Fall 12-1-2015

Bandwidth Estimation for Virtual Networks

Ertong Zhang
University of Nebraska-Lincoln, zhangertong@gmail.com

BANDWIDTH ESTIMATION FOR VIRTUAL NETWORKS

by

Ertong Zhang

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Doctor of Philosophy

Major: Computer Science

Under the Supervision of Professor Lisong Xu

Lincoln, Nebraska

December, 2015

BANDWIDTH ESTIMATION FOR VIRTUAL NETWORKS

Ertong Zhang, Ph.D.

University of Nebraska, 2015

Adviser: Professor Lisong Xu

Cloud computing is transforming a large part of the IT industry, as evidenced by the increasing popularity of public cloud computing services, such as Amazon Web Service, Google Cloud Platform, Microsoft Windows Azure, and Rackspace Public Cloud. Many cloud computing applications are bandwidth-intensive, and thus the network bandwidth information of clouds is important for their users to manage and troubleshoot the application performance.

The current bandwidth estimation methods originally developed for the traditional Internet, however, face great challenges in clouds due to virtualization that is the main enabling technique of cloud computing. First, virtual machine scheduling, which is an important component of computer virtualization for processor sharing, interferes with packet timestamping and thus corrupts the network bandwidth information carried by the packet timestamps. Second, rate limiting, which is a basic building block of network virtualization for bandwidth sharing, shapes the network packets and thus complicates the bandwidth analysis of the packets.

In this dissertation, we tackle the two virtualization challenges to design new bandwidth estimation methodologies for clouds. First, we design bandwidth estimation methods for networks with rate limiting, which is widely used in cloud networks. Bandwidth estimation for networks with token bucket shapers (i.e., a basic type of rate limiters) has been studied before, and the conclusion is that “both capacity and available bandwidth measurement are challenging because of the dichotomy between the raw link bandwidth and the token

bucket rate". Our methods are based on in-depth analysis of the multi-modal distributions of measured bandwidths.

Second, we expand the design space of bandwidth estimation methods to challenging but not rare networks where accurate and correct packet time information are hard to obtain, such as in cloud networks with heavy virtual machine scheduling. Specifically, we design and develop a fundamentally new class of sequence-based bandwidth comparison methods that relatively compare the bandwidth information of multiple paths instead of accurately estimating the bandwidth information of a single path. By doing so, our methods use only packet sequence information but not packet time information, and are fundamentally different from the current bandwidth estimation methods that all use packet time information. Furthermore, we design and develop a new class of sequence-based bandwidth estimation methods by conveying the time information in the packet sequence. Sequence-based bandwidth estimation methods estimate the bandwidth information of a path using the time information conveyed in the packet sequence from another path.

ACKNOWLEDGMENTS

Firstly, I would like to thank my advisor Dr. Lisong Xu for his careful instruction. He spent a lot of his time discussing the research issues with me, devoted himself to helping me improve my writing skills, and provided me with funding support. It is my luck to work with such a nice professor for my Ph.D. study. I also thank my previous advisor Dr. Xue Liu who instructed me for the first semester when I arrived at the University of Nebraska at Lincoln.

It is an honor to work with Professor Byrav Ramamurthy, Professor Hongfeng Yu, and Professor Ming Han on my dissertation. Professor Ramamurthy is an expert in networking and he always discusses networking research problems with me, as well as provides me with good suggestions during my research. He is also my first TA instructor in the first year. Professor Yu joined the CSE department later, and I am glad to meet him in my Ph.D. study. His research is in visualization and graphics, but he also gives me a lot of good advice for my study. Professor Han is from the ECE department. I thank him for spending time reviewing my dissertation and giving me good review suggestions.

I am grateful to my wife Fengyu Dai. We got married at the end of my third year in my Ph.D. Then she moved to the U.S. to live with me. I can have good Chinese food everyday. We also travel a lot in the U.S. This makes my Ph.D. life more interesting. I also thank my family, my parents, brother and sisters. Without their support, I could not have undertaken my Ph.D. study.

I also thank those friends I met in UNL. Graduate students in the Department of Computer Science, Wei Sun, Pan Yi, Junjie Qian, Yaodong Yang, Guangdong Liu, Lina Yu, etc.. My roommates Jingfeng Song, Zhengguo Xiao, Zhongjian Zhang, and Andy Liu, as well as local families in the International Student Fellowship. I was never alone in my study with these nice people around.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 The Epoch of Virtual Technology

In 1943, IBM President Thomas J Watson had said that the world only needed five computers. Watsons idea was to share a super computer among multiple users. This idea is used again in the current cloud computing, where multiple applications are running in a single cloud. The cloud enables the sharing of thousands of servers across multiple application providers through the virtualization technology.

One fundamental virtualization technology is *machine virtualization*. It can create a large number of virtual machines (VM) on a given physical machine. With machine virtualization, the cloud size can be increased by hundreds of times. If a cloud has 1,000 physical computers, it can now provide 100,000 VMs to cloud users if each physical machine has 100 VMs. In addition, all VMs are separated from one another, and applications running on one VM does not influence other applications on different VMs, even if they are located in the same physical machine.

The second virtualization technology is *network virtualization*. Network virtualization splits a physical network into multiple small virtual networks. Different network users can have different virtual networks over the same shared physical network. Network virtualization has many advantages. First, it isolates the virtual network users from one another, so that it is more convenient and secure for them to use the physical network. Second, it is easier for a network provider to manage the whole physical network. For example,

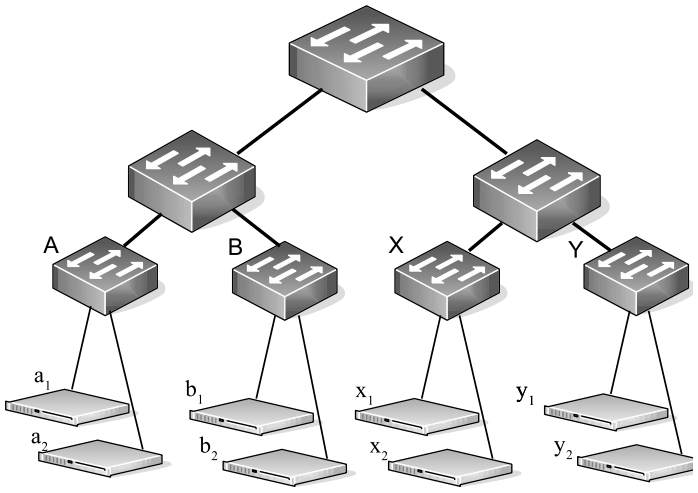
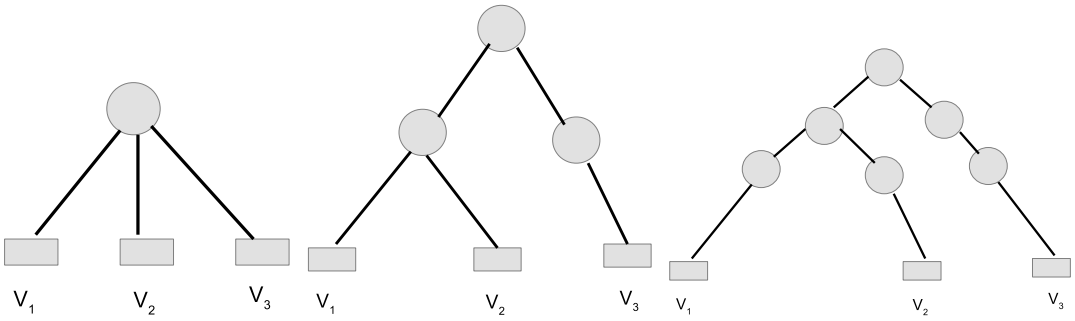


Figure 1.1: FAT tree structure for a data center network.

more bandwidth can be applied to a virtual network, if its user wants more bandwidth and is willing to pay for it. Third, it saves money for both the network provider and network users. The network provider does not need to build a separate physical network for each user, and the users can get their virtual networks at a very low cost.

However, sharing a physcial network is much more complex than sharing a physical server. A network is composed of multiple servers and multiple routers. For example, a typical data center network [3] is shown in Figure 1.1. A network user uses only a small portion of the physical network. This small network portion can be in various network structures. For example, a user uses three VMs. These VMs can be close or far away from one another as shown in Figure 1.2a, 1.2b, and 1.2c where we use a rectangle to represent a VM and a circle to represent a switch. Let us use the number of hops between two VMs to denote the length of network path, and the maximum of hops(denoted as *Hop*) to describe the proximity of the VMs in the network. In Figure 1.2a,1.2b, and 1.2c, the Hop is 2, 4, and 6 respectively. This example shows that the virtual network structures can be very different even if we have only a small number of VMs.

An important question is how the current virtual network is being used by the user.



(a) Hop=2 (b) Hop=4 (c) Hop=6

Figure 1.2: Different virtual networks from the same physical network.

To answer this question, we need some mechanism to describe a virtual network by some metrics. The most important metric is the bandwidth. This dissertation proposes effective algorithms and tools to help users to measure the bandwidth of virtual networks.

1.2 Bandwidth: to describe a virtual network

Regarding bandwidth, my apartment, for example, uses 50 Mbps of xfinity Internet. The 50Mbps is the bandwidth of my Internet connection. This is how we describe a physical network, and it is the same for a virtual network.

To formally describe bandwidth, we usually use the term *capacity*. Capacity is defined as the maximum rate of data transmission that a network path allows a user to send. A links capacity is usually determined by the network cards and the cable connecting the two network cards. As shown in Figure 1.2c, a network path includes more than one link. The whole paths capacity is equal to the minimum capacity of the path, which is called a *bottleneck*.

However, a user usually cannot send data out at its capacity, because the network is shared among multiple users. Therefore, we also use another term *available bandwidth* to describe the network bandwidth. The available bandwidth is the remaining rate of a network path at which data can be sent. Available bandwidth is a dynamical metric used to

Table 1.1: impact of time measurement accuracy on bandwidth estimation

accurate bandwidth	1 Kbps	1 Mbps	1Gbps	10Gbps
time to transfer a 1500 Bytes packet	12s	12 ms	12μs	1.2μs
estimated bandwidth when time error is $\pm 1\mu$s	1 Kbps	1 Mbps	0.92$\sim$1.09Gbps	5.45$\sim$60Gbps

describe a network. The techniques used to measure capacity and available bandwidth of a network are called bandwidth estimation techniques.

There are a wealth of studies of bandwidth estimation techniques. Interested readers can refer to PathChar [16], TailGater [32], CapProbe [29], PBProbe [11], PBM [40], and PathRate [15] for capacity estimation. More techniques about available bandwidth estimation come from Pathload [23], Pathchirp [49], and the system-theoretic approach [35]. These techniques have been shown to fail in virtual networks because of one or more challenges in virtual networks as explained in the next section.

1.3 Challenges to bandwidth estimation in virtual networks

Virtual networks bring more challenges to bandwidth estimation techniques. We explain various challenges to bandwidth estimation in virtual networks below.

Time measurement accuracy: What is the impact of time measurement accuracy on bandwidth estimation? Data is send in packets in a network. Let the packet size be 1500 Bytes, which is the default maximum transmission unit for a network interface card (NIC). Table 1.1 shows the estimated bandwidth where the time measurement error is $\pm 1\mu$s. We can see that the accuracy of a microsecond unit has little or no impact on low-speed bandwidth, but it has a big impact on high-speed bandwidth such as 10 Gbps. A time measurement accuracy of nanosecond is a must for high-speed bandwidth estimation, but current computers cannot measure span of time at the level of a nanosecond. Hence, new methods

are needed to overcome this time measurement accuracy problem.

Hardware factors: Many new features have been added to NICs to improve NIC speeds. For example, *interrupt coalescence* (IC, also called *interrupt moderation*) [25, 44] is commonly used in high-speed NICs, and it reduces the CPU load by generating an interrupt for a group of packets instead of one for each packet. As a result, the packet time information, such as the time difference between two consecutive packets, is changed. Figure 1.3 shows the inter-arrival times of 200 packets from a 1Gbps NIC with and without IC, and we can see that inter-arrival times are considerably affected by IC. The inter-arrival times without IC are 12 μs as shown in Table 1.1, but the inter-arrival times with IC are either much smaller or greater than 12 μs.

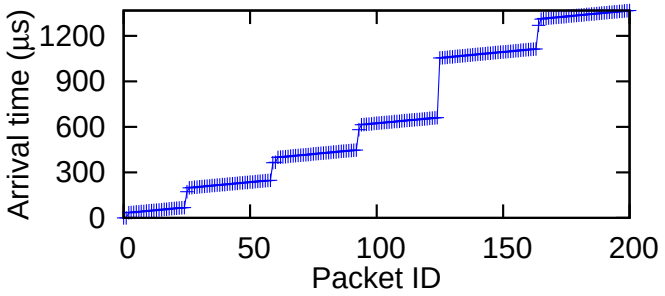


Figure 1.3: Impact of interrupt coalescence on packet inter-arrival times.

VM scheduling: VM scheduling [58, 12] is commonly used in cloud computing, and it enables multiple VMs to share the same pool of CPUs on a physical machine. However, it interferes with packet timestamping of VMs. For example, when a VM is not running, all packets arriving at the VM must wait until the VM is scheduled to run again. As a result, the packet delays and time differences measured by the VM may be drastically different from the actual values.

Rate limiters: Rate limiters are used in data centers to provide guaranteed bandwidth for data center users [8, 18, 24, 43, 50, 51]. A rate limiter works like a set of traffic lights, in which cars can only go through a crossing when the light is green. Likewise, a packet

can send only when it gets permission from the rate limiter. As an example, a Linux token bucket filter(*tbf*) allows packets to be sent at the network capacity when there are enough tokens. When the tokens have been consumed, packets are sent at the token rate. Another example is a Xen token bucket, which only allows a fixed number of packets or bytes to go through in a certain amount of time. The link remains idle when the tokens are consumed until the tokens are updated for the next period.

The challenges listed in this section are urgent; these problems need to be solved for current bandwidth estimation techniques. In this dissertation, we design and develop novel and effective bandwidth estimation tools for virtual networks.

1.4 Improvements to bandwidth estimation in virtual networks

The contribution of this dissertation consists mainly of two points. First we provide a deep understanding of bandwidth in virtual networks. Second, we design and develop novel and effective bandwidth estimation tools for virtual networks. In this section, we offer an overview of this dissertation. Interested readers can select specific chapters of concern based on this overview.

- Contribution 1: We have designed effective bandwidth estimation methods to estimate the bandwidth of networks with token bucket shapers, which are widely used in virtual networks. Bandwidth estimation for networks with token bucket shapers has been studied before, and the conclusion [33] is that ”both capacity and available bandwidth measurement are challenging because of the dichotomy between the raw link bandwidth and the token bucket rate”.

- Contribution 2: We have designed a fundamentally new path capacity comparison method to compare the capacities of two paths without using accurate time information. This method extends the design space of traditional time-based bandwidth

estimation methods by introducing a new class of sequence-based bandwidth comparison methods.

- Contribution 3: We conducted a large scale of measurement study of rate limiting in the three most popular clouds, Amazon EC2, Microsoft Azure, and Google Computer Engine. We found that different rate limiters are used in these clouds, and their traffic characteristics are different.

- Contribution 4: We have designed an available bandwidth estimation method for virtual networks based on the idea that the packet sequence is not affected by packet arrival time. In our contribution 2, we considered only the capacity of a path, and in this work, we have considered the available bandwidth of a path, another important bandwidth metric.

1.5 Organization

We describe our work on token bucket shaper measurement in Chapter 2. In Chapter 3, we present a tool to compare the capacities with packet sequence information. In Chapter 4, we offer the details of our study in EC2, Azure and Google clouds. In Chapter 5, we present our PacketTick tool which is used to measure available bandwidth for a virtual network. We present the conclusion and needs for future work in the last chapter.

2 Capacity and Token Rate Estimation for Networks with Token Bucket Shapers

2.1 Introduction

Cloud computing is transforming a large part of IT industry, as evidenced by the increasing popularity of public cloud computing services, such as Amazon Web Service [6], Google Cloud Platform [17], Microsoft Windows Azure [22], and Rackspace Public Cloud [46]. Many cloud computing applications are bandwidth-intensive, such as MapReduce applications and high performance computing applications, and thus the network bandwidth information of clouds is important for their tenants to manage and troubleshoot the application performance. For example, if the bandwidth information can be estimated, the application performance can be improved by appropriately placing application tasks on virtual machines [31].

Bandwidth estimation methods, such as *pathrate* [15], *capprobe* [29], *tailgater* [32], *pathload* [23], and *spruce* [53], have been successfully used to estimate the capacity and available bandwidth information of a network path in the traditional Internet. However, they face great challenges with clouds. One important reason is that traffic shapers are widely used as a basic building block for rate limiting in clouds, however, traffic shapers interfere with the probing packets of bandwidth estimation methods.

In this chapter, we study token bucket shapers that are a basic type of traffic shapers. Token bucket shapers have been widely used in virtualization software such as VMWare [55] and Xen [9], cloud computing platforms such as Amazon EC2 [6], large-scale virtual net-

works such as PlanetLab [52], software switches such as Open vSwitch [39] and OpenFlow Software Switch [48], and data center resource management schemes such as Seawall [51]. A popular type of token bucket shapers is the Token Bucket Filter (*tbf*) provided in Linux, which regulates traffic according to a token rate and a burst size. *tbf* is the building block of more advanced token bucket shapers, such as the Hierarchy Token Bucket (*htb*) in Linux that regulates traffic using multiple *tbf* shapers and allows token borrowing among different shapers. In this chapter, we focus on *tbf* and *tbf*-like shapers.

The *contribution* of this chapter is that we propose two methods to actively estimate the capacity and token rate, respectively, of a path in a network with potentially multiple *tbf* or *tbf*-like shapers. Specifically, we propose a method called *NarrowLinkCapacity* to estimate the capacity of a path, which is the slowest link capacity among all links in the path. It is an important property of the path, because it determines the average rate of a short train of packets. We also propose a method called *NarrowTokenRate* to estimate the token rate of a path, which is the slowest token rate among all token bucket shapers in the path. It is another important property of the path, because it determines the average rate of a long train of packets.

2.2 Related Work

Network bandwidth estimation methods can be classified into two categories: capacity estimation and available bandwidth estimation. Capacity estimation methods can be further classified into two classes: 1) methods to estimate the capacity of a path, such as *bprobe* [10] , *pathrate* [15], and *capprobe* [29], which mainly use the dispersion of two consecutive probing packets; and 2) methods to estimate the capacity of each individual link in a path, such as *pathchar* [16] and *tailgater* [32]. Available bandwidth estimation methods further fall into two classes: 1) Probe Gap Model (PGM), such as *spruce* [53] and *IGI/PTR* [20],

which use the initial and final time gap information of probing packets; and 2) Probe Rate Model (PRM), such as *pathload* [23] and *pathchirp* [49], which are based on self-induced congestion.

There are very few related works on bandwidth estimation in networks with token bucket shapers. Khandelwal *et al.* [30] study the accuracy of available bandwidth estimation methods, such as *pathload*, on Amazon EC2, and they find that the current methods are "*un-suitable for bandwidth estimation in data center networks*". Lakshminarayanan *et al.* [33] measure the impact of token bucket shapers on bandwidth estimation methods in broadband access networks, and they conclude that "*both capacity and available bandwidth measurement are challenging because of the dichotomy between the raw link bandwidth and the token bucket rate*".

The closest work is *shaperprobe* developed by Kanuparthy and Dovrolis to measure the token bucket characteristics as a traffic shaping service in residential ISP networks [28]. *shaperprobe* detects a level shift in the measured rates, and estimates the token rate as the median rate after the level shift. Our work is different from *shaperprobe* in that we consider general networks whereas *shaperprobe* mainly considers residential ISP networks. There are two important differences between token bucket shapers in general networks and in residential ISP networks. First, token bucket shapers in residential ISP networks usually have bigger burst sizes (e.g., 5-10 MBytes), and as a result the path capacity can be estimated using existing capacity estimation methods. Second, there is no or very little background traffic competing with the probing packets at a token bucket shaper in residential ISP networks, and thus the token rate can be relatively easily estimated.

Table 2.1: Notation used in the chapter

Symbol	Description
c	The capacity of a link or a token bucket shaper
r	The token rate of a token bucket shaper
s	The size of a packet
C	The capacity of a path
R	The token rate of a path
λ	The sending rate of packets by the sender
u	The dispersion rate of two consecutive packets
u_t	The average rate of a train of packets

2.3 Networks with Token Bucket Shapers

In this section, we introduce *tbf* and *tbf*-like shapers, and discuss their impact on the dispersions of a train of packets. The important notation used in this chapter is summarized in Table 2.1.

2.3.1 tbf and tbf-like Shapers

Figure 2.1 illustrates a *tbf* or *tbf*-like shaper. It consists of two buffers: a token bucket and a packet buffer. Tokens are generated and placed into the token bucket at a rate of r bits per second (bps). The token bucket can hold up to σ bits of tokens, and any newly generated token will be discarded if the token bucket is full. When a packet of size s bits arrives at the shaper, if there are at least s bits of tokens available, the packet is immediately transmitted to the outgoing link at its capacity c bps and at the same time s bits of tokens are consumed from the token bucket. Otherwise, the packet will be queued in the packet buffer until there are at least s bits of tokens available.

A *tbf* or *tbf*-like shaper is described by four parameters (r, σ, c, b): 1) token rate r bps,

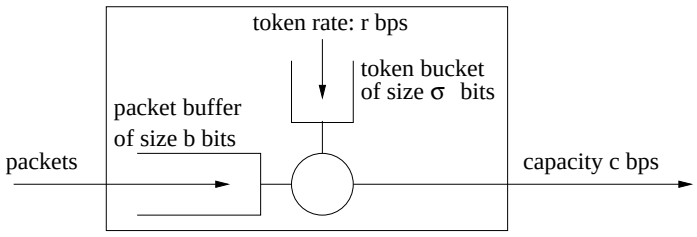


Figure 2.1: A *tbf* or *tbf*-like shaper with four parameters (r, σ, c, b).

2) burst size σ bits (also called token bucket size), 3) capacity c bps, and 4) packet buffer size b bits. Note that, $r \leq c$. In addition, $\sigma \geq \sigma_{min} = MTU$, and this ensures that a packet with the maximum transmission unit (MTU) size can pass though a token bucket. Since the typical MTU is 1500 bytes, we have $\sigma_{min} = 1500 \times 8$ bits. For example, we observe that a PlanetLab [52] node sets its burst size σ to 1600×8 bits.

Considering a case where the token bucket is full, and a train of packets each of size s bits arriving at the shaper at rate $\lambda > r$. In this case, the first K packets will be transmitted at rate $\min(c, \lambda)$, where K is given in Equation (2.1). Note that $K \geq \sigma/s$, because new tokens are being generated during the transmission of the first σ/s packets. After the first K packets, the token bucket becomes empty, and thus the remaining packets will be throttled by token rate r.

$$K = \begin{cases} \lfloor (\sigma/s - r/c)/(1 - r/c) \rfloor, & \text{if } \lambda \geq c \\ \lfloor (\sigma/s - r/\lambda)/(1 - r/\lambda) \rfloor, & \text{if } r < \lambda < c \end{cases} \tag{2.1}$$

2.3.2 Impact on Packet Dispersions

We use the following two simple networks to show the impact of token bucket shapers on a train of packets.

- *Network 1*: A one-hop network without any shaper. The link capacity between the sender and the receiver is c.

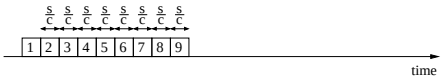


Figure 2.2: In network 1, the dispersion between any two consecutive packet is always s/c.

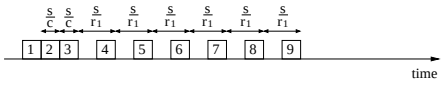


Figure 2.3: In network 2, the first few dispersions are s/c, but the remaining dispersions become s/r_1 due to the regulation of the token bucket shaper.

- *Network 2*: A one-hop network with one token bucket shaper at the sender. The shaper has the following four parameters: $r_1 = c/2$ bps, $\sigma_1 = 2 \times MTU$, c bps, and $b = \infty$ bits. Initially, the token bucket is full.

Let's consider a case when a train of 9 packets are sent at rate $\lambda = c$ and there is no cross traffic. We assume that every packet has the same size of $s = MTU$. We are interested in the *dispersion* between two consecutive packets at the receiver, which is the arrival time difference between their last bits.

- In network 1: The dispersion between any two consecutive packet is always the same, and depends on the link capacity. Figure 2.2 shows the arrival time of each packet at the receiver, and the dispersion is always s/c.

- In network 2: The first two dispersions are still s/c, but all other dispersions become $s/r_1 = 2s/c$ due to the regulation of the shaper as shown in Figure 2.3. Note that the first $K = 3$ packets (obtained using Equation (2.1)) are transmitted back-to-back at capacity c.

These examples show that the dispersion between two consecutive packets depends on the link capacity and the token rates. We have the following observations:

- *Observation 1*: For a short train of packets, the majority of dispersions depend on the link capacity. For example, the first two dispersions in both networks depend on

c. Therefore, *the average rate of a short train of packets highly depends on the link capacity in networks with and without token bucket shapers*.

- *Observation 2*: For a long train of packets, the majority of dispersions depend on the token rate. For example, the last several dispersions in network 2 depend on r_1. Therefore, *the average rate of a long train of packets highly depends on the token rate in a network with token bucket shapers*. Note that, the average rate of a long train of packets in a network without any shaper still depends on the link capacity.

2.4 Design Goals and Challenges

2.4.1 Goals

In this chapter, we design two methods, called *NarrowLinkCapacity* and *NarrowTokenRate*, to actively estimate the capacity and the token rate of a path, respectively, in a multi-hop network with possibly multiple *tbf* or *tbf*-like shapers. Let's consider a path with n links of capacities $c_1, c_2, ..., c_n$ and with m token bucket shapers of token rates $r_1, r_2, ..., r_m$.

- *Capacity C of the path* is defined as the capacity of the *narrow link* of the path, which is the link with the minimum capacity among all links in the path.

$$C = \min_{i=1,...,n} c_i \tag{2.2}$$

- *Token rate R of the path* is defined as the token rate of the *narrow token bucket shaper* on the path, which is the token bucket shaper with the minimum token rate among all token bucket shapers on the path.

$$R = \min_{i=1,...,m} r_i \tag{2.3}$$

If there are multiple links with capacity C, the narrow link is the last one among them. If there are multiple shapers with token rate R, the narrow shaper is the last one among them.

2.4.2 Challenges

There are two major factors that make it challenging to estimate the capacity C and the token rate R of a path. 1) The first factor is the token bucket shapers, each of which regulate the probing traffic depending the availability of their tokens, and greatly complicates the analysis of probing packets. 2) The second factor is random cross traffic, which interferes with the probing packets and changes their dispersions. The packet dispersions could be enlarged or reduced due to cross traffic, and it is hard to distinguish C or R from the noises caused by cross traffic.

2.4.3 Definitions

The dispersion rate histogram $\mathcal{U}$ of a path is commonly used by bandwidth estimation methods [15, 29, 53, 20], because it carries lots of useful information of the path. Both *NarrowLinkCapacity* and *NarrowTokenRate* use $\mathcal{U}$. Below we define dispersion rate histogram $\mathcal{U}$ and some related terms.

Three types of probing traffic are commonly used by bandwidth estimation methods: a single packet, a *packet pair* (two packets), and a *packet train* (more than two packets). In this chapter, we consider only packet pairs and trains, in which all packets have the same size s and are uniformly spaced. The *sending rate* λ of a packet pair or train can be adjusted by controlling the inter-packet gaps at its sender.

The *dispersion* d *of a packet pair* at a receiver as illustrated in Figure 2.4 is the arrival time difference between these two packets at the receiver. The *dispersion rate* u is the ratio

of the packet size s to their dispersion d (i.e., $u = s/d$). We denote the histogram of the dispersion rates of multiple packet pairs by $\mathcal{U}$.

