Srinivasan Seshan, Anees Shaikh, and Ramesh Sitaraman. A measurement-based analysis of multihoming. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 353–364, New York, NY, USA, 2003. ACM Press.

[AP99] Mark Allman and Vern Paxson. On estimating end-to-end network path properties. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 263–274, New York, NY, USA, 1999. ACM.

[BCG$^+$01] Yuri Breitbart, Chee Yong Chan, Minos N. Garofalakis, Rajeev Rastogi, and Abraham Silberschatz. Efficiently monitoring bandwidth and latency in IP networks. In *INFOCOM*, pages 933–942, 2001.

[BGMS04] A. Balk, M. Gerla, D. Maggiorini, and M. Sanadidi. Adaptive video streaming: pre-encoded mpeg-4 with bandwidth scaling. *Comput. Netw.*, 44(4):415–439, 2004.

[Bit] BitTorrent.

[BOP94] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. Tcp vegas: new techniques for congestion detection and avoidance. *SIGCOMM Comput. Commun. Rev.*, 24(4):24–35, October 1994.

[Bro04] N. Brownlee. NeTraMet 5.0b3, 2004. http://www.caida.org/tools/measurement/netramet/.

[CC96a] R. Carter and M. Crovella. Measuring Bottleneck Link Speed in Packet-Switched Networks. Technical Report 96-006, Boston University, 1996.

[CC96b] Robert Carter and Mark Crovella. Measuring Bottleneck Link Speed in Packet-Switched Networks. Technical Report BUCS-TR-1996-006, Computer Science Department, Boston University, March 15 1996.

[CC96c] Robert Carter and Mark Crovella. Measuring bottleneck link speed in packet-switched networks. (1996-006), 15, 1996.

[CCLS01] J. Cao, W. Cleveland, D. Lin, and D. Sun. The effect of statistical multiplexing on internet packet traffic: theory and empirical study. *Bell-Labs Technical Report*, 2001.

[CKC05] David Chua, Eric D. Kolaczyk, and Mark Crovella. A statistical framework for efficient monitoring of end-to-end network properties. *SIGMETRICS Perform. Eval. Rev.*, 33(1):390–391, 2005.

[CL02] Les Cottrell and Connie Logg. Overview of IEPM-BW Bandwidth Testing of Bulk Data Transfer. In *Sc2002: High Performance Networking and Computing*, 2002.

[CP02] F. Coccetti and R. Percacci. Bandwidth measurements and router queues. Technical Report INFN/Code-20 settembre 2002, Instituto Nazionale Di Fisica Nucleare, Trieste, Italy, 2002. http://ipm.mib.infn.it/bandwidth-measurements-and-router-queues.pdf.

[CPB93] Kimberly C. Claffy, George C. Polyzos, and Hans-Werner Braun. Application of sampling methodologies to network traffic characterization. In *SIGCOMM '93: Conference proceedings on Communications architectures, protocols and applications*, pages 194–203. ACM Press, 1993.

[del] Per-flow delay and loss in ns-2 with delaybox. http://dirt.cs.unc.edu/delaybox/.

[DJ03] C. Dovrolis and M. Jain. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. *IEEE/ACM Transactions in Networking*, August 2003.

[DMA$^+$06] D.Antoniades, M.Athanatos, A.Papadogiannakis, E.P.Markatos, and C. Dovrolis. Available bandwidth measurement as simple as running wget. In *Proceedings of Passive and Active Measurement Workshop (PAM)*, March 2006.

[Dov01] C. Dovrolis. Pathrate : A measurement tool for the capacity of network paths. In *ACM SIGCOMM*, 2001. http://www.pathrate.org.

[Dow99] A. Downey. clink: a tool for estimating internet link characteristics, 1999.

[DPJ04] C. Dovrolis, R. Prasad, and M. Jain. Socket Buffer Auto-Sizing for High-Performance Data Transfers. *Journal of Grid Computing*, 1(4), 2004.

[ea07] K. Thompson et. al. Wide-area internet traffic patterns and characteristics. *IEEE Networks*, November/December, 2007.

[FBB01] N. Feamster, D. Bansal, and H. Balakrishnan. the interactions between layered quality adaptation and congestion control for streaming video, 2001.