The *pair-wise dispersions (or dispersions for short) of a packet train* at a receiver as illustrated in Figure 2.5 are the dispersions of all pairs of two consecutive packets of the train at the receiver. The *pair-wise dispersion rates (or dispersion rates for short) of a packet train* are the dispersion rates of all pairs of two consecutive packets of the train. By overloading the symbol, let $\mathcal{U}$ also denote the histogram of the dispersion rates of a packet train. The *average arrival rate* u_t of a train with l packets at the receiver is the ratio of $(l-1)s$ to the arrival time difference between the first and the last packets.

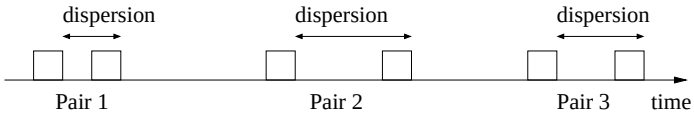


Figure 2.4: Dispersions of multiple packet pairs received by a receiver.

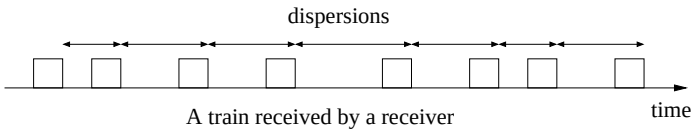


Figure 2.5: Dispersions of a train of packets received by a receiver.

Figure 2.6 illustrates a possible $\mathcal{U}$. Due to the factors explained in Section 2.4.2, $\mathcal{U}$ has multiple modes. A *mode* is a local maximum (i.e., peak). The *strength of a mode* is the frequency or density of the mode (i.e., the height of the peak). Note that, C or R may not be the strongest mode in a $\mathcal{U}$, and sometimes is not even a mode. Thus, it is challenging to estimate C and R.

2.5 Capacity Estimation

In this section, we explain how we design *NarrowLinkCapacity* to estimate the capacity C of a path, which is the capacity of the narrow link on the path. We first demonstrate why it

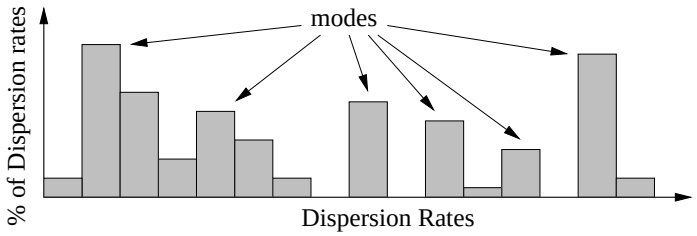


Figure 2.6: Dispersion rate histogram $\mathcal{U}$ is multimodal.

is challenging to estimate C, then describe how we tackle the challenge, and finally present our capacity estimation method.

To estimate the capacity of a path from a sender to a receiver, *NarrowLinkCapacity* sends many packet pairs, and then estimates the path capacity using the dispersion rate histogram $\mathcal{U}$ of the packet pairs. There are two reasons why using packet pairs instead of packet trains. First, a packet pair is less likely to be throttled by the path token rate than a packet train. Second, it has been shown [15] that packet trains are more sensitive to cross traffic than packet pairs.

2.5.1 Simulation setup

We use NS2 simulations to explain how our two methods work in this and next sections, and we will present testbed and Amazon EC2 results in the evaluation section. We consider a multi-hop network with multiple token bucket shapers shown in Figure 2.7. It has 5 links, each with capacity $c_1 = 1200$ Mbps, $c_2 = 1000$ Mbps, $c_3 = 400$ Mbps, $c_4 = 600$ Mbps, and $c_5 = 800$ Mbps, respectively. Therefore, the path capacity C is $c_3 = 400$ Mbps. It has 5 shapers, one for each link, each shaper with token rate $r_1 = 100$ Mbps, $r_2 = 900$ Mbps, $r_3 = 300$ Mbps, $r_4 = 500$ Mbps, and $r_5 = 700$ Mbps, respectively. Therefore, the path token rate R is $r_1 = 100$ Mbps.

All shapers have the same burst size of 1500 bytes, unless otherwise noted. All shapers and all routers have sufficiently large packet buffer so that no packets will be lost. Cross

traffic is generated on each link by multiple Pareto traffic generators with shape parameter 1.9. The packet size of cross traffic is 1500 bytes. Unless otherwise noted, the total amount of cross traffic on each link is set to 50% of the token rate on the link. To simplify the description of the simulation (but without reducing the estimation difficulty), all packets including probing packet and cross packets on a link go through the token bucket shaper on the link.

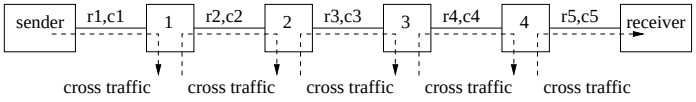


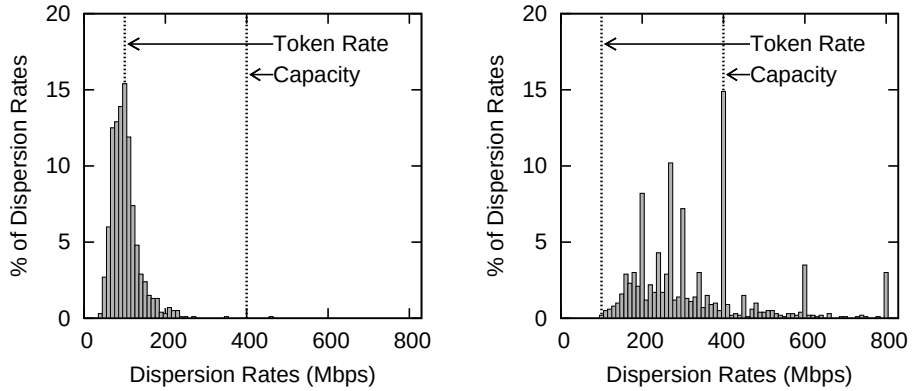
Figure 2.7: A multi-hop network with multiple token bucket shapers.

2.5.2 Why it is challenging?

It is challenging to estimate the capacity C of a path, because C may not be the strongest mode and sometime is not even a mode in the dispersion rate histogram $\mathcal{U}$ due to token bucket shapers and random cross traffic.

Impact of token bucket shapers Figure 2.8 shows two dispersion rate histograms, each of which is obtained using 1000 packet pairs with a bin width of 10 Mbps. Every pair of probing packets are sent at the maximum sending rate (i.e., back-to-back), and the probing packet size is 1500 bytes. We set the burst size of every shaper to 1500 bytes and 150000 bytes, respectively, for the left and right histograms.

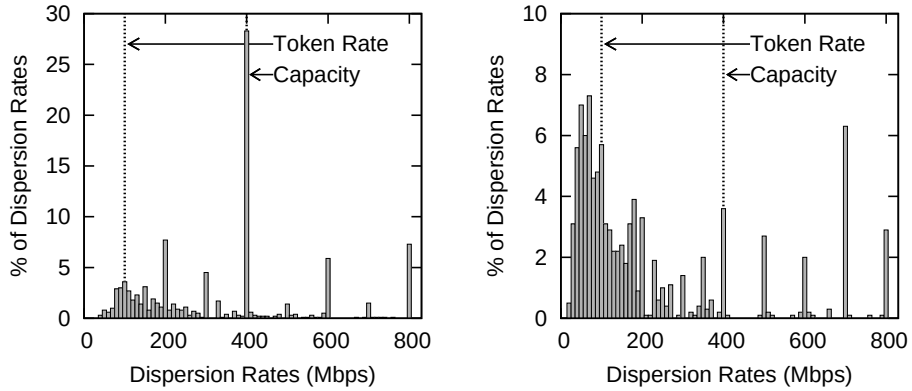
When the probing packet size is the same as the burst size (i.e., 1500 bytes in Histogram 2.8(a)), a single packet can empty the token bucket. As a result, all packet pairs are throttled by the path token rate 100Mbps. When the probing packet size is much smaller than the burst size (e.g., 150000 bytes in Histogram 2.8(b)), many packet pairs can pass the shapers without any delay and are throttled only by the path capacity 400Mbps. We can see that *in order to estimate the path capacity, the probing packet size should be much smaller than the token burst size*.



(a) Burst = 1500 bytes (b) Burst = 150000 bytes

Figure 2.8: The impact of token burst sizes. Cross traffic=50%, Sending rate=max, Probing packet=1500 bytes.

Impact of cross traffic Figure 2.9 shows the impact of cross traffic. We set the amount of cross traffic on each link to 30% and 70% of the token rate on the link, respectively, for the left and right histograms. The probing packet size is 500 bytes, which is much smaller than the burst size 1500 bytes. We observe that *the path capacity with light cross traffic is the strongest mode and easy to estimate, but the path capacity with heavy cross traffic may not be the strongest mode and thus hard to estimate.*



(a) Cross traffic = 30% (b) Cross traffic = 70%

Figure 2.9: The impact of cross traffic. Burst=1500 bytes, Sending rate=max, Probing packet=500 bytes.

There are some dispersion rates spread to the left of $C = 400$ Mbps in both histograms.

They are caused when the second packet of a packet pair is delayed due to cross traffic at a link or shaper. There are also some dispersion rates concentrated at several modes to the right of C in both histograms, such as at 500, 600, 700, and 800 Mbps, which correspond to the link capacities (600 and 800 Mbps) and token rates (500 and 700 Mbps) on the path after the narrow link with c_3 = 400 Mbps. They are caused when the first packet of a packet pair is delayed due to cross traffic at the links and shapers on the path after the narrow link.

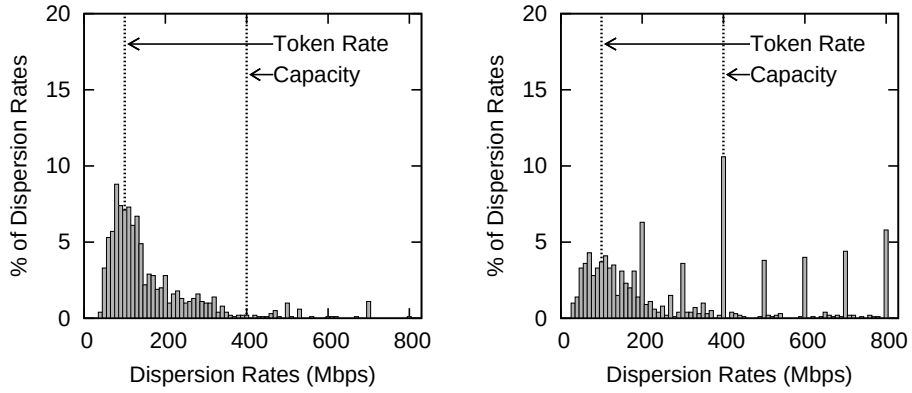
2.5.3 How to tackle the challenge?

To estimate the capacity of a path, we make the path capacity a strong mode in $\mathcal{U}$ by reducing the probing packet size, and then distinguish it from other modes by adjusting the sending rate of packet pairs.

Making the path capacity C a strong mode by reducing the probing packet size There are two reasons why a smaller probing packet size leads to more dispersion rates at C: First, for a packet pair, the smaller their packet size, the less their probability to be throttled by token bucket shapers; Second, consider a packet pair whose dispersion rate becomes C after passing the narrow link. The smaller their packet size, the less their probability to be interfered by cross traffic between the narrow link and the receiver.

This is demonstrated in Figure 2.10. For example, as the probing packet size reduces from 1000 bytes to 500 bytes, the percentage of dispersion rates at $C = 400$ Mbps increases from 0.2% to 11%. However, in order to mitigate the impact of system processing overhead and link-layer frame headers [15], the probing packet size should not be too small. *NarrowLinkCapacity* sets the packet size s to 500 bytes.

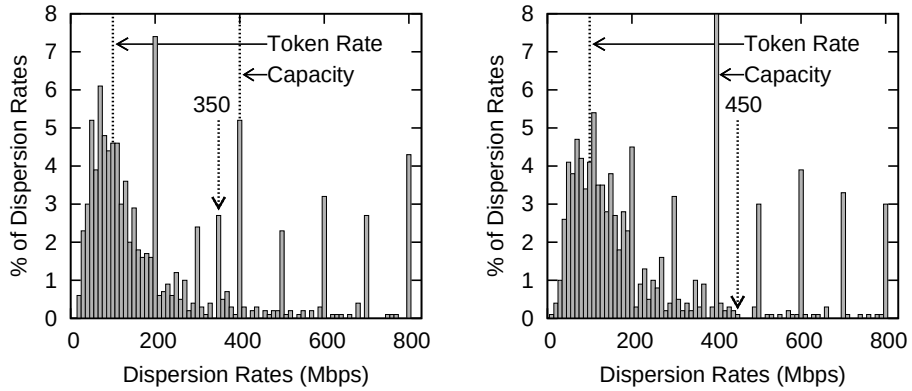
Distinguishing the path capacity mode from other modes by adjusting the sending rate This step is necessary because the path capacity C may not be the strongest mode (e.g., in Histogram 2.9(b)). It is based on the fact that a packet pair with a sending rate $\lambda < C$ is likely to maintain its rate all the way to the receiver, whereas a packet pair with



(a) Packet size=1000 bytes (b) Packet size=500 bytes

Figure 2.10: The impact of probing packet sizes. Cross traffic=50%, Burst=1500 bytes, Sending rate=max.

$\lambda > C$ will definitely be throttled by the narrow link.



(a) Sending rate = 350Mbps (b) Sending rate = 450Mbps

Figure 2.11: The impact of sending rates. Cross traffic=50%, Burst=1500 bytes, Probing packet=500 bytes.

This is demonstrated in Figure 2.11. For example, 350 Mbps is not a mode in Histogram 2.10(b), but it turns to a mode in Histogram 2.11(a), which is obtained using sending rate 350Mbps that is slower than C. As another example, 450 Mbps is not a mode in Histogram 2.10(b), and it is still not a mode in Histogram 2.11(b), which is obtained using sending rate 450 Mbps that is faster than C.

2.5.4 NarrowLinkCapacity: A capacity estimation method

NarrowLinkCapacity has the following four steps.

- Step 1: Obtain histogram $\mathcal{U}$ by sending $L_c = 1000$ packet pairs of size $s = 500$ bytes. Every pair of packets are sent back-to-back, and the inter-pair gap g is 0.02 second.
- Step 2: Identify all local modes in $\mathcal{U}$ in the increasing order: $u_1, u_2, u_3, ...$
- Step 3: Use the binary search algorithm to find a mode u_i, such that
  - Condition 1: Rate $\lambda_1 = (u_{i-1} + u_i)/2$ turns to a mode in the dispersion rate histogram obtained by sending L_c packet pairs at rate λ_1. We do not test this condition for the first mode.
  - Condition 2: Rate $\lambda_2 = (u_i + u_{i+1})/2$ is still not a mode in the dispersion rate histogram obtained by sending L_c packet pairs at rate λ_2. We do not test this condition for the last mode.
- Step 4: The mode u_i is the estimated path capacity.

Implementation remark: In some high-speed or virtual networks, it might be hard to precisely control and measure the rate of a packet pair. In this case, we can replace a packet pair with a very short packet train (like with 3-5 packets). For condition 1 at step 3, we can check whether there is a new mode between u_{i-1} and u_i instead of checking whether $(u_{i-1} + u_i)/2$ is a new mode, and similarly for condition 2.

2.6 Token Rate Estimation

In this section, we explain how we design *NarrowTokenRate* to estimate the token rate R of a path, which is the token rate of the narrow token bucket shaper on the path. We first

demonstrate why it is challenging to estimate R, then describe how we tackle the challenge, and finally present our token rate estimation method.

NarrowTokenRate uses long packet trains, because they are more likely to drain the token bucket of the narrow shaper. Specifically, *NarrowTokenRate* sends two packet trains from a sender to a receiver: a drain train and an estimation train.

First, it sends a *drain train* with L_{dt} packets of size MTU at rate λ_{dt}, in order to drain the token bucket of the narrow shaper. The train size L_{dt} and the sending rate λ_{dt} should be long and high enough to completely drain the tokens, but the sending rate λ_{dt} should not be too high to overflow the network. For all the simulations in this chapter, we set L_{dt} to 1000 packets. *NarrowTokenRate* sets λ_{dt} to αC with $\alpha = 7/8$, which is slightly less than C to avoid network congestion.

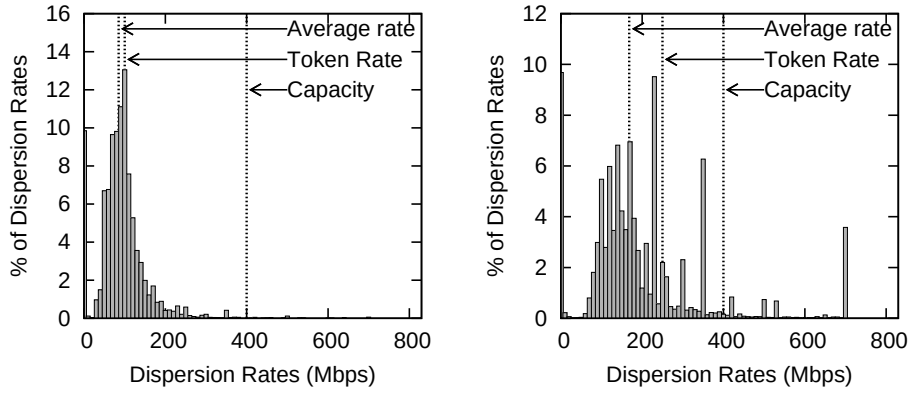
Next, it sends an *estimation train* with L_{et} packets of size s at rate λ_{et}, in order to measure the dispersion rate histogram $\mathcal{U}$, and then find the path token rate R. We set L_{et} to 1000 packets. Below, we discuss how to set the other two parameters s and λ_{et}, in order to accurately find R from $\mathcal{U}$.

2.6.1 Why it is challenging?

It is challenging to estimate the token rate R of a path, because R may not be the strongest mode and sometimes is not even a mode in $\mathcal{U}$ due to token bucket shapers and random cross traffic.

Impact of token bucket shapers Figure 2.12 shows the dispersion rate histograms when we set the token rate r_1 of the network in Figure 2.7 to 100 and 250 Mbps, respectively, for the left and right histograms. In both cases, r_1 is still the token rate R of the path, and the sending rate of the estimation train λ_{et} is set to $\alpha C = 350$ Mbps. We notice that R *is not always the strongest mode*, for example in Histogram 2.12(b).

Figure 2.12 also shows the average arrival rate u_{et} of the estimation train, which is

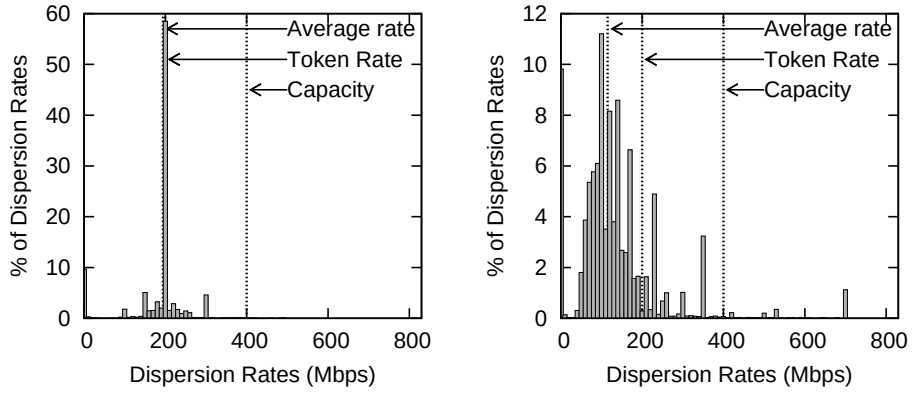


(a) Token rate r_1=100 Mbps (b) Token rate r_1=250 Mbps

Figure 2.12: The impact of path token rates. Cross traffic=50%, Burst=1500 bytes, Sending rate λ_{et}=350 Mbps, Probing packet s=1500 bytes.

defined in Section 2.4.3, and is labelled "average rate" in the figures. We can see that *average arrival rate u_{et} is not an accurate estimate of the path token rate R in general, instead it is a lower bound of R.* Limited by the space, below we only give an intuitive proof: On a path without any shapers, it has been proved [15] that the average arrival rate of a long train is the lower bound of the capacity C of the narrow link due to cross traffic. On a path with shapers, the drain train has drained the token bucket of the narrow shaper, the narrow shaper instead of the narrow link actually throttles the estimation train, and thus u_{et} is a lower bound of R.

Impact of cross traffic Figure 2.13 shows the dispersion rate histograms when we set the amount of cross traffic on each link to 5% and 70% of the token rate on the link. We can see that R is the strongest mode when the cross traffic load is low as in Histogram 2.13(a), but *is not the strongest mode and actually is not even a mode when the cross traffic load is high* as in Histogram 2.13(b).



(a) Cross traffic=5% (b) Cross traffic=70%

Figure 2.13: The impact of cross traffic. Token rate r_1=200 Mbps, Burst=1500 bytes, Sending rate λ_{et}=350 Mbps, Probing packet s=1500 bytes.

2.6.2 How to tackle the challenges?

To estimate the token rate R of a path, we first find a lower bound and an upper bound for R, then make R a strong mode between the two bounds by adjusting the probing packet size s and sending rate λ_{et} of the estimation train.

Finding a lower bound and an upper bound As discussed in the previous subsection, the average arrival rate u_{et} of an estimation train is a lower bound of R. The sending rate λ_{dt} of the drain train is an upper bound of R. Thus, we have $R \in [u_{et}, \lambda_{dt}]$.

Making the path token rate a strong mode We use two techniques to make R a strong mode: *First*, choose a small packet size s for the estimation train. Because the token bucket of the narrow shaper has already been drained by the drain train, all packets of the estimation trains are throttled by the narrow shaper. For these estimation train packets, the smaller their packet size s, the less their probability to be interfered by cross traffic, and thus the more the number of packets maintaining rate R all the way to the receiver. *Second*, choose an appropriate sending rate λ_{et} for the estimation train. Rate λ_{et} should be sufficiently high to keep draining the token bucket, but should not be too high to cause severer congestion that also interferes with the estimation.

The impact of packet size s is demonstrated in Figure 2.14, which shows that *a smaller packet size s leads to a higher percentage of dispersion rates at R* (e.g., from 1.4% to 3.5% when s reduces from 1500 bytes to 500 bytes). However, in order to mitigate the impact of system processing overhead and link-layer frame headers [15], s should not be too small. Therefore, *NarrowTokenRate* sets s to 500 bytes.

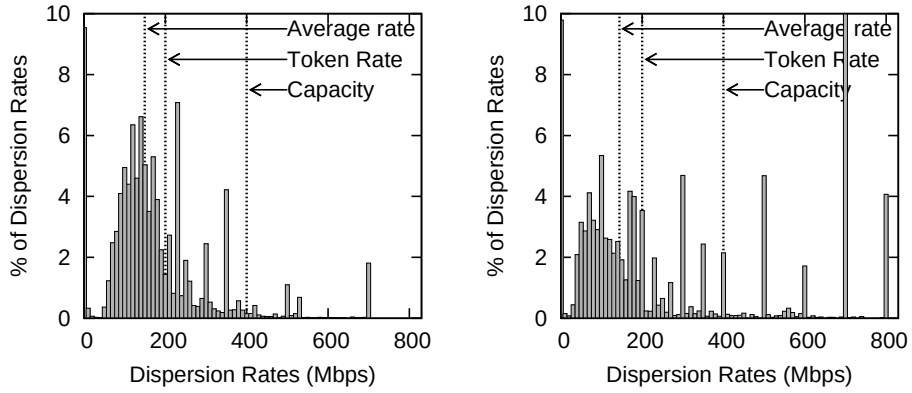
The impact of sending rate λ_{et} is demonstrated in Figure 2.15. By comparing Histogram 2.14(b) with Figure 2.15, we can see that *a sending rate λ_{et} closer to R leads to a higher percentage of dispersion rates at R* (e.g., 3.5% when $\lambda_{et} = 350$ Mbps in Histogram 2.14(b), 5.3% when $\lambda_{et} = R = 200$ Mbps in Histogram 2.15(a), and 3.4% when $\lambda_{et} = 100$ Mbps in Histogram 2.15(b)). This observation is further validated by Figure 2.16 that shows the percentage of dispersion rates at R as the sending rate λ_{et} varies between 20 and 350 Mbps. Without knowing token rate R in advance, *NarrowTokenRate* sets the estimation train sending rate λ_{et} to $(\lambda_{dt} + u_{dt})/2$, which is the average of the drain train sending rate λ_{dt} and the drain train arrival rate u_{dt}, and is used as an initial estimate of R.

Finding the path token rate R from $\mathcal{U}$ Once we get the dispersion rate histogram $\mathcal{U}$ of the estimation train, we estimate R by the strongest mode between lower bound u_{et} and upper bound λ_{dt} in $\mathcal{U}$. For example, R is the strongest mode between the two bounds in Histogram 2.15(a), although it is not the overall strongest mode (i.e., 800 Mbps). Also note that there is another strong mode (i.e., 300 Mbps) between the two bounds in Histogram 2.15(a), which corresponds to the token bucket shaper with $r_3 = 300$ Mbps.

2.6.3 NarrowTokenRate: A token rate estimation method

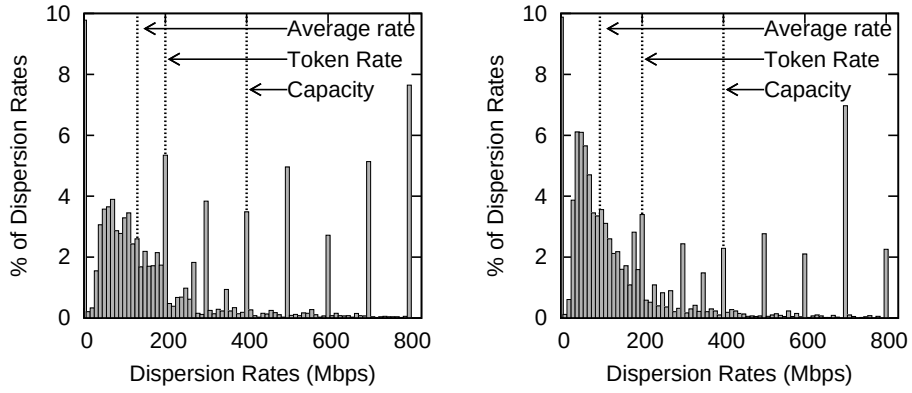
NarrowTokenRate has the following five steps.

- Step 1: Send the drain train at sending rate $\lambda_{dt} = \alpha C$ with $\alpha = 7/8$. The drain train consists of L_{dt} probing packets of size MTU.



(a) Packet size s=1500 bytes (b) Packet size s=500 bytes

Figure 2.14: The impact of probing packet size s. Token rate r_1=200 Mbps, Cross traffic=50%, Burst=1500 bytes, Sending rate λ_{et}=350 Mbps.



(a) Sending rate λ_{et}=200 Mbps (b) Sending rate λ_{et}=100 Mbps

Figure 2.15: The impact of sending rate λ_{et}. Token rate r_1=200 Mbps, Cross traffic=50%, Burst=1500 bytes, Probing packet s=500 bytes.

- Step 2: Measure the average arrival rate u_{dt} of the drain train at the receiver.
- Step 3: Send the estimation train at sending rate $\lambda_{et} = (\lambda_{dt} + u_{dt})/2$. The estimation train consists of $L_{et} = 1000$ packets of size s = 500 bytes.
- Step 4: Measure the average arrival rate u_{et} of the estimation train at the receiver, and measure the dispersion rate histogram $\mathcal{U}$ of the estimation train at the receiver.
- Step 5: The path token rate R is estimated by the strongest mode between u_{et} and

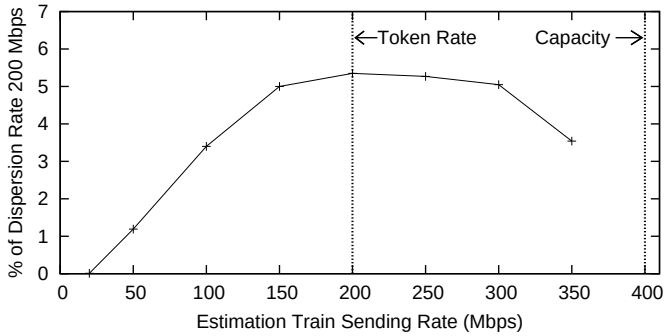


Figure 2.16: The impact of sending rate λ_{et} on the percentage of R in $\mathcal{U}$. Token rate r_1=200 Mbps, Cross traffic=50%, Burst=1500 bytes, Probing packet s=500 bytes.

λ_{dt} in $\mathcal{U}$. In case of multiple strongest modes (e.g., corresponding to multiple token rates), choose the slowest one (i.e., the narrow shaper).

NarrowTokenRate has one parameters: L_{dt} is the number of packets in the drain train. A user should specify the value of L_{dt} depending on how much probing traffic the user can afford. The larger the L_{dt}, the larger the burst size *NarrowTokenRate* can work with. Specifically, L_{dt} should be greater than K given in Equation (2.1).

Discuss: *shaperprobe* [28] is designed for detecting a token bucket shaper in a residential ISP network, where there is no or very little cross traffic competing with the probing packets at a token bucket shaper. Thus, it is mainly based on the average train arrival rate, which however is sensitive to the cross traffic in a general network.

2.7 Evaluation

In this section, we evaluate the accuracy of *NarrowLinkCapacity* and *NarrowTokenRate* using our lab test-bed, Amazon EC2 [6], and NS2 simulations. We use their default parameter values and set $L_{dt} = 1000$ packets for *NarrowTokenRate*, unless otherwise noted.

We compare our methods with two other methods. 1) *pathrate* [15] that is one of the most accurate and tested capacity estimation methods. In the experiments, we use

the original *pathrate* code released by the authors. 2) *shaperprobe* [28] that is designed to detect the token rate of a residential ISP network. Without its source code, we re-implement it by following the algorithm described in the *shaperprobe* paper [28].

2.7.1 Test-bed Results

The test-bed topology is similar to Figure 2.7, but with one more link. Between the sender and the receiver, there are a total of five switches, including two gigabit switches, one 10-gigabit switch, and two switches emulated using Dell servers. Among the six links, two links have capacity 10 Gbps, and all others have capacity 1 Gbps. Therefore, the path capacity C is 1 Gbps. There is one token bucket shaper created using a Linux Token Bucket Filter (tbf) on an emulated switch, and therefore its token rate is the path token rate R. There are several other Dell servers to generate random Poisson cross traffic over each link using MGEN [1]. The interrupt coalescing feature of every network card is turned off to improve the packet timestamp accuracy.

Capacity experiments: In this group of experiments, we evaluate the accuracy of *NarrowLinkCapacity* and *pathrate*. We vary the token rate R from 200, to 400 and 600 Mbps, and we also vary the burst size from 1.6, to 10 and 100 Kbytes. Figure 2.17 shows the experiment results. We can see that *NarrowLinkCapacity* can accurately estimate the path capacity (i.e., 1 Gbps) for all tested token rates and burst sizes. However, *pathrate* cannot correctly estimate the path capacity, and its results depend on both the token rates and burst sizes. Especially when the burst size is small, *pathrate* actually estimates the token rate instead of the capacity.

There are several reasons that *pathrate* does not work well in networks with shapers. First, it varies the packet size between 550 and 1500 bytes. However, packet pairs of large packet sizes are more likely to be throttled by the path token rate instead of the path capacity, when the burst size is small. Second, it uses packet trains for quick estimation.

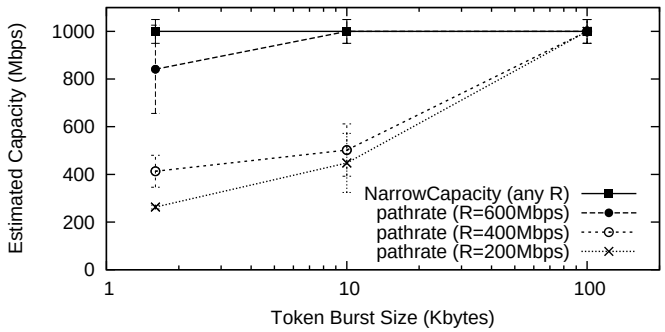


Figure 2.17: Capacity results estimated by *NarrowLinkCapacity* and *pathrate*.

However, packet trains are more likely to be throttled by the path token rate. Third, it uses the average arrival rate of a long packet train as a lower bound when selecting the path capacity. However, due to the regulation of the shapers, the average arrival rate of a long packet train could be lower the path token rate, and then much lower than the path capacity. As a result, it is not a good lower bound for the path capacity.

Token rate experiments: In this group of experiments, we evaluate the accuracy of *NarrowTokenRate* and *shaperprobe*. We still vary the token rate R from 200, to 400 and 600 Mbps, and we also vary the percentage of cross traffic on each link from 10% to 50%. Figure 2.18 shows the experiment results. We can see that *NarrowTokenRate* can accurately estimate the token rate R for all tested token rates and cross traffic. However, the result estimated by *shaperprobe* is only a lower bound of the token rate, and is sensitive to the cross traffic. This is because *shaperprobe* is mainly designed to detect a token bucket shaper in a residential ISP network, where there is no or very little cross traffic competing with the probing packets at a token bucket shaper.

2.7.2 Amazon EC2 Results

We evaluate *NarrowLinkCapacity* and *NarrowTokenRate* using virtual machines on Amazon Elastic Compute Cloud (EC2) [6], which is a very popular public cloud computing platform. The EC2 facilities are located at multiple locations, and we choose the one in the US

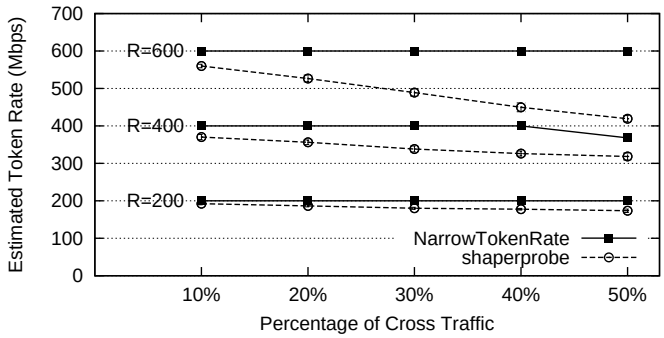


Figure 2.18: Token rate results estimated by *NarrowTokenRate* and *shaperprobe*.

West (Oregon) region. EC2 provides different types of virtual machines, called instances, with different computing and networking capacity. We select a micro instance (t1.micro), a small instance (m1.small), and a medium instance (m3.medium) as three senders, and select an xlarge instance (m3.xlarge) as the receiver to make sure that the receiver is not the bottleneck. We are interested in whether EC2 traffic is throttled by traffic shapers, and if so what the path capacity and the path token rate from each sender to the receiver are.

Figure 2.19 shows the experiment results. From the big differences between *NarrowLinkCapacity* and *NarrowTokenRate* results, it is clear that the traffic from all three senders are throttled by traffic shapers. The capacity of all three senders is 1 Gbps, and this is possibly because they use Gigabit network cards. They have slightly different token rates, in the increasing order of micro, small, and medium instances. This is consistent with their computing capacity ordering specified by EC2 [6].

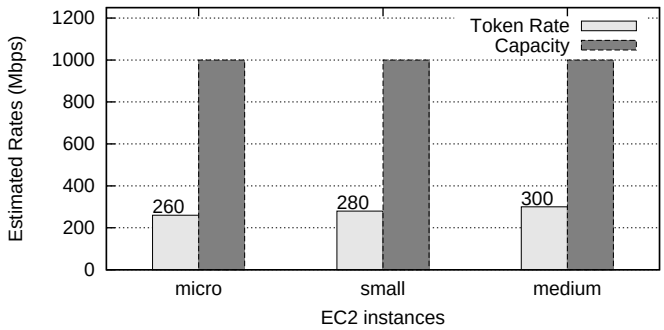


Figure 2.19: The path capacity and token rate of EC2 instances.

Since the detailed network capability of each instance is not provided by EC2, we also run *iperf* on these machines in order to verify our results to some extent. We run *iperf* on each sender for 10 seconds to get a stable throughput, and then calculate the average rate. We find that the *iperf* results are close to the token rate estimated by *NarrowTokenRate*. Note that, *NarrowTokenRate* reports more consistent results than *iperf*, and requires much less traffic than *iperf*.

We also evaluate *pathrate* and *shaperprobe* on EC2. *pathrate* does not work and it reports that interrupt coalescing is detected and there is an insufficient number of packet dispersion estimates. This is possibly due to interference of interrupt coalescing and traffic shapers. *shaperprobe* reports very similar results as *NarrowTokenRate*, and this is because we do not generate any cross traffic at the token bucket shaper in these experiments.

Finally, we notice that the same type of EC2 instances in different zones or regions may have different networking capability, such as different token rates (usually slightly but sometime significantly different), and we are interested in conducting a comprehensive measurement of EC2 instances using our tools in the future.

2.7.3 Simulation Results

We also comprehensively evaluate our methods using NS-2 for much more network topologies and parameters. Limited by the space, we present only some of our simulation results.

We simulate a network as illustrated in Figure 2.7 with 5 links of capacities 2, 1, 2, 1, and 2 Gbps, respectively. Therefore, the path capacity is 1 Gbps. The cross traffic of a link is set to 20% of the link capacity if the link does not have a shaper, or of the token rate if the link has a shaper.

Impact of the location of a token bucket shaper: We add just one token bucket shaper into the network, but at different locations: at the sender, router 1, router 2, router 3, and router 4. If the shaper is added at the sender, it regulates all packets on the link between

the sender and router 1. The token rate is always 600 Mbps, and the burst size is 2000 bytes. Figure 2.20 shows that our methods can accurately estimate the path capacity and path token rate in all cases independent of the shaper location.

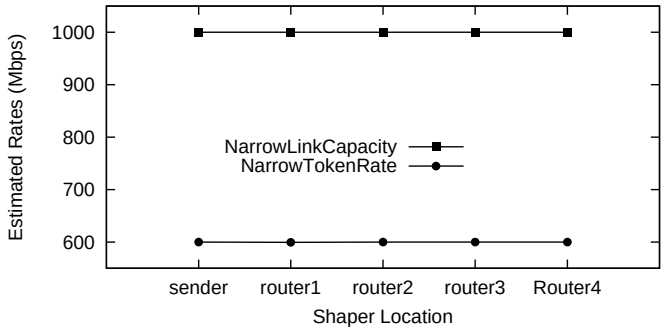


Figure 2.20: Our methods can accurately estimate the path capacity and path token rate, respectively, independent of the shaper location.

Multiple token bucket shapers: We add multiple shapers into the network as follows:

- Case 1) we add the first shaper with token rate 800 Mbps and burst size 10K bytes at router 1.
- Case 2) add the second shaper with token rate 600 Mbps and burst size 20 Kbytes at router 2.
- Case 3) add the third shaper with token rate 400 Mbps and burst size 40 Kbytes at router 3.
- Case 4) add the fourth shaper with token rate 200 Mbps and burst size 40 Kbytes at router 4.

Therefore, the path capacity remains 1 Gbps in all cases, but the path token rate reduces from 800 Mbps to 200 Mbps from case 1 to case 4. Figure 2.21 shows that our methods can accurately estimate both the path capacity and path token rate in all cases with multiple shapers.

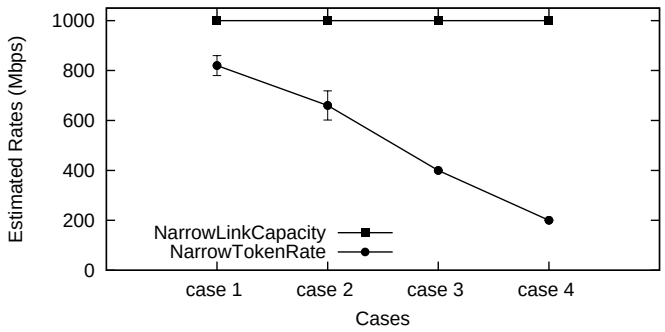


Figure 2.21: Our methods can accurately estimate the path capacity and path token rate in case of multiple shapers.

3 Network Path Capacity Comparison without Accurate Packet Time Information

3.1 Introduction

A rich body of bandwidth estimation methods [45] have been proposed and studied in the past two decades, due to the wide range of applications of bandwidth estimation. However, there is a fundamental problem with the current bandwidth estimation methods. Most (if not all) of them need to accurately measure certain time information of network packets, such as the arrival time difference (ATD) between two consecutive packets [15], the one-way or round-trip delay of each packet [29], and the queueing delay of each packet [23]. However, it is hard and sometimes impossible to accurately measure these time information in an increasing number of network environments, such as widely deployed high speed networks, and emerging cloud computing networks.

There are two major reasons why it is sometimes hard to accurately measure the packet time information. First, it takes very short times to send or receive packets at very high speeds. However, it is hard to measure such short times due to the limited system capability [25, 41]. Second, various software and hardware factors at the receiver of packets, such as interrupt moderation [44, 25] (commonly used in high speed network cards) and virtual machine (VM) scheduling [58, 12] (commonly used in cloud computing), greatly change the original packet time information. As a result, the packet time information measured by the packet receiver is not correct.

Our work is motivated by the observation that many applications only need to relatively

compare the bandwidth information of different paths. For example, in a peer-to-peer (P2P) network, a new peer needs to select several fastest peers as its neighbors from a set of existing peers. More motivating examples will be discussed in Section 3.2. In these cases, we do not need to measure the actual bandwidth information of each path, instead, we only need to relatively compare the bandwidth information of different paths, and then rank them according to their bandwidth information.

In this chapter, we study how to relatively compare the bandwidth information of multiple paths without requiring accurate packet time information. There are several important bandwidth metrics [45]. As the first step, this chapter considers only the capacity of a path that is the capacity of the narrow link in the path, and the narrow link of a path is the link with the smallest capacity among all links in a path. The path capacity is a basic bandwidth metric and will provide useful information for studying other bandwidth metrics, such as available bandwidth and bulk TCP throughput, which will be considered in our future work.

Specifically, this chapter proposes a capacity comparison method, called PathComp, which can relatively compare the path capacities from two senders to the same receiver. Basically, PathComp actively sends probing packets from both senders to the receiver, measures the arrival sequence of these packets at the receiver, and then relatively compares the capacities of the two paths.

PathComp is based on the fact that the inter-arrival gap between two consecutive packets from the same sender is related to the capacity of their path. This fact is also the basis of the current capacity estimation methods [15, 29]. The uniqueness of PathComp is that it measures the packet inter-arrival gap using the packet arrival sequence information, whereas the current capacity estimation methods measure the packet inter-arrival gap using the packet arrival time information. Therefore, PathComp does not require any accurate packet time information, and is fundamentally different from the current capacity estimation methods.

The contributions of this chapter are as follows. First, *we expand the design space of traditional time-based bandwidth estimation methods by introducing a new class of sequence-based bandwidth comparison methods*. Note that bandwidth comparison methods are inherently more scalable than traditional bandwidth estimation methods in terms of the measurement time for a large number of paths. This is because bandwidth comparison methods are designed to simultaneously measure multiple paths, whereas traditional bandwidth estimation methods are designed to measure a single path and are sensitive to the interference among multiple concurrent measurements [14].

Second, *we propose a capacity comparison method, called PathComp, which can determine not only which path is faster but also how much faster in terms of the path capacity*. In the chapter, we thoroughly study the impact of various types of cross traffic on capacity comparison, and we also discuss some implementation challenge, such as Receiver Side Scaling [38]. Our testbed, campus network, and Amazon EC2 [6] experiments show that PathComp can accurately compare the capacities of two paths in a variety of network environments.

3.2 Motivation

Bandwidth Comparison Scenarios

Our work is motivated by the observation that many applications only need to relatively compare the bandwidth information of different paths.

P2P neighbor selection: When a new peer joins a P2P network [36], it usually needs to select its neighbors from a set of existing peers. Typically, the new peer selects the existing peers with fast network bandwidth as its neighbors so that it can quickly download data from its neighbors. Bandwidth comparison methods can be used to quickly select several fastest peers from a set of existing peers.

Network-aware task placement [31]: Consider a bandwidth-intensive cloud application with three tasks: T_1, T_2, and T_3, and a cloud consisting of three interconnected VMs: V_1, V_2, and V_3. Assume that tasks T_1 and T_2 communicate often with task T_3, but not much with each other. If we find that the path between V_1 and V_2 is the slowest one using a bandwidth comparison method, and network measurements show that the latency between any two of these three VMs is the same, then the application performance can be improved with the optimal task placement that places task T_3 on V_3, and places the other two tasks on the other two VMs.

Difficulties in Obtaining Accurate Packet Time Information Another motivation of our work is that the current time-based capacity measurement algorithms do not work well in some network environments, such as high speed networks and cloud computing networks, where it is hard to accurately measure the packet time information. As discussed the challenges in the Introduction part of this proposal, there are two main reasons. First, it takes very short times to send or receive packets at very high speeds. For example, it takes only 12 μs to send or receive a 1500-byte packet at 1 Gbps, and only 1.2 μs at 10 Gbps. However, it is hard to accurately measure such short times due to the limited system capability [25, 41], such as clock time resolutions, clock frequency differences between the sender and the receiver, and the system call overhead.

Second, there are various software and hardward factors at the receiver of packets, such as interrupt coalescence [25, 44], context switching, and virtual machine scheduling [58, 12], which change the original packet time information, and thus the packet time information measured by the packet receiver is not correct.

3.3 Design Space and Related Work

We discuss the design space of capacity estimation methods in Figure 3.1, which helps us to understand the relation between the current capacity estimation methods and our proposed capacity comparison method. Some methods measure the capacity information of the path from a computer to another computer, and we refer to the first computer as the sender and the second computer as the receiver. Some other methods measure the capacity information of the round-trip path from a computer to another computer and then back to the first one. For these methods, we refer to the first computer as both the sender and receiver.

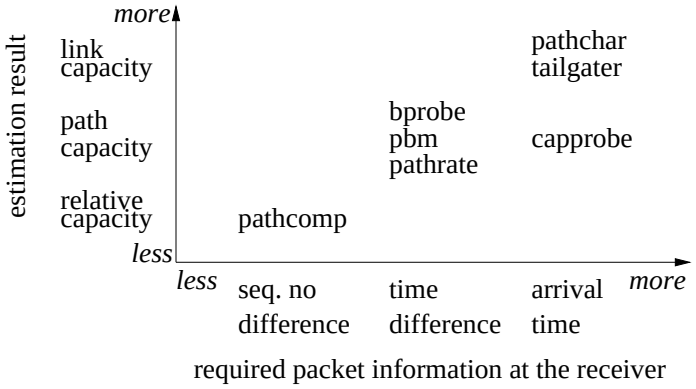


Figure 3.1: The design space of capacity estimation methods.

The design space shown in Figure 3.1 is based on the required information of probing packets at the receiver. PathChar [16] and TailGater [32] estimate the capacity of each individual *link* in the path using the packet *arrival times* at the receiver. CapProbe [29] and PBProbe [11] estimate the path capacity using the packet arrival times. BProbe [10], PBM [40], and PathRate [15] estimate the *path* capacity using the packet *arrival time differences* at the receiver (defined in Section 3.4). Our proposed PathComp *relatively* compares the path capacities from two senders to the same receiver using the packet *arrival sequence number differences* at the receiver (defined in Section 3.4).

Note that if we know the capacity of each individual link of a path, we can infer the

capacity of the path. If we know the capacities of two paths, we can infer their relative capacity ratio. Also note that the arrival times can be used to calculate the arrival time differences, and the arrival time differences can be used to infer the arrival sequence number differences. Therefore, we can see that the less the estimated capacity information, the less the required packet information.

Further more, the arrival time differences are relatively easier to accurately measure than the arrival times. For example, they are not sensitive to clock time differences between the sender and the receiver. The arrival sequence number differences can be more accurately measured than the arrival time differences. For example, they are not sensitive to the interrupt moderation at the receiver. Overall, we can see that *the less the estimated capacity information, the less the required packet information, and the more robust the method*.

3.4 Capacity Comparison

In this section, we explain the difference between the traditional capacity estimation problem and our proposed capacity comparison problem, and explain the difference between the traditional time-based capacity estimation methods and our proposed sequence-based capacity comparison method.

3.4.1 Capacity Estimation and Comparison Problems

We use an example illustrated in Figure 3.2 to describe the difference between the traditional capacity estimation problem and our proposed capacity comparison problem. There are two paths in Figure 3.2: path *a* is from sender *SNDa* to receiver *RCV*, and path *b* is from sender *SNDb* to the same receiver *RCV*. Both paths merge with each other at router *R5*. Network 5 in the figure represents everything between *R5* (including *R5*) and *RCV*. Network 1

represents everything between *SNDa* and the narrow link of path *a* (called narrow link *a*), and network 2 for everything between narrow link *a* and *R5*. Similarly networks 3 and 4 for path *b*. Let C_a denote the capacity of path *a* that is the capacity of narrow link *a*, and let C_b denote the capacity of path *b* that is the capacity of narrow link *b*.

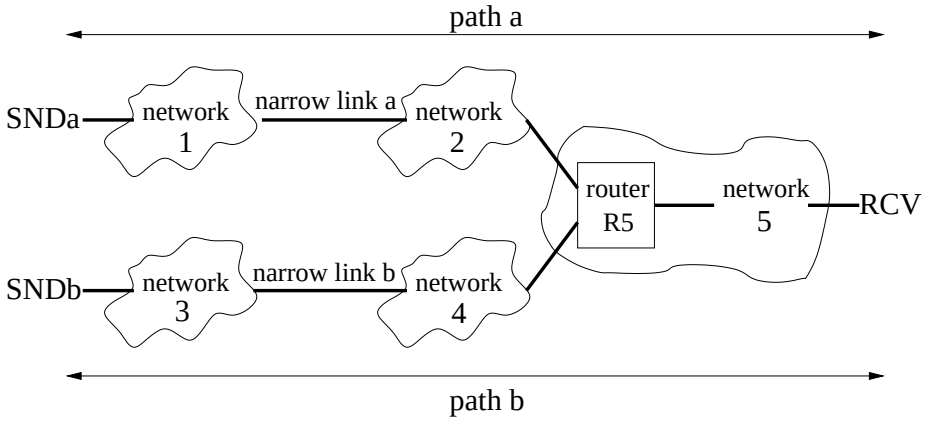


Figure 3.2: Two paths: path *a* is from sender *SNDa* to receiver *RCV*, and path *b* is from sender *SNDb* to the same receiver *RCV*.

The traditional capacity estimation problem considers the capacity of the narrow link of a single path. For example, for the two paths in Figure 3.2, the traditional capacity estimation problem separately estimates C_a and C_b.

Our proposed capacity comparison problem considers the capacity ratio of the narrow links of two paths. For example, for paths *a* and *b* in Figure 3.2, the capacity comparison problem estimates the capacity ratio of C_a and C_b. That is, it relatively compares the link capacities of these two narrow links. *The capacity ratio* γ *of two paths* a *and* b is defined as follows. Note that γ is a real number at least 1.

$$\gamma = \begin{cases} C_a/C_b, & \text{if } C_a \geq C_b \\ C_b/C_a, & \text{otherwise.} \end{cases} \tag{3.1}$$

We also define the *rounded capacity ratio* as follows, which is an integer at least 1.

$$\Gamma = round(\gamma) \tag{3.2}$$

Note that Figure 3.2 assumes that paths *a* and *b* do not have a shared narrow link (i.e., if the narrow link is located in network 5). This is a reasonable assumption for a variety of scenarios. For example, consider the P2P neighbor selection problem described in Section 3.2. The narrow link of the path from a neighbor to a peer is usually the upload link of the neighbor, and thus different neighbors usually do not have a shared narrow link. As another example, consider the network-aware task placement problem in Section 3.2. The narrow link of the path from a sender VM to another receiver VM is usually located near the sender VM due to the rate limiting of the sender VM, and thus the paths from different sender VMs usually do not have a shared narrow link.

In cases where two paths have a shared narrow link, there are two options. First, capacity comparison reports the capacity ratio of the narrow links of the distinct segments of the two paths. Second, capacity comparison does not report anything, if a shared narrow link is detected. However, the method to detect a shared narrow link is out of the scope of this chapter. We choose the first option in this chapter.

3.4.2 Traditional Time-based Capacity Estimation

The traditional capacity estimation methods, such as PathRate [15], and CapProbe [29], are usually based on packet arrival time differences (also called inter-arrival times, and dispersion times). The packet *arrival time difference (ATD, denoted by τ) of two packets* is the time difference between their arrival times. For example, Figure 3.3 shows two *SNDa* packets, a_1 and a_2, on the link from network 5 to receiver *RCV*, and the time difference τ is their ATD at *RCV*.

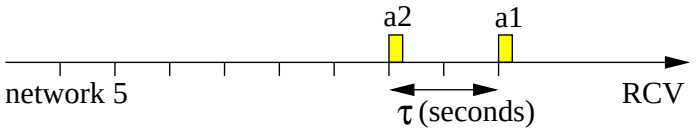


Figure 3.3: The ATD τ at *RCV* between two *SNDa* packets (a_1 and a_2) is their arrival time difference at *RCV*.

Assuming that there is no cross traffic on path *a*, and assuming that *RCV* can accurately measure the ATD, the capacity C_a can be obtained as follows [15, 29], where S is the packet size and τ is the ATD.

$$C_a = S/\tau \tag{3.3}$$

The traditional capacity estimation methods mainly differ in how to accurately estimate C_a in the presence of cross traffic. However, if *RCV* cannot accurately measure the ATD, none of these methods works.

3.4.3 Proposed Sequence-based Capacity Comparison

ASND Definition We propose to tackle the capacity comparison problem using packet arrival sequence number differences instead of packet arrival time differences, so that our method does not require accurate packet time information. Below we use an example to explain the concept of the packet arrival sequence number differences.

Each of the two senders, *SNDa* and *SNDb*, sends a train of $L = 5$ packets of the same packet size S to the receiver *RCV* at approximately the same time. These packets are sent back-to-back by their senders (i.e., at their maximum rates). In this example, we assume that there is no cross traffic in all 5 networks in Figure 3.2.

The top line of Figure 3.4 shows the 5 *SNDa* packets on the link from network 2 to router *R5*. Since there is no cross traffic, the ATD between every two consecutive *SNDa* packets at router *R5* is inversely proportional to the capacity C_a of narrow link *a*. The bottom line of Figure 3.4 shows the 5 *SNDb* packets on the link from network 4 to router

R5, and the ATD between every two consecutive *SNDb* packets at the router is inversely proportional to the capacity C_b of narrow link *b*. In this example, we set $C_a = C_b/2$, and thus the ATD between two consecutive *SNDa* packets is twice the ATD between two consecutive *SNDb* packets as illustrated in Figure 3.4.

These 10 packets merge with one another at router *R5*. In this example, we assume that these packets arrive at *RCV* in the order of their arrival times at router *R5*. For example, Figure 3.4 shows that packet b_1 arrives at *R5* earlier than packet a_1, and thus packet b_1 arrives at *RCV* earlier than packet a_1. In Section 3.6.3, we will discuss cases where this assumption does not hold and describe our solution. *RCV* assigns the first received packet an arrival sequence number of 1, and the next received packet an arrival sequence number of 2, and so on. Figure 3.5 shows the arrival order of these 10 packets at *RCV*, and their corresponding arrival sequence numbers.