[GCM00] Liang Guo, Mark Crovella, and Ibrahim Matta. Tcp congestion control and heavy tails. Technical report, Boston, MA, USA, 2000.

[HCSJ04] F. Hernandez-Campos, F. D. Smith, and K. Jeffay. Generating realistic tcp workloads. *Proceedings of CMG*, pages 273–284, 2004.

[HLM$^+$04] Ningning Hu, Li (Erran) Li, Zhuoqing Morley Mao, Peter Steenkiste, and Jia Wang. Locating internet bottlenecks: algorithms, measurements, and implications. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 41–54, New York, NY, USA, 2004. ACM Press.

[HS03] N. Hu and P. Steenkiste. Evaluation and Characterization of Available Bandwidth Probing Techniques. *IEEE JSAC Internet and WWW Measurement, Mapping, and Modeling*, 2003.

[HS05] N. Hu and P. Steenkiste. Exploiting internet route sharing for large scale available bandwidth estimation. In *Internet Measurements Conference(IMC)*, 2005.

[Hyu04] Y. Hyun. Running Bandwidth Estimation Tools on Wide-Area Internet Paths, 2004. http://www.caida.org/projects/bwest/reports/tool-comparison-supplement.xml.

[Inc92] The Math Works Inc. *MATLAB, High-performance Numeric Computation and Visualization Software. User's Guide*. 1992.

[ipe] Iperf. http://dast.nlanr.net/Projects/Iperf/.

[Jac] V. Jacobson. pathchar. ftp://ftp.ee.lbl.gov/pathchar/.

[JD02a] M. Jain and C. Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth, 2002.

[JD02b] M. Jain and C. Dovrolis. Pathload: an available bandwidth estimation tool. In *PAM*, 2002.

[JD04] M. Jain and C. Dovrolis. Ten fallacies and pitfalls on end-to-end available bandwidth estimation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, October 2004.

[JD05a] Manish Jain and Constantinos Dovrolis. End-to-end estimation of the available bandwidth variation range. In *Proceedings of ACM Sigmetrics 05*, June 2005.

[JD05b] Hao Jiang and Constantinos Dovrolis. Why is the internet traffic bursty in short time scales? *SIGMETRICS Perform. Eval. Rev.*, 33(1):241–252, 2005.

[Jin04] Guojun Jin. netest-2, 2004. http://www-didc.lbl.gov/NCS/netest.html.

[Jor04] L. Jorgenson. Size Matters: Network Performance on Jumbo Packets. In *Joint Techs Workshop Columbus, OH*, July 2004.

[JT03] G. Jin and B.L. Tierney. System capability effects on algorithms for network bandwidth measurement. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference*, October 2003.

[JYCA01] G. Jin, G. Yang, B. R. Crowley, and D. A. Agarwal. Network Characterization Service (NCS). Technical report, LBNL, 2001.

[KK06] R. Kumar and J. Kaur. Practical beacon placement for link monitoring using network tomography. *IEEE Journal on Selected Areas in Communication*, 2006.

[kme] Kmeans. http://people.revoledu.com/kardi/tutorial/kMean/matlabkMeans.htm.

[KMF04] Thomas Karagiannis, Mart Molle, and Michalis Faloutsos. Long-range dependence: Ten years of internet traffic modeling. *IEEE Internet Computing*, 8(5):57–64, 2004.

[KMK$^+$01] K. Keys, D. Moore, R. Koga, E. Lagache, M. Tesch, and k Claffy. The architecture of CoralReef: an Internet traffic monitoring software suite. In *Passive and Active Network Measurement (PAM) Workshop, Amsterdam, Netherlands*, April 2001.

[KV06] George Kola and Mary K. Vernon. Quickprobe: available bandwidth estimation in two roundtrips. *SIGMETRICS Perform. Eval. Rev.*, 34(1):359–360, 2006.

[LDS06] Li Lao, Constantine Dovrolis, and M. Y. Sanadidi. The probe gap model can underestimate the available bandwidth of multihop paths. *SIGCOMM Comput. Commun. Rev.*, 36(5):29–34, 2006.