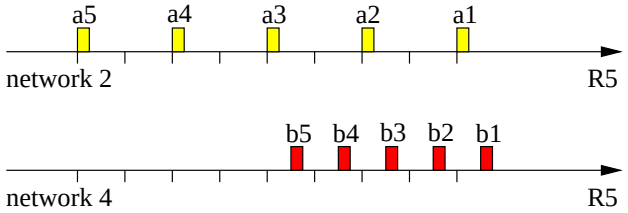


Figure 3.4: 5 *SNDa* packets on the link from network 2 to router *R5* (top line), and at the same time 5 *SNDb* packets on the link from network 4 to *R5* (bottom line).

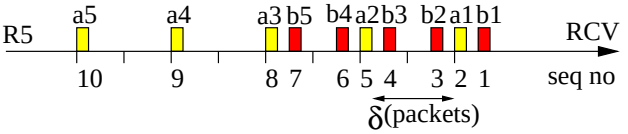


Figure 3.5: The arrival sequence numbers of all 10 packets at *RCV*. The ASND δ between packets a_1 and a_2 is the difference between their packet arrival sequence numbers minus one.

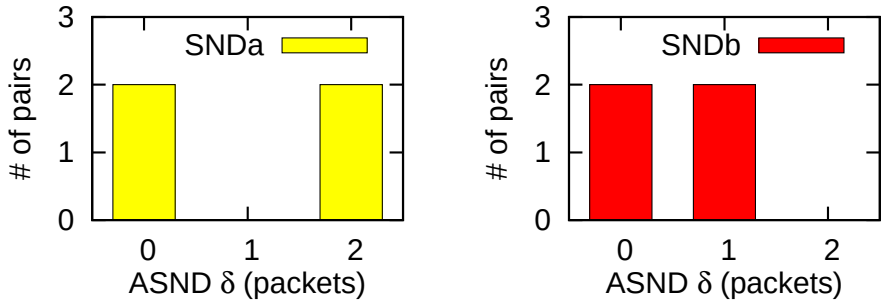
The packet arrival sequence number difference (ASND, denoted by δ) of two packets is defined to be the difference between their arrival sequence numbers minus one. For example, the arrival sequence number of packet a_1 is 2 in Figure 3.5, and that of packet a_2

is 5, thus their ASND is $\delta = (5-2)-1 = 2$ packets. Intuitively, this means that there are two other packets between packets a_1 and a_2.

ASND Histograms We can infer the capacity ratio γ of paths *a* and *b* by analyzing their ASND histograms. In this subsection, we present the concept of ASND histograms, and in Section 3.5, we will thoroughly study the impact of cross traffic on the ASND histograms.

Let $H_a(i)$ (respectively, $H_b(i)$) denote the total number of pairs of two consecutive packets that are sent by *SNDa* (respectively, *SNDb*) and separated by $\delta = i$ packets at *RCV*. For example, $H_a(0) = 2$ pairs in Figure 3.5, because the ASND between packets a_3 and a_4 is 0 and that between packets a_4 and a_5 is also 0. As another example, $H_a(2) = 2$ pairs, because the ASND between packets a_1 and a_2 is 2 and that between packets a_2 and a_3 is also 2.

The ASND histogram of the SNDa *train is vector* $H_a = (H_a(0), H_a(1), H_a(2), ...)$, and *that of the* SNDb *train is vector* $H_b = (H_b(0), H_b(1), H_b(2), ...)$. For example, Figure 3.6 shows ASND histograms H_a and H_b for *SNDa* and *SNDb* trains, respectively.



(a) Histogram H_a (b) Histogram H_b

Figure 3.6: The ASND histograms of the two trains in Figure 3.5.

We have the following theorem to simplify our analysis of ASND histograms.

Theorem 1. $|H_a(0) - H_b(0)| \leq 1$, *if two trains have the same number* L *of packets.*

Proof. Let symbols a and b (without the subscripts) to denote a packet of *SNDa* and *SNDb*,

respectively. The arrival order of the $2L$ packets can be described by a string consisting of L symbol a's and L symbol b's. For example, if $L = 5$, the arrival order of the 10 packets in Figure 3.5 can be described by string $aaabbabbab$, where the rightmost symbol (i.e., b) is the first packet received by *RCV* (i.e., b_1), and the leftmost symbol (i.e., a) is the last packet received by *RCV* (i.e., a_5).

A string of L symbol a's and L symbol b's can be classified into the following four cases, according to the leftmost and rightmost symbols. We will prove only cases 1 and 2, and the other two cases can be proved very similarly.

- Case 1: The leftmost one: a, and the rightmost one: a.
- Case 2: The leftmost one: a, and the rightmost one: b.
- Case 3: The leftmost one: b, and the rightmost one: a.
- Case 4: The leftmost one: b, and the rightmost one: b.

Case 1: For a string with $2L$ symbols, there are a total of $2L-1$ pairs of two consecutive symbols. Let $n(aa)$, $n(ab)$, $n(ba)$, and $n(bb)$ denote the number of pairs aa, ab, ba, and bb, respectively. Let $n(*a)$ and $n(*b)$ denote the number of pairs whose right symbol is a and b, respectively. By definition, we have $n(*a) = n(ba)+n(aa)$, and $n(*b) = n(ab)+n(bb)$.

We have $n(*a) = L - 1$, because the leftmost a cannot be the right symbol of a pair. We also have $n(*b) = L$. Therefore, we have $n(*a) = n(*b) - 1$.

Since both the leftmost and the rightmost symbols are a, we have $n(ab) = n(ba)$. Therefore, $H_a(0) = n(aa) = n(*a)-n(ba) = (n(*b)-1)-n(ab) = n(bb)-1 = H_b(0)-1$.

Case 2: We have $n(*a) = L - 1$, because the leftmost a cannot be the right symbol of a pair. We also have $n(*b) = L$. Therefore, we have $n(*a) = n(*b) - 1$.

Since the leftmost symbol is a and the rightmost one is b, we have $n(ab) = n(ba) + 1$. Therefore, $H_a(0) = n(aa) = n(*a)-n(ba) = (n(*b)-1)-(n(ab)-1) = n(bb) = H_b(0)$.

□

For example, $H_a(0) - H_b(0) = 2 - 2 = 0$ in Figure 3.6. Note that, Theorem 1 holds no matter whether there is cross traffic or not and no matter how long the train size L is.

We will not show and will not use $H_a(0)$ and $H_b(0)$ in the rest of the chapter for the following two reasons. First, usually the *SNDa* and *SNDb* trains only partially overlap with each other, and thus $H_a(0)$ and $H_b(0)$ are mainly due to the non-overlapping packets of the two trains. Second, Theorem 1 shows that $H_a(0)$ and $H_b(0)$ are very close to each other, and thus do not provide much useful information.

A peak (also called a mode) in a histogram is a local maximum that is higher than its right neighbors and no less than its left neighbor (if exists). For example, histogram H_a in Figure 3.6 has a peak at 2 packets, and histogram H_b has a peak at 1 packet (note that it does not have the left neighbor, since we do not consider $H_b(0)$).

We introduce the second theorem about the peaks in ASND histograms. Without loss of generality, this theorem considers only case $C_a \leq C_b$.

Theorem 2. *In the absence of cross traffic, if $C_a \leq C_b$, histogram H_a has only one peak and the peak is located at Γ packets, and histogram H_b has only one peak and the peak is located at 1 packet. The capacity ratio γ can be obtained as follow, where $H_a(\Gamma - 1)$ should be set to 0 if $\Gamma = 1$.*

$$\gamma = \frac{(\Gamma - 1)H_a(\Gamma - 1) + \Gamma H_a(\Gamma) + (\Gamma + 1)H_a(\Gamma + 1)}{H_a(\Gamma - 1) + H_a(\Gamma) + H_a(\Gamma + 1)} \tag{3.4}$$

Proof. When there is no cross traffic, the ATD of a pair of two consecutive *SNDa* packets is S/C_a. The average number of *SNDb* packets that can be transmitted during an S/C_a interval is $(S/C_a) \times (C_b/S) = C_b/C_a$. Therefore, the average number of *SNDb* packets

between a pair of two consecutive *SNDa* packets is C_b/C_a. We consider the following three possible cases:

Case 1: C_b/C_a is a positive integer. That is, $\Gamma = \gamma = C_b/C_a$. In this case, there are exactly Γ *SNDb* packets between a pair of two consecutive *SNDa* packets. Therefore, the peak of H_a is at Γ packets. In this case, there are either 0 or 1 *SNDa* packet between a pair of two consecutive *SNDb* packets. Therefore, the peak of H_b is at 1 packet. Note that, $H_a(\Gamma - 1) = H_a(\Gamma + 1) = 0$, and thus Equation (3.4) can be proved.

Case 2: C_b/C_a is a decimal greater than 1, and $\Gamma = \lfloor C_b/C_a \rfloor$ and $\Gamma + 1 = \lceil C_b/C_a \rceil$. In this case, there are either $\lfloor C_b/C_a \rfloor$ or $\lceil C_b/C_a \rceil$ *SNDb* packets between a pair of two consecutive *SNDa* packets. Because $\Gamma = round(\gamma) = \lfloor C_b/C_a \rfloor$, we have $H_a(\Gamma) > H_a(\Gamma + 1) > 0$. Therefore, the peak of H_a is at Γ packets. In this case, there are either 0 or 1 *SNDa* packet between a pair of two consecutive *SNDb* packets. Therefore, the peak of H_b is at 1 packet. Note that, $H_a(\Gamma - 1) = 0$, and thus Equation (3.4) can be proved.

Case 3: C_b/C_a is a decimal greater than 1, and $\Gamma - 1 = \lfloor C_b/C_a \rfloor$ and $\Gamma = \lceil C_b/C_a \rceil$. This case can be proved in a similar way to case 2.

□

For example, in Figure 3.6, because $C_a < C_b$, histogram H_a has a peak at $\Gamma = round(C_b/C_a) = 2$ packets, and histogram H_b has a peak at 1 packet.

ASND-based Capacity Comparison Theorem 2 provides the foundation of our proposed capacity comparison method in the absence of cross traffic. Given the histogram H of the slower path, algorithm EST-RATIO can estimate the capacity ratio γ using Theorem 2. Since initially we do not know which path is slower, algorithm COMPARE calculates two ratio estimates: γ_a assuming path *a* is slower, and γ_b assuming path *b* is slower. Then it selects the bigger ratio as the final result.

Algorithm 1 Estimate the capacity ratio from histogram H using Theorem 2 in the absence of cross traffic

function EST-RATIO(H)
  $\Gamma \leftarrow \max(H(1), H(2), H(3), ...)$ ▷ Find the peak
  $\gamma \leftarrow \frac{(\Gamma-1)H(\Gamma-1)+\Gamma H(\Gamma)+(\Gamma+1)H(\Gamma+1)}{H(\Gamma-1)+H(\Gamma)+H(\Gamma+1)}$
  return γ
end function

Algorithm 2 Compare the capacities of two paths using their histograms H_a and H_b in the absence of cross traffic

function COMPARE(H_a, H_b)
  $\gamma_a \leftarrow$ EST-RATIO(H_a) ▷ Assuming *a* is slower
  $\gamma_b \leftarrow$ EST-RATIO(H_b) ▷ Assuming *b* is slower
  if $\gamma_a == \gamma_b$ **then**
    print Path *a* is as fast as path *b*.
  else if $\gamma_a > \gamma_b$ **then**
    print Path *a* is slower than path *b*.
    print $C_b/C_a = \gamma_a$
  else
    print Path *a* is faster than path *b*.
    print $C_a/C_b = \gamma_b$
  end if
end function

3.5 Impact of Cross Traffic

In this section, we study the impact of various types of cross traffic on the ASND histograms using our lab testbed.

We study five possible types of cross traffic as illustrated in Figure 3.7, which is very similar to Figure 3.2 and just simplifies each network to a single router. The capacity of each link is chosen to demonstrate the impact of the cross traffic on that link. We emulate this network using our 10Gbps testbed, and each link is emulated by a Linux token bucket filter (tbf).

The narrow link of path *a* from *SNDa* to *RCV* is the link between routers *R1* and *R2*, and thus the capacity of path *a* is $C_a = 200$Mbps. The narrow link of path *b* from *SNDb* to *RCV*

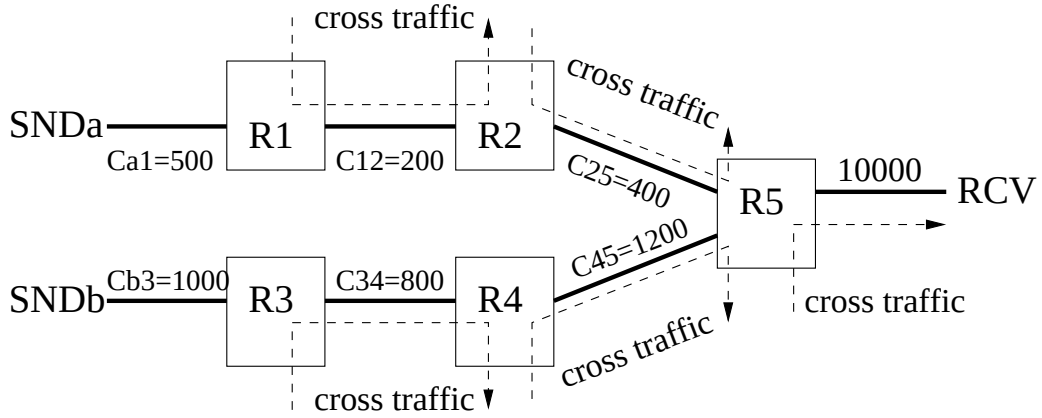


Figure 3.7: Five possible sources of cross traffic. Link capacity unit: Mbps.

is the link between routers *R3* and *R4*, and thus the capacity of path *b* is $C_b = 800$Mbps. Therefore, we have $\Gamma = \gamma = 4$.

In each of the following experiments, each sender sends a train of $L = 500$ packets at approximately the same time, and *RCV* measures the ASND histograms. Because path *b* is the faster one, all *SNDb* histograms concentrate at 0 and 1 packet (similar to Figure 3.6b), and thus we do not show the *SNDb* histograms. For the SNDa histograms, we do not show the result for 0 packet, as explained in Section 3.4.

No Cross Traffic As a reference case, first we do not generate any cross traffic. Since $\gamma = 4$, there should be 4 *SNDb* packets between a pair of two consecutive *SNDa* packets, as illustrated in Figure 3.8. The SNDa histogram is shown in Figure 3.13. As we expect, the ASND of most *SNDa* pairs is 4 packets. But there are a small number of *SNDa* pairs with other ASNDs, which are mainly caused by the randomness of the routers that are emulated using our lab computers and Linux tbf.

Cross Traffic between *R1* and *R2*

This experiment shows the impact of cross traffic before or on the narrow link of path *a* (i.e., the slower path). Random cross traffic is generated using MGEN [1] at an average rate of $200 * 50\% = 100$ Mbps between *R1* and *R2*.

Let's consider the example shown in Figure 3.9. There are still the same 8 *SNDb* packets

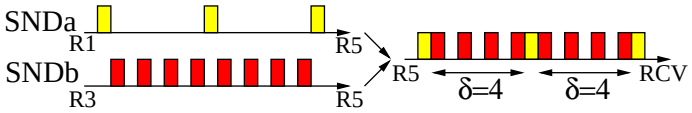


Figure 3.8: No cross traffic. Each box indicates a packet on the link.

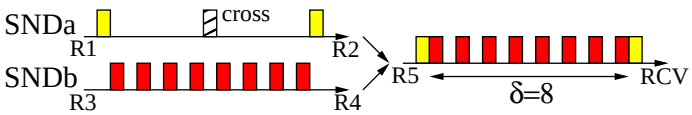


Figure 3.9: Cross traffic between *R1* and *R2*.

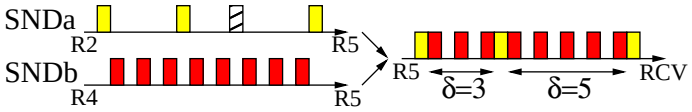


Figure 3.10: Cross traffic between *R2* and *R5*.

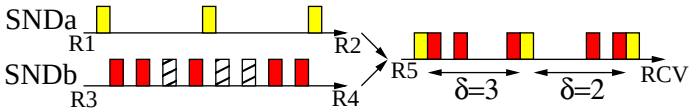


Figure 3.11: Cross traffic between *R3* and *R4*.

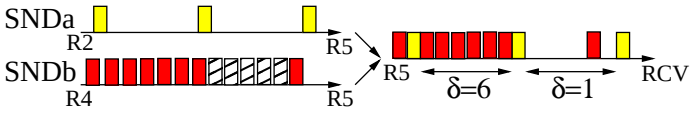


Figure 3.12: Cross traffic between *R4* and *R5*.

passing the link between *R3* and *R4* as in Figure 3.8. But during this time interval, a cross traffic packet is inserted between the first (i.e., the rightmost one) and second *SNDa* packets (the third *SNDa* packet is further delayed, and not shown in the figure). As a result, the ASND between the first and second *SNDa* packets is doubled and becomes 8 packets.

This is why the *SNDa* histogram in Figure 3.14 has a non-negligible number of *SNDa* pairs with $\delta = 8$ packets. Further more, the numbers of *SNDa* pairs with $\delta = \gamma i = 4i$ packets approximately follow a Geometric distribution described by Equation (3.5), where N is the total number of pairs with $\delta = 4i$ packets, and p is the occurrence probability of a cross traffic packet. For example, the dotted line in Figure 3.14 is obtained using Equation (3.5) with the corresponding N and p.

$$H_a(i * 4) = Np^{i-1}(1-p) \qquad 1 \leq i \tag{3.5}$$

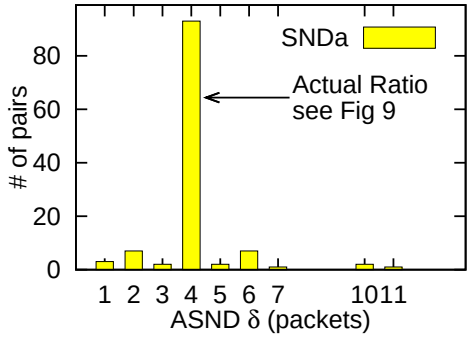


Figure 3.13: No cross traffic.

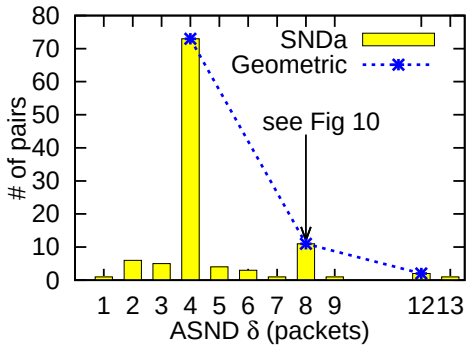


Figure 3.14: 50% cross traffic between *R1* and *R2*.

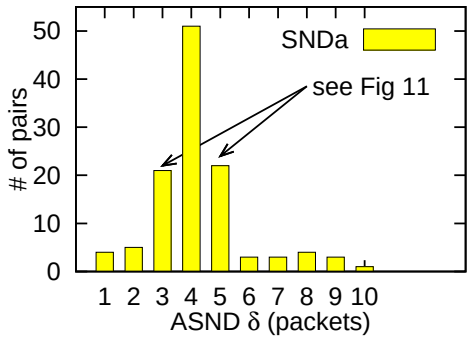


Figure 3.15: 80% cross traffic between *R2* and *R5*.

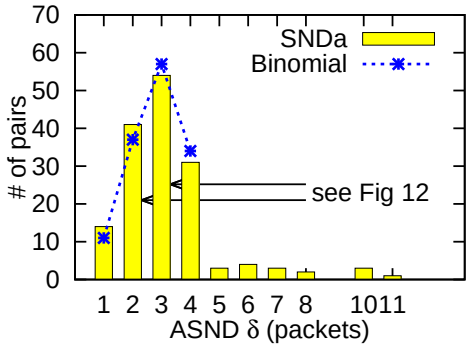


Figure 3.16: 50% cross traffic between *R3* and *R4*.

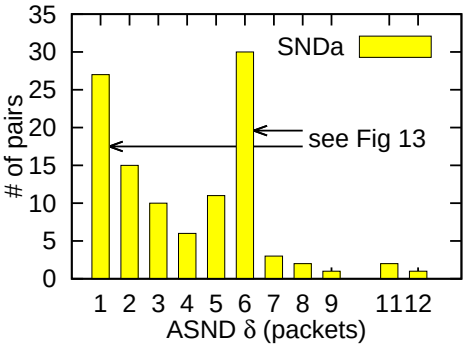


Figure 3.17: 50% cross traffic between *R4* and *R5*.

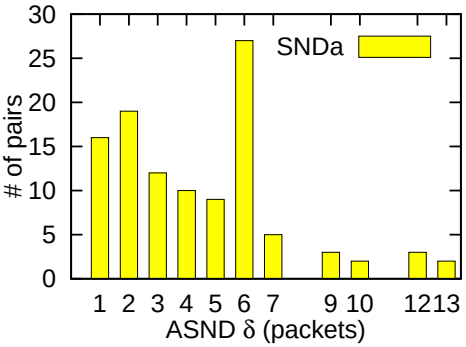


Figure 3.18: 50% cross traffic on all 5 links.

Cross Traffic between *R2* and *R5*

This experiment shows the impact of cross traffic beyond the narrow link of path *a* but still before the shared segment. Random cross traffic is generated at an average rate of $400 * 80\% = 320$ Mbps between *R2* and *R5*.

In the example shown in Figure 3.10, there are still the same 8 *SNDb* packets passing the link between *R4* and *R5* as in Figure 3.8. But a cross traffic packet is inserted between the first and second *SNDa* packets. Because the link capacity between *R2* and *R5* is twice that between *R1* and *R2*, the third *SNDa* packet can still be transmitted at the original time as in Figure 3.8. As a result, the ASND between the first and second *SNDa* packets increases to 5 packets, but the ASND between the next two *SNDa* packets decreases to 3 packets.

This is why the *SNDa* histogram shown in Figure 3.15 has a non-negligible number of *SNDa* pairs with ASNDs around 4 packets, such as 3 and 5 packets.

Cross Traffic between *R3* and *R4*

This experiment shows the impact of cross traffic before or on the narrow link of path *b* (i.e., the faster path). Random cross traffic is generated at an average rate of $800 * 50\% = 400$ Mbps between *R3* and *R4*.

In the example shown in Figure 3.11, there are still the same 3 *SNDa* packets passing the link between *R1* and *R2* as in Figure 3.8. But three cross traffic packets are inserted between these *SNDb* packets (the rightmost three *SNDb* packets in Figure 3.8 are further delayed, and not shown in Figure 3.11). As a result, the ASND between the first and second *SNDa* packets decreases to 2 packets, and the ASND between the next two *SNDa* packets decreases to 3 packets.

This is why the *SNDa* histogram in Figure 3.16 has a large number of *SNDa* pairs with ASND less than 4 packets. The numbers of *SNDa* pairs with ASNDs between 1 and 4 packets follow a Binomial distribution described by Equation (3.6), where N is the total number of *SNDa* pairs with ASNDs between 1 and 4 packets, and p is the occurrence probability of a cross traffic packet. The dotted line in Figure 3.16 is obtained using Equation (3.6) with the corresponding N and p.

$$H_a(i) = N\binom{4}{i}(1-p)^i p^{4-i}/(1-p^4) \qquad 1 \leq i \leq 4 \tag{3.6}$$

Cross Traffic between *R4* and *R5*

This experiment studies the impact of cross traffic beyond the narrow link of path *b* but still before the shared segment. Random cross traffic is generated at an average rate of $1200 * 50\% = 600$ Mbps between *R4* and *R5*.

In the example shown in Figure 3.12, there are still the same 3 *SNDa* packets passing the link between *R2* and *R5* as in Figure 3.8. But during this time interval, several cross traffic packets are inserted between the first and second *SNDb* packets. Because the link capacity between *R4* and *R5* is higher than that between *R3* and *R4*, the remaining *SNDa* packets are only slightly delayed than in Figure 3.8. As a result, the ASND between the first and second *SNDa* packets decreases to 1 packet, and the ASND between the next two *SNDa* packets becomes 6 packets, which is the capacity ratio of the link between *R4* and *R5* to the link between *R1* and *R2* (i.e., 1200/200=6).

This is why the *SNDa* histogram shown in Figure 3.17 has a large number of *SNDa* pairs with ASND not equal to 4 packets, such as 1 and 6 packets.

Cross Traffic between *R5* and *RCV*

This experiment shows the impact of cross traffic in the shared segment of both paths. As we expect, this type of cross traffic does not have any impact on the histograms.

Cross Traffic on All Five Links

Finally, we generate all five types of cross traffic. The *SNDa* histogram is shown in Figure 3.18, and it shows the combination of the impact of all five types of cross traffic.

Summary We have the following observations about the ASND histogram of the slower path.

Observation 1: In practice, the ASND histogram could have a small number of ASND values caused by the randomness of the end-systems and the networks. These ASND values should be treated as noises and be discarded.

Observation 2: With little or no cross traffic, there is only one peak and the peak is lo-

cated at the rounded capacity ratio Γ (e.g., Figure 3.13). This is consistent with Theorem 2.

Observation 3: In the presence of cross traffic, it is possible that there are multiple peaks (also called multi-mode), and Γ may or may not be the location of a peak. For example, Figure 3.17 has peaks at 1 and 6 packets, but no peak at 4 packets. We observe that *multiple peaks are usually caused by the cross traffic on the faster path.* More specifically, they are usually caused by the cross traffic beyond the narrow link of the faster path, e.g., in Figures 3.17 and 3.18.

Observation 4: In the presence of cross traffic, if there is only a single peak, the peak location tends to be a lower bound of Γ (e.g., Figure 3.15 has a single peak at 4 packets, and Figure 3.16 has a single peak at 3 packets). Intuitively, this is because an ASND value greater than Γ usually leads to another ASND value smaller than Γ (e.g., Figures 3.10 and 3.12). Therefore, if there is a peak to the right of Γ, then there is usually another peak to the left of Γ. That is, there will be multiple peaks.

Observation 5: We have calculated and verified that the average of all ASND values of a histogram could be higher than or lower than γ (i.e., neither an upper bound nor a lower bound), depending on the amounts and locations of cross traffic on both paths.

3.6 PathComp

In this section, we present our proposed PathComp to relative compare the capacity ratio of two paths to the same receiver without requiring accurate packet time information.

3.6.1 The PathComp Method

PathComp follows the basic idea of algorithms EST-RATIO and COMPARE as described in Section 3.4.3. However, there are two problems with algorithm EST-RATIO in the presence of cross traffic. 1) It is possible to have multiple peaks in a histogram mainly due to the

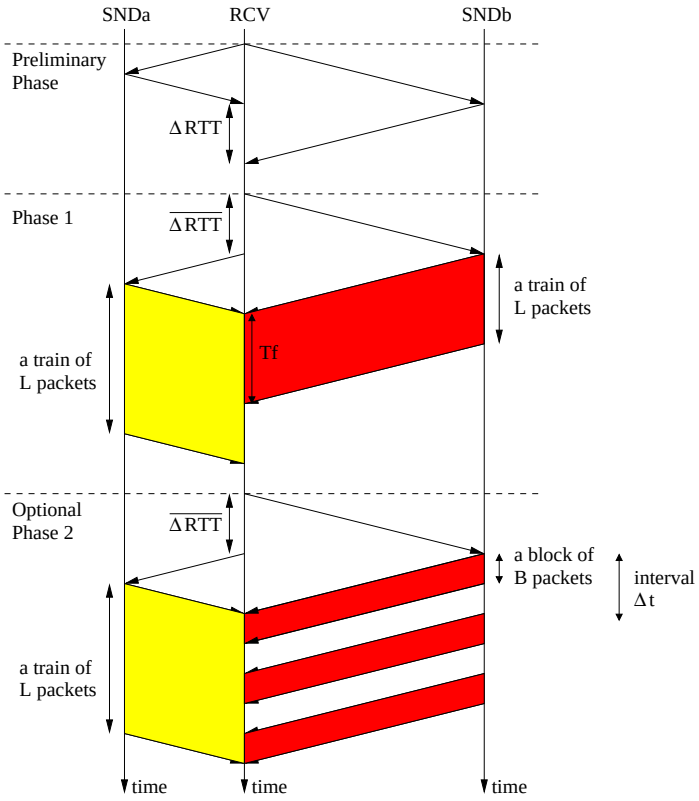


Figure 3.19: PathComp has three phrases.

cross traffic on the faster path (i.e., Observation 3 in Section 3.5), however EST-RATIO assumes only one peak in a histogram. To tackle this problem, we divide the long packet train on the faster path into multiple short packet blocks, in order to reduce the impact of cross traffic. 2) If there is a single peak in the histogram of the slower path, the peak location tends to be a lower bound of Γ (i.e., Observation 4 in Section 3.5). To tackle this problem, we estimate Γ by the peak of the weighted histogram.

PathComp consists of three phases as shown in Figure 3.19.

- 1) *Preliminary phase* measures some basic network information, such as the round-trip times (RTTs).
- 2) *Phase I* measures the histograms of the two paths. If there is a single peak in the histogram of the slower path, PathComp estimates γ using algorithms COMPARE

and EST-RATIO2; otherwise, it starts Phase II.

- 3) *Phase II* re-measures the histograms using multiple packet blocks on the faster path, and then estimates γ using algorithm EST-RATIO2.

Figure 3.19 still considers the two paths shown in Figure 3.2. But to simplify the figure, Figure 3.19 assumes that there is only one link between *SNDa* and *R5* that is the narrow link of path *a*, and there is only one link between *SNDb* and *R5* that is the narrow link of path *b*.

Preliminary Phase This phase measures the RTT difference $\triangle RTT$ between *SNDa-RCV* and *SNDb-RCV* as illustrated in Figure 3.19, so that in the next two phases the packets of *SNDa* and *SNDb* can overlap with each other. PathComp measures $\triangle RTT$ multiple times, and calculates the mean (denoted by $\overline{\triangle RTT}$) and standard deviation (denoted by $\sigma(\triangle RTT)$) of measured $\triangle RTT$ values.

Phase I *RCV* first tells the sender with a longer RTT (i.e., *SNDb* in Figure 3.19) to start its packet transmission, and after a delay of $\overline{\triangle RTT}$, RCV then tells the sender with a shorter RTT (i.e., *SNDa*) to start its packet transmission. Each sender sends a train of L consecutive packets with the same packet size S. In Figure 3.19, the capacity C_a of path *a* is lower than the capacity C_b of path b, so *SNDa* takes a longer time to transmit the same number L of packets than *SNDb*.

After *RCV* receives these two trains, PathComp measures the ASND histograms H_a and H_b of the two paths, and uses Algorithms EST-RATIO2 and COMPARE2 to estimate the capacity ratio. The difference between EST-RATIO and EST-RATIO2 is that the former selects the peak from the original histogram $H = (H(1), H(2), H(3), ...)$, whereas the latter selects the peak from the weighted histogram $(H(1), 2H(2), 3H(3), ...)$. This is motivated by Observation 4 in Section 3.5. Note that the peak location of the weighted histogram is greater than or the same as that of the original histogram. The difference between

COMPARE and COMPARE2 is that the former calls EST-RATIO whereas the latter called EST-RATO2. In addition, if multiple peaks are detected in the histogram of the slower path, Algorithm COMPARE2 starts phase II.

Algorithm 3 Estimate the capacity ratio from histogram H in the presence of cross traffic

1: **function** EST-RATIO2(H)
2: Remove measurement noises from H
3: $\Gamma \leftarrow \max(H(1), 2H(2), 3H(3), ...)$ ▷ Weighted
4: $\gamma \leftarrow \frac{(\Gamma-1)H(\Gamma-1)+\Gamma H(\Gamma)+(\Gamma+1)H(\Gamma+1)}{H(\Gamma-1)+H(\Gamma)+H(\Gamma+1)}$
5: **return** γ
6: **end function**

Parameter Setting If $\sigma(\triangle RTT) = 0$, the two trains should arrive at *RCV* at the same time as illustrated in Figure 3.19. In practice, $\sigma(\triangle RTT) > 0$, and the train size L should be sufficiently long so that the two trains can still overlap with each other. For example, consider a cloud computing network in a data center with $\sigma(\triangle RTT)$=1 ms and with the capacity=1 Gbps, L should be at least 83 packets longer to compensate for the RTT variance if packet size S is 1500 Byte. By default, PathComp sets the train size L to 500 packets.

If the two trains could not overlap with each other, or overlap for only a small portion of each train, PathComp increases the train size and re-sends the two trains. However, if excessive packet loss is detected at *RCV*, PathComp quits the estimation.

By default, PathComp sets the packet size S to 1500 bytes. This is because our experiments show that ASND histograms become hard to predict and analyze when the packet size is small. Intuitively, this is because the randomness of the end-systems and networks have a big impact on small packets, and thus there are much more noises in the ASND histograms.

Phase II PathComp enters this phase, if there are multiple peaks in the histogram of the slower path. As observed in Section 3.5, this is usually due to the high cross traffic load on the faster path. Therefore, we divide the long packet train on the faster path into multiple

Algorithm 4 Compare the capacities of two paths using their histograms H_a and H_b in the presence of cross traffic

```
function COMPARE2(H_a, H_b)
    γ_a ← EST-RATIO2(H_a)        ▷ Assuming a is slower
    γ_b ← EST-RATIO2(H_b)        ▷ Assuming b is slower
    if γ_a == γ_b then
        print Path a is as fast as path b.
    else if γ_a > γ_b then
        print Path a is slower than path b.
        if H_a has multiple peaks then
            starts Phase II
        else
            print C_b/C_a = γ_a
        end if
    else
        print Path a is faster than path b.
        if H_b has multiple peaks then
            starts Phase II
        else
            print C_a/C_b = γ_b
        end if
    end if
end function
```

short packet blocks, in order to reduce the impact of cross traffic.

Specifically, PathComp still sends a train of L packets back-to-back on the slower path. But on the faster path, PathComp sends a block of B packets back-to-back every Δt time interval, until all L packets have been sent out, as illustrated in Figure 3.19. After *RCV* receives these two trains, PathComp measures only the ASND histogram of the slower path, and uses Algorithm EST-RATIO2 to estimate the capacity ratio.

Parameter Setting The block size B should be much larger than the capacity ratio γ, because B limits the maximum ASND between two consecutive packets on the slower path. By default, PathComp sets B to 20 packets, which is larger than most typical ratios, such as 2 and 10.

The interval Δt should be long enough in order to sufficiently separate different packet

blocks, but should not be too long so that most packets on the faster path can still overlap with the packets on the slower path. By default, PathComp sets Δt to $2T_f/(L/B) = 2BT_f/L$, so that the average transmission rate of all packets is approximately reduced by half and the total transmission time of all packets is approximately doubled. T_f is the time for *RCV* to receive the packet train from the faster path in Phase I.

PathComp checks whether Δt is too long or too short as follows. If less than half of the packet blocks on the faster path overlap with the packet train on the slower path, it is likely that Δt is too big. If the ASND histogram of the faster path contains very few large ASND values (e.g. $\delta \geq 5$), it is likely that Δt is too short. In these cases, PathComp adjusts the interval Δt and re-sends the packets.

As an example, Figure 3.18 shows the original *SNDa* histogram with multiple peaks obtained using the packet train on the faster path, and Figure 3.20 shows the new *SNDa* histogram obtained using packet blocks on the faster path. We can see that in the new histogram, there are still two peaks, but that there is a peak at $\Gamma = 4$, and it is the highest peak.

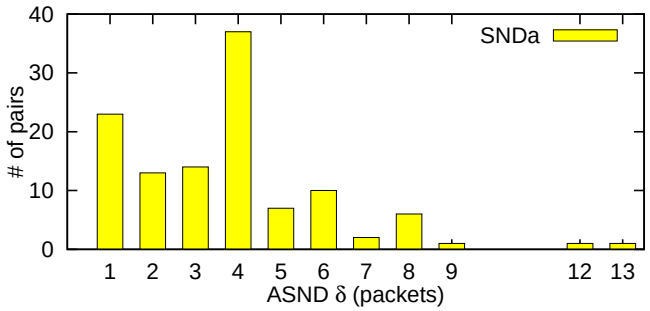


Figure 3.20: The difference between this figure and Figure 3.18 shows the effectiveness of Phase II in the presence of cross traffic.

3.6.2 Packet Time Information used in PathComp

PathComp uses only two types of coarse packet time information: the $\triangle RTT$ between two paths, and the time T_f for *RCV* to receive the packet train from the faster path in Phase I. None of them needs to be accurately measured.

$\triangle RTT$ is used in both Phases I and II so that the packets on both paths will arrive at *RCV* at approximately the same time. The inaccuracy in measuring $\triangle RTT$ can be mitigated by using longer packet trains.

T_f is used in Phase II to calculate block interval Δt. Note that time T_f is the time for receiving a train of L packets, not a single packet. Therefore, due to the relatively large value of L (e.g., 500), T_f is a relatively long time (e.g., 0.6 ms at 10Gbps). In addition, too large or too small interval Δt due to the inaccuracy in measuring T_f will be detected and adjusted by PathComp in Phase II.

3.6.3 An Implementation Challenge: RSS and IC

Two features of high-speed NICs may interfere with PathComp: Receiver Side Scaling (RSS) and Interrupt Coalescence (IC). Each of them alone doe not affect PathComp, but when both of them are enabled, they greatly interfere with PathComp. Below we explain the reasons and our solution.

RSS [38] is a relatively new NIC feature to allow a NIC to balance interrupts among multiple CPUs in a computer. RSS distributes incoming packets into different NIC Rx queues according to their hash values calculated using the packet information, such as source IP. As a result, the probing packets from two different senders are placed into different NIC Rx queues and handled by different CPUs on *RCV*. IC [25] is a NIC feature to reduce the CPU load by generating an interrupt for a group of packets instead of each packet.

When an interrupt is generated as each packet arrives (i.e., IC disabled), RSS alone does not affect PathComp because the interrupt sequence follows the packet arrival sequence. When there is only a single NIC Rx queue (i.e., RSS disabled), IC along does not affect PathComp because IC changes only the packet arrival times but not the packet arrival sequence. However, when both RSS and IC are enabled, they greatly interfere with PathComp as illustrated in Figure 3.21. Packets from different senders are placed into different NIC Rx queues, and an interrupt is generated only for a group of packets from a Rx queue. As a result, the packet arrival sequence measured by PathComp is different from the original packet arrival sequence at the NIC.

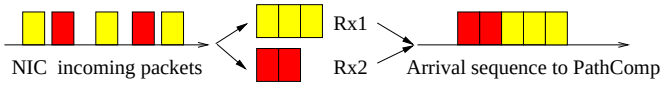


Figure 3.21: Impact of RSS and IC on the packet arrival sequence.

A simple solution is IP address spoofing. We modify the packet source IP address of one sender to the same as that of the other sender, in order to conceal *RCV* that all packets are from the same sender. *RCV* therefore places all packets to the same Rx queue. Although packets with a forged source IP address may be filtered by some firewall, this is a more practical solution compared with disabling either RSS or IC on *RCV*. We have successfully tested this solution on our campus network, Amazon EC2 [6], and PlanetLab [52].

Figure 3.22 shows the *SNDa* histogram when both RSS and IC are enabled. It is obtained with exactly the same testbed setting (including cross traffic, RSS, and IC) as Figure 3.13, except that the latter uses IP address spoofing. We can see that Figure 3.22 is greatly different from Figure 3.13. That is, without IP address spoofing, RSS and IC greatly change the packet arrival sequence.

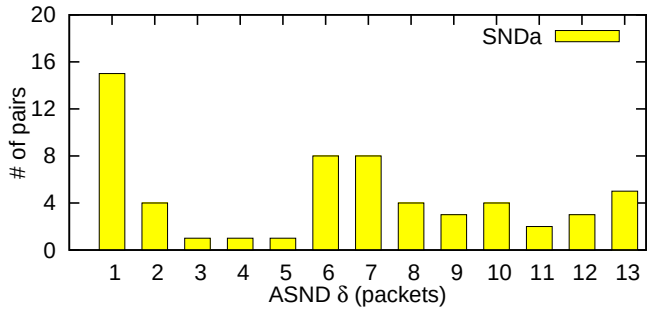


Figure 3.22: The difference between this figure and Figure 3.13 shows the impact of RSS and IC on the histogram.

3.7 Evaluation

In this section, we evaluate PathComp using our lab testbed, our campus network, and Amazon EC2.

3.7.1 Testbed Results

We conduct the following three groups of testbed experiments to evaluate PathComp with default parameters. For each experiment, we run it for 50 times, and report the average with a 95% confidence interval. The emulated network topology is the same as the one shown in Figure 3.7 but with different link capacities. We use Linux tbf with the minimum token burst size to emulate a link capacity, except 100 Mbps, 1 Gbps, and 10 Gbps. We notice that Linux tbf on our testbed can only emulate up to 1.6 Gbps links due to limited system capability. Thus, the maximum link capacity in our testbed experiments is 1.6 Gbps, except 10 Gbps.

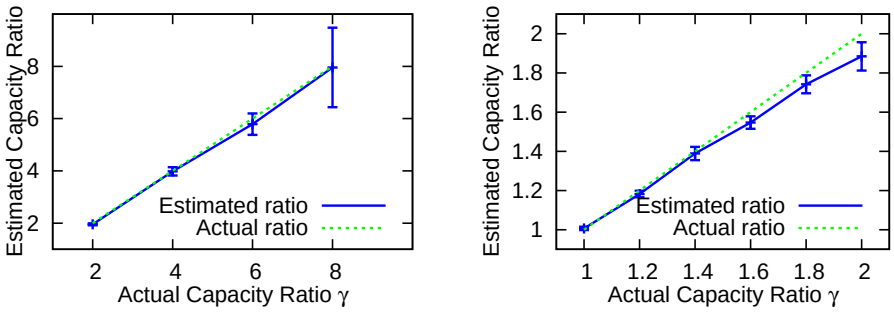
Group 1 - Impact of Large Capacity Ratios: This group of experiments study the accuracy of PathComp when two paths have a capacity ratio at least 2. For path *a* in Figure 3.7, we set $C_{a1} = 500$ Mbps, $C_{12} = 200$ Mbps, $C_{25} = 1$ Gbps, and thus the capacity of path *a* is $C_a = C_{12} = 200$ Mbps. For path *b*, we set $C_{b3} = 1.6$ Gbps, $C_{34} = 400$ Mbps to 1.6 Gbps, $C_{45} = 10$ Gbps, and thus the capacity of path *b* is $C_b = C_{34}$. Therefore, the

capacity ratio $\gamma = C_b/C_a$ varies from 2 to 8.

The estimated capacity ratios are shown in Figure 3.23a, where each link marked in Figure 3.7 has 30% cross traffic. We can see that PathComp can accurately measure these large capacity ratios. The large confidence interval at $\gamma = 8$ is partially because that Linux tbf has almost reached its max performance limit on our testbed.

Group 2 - Impact of Small Capacity Ratios: This group of experiments study the accuracy of PathComp when two paths have a capacity ratio no more than 2. Path *a* has the same link capacities as in group 1, and thus the capacity of path *a* is still $C_a = C_{12} = 200$ Mbps. For path *b*, we set $C_{b3} = 1$ Gbps, $C_{34} = 200$ Mbps to 400 Mbps, $C_{45} = 1$ Gbps, and thus the capacity of path *b* is $C_b = C_{34}$. Therefore, the capacity ratio $\gamma = C_b/C_a$ varies from 1 to 2.

The estimated capacity ratios are shown in Figure 3.23b, where each link marked in Figure 3.7 has 30% cross traffic. We can see that PathComp can accurately measure these small capacity ratios. Even when $\gamma = 2$, the average estimated ratio is 1.88, and is very close to the actual ratio. Note that results with $\gamma = 2$ in Figures 3.23a and 3.23b are obtained using different link capacities (e.g., C_{45}) and then different amounts of cross traffic. In the latter, C_{45} is smaller, and thus its link is more congested. This is why the estimation error with $\gamma = 2$ in the latter is larger than that in the former.



(a) Capacity ratio in range (2, 8)

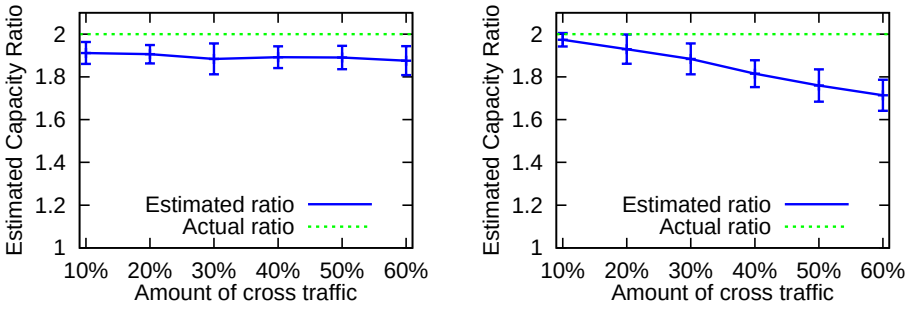
(b) Capacity ratio in range (1, 2)

Figure 3.23: Impact of large and small capacity ratios.

Group 3 - Impact of Cross Traffic: This group of experiments study the accuracy of PathComp under different amounts of cross traffic. We use the same link capacities as in group 2, except that we set C_{34} to 400 Mbps. Therefore, $\gamma = C_b/C_a$ is fixed to 2.

Figure 3.24a shows the estimated capacity ratios when the cross traffic on path *a* varies from 10% to 60% and that on path *b* is fixed to 30%. Figure 3.24b shows the estimated capacity ratios when the cross traffic on path *a* is fixed to 30%, and that on path *b* varies from 10% to 60%.

We can see that cross traffic on path *b* (i.e., the faster path) has a bigger impact than that on past *a* (i.e., the slower path). The reason is the probing traffic on path *b* is sent at a higher rate. With the same percentage of crossing traffic, path *b* is more congested than path *a*. For example, with 60% crossing traffic, the link utilization between *R4* and *R5* on path *b* can reach up to $0.6 + 400/1000 = 100\%$, but only up to $0.6 + 200/1000 = 80\%$ for the link between *R2* and *R5* on path *a*. This is consistent with our observation in Section 3.5, and this is also the motivation why PathComp in Phase II divides a long packet train into multiple short packet blocks on the faster path.



(a) Varying cross traffic on path *a* (b) Varying cross traffic on path *b*

Figure 3.24: Impact of cross traffic.

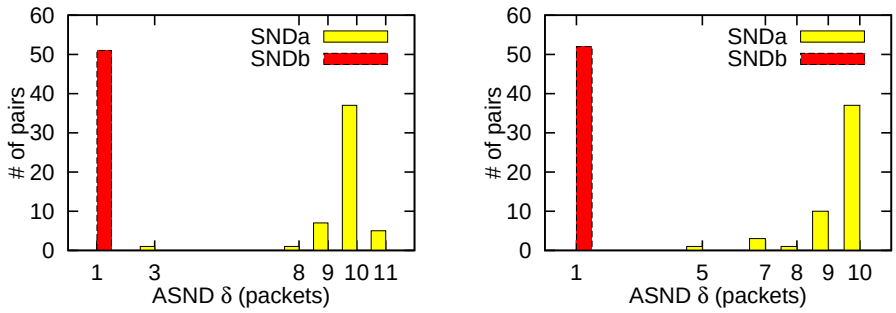
Remarks: We also run PathRate [15] on our testbed, which is one of the most well studied and widely used capacity estimation methods. However, it could not accurately estimate the capacity of a path on our testbed. For example, in Group 1, it reports a capacity

of 1100∼1400 Mbps (results of multiple runs) for path *a*, and reports an insufficient number of packet dispersion estimates for path *b*. This is partially due to interference of IC and Linux tbf.

3.7.2 Campus Network Results

We also evaluate PathComp using some servers in our campus network, where we know the network and server information.

Intra-Department Network: We choose three servers, denoted by *SNDa*, *SNDb*, and *RCV*, in our department. *SNDa* is connected to the department 1 Gbps network through a 100 Mbps switch, and both *SNDb* and *RCV* are connected to the department network through 1 Gbps Ethernet. Figure 3.25a shows the ASND histograms of *SNDa* and *SNDb*, and note that there are some ASND values at 9 and 11 packets which are caused by cross traffic. PathComp correctly estimates that the capacity ratio is 10 (corresponding to the peak at $\delta = 10$ packets). We also run PathRate, and it correctly estimates the capacity between *SNDa* and *RCV* as 100 Mbps, but it mistakenly reports the capacity between *SNDb* and *RCV* as 1900∼2100 Mbps.



(a) Intra-Department Network (b) Inter-Department Network

Figure 3.25: Campus network experiments.

Inter-Department Network: We choose three servers, denoted by *SNDa*, *SNDb*, and *RCV*, in three different departments in our campus network. *SNDa* has a 100 Mbps NIC,

and both *SNDb* and *RCV* have a 1 Gbps NIC. All three servers are connected to the campus 1 Gbps network. Each of the two paths passes four routers, and they share only the last router just before *RCV*. Figure 3.25b shows the ASND histograms of *SNDa* and *SNDb*, and PathComp correctly estimates that the capacity ratio is 10. We also run PathRate, and it correctly estimates the capacities of both paths: *SNDa*: 100 Mbps, and *SNDb*: 970∼990 Mbps.

3.7.3 Amazon EC2 Results

We also evaluate PathComp using VMs on Amazon Elastic Compute Cloud (EC2) [6], which is a very popular public cloud computing platform. The EC2 facilities are located at multiple locations, and we choose the one in the US West (Oregon) region that includes three zones. We select three micro instances from different zones as three senders denoted by *SNDa*, *SNDb*, and *SNDc*, and we select one medium instance as the receiver *RCV*.

We relatively compare the path capacities from the three senders to the receiver for 100 times, and Figure 3.26 shows the cumulative distribution function (CDF) of the estimated capacity ratios. PathComp reports that *SNDa* is slightly faster than *SNDb*, *SNDb* is about 2.2∼2.4 times faster than *SNDc*, and *SNDa* is about 2.4∼2.7 times faster than *SNDc*. We can also see that the results are highly consistent. For example, among estimated ratios between *SNDc* and *SNDa*, most of them are about 2.4∼2.7, and about 10% of them are smaller than 2.4. This is possibly due to the interference of VM scheduling, as micro instances are scheduled much more frequently than other types of instances.

In order to verify our estimated capacity ratios, we also run PathRate and iperf on EC2. PathRate reports that IC is detected and there is an insufficient number of packet dispersion estimates. Since this section considers the capacity of a path that indicates the short-term peak rate of the path, we use the iperf/tcp highest 1-second throughput in its first ten seconds. For *SNDa*, the iperf results are 540∼980 Mbps. For *SNDb*, the iperf results

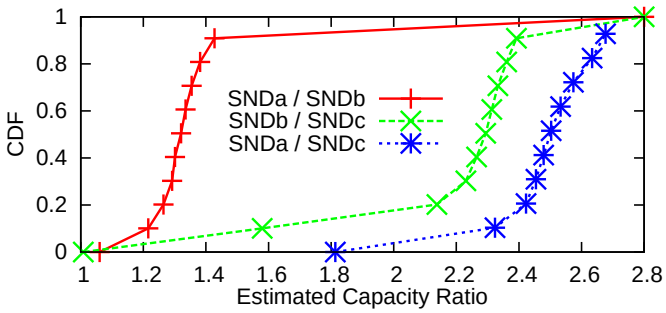


Figure 3.26: Amazon EC2 experiments

are 530∼760 Mbps. For *SNDc*, the iperf results are 280∼290 Mbps. The iperf results are consistent with our estimated capacity ratios. We guess that the *SNDa* capacity is possibly 1 Gbps, and the *SNDb* and *SNDc* capacities are limited possibly by the virtual machine capability and by rate limiters (e.g., a token bucket shaper).

Note that PathComp sends out much less traffic than iperf. For example, PathComp sends less than 1 MBytes from *SNDa*, whereas iperf sends 65∼117 MBytes just in the first second.

4 Rate Limiting in Public Clouds

4.1 Introduction

Because it is cheaper and more scalable to rent virtual machines (VM) than to buy servers, a growing number of corporate and government entities are choosing public clouds to run their applications because it is cheaper and more scalable to rent virtual machines (VM) than to buy servers. For example, Dropbox is a large IT company using Amazons S3 as file storage, and Amazons EC2 [7] instances to provide synchronization and collaboration[57]. Besides Amazon, there are also many other VM vendors such as Microsoft Azure [22] and Google Compute Engine [17].

Though it is more convenient and scalable to use VMs, cloud users share these resources with thousands of other users. Users want to know what they get for their money. For example, if current bandwidth cannot satisfy a large burst of customer requests, they can pay more to increase the bandwidth. However, current bandwidth estimation tools cannot be used in public clouds [56]. One main reason is that bandwidth provided to cloud users is maximum bandwidth, but not an actual available bandwidth. The actual available bandwidth may be lower than the maximum bandwidth if more users are sharing the network. Many reliable network structures have been proposed [24, 8, 18, 50, 47, 51]. Most of the new structures use rate limiters to shape bandwidth, such as the *tbf-like* rate limiter [21] and the *Xen-like* rate limiter [9]. For public clouds, an important question is how is bandwidth shared between multiple users.

To answer this question, this chapter offers detailed information about rate limiting in public clouds through a deep study of rate limiting in three popular public clouds: Amazon EC2, Microsoft Azure, and Google Compute Engine. We find below that the rate is limited in two aspects in public clouds.

- *First:* The traffic in public clouds is shaped by VM scheduling. A VM is scheduled out when it consumes its credit, and scheduled back after an interval. We define a VM's sending rate as the sending capability of a VM, which is mainly determined by the VM scheduling. In our study, we observe Amazon and Google micro instances with high sending rates and Azure micro instances with lower sending rate.

- *Second:* Two typical rate limiters are found in Amazon, Google and Azure public clouds. Azure instances use a *Xen-like* rate limiter to shape the traffic after it is shaped by VM scheduling in the first step. Amazon and Google clouds use a Linux *tbf-like* rate limiter to shape traffic.

Chapter Organization. We provide the background related to bandwidth allocation for a VM and rate limiters in section 4.2. We describe our measurement tool and data, and offer an overview of our work in section 4.3. Secontion 4.4 offers a discussion of a VM's sending rate in clouds. Section 4.5 measures the rate limiters in clouds. Section 4.6 comprises the conclusion to this chapter.

4.2 Background

4.2.1 Bandwidth allocation in VMs

We use Xen as an example to explain how bandwidth is allocated to a VM. The networking processsing path is shown in Figure 4.1.

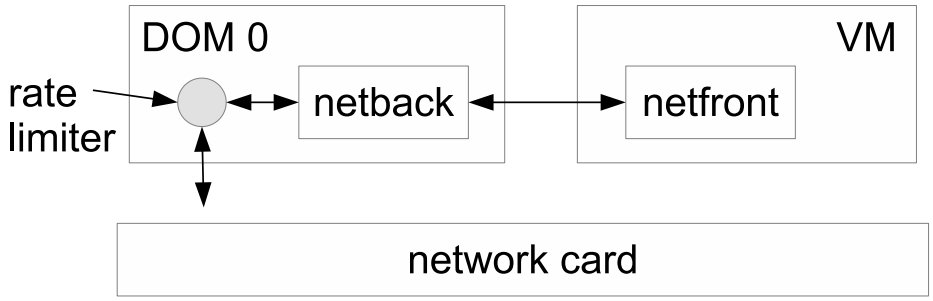


Figure 4.1: Networking processing path in Xen.