[LFV07a] J. Liebeherr, M. Fidler, and S. Valaee. A min-plus system of bandwidth estimation. *IEEE Infocom 2007*, pages 29–34, 2007.

[LFV07b] Jörg Liebeherr, Markus Fidler, and Shahrokh Valaee. A min-plus system interpretation of bandwidth estimation. In *Proceedings of IEEE INFOCOM*, May 2007.

[Li05] Ningning Hu Li. A measurement study of internet bottlenecks. In *Proceedings of IEEE INFCOM*, 2005.

[LRL04] X. Liu, K. Ravindran, and D. Loguinov. Evaluating the potential of bandwidth estimators. In *4th New York Metro Area Networking Workshop (NYMAN)*, September 2004.

[LRL05] Xiliang Liu, Kaliappa Ravindran, and Dmitri Loguinov. Multi-hop probing asymptotics in available bandwidth estimation: Stochastic analysis. In *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet measurement*, New York, NY, USA, 2005. ACM Press.

[LRLL04] Xiliang Liu, Kaliappa Ravindran, Benyuan Liu, and Dmitri Loguinov. Single-hop probing asymptotics in available bandwidth estimation: sample-path analysis. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 300–313, New York, NY, USA, 2004. ACM Press.

[LTWW93] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of ethernet traffic. *SIGCOMM Comput. Commun. Rev.*, 23(4):183–193, 1993.

[Mah00] B. Mah. pchar: A tool for measuring internet path characteristics. In *ISMA*, 2000. http://www.employees.org/b̃mah/Software/pchar/.

[Mat03] M. Mathis. The Web100 Project, 2003. http://www.web100.org.

[MBG00] B. Melander, M. Bjorkman, and P. Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Global Internet Symposium*, November 2000.

[Mos08] Kostas Moschos. Digitisation of audio music files, survey on existing software solutions, 2008.

[Nav03] J. Navratil. ABwE: A Practical Approach to Available Bandwidth. In *PAM*, 2003.

[NLA04] NLANR. Passive Measurement Analysis Datacube, August 2004. http://pma.nlanr.net/Datacube/.

[NN98] Jim M. Ng and Peoy Khee Ng. Cost-delay path selection function for real-time multicast routing. In *MASCOTS '98: Proceedings of the 6th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 56, Washington, DC, USA, 1998. IEEE Computer Society.

[NS2] Network simulator-2 ns2 (http://www.isi.edu/nsnam/ns/).

[Pax97] Vern Paxson. End-to-end Internet packet dynamics. In *Proceedings of the ACM SIGCOMM '97 conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, volume 27,4 of *Computer Communication Review*, pages 139–154, Cannes, France, September 1997. ACM Press.

[PDM02] R. Prasad, C. Dovrolis, and B. Mah. The effect of layer-2 store-and-forward devices on per hop capacity estimation. In *IMW*, 2002.

[PJD04] R. Prasad, M. Jain, and C. Dovrolis. Effects of Interrupt Coalescence on Network Measurements. In *PAM*, 2004.

[PKC96] Kihong Park, Gitae Kim, and Mark Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *ICNP '96: Proceedings of the 1996 International Conference on Network Protocols (ICNP '96)*, page 171, Washington, DC, USA, 1996. IEEE Computer Society.

[Pla] Planetlab. http://www.planet-lab.org/.

[PP] KyoungSoo Park and Vivek Pai. Comon: A monitoring infrastructure for planetlab. http://comon.cs.princeton.edu/.

[PV02a] A. Pasztor and D. Veitch. Precision based precision timing without gps. In *Proceedings of ACM SIGMETRICS*, June 2002.

[PV02b] Attila Pasztor and Darryl Veitch. On the Scope of End-to-end Probing Methods. *Communications Letters, IEEE*, 6(11), November 2002.

[Qur04] A Qureshi. Exploring proximity based peer selection in bittorrent-like protocol. *MIT 6.824 student project*, 2004.

[rat] Planetlab:bandwidth limits. http://www.planet-lab.org/doc/BandwidthLimits.php.

[Rib03] V. Ribeiro. pathChirp: Efficient Available Bandwidth Estimation for Network Path. In *PAM*, 2003.