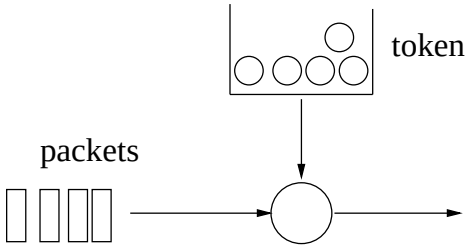


Figure 4.2: Token bucket model.

When a VM is scheduled to work based on a credit scheduling policy [13] it can use CPU resources to process packets, i.e. sending or receiving packets. The traffic goes through DOM0 (also called hypervisor) by a netfront/netback channel [37]. DOM0 controls all VM traffic, and decides when to help a VM send a packet out and when to deliver a packet to a VM from outside. This function is implemented in a rate limiter. Each VM has a rate limiter in DOM0, and the rate limiter is different from each other based on its services. We consider two of the most popular rate limiters used in the literature: the Linux tbf-like rate limiter, and the Xen-like rate limiter. We will discuss them in the next section.

4.2.2 Rate limiting

The token bucket model is shown in Figure 4.2. A packet is allowed to pass when it gets a token. The token is removed from the bucket when it is allocated to a packet. The bucket size is the maximum tokens available for use. When the bucket is empty, all packets are buffered in order to wait for new tokens. Thus, packets are shaped to fit the token rate.

With the *Linux tbf-like rate limiter*, there are a large number of tokens at the beginning (the maximum is called *burst size b*). Packets can be sent out at a high rate, called *peak rate P*, as there are enough tokens allocated for the packets. Tokens are replenished at a lower rate, *token rate R*. When the tokens are consumed, packets are sent out at this token rate R. A Linux token bucket filter can be described by (P, R, b).

A *Xen-like rate limiter*, on the other hand, updates the token after an interval T. Each time the added token size is Δ. If Δ tokens are consumed in a short time, the packets are saved in a local buffer for the next token update period. Comparing it with the Linux tbf-like filter, which updates the available tokens when a packet arrives, the Xen rate limiter updates the token less frequently. A Xen rate limiter can be described by (T, Δ).

In our measurement, if the measured rate can be described by (P, R, b), i.e. a high constant rate P at the beginning and a lower constant rate R later, we consider the network to be using a tbf-like rate limiter. If the packet trace follows a (T, Δ) pattern, i.e. Δ packets are sent every interval T, we consider the network to be using a Xen-like rate limiter.

4.3 Data and overview

In this section we describe the data we use in this chapter and offers an overview of our work. We design a UDP probing tool including ProbeSnd and ProbeRcv. ProbeSnd sends UDP packets into a cloud network at a maximum rate, and ProbeRcv receives UDP packets sent by ProbeSnd. We record each packet's id, sending time and receiving time. We run our ProbeSnd and ProbeRcv on VMs from the selected public clouds.

Table 4.1 shows the clouds, selected instance types and short names used in this chapter.

Note that we are not going to try to determine which instance or cloud is better. This study only offers technical details of rate limiting in public clouds. Cloud users should not decide which VM instance to use based simply on our results.

Table 4.1: Public clouds and instance types

Public Cloud	Instance Type	Short Name
Google Compute Engine US-Central1	f1-micro	G_1
	n1-standard-1	G_2
	n1-standard-2	G_3
Microsoft Azure South Central US	basic-a0	M_1
	basic-a1	M_2
	basic-a2	M_3
Amazon EC2 US West (Oregon)	t2.micro	E_1
	m3.meduim	E_2
	m3.xlarge	E_3

The overview of our measurement is as follows:

(1), We use our UDP probing tool to measure each instance's *sending rate*. The main contribution to the sending rate is the VM scheduling. We describe the scheduling process using two periods the *active period*, and the *sleeping period*. A detailed study of *active period*, *sleeping period* and *sending rate* is in Section 4.4.

(2), We measure the rate limiters in the networking path. We use iperf for EC2 and Google Compute Engine instances as we find the burst size is very large for these instances. We find that a Linux *tbf-like* rate limiter is used in EC2 and Google Compute Engine G_1 instances in both directions. Moreover, G_1 gives its instances a large number of tokens every 20 seconds. By using our UDP probing tool, we find that Azure instances use a *Xen-like* rate limiter.

4.4 Rate limiting in a sender VM

In this section, we discuss a sender VM's performance. Rate limiting in a networking path is considered in the next section. We define the sending rate of a VM first, and then measure the sending rate in public clouds.

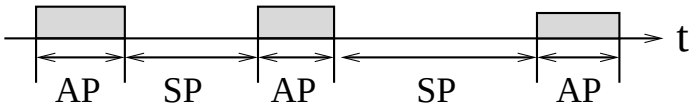


Figure 4.3: Active and sleeping periods.

4.4.1 How does a sender VM shape traffic?

When a number of packets are sent from a sender VM, the packets are sent out periodically as shown in Figure 4.3. A virtual machine sends traffic into the network in a given period, called *active period AP*. Then the virtual machine is suspended for a certain interval, called *sleeping period SP*. Figure 4.3 shows the packet trace from a sender VM. Packets are sent out in the active periods in shadow. No packets are sent during the sleeping periods because the VM is sleeping.

We define a VM's sending rate based on the active period and sleeping period. Consider the number of packets generated during an AP as N, packet size as $size$, and the length of AP and SP as A, and S. Sending rate of a virtual machine r is defined by,

$$r = \frac{N \times size}{A + S} \tag{4.1}$$

Summary: Sending rate is the capability of a VM of transmitting packets. The sending rate is the bottleneck of a path when it is lower than the path's bandwidth. In other words, the traffic from the VM is limited by the sending rate if the sending rate is too low .

4.4.2 Real data analysis regarding sending rate

Measurement Method, We use our UDP probing tool to probe a large number of packets into the network. We keep a record of the sending time of every packet. The intervals between every two packets are calculated. The intervals can be one of the following cases. (1)Dispersion time to send a packet counts more than 90% of all the time intervals. A

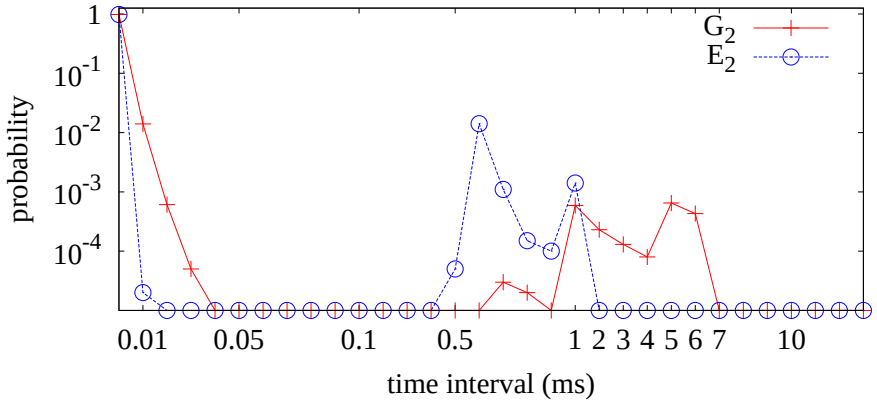


Figure 4.4: Probability distribution of time intervals between two consecutive packets

dispersion time is usually less than 10 μs as modern processors and NICs are very fast. (2) Interrupt coalescence (IC) or offloading time is longer than dispersion time. The interval falls in the range $10 \sim 100\mu s$. (3) VM sleeping period is the longest one and usually longer than 1ms.

Figure 4.4 shows the probability density distribution of time intervals from two instances G_2 and E_2. The sleeping period is the second peak area shown in Figure 4.4. The probability of the time intervals between 0.05 ms and 0.5 ms is very low as IC time is shorter than 0.05 ms and the sleeping period is longer than 0.5 ms. Hence, we select the second peak area as the sleeping period, i.e. if an interval is located in the second peak area we consider it a sleeping period.

Figure 4.4 shows that the measured sleeping period S_i lies in a large range. For example, for a G_2 instance in Figure 4.4, S_i is in the range of 0.6$\sim$6 ms. We can choose the peak to estimate the sleeping period. This is true for most instances such as Amazon EC2 instance E_2. However, some instances such as G_2 have two peak points. In other words, a G_2 instance may be scheduled out for 1ms or 5ms. In this case, we use the median of peaks to estimate the length of a sleeping period.

Figure 4.5 shows the average length of APs and SPs of different VM instances with a

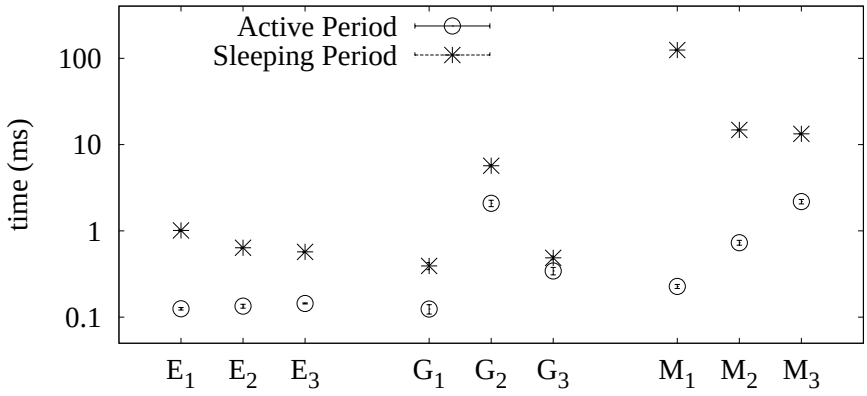


Figure 4.5: Average deviation of APs and SPs of different VM instances

95% confidence interval obtained using at least 30 runs. The instances from different zones in the public clouds are in Table 4.1. Figure 4.5 shows that the sleeping period can be as short as 400 μs, and as long as 120 ms. Overall, an active period is much shorter than a sleeping period.

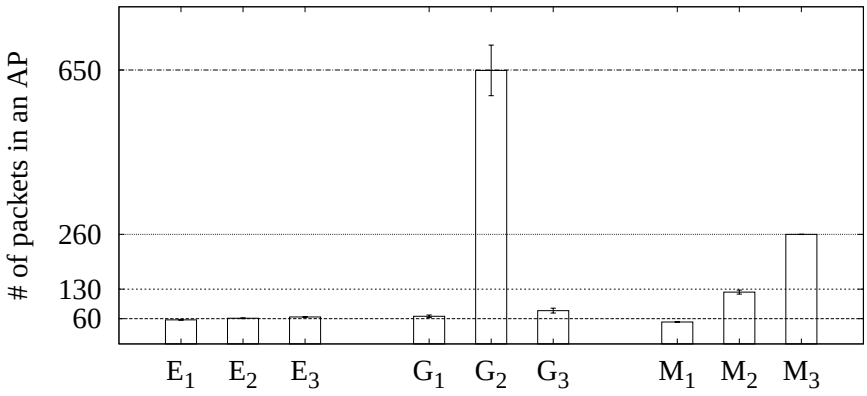


Figure 4.6: The average number of packets in one active period

Figure 4.6 shows the average number of packets in an active period with 95% confidence interval. We find that in most cases, the number of packets N_i in an active period AP_i does not change whether S_i is long or short. In a few cases, N_i increases if S_i is longer, such as G_2. Similar to SP estimation, we use the median of N_i and A_i, to estimate N and

A. Then r is calculated based on Equation 4.1.

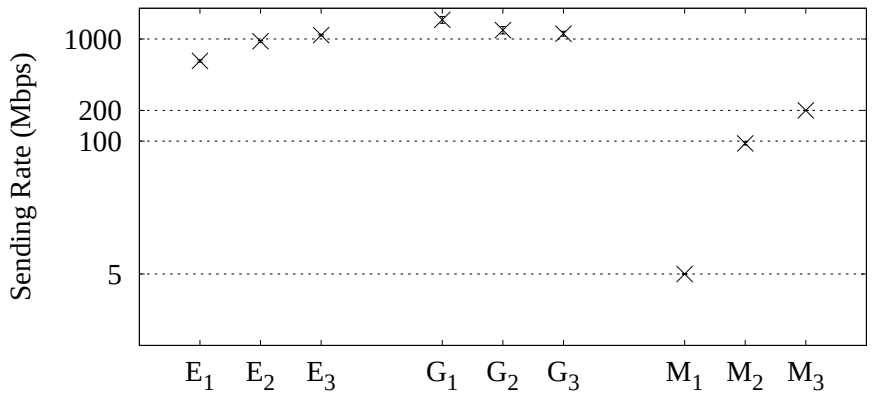


Figure 4.7: The sending rate for different VM instances

Figure 4.7 shows the average sending rate of different instances with a 95% confidence interval. We conclude that the sending rate of each instance is as follows.

- The sending rate of E_1 is a little below 1Gbps, and the E_2 and E_3 instances' sending rates are very close and above 1 Gbps.
- Google instances' sending rates are all above 1Gbps. G_1's sending rate is high because it offers "bursting capabilities" [17].
- Azure instances' sending rates are set to be the same as the token rate (we will discuss the token rate in next section).

4.5 Rate limiters in a network path

In this section, we describe the rate limiters observed in Amazon EC2, Google Compute Engine, and Azure data centers. We use a bandwidth measurement tool iperf in EC2 and Google Compute Engine. We use our UDP probing tool to analyze how Azure shapes the traffic. For each cloud, we offer an example of one instance, then answer the following

three questions: What type of rate limiter is it, why it is a rate limiter, and how are the other instances in the cloud shaped?

4.5.1 EC2's rate limiter

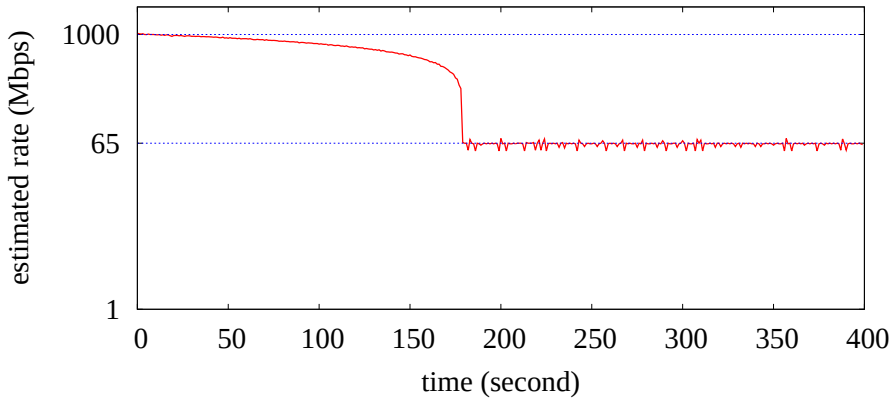


Figure 4.8: Iperf result for EC2 instance E_1

Amazon EC2 uses a **Linux tbf-like rate limiter** in the network. The iperf result demonstrate that the rate follows a $(P,\ R,\ b)$ pattern. In this experiment, the receiver is a large instance with high throughput, and the senders are the instances in Table 4.1. Figure 4.8 illustrates the iperf result of an instance E_1. This figure shows the probing rate is $P = 1Gbps$ at the beginning, and later it changes to $R = 65Mbps$, and stays at the rate R.

The property of the rate limiter in EC2 is that it shapes traffic in both directions with the same token rate. We use iperf to send packets from a high throughput instance to instance E_1, and the result curve is the same as the curve in Figure 4.8. The change from peak rate to token rate should take a time as short as G_1 instance in Figure 4.2, but it takes a long time to change the rate from 1 Gbps to 600 Mbps, and then changes to the token rate quickly. We think some techniques in the rate limiter make this change smoothly.

We measure different instances' token rates on Amazon EC2 by using iperf. The result is shown in Table 4.2. Not all E_1 instances can get the 1Gbps, which can be explained by

Table 4.2: Peak rate and token rate for EC2 instances

Instance type	Peak rate (Mbps)	Token rate (Mbps)
E_1	400∼1000	65
E_2	1000	300
E_2	1000	700

its sending rate below 1 Gbps.

4.5.2 Google's rate limiter

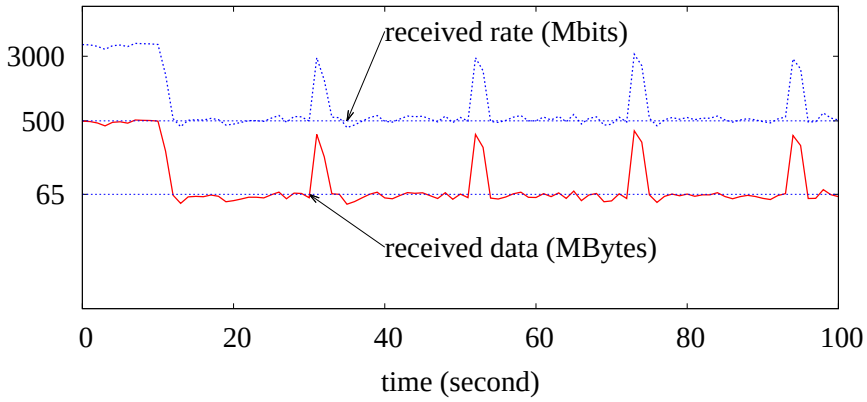


Figure 4.9: Iperf Result for Google instance G_1

G_1 uses a **Linux tbf-like rate limiter** in the network. The conclusion is based on the fact that the iperf result demonstrates that the rate follows $(P,\ R,\ b)$ pattern. As shown in Figure 4.9 about an iperf result of G_1 instance, the peak rate is $P = 3.2\ Gbps$ at beginning, and later changes to $R = 500Mbps$. We observe that the peak rate is varied when the receiver changes from n1-standard-2, to n1-standard-4, and n1-standard-8, The peak rate changes from 1.5 Gbps to 2.2 Gbps, and 3.2 Gbps. This is due to that a high performance instance can send data at a high speed.

The rate limiter in G_1 has two characteristics. First, it shapes traffic in both directions but with different token rates. Figure 4.9 shows the token rate for outgoing traffic is around 500Mbps. The incoming traffic is limited at the rate around 300 Mbps from the iperf result.

Second, the measured rate increases to a high rate every 20 seconds. The received data also increase every 20 seconds as shown in Figure 4.9. We think that Google offers a reward of a large number of tokens to the bucket every 20 seconds.

We also measured G_2 and G_3 by using iperf, and found that there is no change in the rate and the rate stays above 1 Gbps. We think there may be no rate limiters in the high performance instances G_2 and G_3, or we need other bandwidth measurement tools for high speed networks.

4.5.3 Azure's rate limiter

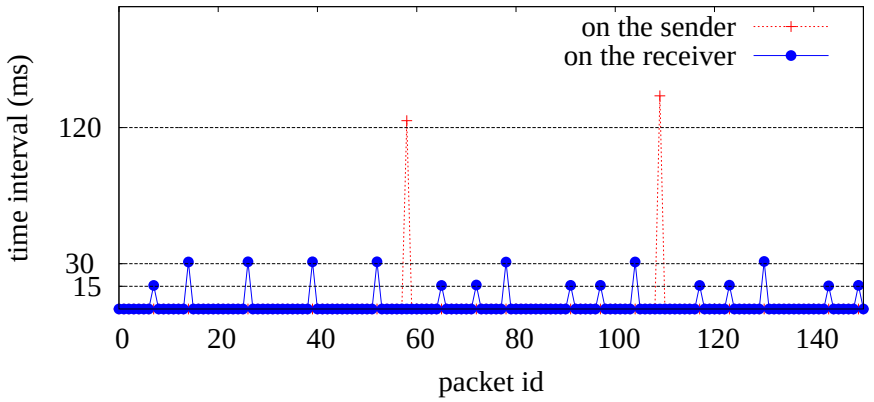


Figure 4.10: Time intervals on sender and receiver side for Azure instance M_1

In Azure, M_1 uses a **Xen-like rate limiter** in the network. The reason is the traffic follows a $(T,\ \Delta)$ pattern. Using our UDP probing tool, we measure the time intervals between two consecutive packets on both the sender side and the receiver side as shown in Figure 4.10. The large interval is around 120 ms on the sender side (sleeping period), and on the receiver side it is 15 or 30 ms. Moreover, we find that 6.5 packets are received on average when the interval is 15ms, and 13 packets are received when the interval is 30 ms. Hence, if $T = 15\ ms$, then $\Delta = 6.5$ packets; if $T = 30\ ms$, then $\Delta = 13$ packets.

Table 4.3: Peak rate and token rate for EC2 instances

Instance Type	T (ms)	Δ (packets)
M_1	15	6.5
M_2	15	130
M_3	15	260

Both intervals and packets follow the $(T,\ \Delta)$ pattern. We use the small one to show the instance's pattern.

The significant property of the rate limiter in Azure is that it only shapes the outgoing traffic, not the incoming traffic. When we change the sender from a low throughput instance to a high throughput instance, we find the estimated rate by iperf also increases on the same receiver. This proves the incoming traffic is not shaped.

Why is a Xen-like rate limiter still used for Azure instances when the rate has already been shaped by the sender VM? By separating the large number of packets from one active period into several small groups, the Xen-like rate limiter smoothes the traffic. A possible replacement of the Xen-like rate limiter is to schedule the VM every 15 or 30 ms. However, VM scheduling causes more overhead, compared with the rate limiter implemented in a hypervisor.

We measure different instances' token rates on Azure by using the UDP probing tool. M_1 is shaped by the Xen rate limiter, and M_2 and M_3 are shaped by VM scheduling. The result is shown in Table 4.3, in which the packet size is 1500 Bytes.

4.5.4 Summary of rate limiting in public clouds

We summarize the rate limiting and property in the selected instances in Table 4.4.

Table 4.4: All instances's rate limiters and property

Instance type	Rate limiter type	Properties
E_1, E_2, E_3	tbf-like	both directions, same rate
G_1	tbf-like	both directions, different rate
G_2, G_3	NA	high speed
M_1	Xen-like	single direction
M_2, M_3	VM scheduling	single direction

4.6 Conclusion

In this chapter, we described how rates are limited in public clouds. We measure the three most popular public clouds, Amazon EC2, Google Compute Engine, and Microsoft Azure, and found two main rate limiting methods. First, VM scheduling itself can shape the traffic, proven by the observations from the Azure instances. Second, rate limiters are used in the network paths to shape traffic. We considered two main rate limiters, the Linux tbf-like rate limiter and the Xen-like rate limiter. These rate limiters are found in the public clouds.

Our data was measured in September of 2014. This is a little different from our measurement in June 2014, as the cloud providers update their networks and instances. It is also possible that our result would be different from other studies done in the future. Even so, our measurement can be used to compare the degree to which public clouds improve their networking performance.

5 Packet Ticking: A New Timing Mechanism and Its Application in Bandwidth Estimation

5.1 Introduction

Current high-speed networking and cloud computing technologies have been using interrupt coalescence and virtual machine scheduling well for decades. However, the interrupt coalescence [60] and virtual machine (VM) scheduling [58, 12] can change probing packets arrival time. Thus it is hard to estimate available bandwidth (AB) accurately based on the packet arrival time. An accurate AB can be put to good use in rate-based streaming applications [2], task scheduling in data centers [4], resource allocation in grids/clouds using optical network architecture [61], and congestion control for TCP in data center networks [59, 54]. To provide an accurate AB, it is urgent to develop a more efficient AB estimation method that works in high-speed networking and virtual environment.

There are a few methods to handle the inaccuracy in arrival time caused by interrupt coalescence, but no methods have been proposed for inaccurate arrival time in virtual environment. The first class of methods use *smoothing techniques*. The packet arrival time comes with noise caused by interrupt delay. For example, IMR-pathload [26] uses the wavelet-based and average-based methods to eliminate the noise. PRC-MT [27] tunes the packet train size to reduce the impact of noise. BASS [60] smooths packet arrival times in an interrupt coalescence interval to improve AB estimation accuracy in high-speed networks. The smoothing techniques simply estimate AB based on packets' average arrival

rate in a short interval. If the interval is longer such as VM scheduling time, the smoothing techniques may not work. The new AB estimation method proposed in this chapter does not use smoothing techniques, and is the first method to estimate AB in a virtual environment.

The second class involves getting more accurate packet arrival time. For example, MinProbe [19] is based on special physical layer idle symbols that are transmitted and analyzed by a software-defined NIC [34]. Only through this software defined NIC, MinProbe can access the physical layer in real time, so it cannot be used in a general network card. Another example is PathComp [62], a recently proposed capacity measurement tool. PathComp relies on packet sequence information to compare the capacity of two paths by a capacity ratio. Packet sequence is not affected by interrupt coalescence and VM scheduling. Our proposed method is inspired by the work of PathComp and also uses the packet sequence information, but our method can estimate the AB of a path, whereas PathComp can only get the capacity ratio of two paths.

In this chapter, we propose a novel timing mechanism, called packet ticking, which can provide high-accuracy timing for measuring the arrival time of *probing packets* from a sender denoted by SND1 to a receiver denoted by RCV. Packet ticking relies on an additional sender, denoted by SND2, to send special *ticking packets* to the same receiver RCV. The sending rate of these ticking packets is carefully chosen to be sufficiently low so that their inter-packet gaps Δ are long and less affected by crossing traffic. On the receiver RCV, the inter-arrival gaps between the ticking packets are approximately Δ, and thus can be used as time units to measure the arrival times of the probing packets from SND1. Our analysis shows that packet arrival time calculated from the ticking packets can accurately detect increasing trend of *one way delay* (OWD), which is a key metric used in current AB estimation tools such as PathLoad [23].

The contribution of this chapter consists of three main points.

- We analyze current AB estimation tools and show that they do not work well with interrupt coalescence and VM scheduling. Using Pathload as an example, we show how it is affected by inaccurate actual arrival times.

- We propose a new timing mechanism, *packet ticking*, for AB estimation tools. To demonstrate the application of packet ticking, we design a new AB estimation tool **PacketTick**. PacketTick uses a similar algorithm to Pathload with the new timing mechanism. We analyze and discuss the impact of various factors, such as Δ, ticking packet size, and crossing traffic, on the accuracy of PacketTick.

- We compare the accuracy of PacketTick with Pathload in testbed and Amazon EC2. In our local testbed, we emulate a virtual network. The results in tested and wild network both validate that PacketTick can estimate AB more accurately than Pathload in virtual environment.

The chapter is organized in the following sections. In Section 5.2, we offer background on available bandwidth definition and related challenges. In section 5.3, we present the Pathload algorithm and explain why Pathload fails with actual arrival time. In Section 5.4, we propose packet ticking and how to use it to estimate AB. In Section 5.5, we analyze factors that affect packet tick, including packet train size and background traffic. In Section 5.6, we propose our AB estimation tool PacketTick. We evaluate PacketTick in testbed and Amazon EC2 in Section 5.7, and conclude our work in the end of this chapter.

5.2 Background

In this section, we present the background of AB estimation techniques, including the AB definition and two main AB estimation models first. Then we discuss the challenges in AB estimation.

5.2.1 Background on AB Estimation

When we talk about AB estimation, AB refers to the maximum bandwidth that users can access along a path from a sender SND1 to a receiver RCV, which may include multiple links. Denote the link i's capacity as C_i. Crossing traffic through the link i at time t is $A_i(t)$. Available bandwidth on the path SND1→RCV is the minimum one of all links' AB.

$$B(t) = \min_i \{C_i - A_i(t)\} \tag{5.1}$$

Instead of measuring $A_i(t)$ at a specific time, we estimate $A_i(t)$ as an average rate $\hat{A}_i(t)$ in a time range T.

$$\hat{A}_i(t) = \frac{1}{T} \int_t^{t+T} A_i(\tau) d\tau \tag{5.2}$$

There are two AB estimation models, Probing Gap Model (PGM) and Probing Rate Model (PRM). PGM sends a pair of packets with an inter-packet gap δ_s and receives with an inter-packet gap δ_r. If packets are sent at the maximum rate C, the delay difference $\delta_r - \delta_s$ is caused by crossing traffic. Thus, the crossing traffic can be calculated by

$$\hat{A}(t) = \frac{\delta_r - \delta_s}{\delta_s} \times C \tag{5.3}$$

PRM introduces self-induced congestion into the network. When packets are sent out at a rate lower than AB, they are received at a rate equal to the sending rate. If the sending rate is higher than AB, packets are delayed in the router buffer, so receiving rate is lower than the sending rate. The later packets are delayed longer than the earlier packets. PRM tools, such as Pathload [23], Pathchirp [49], and the system-theoretic approach [35], check packet delays to see if the sending rate exceeds AB. The new AB estimation tool proposed in this chapter uses the PRM model, specially a revised Pathload algorithm, to estimate

AB.

One way delay (OWD) is the total delay for a packet to transfer from SND to RCV. It includes each link's packet processing delay, queuing delay, transmission delay, and propagation delay. Among those delays, only queueing delay changes based on queue size. If we send a *packet pair* at a rate $R_1 > AB$, packets are delayed in a router buffer because they cannot be sent out at R_1 in the bottleneck link. The second packet delays a longer time because its delay includes the first packet's processing time, thus $OWD_1 > OWD_2$. Accordingly if we send a *packet train* (multiple packets or a group of packets), the related OWDs show an increasing trend if $R_1 > AB$. If $R_1 < AB$, a packet has been processed before the following packets enters in the same queue. Thus OWD_1 and OWD_2 are independent. The related OWDs of a packet train should show a stable trend. If we send packet trains at different rates, AB is the turning rate that OWD starts increasing.

5.2.2 Challenges in AB Estimation

We present two challenges for AB estimation, and we pay more attention to the second one because current AB estimation tools are not able to solve this challenge.

The first challenge for AB estimation is stochastic crossing traffic. All existing AB estimation tools aim to improve AB estimation accuracy under different crossing traffic. For example, Pathload [23] uses a binary search algorithm to find an AB range with which to approach the accurate AB. Pathchirp uses an exponentially spaced packet train to estimate AB. Our proposed tool PacketTick uses a *Pathload-like* algorithm. Consequently, PacketTick is as accurate as Pathload with the same crossing traffic. The main advantage over Pathload is a better timing metric.

The second challenge for AB estimation is the time requirement. For high-speed networks, it requires high-fidelity instruments to estimate a high-fidelity AB. It is impossible to get a high-fidelity timestamp based on current technology. Many factors may affect the

time accuracy, such as OS process scheduling, interrupt coalescence, and VM scheduling. We offer two case studies to show why the timing is inaccurate in estimating AB.

Case 1: Interrupt Coalescence (IC), Interrupt Coalescence delays an interrupt to inform packet arrival time. Usually multiple packets are transferred to the kernel in an interrupt. Figure 5.1 is an example in our testbed. We send packets at 1Gbps through a 10 Gbps network card. The inter-packet gaps should be 12 μs when the packet size is 1500 Bytes. Because the interrupt coalescence feature is enabled in the network card driver by default, packets are transferred to the kernel by group. The inter-packet gaps in a group are very small, and inter-packet gaps between groups are large. Researchers [5] try to introduce hardware taping to decrease the influence of interrupt coalescence. But using the interrupt coalescence feature to increase a network card's processing speed is unavoidable.

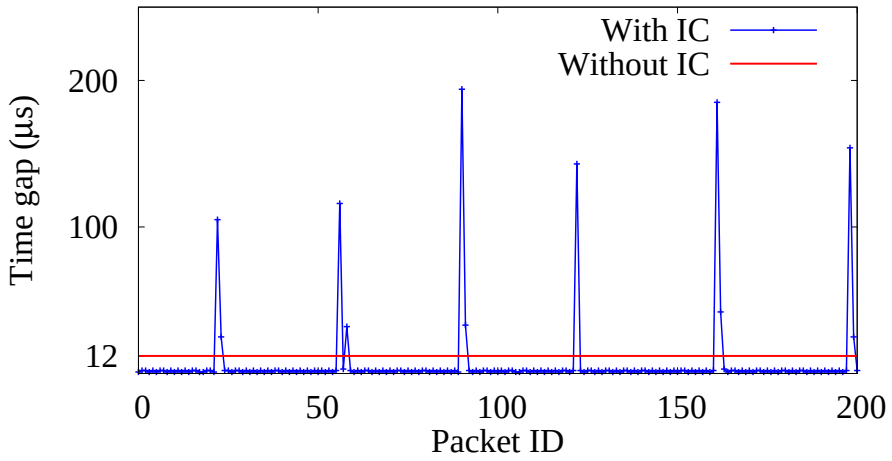


Figure 5.1: Inter-packet gaps for a packet train through a 10 Gigabit network card. The sending rate is 1Gbps, and the packet size is 1500 Byte.

Case 2: VM scheduling, a VM is periodically scheduled to share CPU, networking and other resources with multiple VMs residing in the same physical machine. When a VM runs out of its allocated resources, it is suspended and waits for the next allocated resources. Packets that arrive during the suspending period are buffered and delivered to

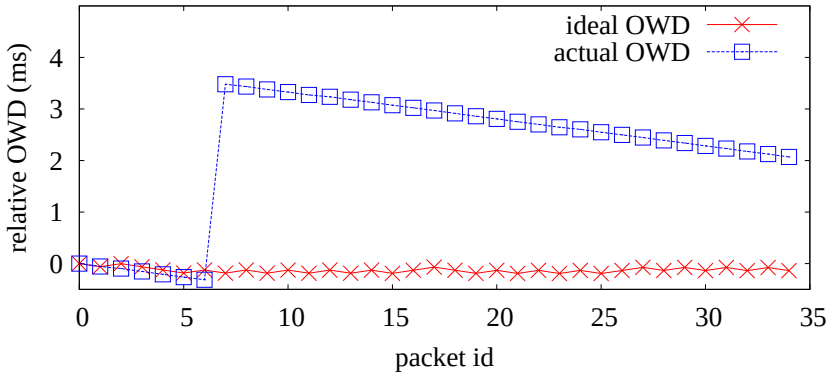
the VM when it is scheduled to work. The timestamp from the buffered packets cannot be used to estimate AB.

We use two VMs from Amazon EC2, one small instance as RCV and one medium instance as SND1, and measure the OWD between SND1 and RCV. To dismiss clock synchronization, we calculate a relative OWD, which is the OWD difference with the first packet's OWD. The relative OWD can be also used to check the OWD increasing trend.

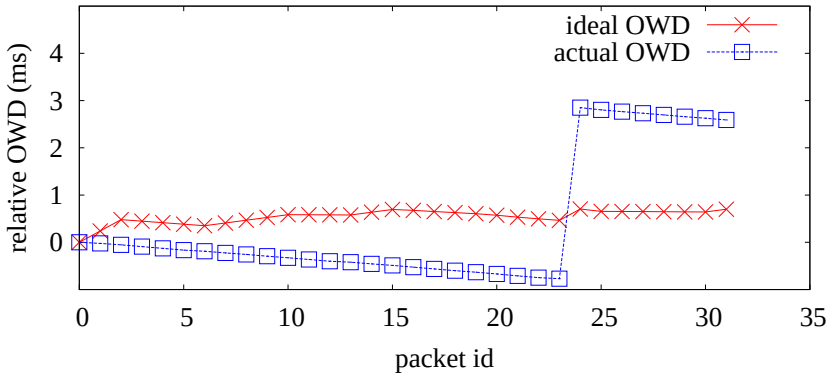
We send packets at two different rates, a very low rate, and a very high rate. OWD should be stable in Figure 5.2(a) and increasing in Figure 5.2(b). However, it is impossible to check the OWD increasing trend using the actual arrival time on RCV. Figure 5.2 shows that there is a large gap of about 4 ms in the OWD. This is caused by a VM scheduling. The following OWD decreases because it has a shorter queuing delay. The ideal OWD in Figure 5.2 is estimated by the new timing mechanism that we propose in this chapter.

(N_0, T_0) Traffic Pattern. Through the case studies, we found that the time information on RCV is not accurate enough to estimate AB in the above cases. Packets may be delayed for a given time T_0 before being delivered to RCV. To simplify our simulation, we modeled the packets received on RCV by a pattern (N_0, T_0), where N_0 is the number of packets continuously received by RCV before RCV is suspended for a T_0 interval. In the interrupt coalescence, N_0 and T_0 can be set by the parameters rx-frames and rx-usecs in a Linux system. In public clouds, N_0 and T_0 are based on VM types and workloads, and previous studies such as [56] validate the existence of the (N_0, T_0) traffic pattern in Amazon EC2.

A detailed analysis of N_0 and T_0 distributions is outside of the scope of this dissertation. We made a preliminary study using over 100 EC2 instances and 50 Azure instances during 2014 and 2015. The result is that N_0 is around 30$\sim$200, and T_0 can be 1ms, 2ms, 4ms, 8ms or the other intervals with different probabilities. In our evaluation section, we will set N_0 and T_0 to these values to simulate a real public cloud environment.



(a) OWDs should be stable



(b) OWDs should be increasing

Figure 5.2: Sending rate in (a) is 100 Mbps, and in (b) 500 Mbps.

5.3 Pathload

5.3.1 Overview of Pathload

Pathload is one of the most well studied and accurate AB estimation tools to estimate the AB along a path between two hosts SND1 and RCV. The process of estimation is (1) SND1 sends a packet train to RCV at a specific rate by setting the inter-packet interval of the packet train; (2) RCV receives all the packets, calculates each packet's one way delay (OWD) which is the total delay from sending to receiving a packet, and determines the next rate; (3) Pathload uses a binary search algorithm to find a range to estimate AB. We discuss

Table 5.1: Notation used in the chapter

Symbol	Description
p_i, D_i, s_i	Packet i, its OWD, and sending time
δ_i^r	Inter-packet gap between p_i and p_{i-1}
t_i	Ticking packet i, also used as its arrival time
r_i^I, r_i^A, r_i^P	Receiving time for p_i by ideal time, actual time, and packet tick
$S_{PDT}^I, S_{PDT}^A, S_{PDT}^P$	PDT calculated by ideal time, actual time, and packet tick
δ^s	Inter-packet gap at SND1
Δ, Δ_r	Inter-tick gap at SND2 and at RCV
R_1, R_2	Sending rate at SND1 and SND2

the OWD calculation and how to determine the next rate in Pathload.

OWD is the time difference between s_i and r_i for a packet p_i. We have three types of receiving times in this chapter, (1) r_i^I, the ideal time which is the accurate time that p_i arrives on RCV; (2) r_i^A, the actual time that users get from an OS function such as gettimeofday(); (3) r_i^P, a new time that we propose in this chapter.

Pathload uses the OWD increasing trend to check if the packet train sending rate is $R > AB$ or $R < AB$. Specifically, Pathload[23] uses the Pairwise Difference Test (PDT) and Pairwise Comparison Testing (PCT) criterion to check the increasing trend. PDT is defined by Equation 5.4, where D_i is the OWD of packet p_i. The range of S_{PDT} is $[-1, 1]$. If $S_{PDT} \to 0$, the OWDs are stable or independent. If $S_{PDT} \to 1$, the OWDs are increasing.

$$S_{PDT} = \frac{D_N - D_0}{\sum_{i=1}^{N} |D_i - D_{i-1}|} \tag{5.4}$$

Pathload changes the sending rate based on the increasing trend. If the OWDs are

increasing, which means the current sending rate is greater than AB, the new sending rate is going to be lower. If the OWDs are stable, which means the current sending rate is less than AB, the next sending rate should be greater than the current sending rate. A binary search process is used to find a range to estimate AB.

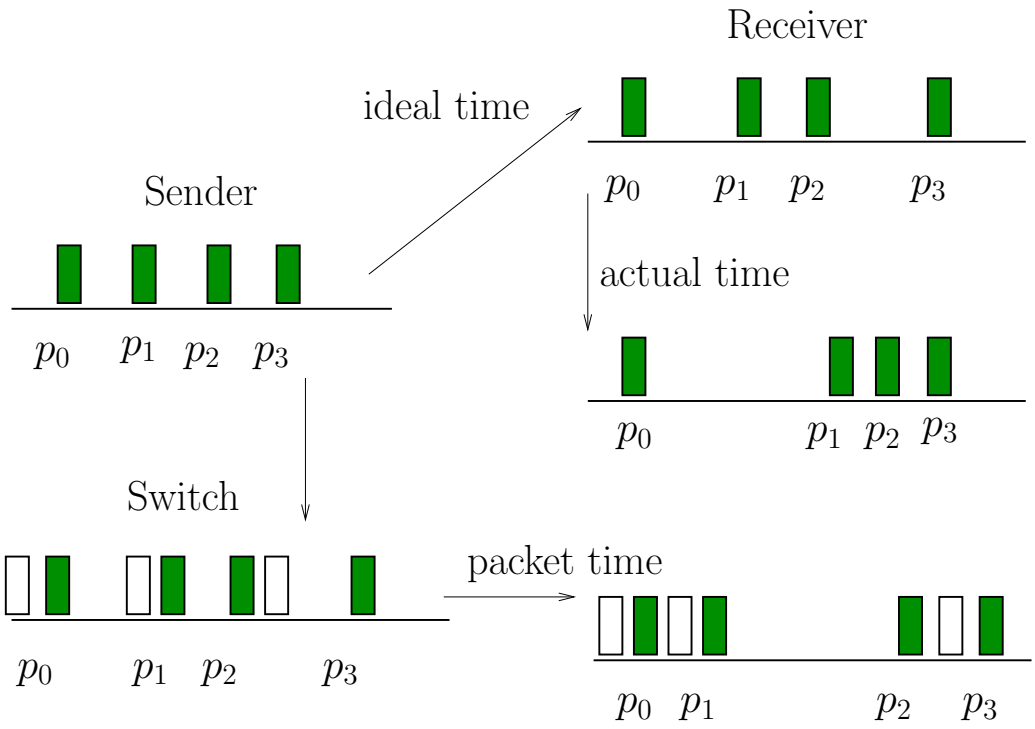


Figure 5.3: Three arrival times- ideal time, actual time, and packet ticks.

5.3.2 Pathload failures with actual arrival times

Pathload cannot estimate AB correctly with actual arrival times. Figure 5.3 shows the difference between ”ideal time” and ”actual time”. A packet arrives at the RCV network card at an ideal arrival time. However, the ideal time cannot be captured by a common network card or a user-level software. The time used in a user-level software is the actual time, which is often delayed by network card features such as interrupt coalescence or VM scheduling. The actual arrival time of four packets $p_0 \sim p_3$ is shown in Figure 5.3. The

inter-arrival time between packets p_0 and p_1 is caused by an interrupt delay. A short inter-arrival time is between p_1, p_2 and p_3 because they are transferred to the kernel together.

When the inter-arrival time is shorter, such as the one between p_1 and p_2, the related S^A_{PDT} is lowered. Denote $\delta^r_i = r^A_i - r^A_{i-1}$ as p_i's inter-arrival time on RCV, and $\delta^s = s_i - s_{i-1}$ as inter-sending time on SND. As we know, $D_i - D_{i-1} = r^A_i - s_i - (r^A_{i-1} - s_{i-1}) = r^A_i - r^A_{i-1} - (s_i - s_{i-1})$. Therefore we have

$$D_i - D_{i-1} = \delta^r_i - \delta^s \tag{5.5}$$

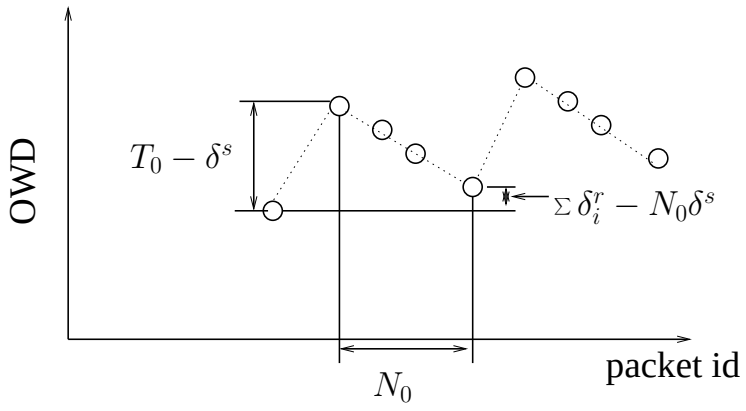


Figure 5.4: OWD of a (N_0, T_0) pattern packet train, every N_0 packets delay T_0.

For example, in a (N_0, T_0) pattern packet train, every N_0 packets delay T_0. As shown in Figure 5.4, the OWD difference between D_0 and D_1 is $T_0 - \delta^s$ based on Equation 5.5. For $p_i, 1 < i < N_0$, $D_i - D_{i-1} = \delta^r_i - \delta^s$. The difference between D_0 and D_{N_0} is $\sum \delta^r_i - N_0\delta^s$. Figure 5.4 is a special example that we can use the first N_0 packets to estimate S^A_{PDT}. The S^A_{PDT} is

$$S^A_{PDT} = \frac{D_N - D_0}{\sum_{i=1}^{N} |D_i - D_{i-1}|} = \frac{\sum \delta^r_i - N_0\delta^s}{2(T_0 - \delta^s) - \sum \delta^r_i - N_0\delta^s}$$

Therefore, S^A_{PDT} is lowered when T_0 increases. It is highly possible that S^A_{PDT} cannot check the increasing trend because T_0 is much longer than $\sum \delta^r_i - N_0\delta^s$.

5.4 Packet Ticking: a new timing mechanism

5.4.1 Overview of Packet Ticking

From the case study of Pathload, we have found that we need a new timing mechanism that can work even if the RCV is suspended. An obvious solution is to keep a record of the ideal arrival time for each packet. If users use special hardware such as DAG, it is easy to get the ideal arrival time or add the ideal arrival time to the packet header. However, this method requires specific hardware and a super user privilege.

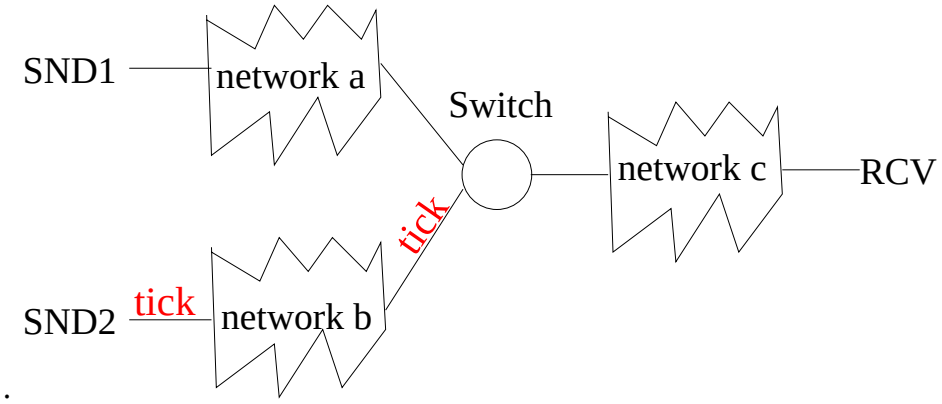


Figure 5.5: Packet tick example.

We propose *packet ticking*, a software solution that provides a new timing mechanism to record the arrival time of a packet. Packet ticking sends special ticking packets at a specific rate R_2 from a different path $SND2 \rightarrow RCV$. These ticking packets are called *ticks*. We call this timing mechanism *packet ticking*. To differentiate the packets from SND1 and SND2, below we call packets from SND1 *probing packets*, and packets from SND2 *ticking packets*.

A probing packet's arrival time is estimated by the closest ticking packet. For example, in Figure 5.6 which is part of Figure 5.3. Each ticking packet is denoted as a white rectangle, and each probing packet as a green rectangle. Then we use the latest ticking packet's arrival time to estimate a probing packet's arrival time. For example, $r_0^P = t_0$, where t_0 is

the arrival time of the first ticking packet. Though we do not know the exact time of t_0, we know the inter-ticking packet gap (or the length of one tick) $t_i - t_{i-1}$, which is assumed to be a constant and does not change after the ticking packets are sent from SND2. It can be calculated by the ticking packet size S and sending rate R_2. Denote the inter-ticking packet gap as Δ, then $\Delta = \frac{S}{R_2}$. A ticking packet's arrival time is estimate as

$$t_i = i\Delta + t_0$$

. We assume $t_0 = 0$, so $t_i = i\Delta$.

Probing packets and ticking packets intersect with each other at the Switch before the RCV as shown in Figure 5.5. So the first point at which probing packets and ticking packets meet is not at RCV, but at the Switch. After the Switch, the packet sequence is not changed. Hence, when we use packet tick to estimate the probing packets' arrival time, we estimate the packets' arrival time at the Switch. In this chapter, we assume that the bottleneck is always in *network a*, and the probing packets and ticking packets are not dropped in *network c*.

One advantage of packet tick is that we get time in the network. It is known that a user cannot get time from a switch directly unless he is a network administrator and the switch is also required to provide the packets' arrival time. Using the arrival time at a switch, we can estimate the AB along the path SND1 $\rightarrow$ Switch, which is the AB of the path SND1$\rightarrow$RCV because we assume that the bottleneck is in *network a*.

Another advantage is that the arrival time is saved in the packet sequence. As long as we get the sequence such as the one shown in Figure 5.6, we can estimate all probing packets' arrival time. A detail of how we get the arrival time and use it to estimate the AB is presented in the following section.

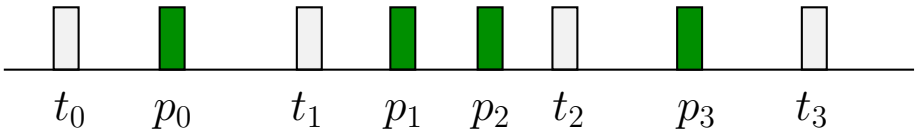


Figure 5.6: Packet sequence at RCV.

5.4.2 Use packet ticking to estimate AB

In this section, we present the use of packet ticking to estimate available bandwidth. The first step is to calculate the one way delay of probing packets based on ticking packets. The second step is to check OWD's increasing trend to calculate the next sending rate.

The procedure to calculate OWD using packet ticking is described as follows.

Procedure 1

- 1) Send probing packets from SND1 to RCV at the rate R_1, and send ticking packets from SND2 to RCV with an inter-tick gap Δ. Record each probing packet's sending time s_i. RCV receives the packets in a sequence that includes all probing and ticking packets.

- 2) Calculate a ticking packet's arrival time as $t_i = i\Delta$. A probing packet's arrival time is estimated as the latest ticking packet's arrival time. For example, in Figure 5.6, $r_0^P = t_0$, $r_1^P = r_2^P = t_1$ and $r_3^P = t_2$.

- 3) Calculate a probing packet's OWD by $D_i = r_i^P - s_i$.

One difference between the packet ticking and the actual times is that we do not use all of the probing packets. When more probing packets arrive in one tick we take the first packet to calculate its OWD. As the example shown in Figure 5.6, two probing packets p_1 and p_2 fall in one tick, and both are estimated to arrive at t_1 at the Switch. It is more accurate to estimate p_1's arrival time as t_1 than p_2's arrival time as t_1 because p_2 arrives later than p_1. Hence, we use p_1's OWD, and not p_2's.

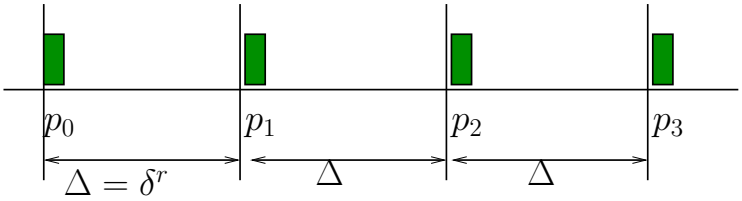


Figure 5.7: Example: one probing packet in one tick.

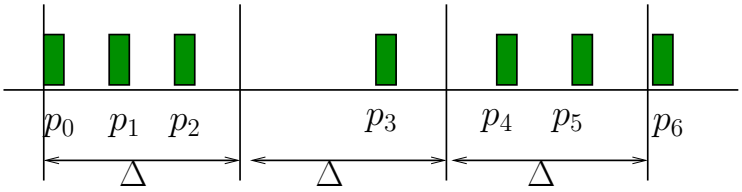


Figure 5.8: Example: multiple probing packets in one tick.

Pathload [23] is a state-of-art available bandwidth estimation tool, and uses PDT to check the OWD increasing trend. We have an equation similar to Equation 5.5 regarding to the OWD difference and probing packets' inter-packet gap δ^s. Consider m probing packets arrive in one tick, for instance the packet sequence is $t_i, p_{j+1}, ..., p_{j+m}, t_{i+1}, p_{j+m+1}$. Based on the above analysis, only p_{j+m+1} and p_{j+1} are used to calculate OWD. The OWD difference is calculated by Equation 5.6.

$$D_{i+1} - D_i = \Delta - m\delta^s \tag{5.6}$$

We use Theorem 1 to show that packet ticking can achieve the same accuracy as ideal times in some special cases, 1) $S^I_{PDT} = 0$ when OWDs are stable and 2) $S^I_{PDT} = 1$ when OWDs are increasing.

Theorem 1, $\exists\Delta$, when $S^I_{PDT} = 0$, $S^P_{PDT} = 0$; $\exists\Delta$, when $S^I_{PDT} = 1$, $S^P_{PDT} = 1$.

Proof. Let N be the total number of probing packets, which covers K ticks. D^I_i is the OWD calculated with idea times, and D^P_i with packet ticking.

(1), When $S^I_{PDT} = 1$, $S^P_{PDT} = 1$.

When $S^I_{PDT} = 1$, i.e. $\frac{D^I_N - D^I_0}{\sum_{i=1}^{N} |D^I_i - D^I_{i-1}|} = 1$, then $D^I_N - D^I_0 = \sum_{i=1}^{N} |D^I_i - D^I_{i-1}|$. This is true if and only if $\forall i, D^I_i - D^I_{i-1} > 0$. Based on Equation 5.5, $S^I_{PDT} = 1$ only when $\forall i, \delta^r_i - \delta^s > 0$, i.e. $\delta^r_i > \delta^s$. Let $\Delta \in [\delta_s, \min \delta^r_i]$. Then there is at most one packet in a tick on the Switch as shown in Figure 5.7. If there is only one packet p_i, then $D^P_i - D^P_{i-1} = \Delta - \delta^s > 0$. If there is no packet in a tick, for example the packet sequence as $t_j, p_i, t_{j+1}, t_{j+2}, p_{i+1}$, then the OWD difference between p_{i+1} and p_i is $D^P_{i+1} - D^P_i = 2\Delta - \delta^s > 0$. So we can get $\forall i, D^P_i - D^P_{i-1} > 0$. Therefore $S^P_{PDT} = 1$.

(2), When $S^I_{PDT} = 0$, $S^P_{PDT} = 0$.

When $S^I_{PDT} = 0$, i.e. $D^I_N - D^I_0 = 0$, $D^I_N - D^I_0 = \sum_{i=1}^{N} \delta^r_i - N\delta^s$ based on Equation 5.5, so $\sum_{i=1}^{N} \delta^r_i = N\delta^s$. When using packet tick, $D^P_N - D^P_0 = K\Delta - N\delta^s$. If we set $\Delta = \frac{N\delta^s}{K}$, $D^P_N - D^P_0 = 0$, then $S^P_{PDT} = 0$. □

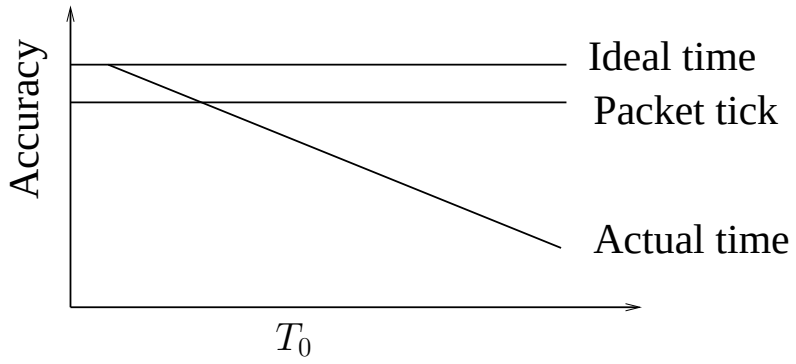


Figure 5.9: The accuracy of using packet tick, actual time and ideal time to estimate AB.