[RK03] S. Rewaskar and J. Kaur. Testing the scalability limits of overlay routing infrastructures. *Technical Report, Department of Computer Science, University of North Carolina at Chapel Hill*, May 2003.

[RRB$^+$03] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. pathChirp: Efficient available bandwidth estimation for network paths. In *Passive and Active Measurement Workshop* , April 2003.

[RRB04] V. Ribeiro, R. Riedi, and R. Baraniuk. Spatio-Temporal Available Bandwidth Estimation with STAB. In *ACM Sigmetrics/Performance*, New York, NY, June 2004.

[San04] San Diego Supercomputer Center . CalNGI Network Performance Reference Lab (NPRL), 2004. http://www.calngi.org/about/index.html.

[SK06] A. Shriram and J. Kaur. Empirical study of the impact of sampling timescales and strategies on measurement of available bandwidth. In *Proceedings of Passive and Active Measurement Workshop (PAM)*, March 2006.

[SK07] Alok Shriram and Jasleen Kaur. Empirical evaluation of techniques to measure available bandwidth. In *INFOCOM*. IEEE Communications Society, 2007.

[SKK03a] Jacob Strauss, Dina Katabi, and Frans Kaashoek. A Measurement Study of Available Bandwidth Estimation Tools. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference ’03*, Miami, Florida, October 2003.

[SKK03b] Jacob Strauss, Dina Katabi, and Frans Kaashoek. A measurement study of available bandwidth estimation tools. In *IMW*, 2003.

[SLA04] SLAC. Internet End-to-end Performance Monitoring - Bandwidth to the World (IEPM-BW) Project, August 2004. http://www-iepm.slac.stanford.edu/bw/.

[SMH$^+$05] Alok Shriram, Margaret Murray, Young Hyun, Nevil Brownlee, Andre Broido, Marina Fomenkov, and Kimberly C. Claffy. Comparison of public end-to-end bandwidth estimation tools on high-speed links. In *PAM*, pages 306–320, 2005.

[Spi04a] Spirent Corp. Smartbits 6000B, 2004. http://spirentcom.com/analysis/view.cfm?P=141.

[Spi04b] Spirent Corp. Smartflow, 2004. http://spirentcom.com/analysis/view.cfm?P=119.

[SQZ06] Han Hee Song, Lili Qiu, and Yin Zhang. Netquest: a flexible framework for large-scale network measurement. *SIGMETRICS Perform. Eval. Rev.*, 34(1):121–132, 2006.

[Ter] TeraGrid. Teragrid. http://www.teragrid.org/.

[tie] Tier-1 service provider article on wikipedia. http://en.wikipedia.org/wiki/Tier1carrier.

[TUAK04] Turhan Tunalimath, Nukhet Uzbek, Koray Anar, and Aylin Kantarc. Bandwidth-aware scaling for internet video streaming. *LNCS Computer and Information Sciences - ISCIS 2004*, 3280, 2004.

[Tur04] A. Turner. tcpreplay 2.2.2 - a tool to replay saved tcpdump files at arbitrary speed, July 2004. http://tcpreplay.sourceforge.net/.

[WAHC$^+$06] M.C. Weigle, P. Adurthi, F. Hernandez-Campos, K. Jeffay, and F.D. Smith. Tmix: A tool for generating realistic application workloads in ns-2. *ACM SIGCOMM Computer Communication Review*, 26(3):67–76, 2006.

[Wol98] Richard Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998.

[WY00] Kun-Lung Wu and Philip S. Yu. Latency-sensitive hashing for collaborative web caching. In *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking*, pages 633–644, Amsterdam, The Netherlands, The Netherlands, 2000. North-Holland Publishing Co.

[YSB$^+$06] P. Yalagandula, P. Sharma, S. Banerjee, S.-J.Lee, and S. Basu. S3: A scalable sensing service for monitoring large networked systems. In *Internet Network Management (INM)*, 2006.

[ZDA06] Yong Zhu, Constantinos Dovrolis, and Mostafa Ammar. Dynamic overlay routing based on available bandwidth estimation: a simulation study. *Comput. Networks*, 50(6):742–762, 2006.