5.4.3 Comparison of Packet tick, Actual time and Ideal time

Figure 5.9 shows the accuracy of using packet ticking, actual times and ideal times to estimate AB. First, ideal arrival times achieve the highest accuracy. The accuracy depends on the Pathload algorithms if we use Pathload to estimate AB. Second, actual arrival times are very close to ideal times when the interrupt time T_0 is short. However, its accuracy

decreases as T_0 increases. Third, our proposed packet ticking can achieve a high accuracy and is not affected by T_0.

The accuracy achieved by packet ticking depends on inter-tick gap Δ and the crossing traffic's effect on Δ. Theorem 1 shows that under some special cases packet ticking can achieve the same accuracy as ideal times. However, the special cases require an ideal Δ which is hard to get if we do not know AB in advance. In Section 5.5, we analyze the accuracy of packet ticks.

5.5 Analysis of Packet Ticking

In this chapter, we analyse the precision and accuracy of packet ticking. Basically, Δ is the precision of packet ticking, and it determines the AB estimation accuracy. We analyze how Δ affects S^P_{PDT}, and discuss the factors that affect Δ.

5.5.1 Δ: precision and accuracy

The packet ticking precision is defined by Δ. If Δ is a very small value, i.e. $\Delta \rightarrow o$, the packet ticking becomes a computer clock with the highest precision. Simply speaking, when Δ is longer, it has a low precision, and more packets from SND1 are located in one tick. In Figure 5.7, Δ is small and there is one packet in each tick. In Figure 5.8, Δ is large and there are multiple packets in every tick.

The accuracy of packet ticking is the percentage of Δ remaining the same after ticking packets are sent from SND2. The inter-tick gaps affect by crossing traffic at each hop. Intuitively, when Δ is small, the packet ticking is less accurate and more affected by crossing traffic. When Δ is large, the packet ticking is more accurate.

Using a large Δ or a small Δ is a trade-off. The advantage of a smaller Δ is that the packet ticking is more precise. More packets are used to calculate OWD, so we can get

more information from the packet sequence. The disadvantage is causing more network overhead and less accuracy. Figure 5.10 is an example of Δ_r distribution measured in Amazon EC2. We select a medium instance (SND1) and a micro instance (RCV), and keep sending packets at a specific rate R_2. We use the inter-arrival time Δ_r on the receiver to estimate Δ_r on the Switch. We find that when the sending gap is set as Δ=120ms, 90% of Δ_r on the receiver is around 120ms. However, if the sending rate is high, less Δ_r on the receiver is equal to the sending gap. This example indicates that if we send ticks at a high rate, i.e. a smaller Δ, Δ_r is less accurate on the Switch.

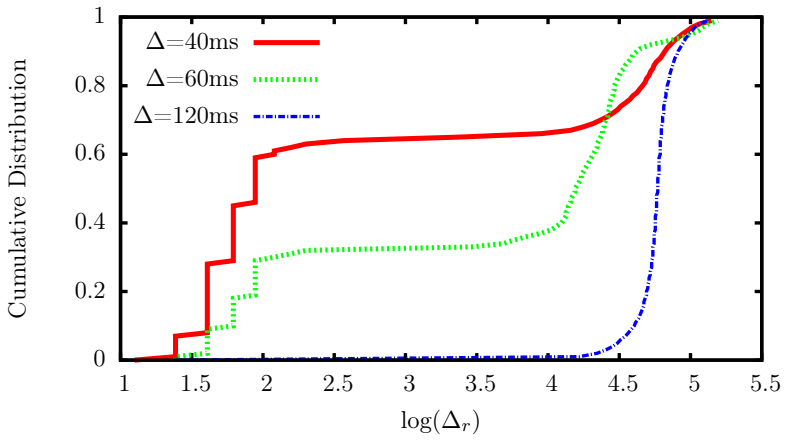


Figure 5.10: The distribution of different Δ measured in Amazon EC2.

The procedure to select a right Δ with high precision and accuracy is as follows.

Procedure 2

- 1) Sending K ticking packets from SND2 to RCV with an inter-tick gap Δ_0. Record each packet's receiving time r_i^A.
- 2) Calculate the inter-arrival gap Δ_r based on r_i^A and r_{i-1}^A.
- 3) If π% of Δ_r locates in the range $((1-\epsilon)\Delta_0, (1+\epsilon)\Delta_0)$, choose $\Delta = \Delta_0$. Otherwise, set $\Delta_0 = \lambda\Delta_0$, and go back to Step 1.

K is the number of probing packets we send, and the default value is set as 500, the same length used in Pathload. The initial Δ_0 is set as 20 μs, which is a very high precision for current 1 Gbps and 10 Gbps networks. Users can select a low sending rate or a larger Δ_0 if they have an idea of the upper bound of AB. We use inter-arrival gap Δ_r to estimate the gap at the Switch because this is the only time we can get. π is the confidence to use Δ, and we use 90% in our packet ticking. It may change depending on specific environments. For example 90% may not be achievable then users can use a small one. ϵ is the allowable variance for Δ_r from Δ, and we use 10%. λ is the parameter for finding a larger Δ_0, and we set default value as 1.5.

Users can adapt $(K, \Delta_0, \epsilon, \lambda, \pi)$ to find a suitable Δ, or set it as a constant if they know the AB range. For example, if AB>100Mbps, set $\Delta = 120\mu s$.

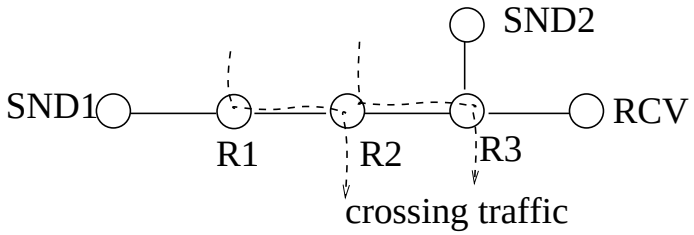


Figure 5.11: A multi-hop network used for simulation in NS2.

5.5.2 Impact of Δ on S^P_{PDT}

What is the difference between Δ_1 and Δ_2 if they both have high accuracy but a different precision, i.e. $\Delta_1 < \Delta_2$?

We use a simulation with NS2 to discuss the impact of Δ on S^P_{PDT}. Using a simulator, we can get the same S^I_{PDT} while changing different Δs. Figure 5.11 illustrates the network for the simulation. The capacity of all the links in the network is 1Gbps, and we use two one-hop crossing traffic 600Mbps and 400Mbps. So the AB is 400Mbps.

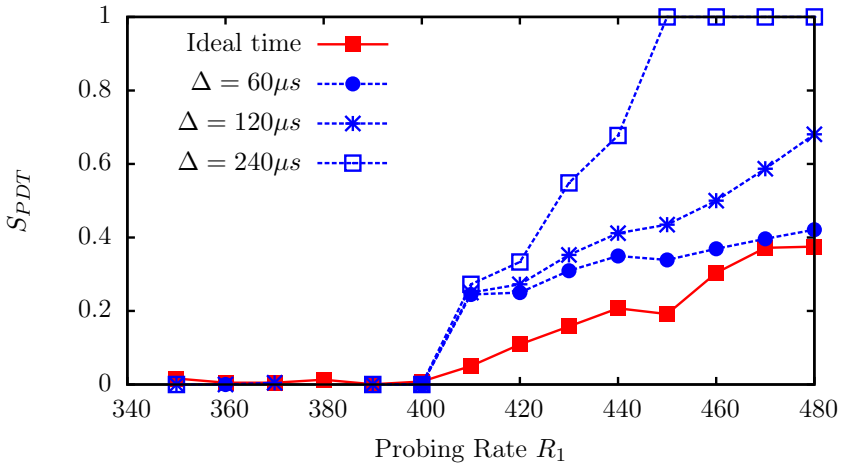


Figure 5.12: The comparison of S_{PDT} when using different Δs.

Figure 5.12 calculates the S_{PDT} with different Δs, 60 μs, 120 μs, and 240 μs, and the corresponding sending rate is 200Mbps, 100Mbps, and 50Mbps respectively. We change the probing rate on SND1. The ideal S_{PDT} is $S_{PDT} = 1$ when $R_1 > 400$, and $S_{PDT} = 0$ when $R_1 < 0$. However, S^I_{PDT} is the worst in our simulation because it is more sensitive to crossing traffic. Comparing $\Delta = 60\mu s$, 120 μs, and 240 μs, 240 μs is better than the others because it reaches $S^P_{PDT} = 1$ with the lowest probing rate $R_1 = 450$Mbps. The estimated AB is closest to the correct AB: AB=400Mbps.

On the other hand, a large Δ requires a large packet train for probing packets. In our simulation, when $\Delta = 60\mu s$, there are about 2 probing packets in one tick. When we use $\Delta = 240\mu s$, there are about 7 packets in one tick. We set the packet train size as $N = 200$. For $\Delta = 60\mu s$, this packet train covers $K = 100$ ticking packets. However, for $\Delta = 240\mu s$, it covers less than 30 ticking packets. That is why for a large Δ we have to send more probing packets to get the same number of ticking packets to estimate AB.

From this experiment, we find that a small Δ can estimate AB more correctly. A large Δ with a high precision is more sensitive to crossing traffic. However, small Δ requires a large packet train in AB estimation.

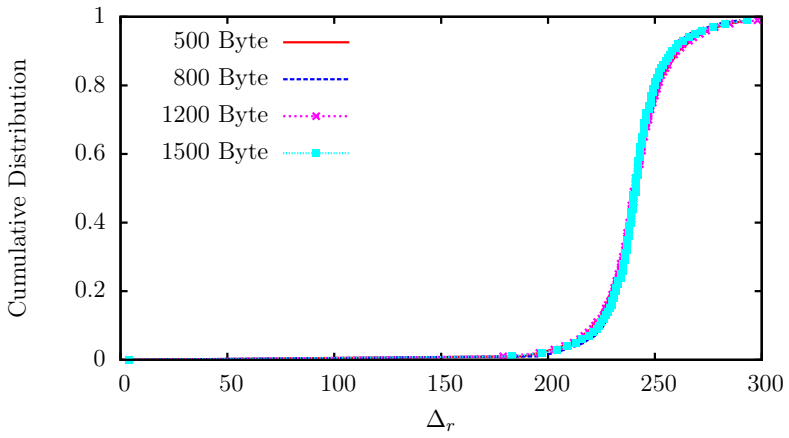


Figure 5.13: Effect of ticking packet size on Δ.

5.5.3 Ticking packet size

Δ is independent of the ticking packet size because Δ is comparably very large with the sending time, which is determined by the ticking packet size. A large ticking packet takes a longer propagation time to send at each hop. For example, in Figure 5.6, the ticking packet size determines the rectangle width. Δ is relatively longer than the rectangle width. So Δ is independent of the ticking packet size. We use an example in Amazon EC2, and the inter-packet gap is set as $\Delta = 240\mu s$. Figure 5.13 shows the distribution of Δ_r measured on RCV; it clearly indicates that Δ_r is independent with the packet size.

5.5.4 Actual Δ_r is inconstant

Though the inter-ticking packet gap Δ is set as a constant, the actual inter-ticking packet gap Δ_r on the Switch is inconstant because of crossing traffic in *network b*. However, we use the constant Δ to estimate the probing packets' arrival time. In this section, we analyze the effect of Δ_r.

There is a rich body of related work in the study of inter-packet gap distribution. For example, Piratala used a log-logistic model to describe a packet train's inter-packet gap

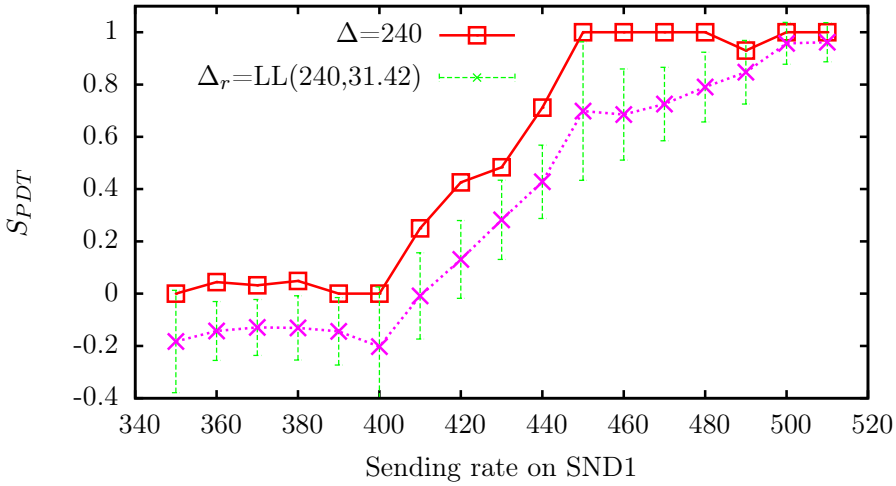


Figure 5.14: S^P_{PDT} when actual Δ is inconstant.

distribution [42]. This distribution is also validated by our measurement in Amazon EC2 network as shown in Figure 5.13.

The cumulative distribution function for a log-logistic distribution $LL(\alpha, \beta)$ is

$$F(x, \alpha, \beta) = \frac{x^\beta}{\alpha^\beta + x^\beta}$$

where α is the median of x, and β is a shape parameter. In our measurement, the inter-packet gap follows a log-logistic distribution LL(241.18, 31.42).

We use the same settings from our measurement in Amazon EC2 to simulate a packet train in NS2. We generate ticking packets with random gaps which follows a LL(240,31.42) distribution. Figure 5.14 compares the S^P_{PDT} calculated by constant Δ and Δ_r. This figure is the average result of 50 runs with a 95% confidential interval. Figure 5.14 indicates that S^P_{PDT} is lowered when it is calculated by Δ_r. The estimated AB based on Δ_r will be larger than one based on Δ. If we use $S^P_{PDT} = 0.5$ as a criterion to estimate AB, the final AB range is 400Mbps$\sim$430Mbps based on Δ, whereas the final AB is 400Mbps$\sim$450Mbps based on Δ_r.

The accuracy of Δ_r in estimating AB depends on the shape of LL, i.e. β. If β is smaller, more actual inter-ticking packets gaps fall out of $((1-\epsilon)\Delta, (1+\epsilon)\Delta)$. The corresponding S^P_{PDT} is smaller. The estimated AB is going to be a much larger range. This explains why in Procedure 2, to select a good Δ, we require that π% of G_i is located in the range $((1-\epsilon)\Delta_0, (1+\epsilon)\Delta_0)$.

5.6 PacketTick: a new AB estimation tool

Putting it all together, we designed a new AB estimation tool, called PacketTick. PacketTick is based on the new timing mechanism packet ticking, and uses the same algorithm as Pathload to estimate AB. Basically, the main difference between PacketTick and Pathload is that PacketTick requires an additional sender SND2 to send ticks to RCV. The times used in AB estimation is estimated from the ticking packets from SND2.

The process of PacketTick is described by as follows.

Procedure 3

- 1) Select a new sender SND2. The bottleneck of SND1 $\rightarrow$ RCV should be before the Switch.
- 2) Use Procedure 2 to select a good Δ to set the sending rate R_2 on SND2.
- 3) Send a packet train on SND1 at rate R_1, simultaneously sending ticking packets on SND2 at rate R_2.
- 4) Use Procedure 1 to calculate OWD, and use the Pathload algorithm to check the increasing trend. Stop if we get a proper AB range; otherwise set a new R_1 based on the Pathload algorithm and go to step 3.

In step 1, the selection of SND2 is discussed in Section IV-A. Users can use software such as traceroute to check the links in *network a*. Or users can use PacketTick to estimate the AB along the path SND1→Switch.

In step 3, to minimize the overhead caused by ticking packets, SND2 sends ticking packets to RCV at first, and then sends a request to SND1 to ask it to send a packet train at R_1. After SND1 sends the packet train, it sends a request to SND2 to stop sending ticking packets. This process can guarantee that ticking packets arrive at the Switch before probing packets, and SND2 stops sending ticking packets after all probing packets go through the Switch.

In step 4, the Pathload algorithm is reviewed in Section III. For more details of its implementation, users can refer to Pathload [23].

Moreover, PacketTick can implement the other algorithms such as Pathchirp [49] to estimate AB. Due to this chapter's limits, we only discuss PacketTick based on the Pathload algorithm.

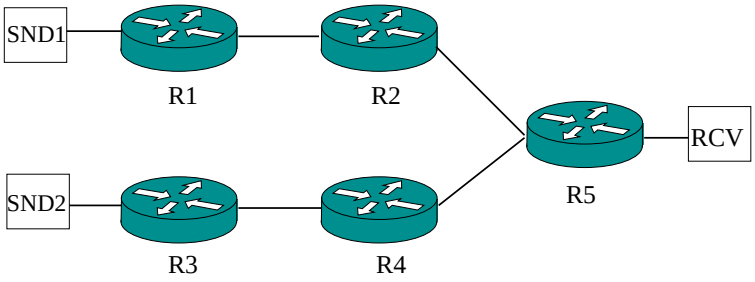


Figure 5.15: Testbed topology.

5.7 Evaluation

We evaluated the performance of PacketTick in our local testbed, and Amazon EC2. We also compared it with the existing AB estimation tool Pathload. The result in the figures is the average value of 50 times running with a 95% confidence interval.

5.7.1 Testbed setup

The network topology of our 10 Gigabit testbed is shown in Figure 5.15. Routers 1, 2, 3, 4 and RCV are Linux servers, and each server has two Intel 82599 10 Gigabit network cards. We use Linux tc to set different link capacities. Router 5 is a 10 Gigabit Netgear switch.

We use Poisson crossing traffic generated by MGEN. The bottleneck in the network is on Router 2. We add different crossing traffics before Router 2 and after Router 2, i.e. a crossing traffic from R1 to R2, and from R2 to R3. The computers generating crossing traffic and receiving crossing traffic are not shown in Figure 5.15.

We emulate a data center network environment in our testbed which follows a (N_0, T_0) traffic pattern. This emulation is implemented by modifying PacketTick and Pathload to send every N_0 packets as usual, and then keeps the program sleeping for T_0.

5.7.2 Impact of crossing traffic

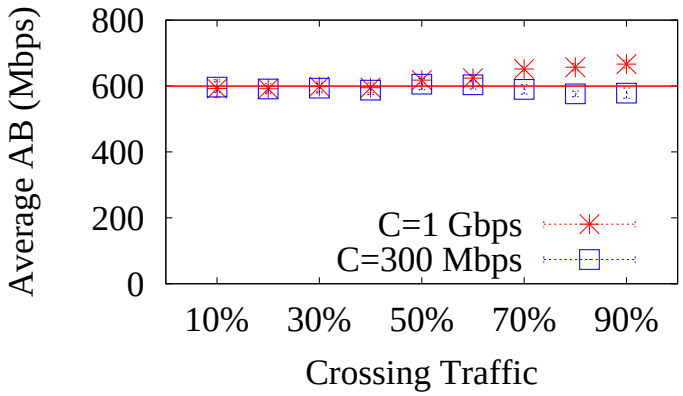


Figure 5.16: We change the crossing traffic in *network b* from 10% to 90%, and estimate the AB in *network a* by PacketTick. We consider two settings in *network b* where (1) capacity = 1Gbps, and (2) capacity = 300 Mbps.

Section 5.5.4 discuss the impact of the inconstant Δ on S_{PDT}. The inconstant Δ is caused by crossing traffic on the path SND2$\rightarrow$ RCV. In this experiment, we study the

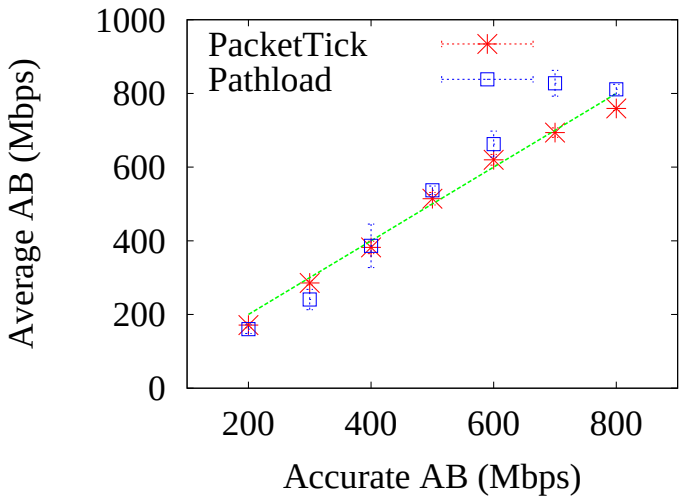


Figure 5.17: Average AB estimated by Pathload and PacketTick in our testbed.

effect of crossing traffic on AB estimation. We change the crossing traffic on the path SND2→ RCV. Then we run PacketTick to estimate AB on the path SND1→ RCV. The correct value is 600 Mbps. Figure 5.16 shows that PacketTick can correctly estimate AB when the crossing traffic is below 60%. When the crossing traffic is above 60%, estimated AB is close to the correct AB.

Another study is the different capacities in *network b*. Two capacities are considered, 1 Gbps and 300 Mbps. We find that PacketTick works better when the capacity is 300 Mbps, especially when the crossing traffic is above 60%. The reason is that Δ is large when C is small. A larger Δ can increase S_{PDT} values, resulting in a more correct AB based on the analysis in Section 5.5.2.

5.7.3 Testbed Evaluation

We implemented two experiments to compare Pathlaod and Packettick on our testbed. First, we compare the AB estimation accuracy with Pathload and PacketTick. Second, we simulate the data center environment (N_0, T_0); and compare the performance of Pathload and PacketTick.

Table 5.2: Compare Pathload and PacketTick

Actual AB	Send data		Time		Measured AB Range	
	PL	PT	PL	PT	PL	PT
200	7.40	19.4	8.63	6.35	72∼248	164∼178
300	21.9	20.5	13.9	5.56	190∼291	280∼291
400	11.0	13.9	8.71	4.32	322∼449	374∼390
500	19.9	11.5	11.7	3.56	471∼603	503∼524
600	11.0	15.4	8.68	4.7	499∼825	610∼629
700	11.0	14.5	8.66	4.44	743∼912	686∼700
800	9.20	9.86	8.05	3.09	775∼846	753∼765

(PT: PacketTick; PL: Pathload.)

Basic networks : In a basic network, we compare the performance of PacketTick and Pathload on the following aspects: average AB, overhead, time, and AB range. Figure 5.17 shows the result of average. Both Pathload and PacketTick provide users a range of AB (R_{min}, R_{max}). The average AB is the mean value of the range, i.e. $(R_{min} + R_{max})/2$. Figure 5.17 shows the result of the average $(R_{min} + R_{max})/2$ with a 95% confidence. From Figure 5.17, we can see that both Pathload and PacketTick can be used to estimate AB, which approaches the accurate AB. PacketTick works better than Pathload especially when the accurate AB is higher.

Comparing the estimated AB range, shown in Table 5.2, PacketTick estimates the AB range more accurately than Pathload. Pathload's AB range is very large. This can be explained by the Section 5.5.2, where $S^I_{PDT} < S^P_{PDT}$, resulting in a large AB range, and S^A_{PDT} is very close to S^I_{PDT}. When the accurate AB is higher, Pathload cannot estimate it correctly. For example, Pathload's result is 743∼912 Mbps when the accurate AB is 700 Mbps, but PacketTick is more accurate, and its result is 686∼700 Mbps. When AB=800 Mbps, the estimated AB is less accurate because we cannot generate packets correctly at a high rate.

We also studied the overhead and time to measure AB by Pathload and PacketTick. The result is shown in Table 5.2. PacketTick uses more probing packet than Pathload because we add one more path to send ticks. The additional cost of the ticks is not very heavy in view of fact that we use small-sized packets (500 Byte), and ticking packets are sent at a relatively low rate. Comparing the time needed to estimate AB, PacketTick takes less time than Pathload.

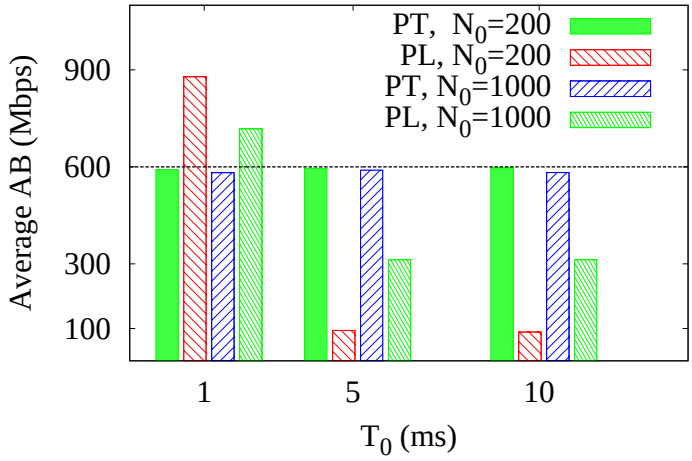


Figure 5.18: We simulate data center networks in our testbed, and compare the AB estimated by PacketTick and Pathload. The settings are $T_0 = 1$, 5, and 10 ms, and N_0=200, and 1000 packets.

Data center networks : We simulate a data center network in our testbed that follows the (N_0, T_0) traffic pattern. We modify the PacketTick and Pathload programs to suspend them for a time T_0 after receiving N_0 packets. Figure 5.18 shows the average AB estimated by Pathload and PacketTick when T_0=1, 5, and 10 ms, N_0=200, and 1000. The accurate AB is also set as 600 Mbps.

Pathload works only when N_0 is large and T_0 is small, i.e. $N_0 = 1000$ and $T_0 = 1$ ms. In the other cases $N_0 = 200$, and $T_0 = 5$ and 10 ms, Pathload estimates the AB as 100 Mbps. When $N_0 = 1000$, and $T_0 = 5$ and 10 ms, Pathload estimates the AB as 300 Mbps.

Incidentally, the result of Pathload in Figure 5.18 is the average of valid values, and the other invalid values include 0 Mbps or aborting by Pathload.

In contrast, PacketTick can estimate the exact AB in all settings of N_0 and T_0. This shows PacketTick can be used to estimate the (N_0, T_0) pattern traffic.

5.7.4 PacketTick in the wild

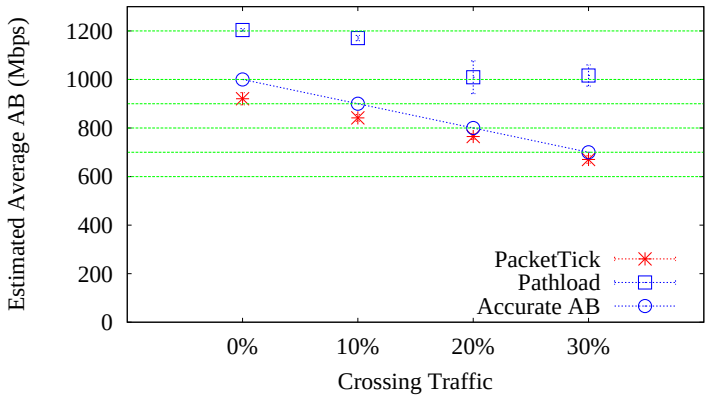


Figure 5.19: The comparison of PacketTick and Pathload in Amazon EC2 instances. The crossing traffic is changed from 0 to 300 Mbps.

We show that PacketTick can be used in public clouds to estimate AB. We select one small instance as RCV and one medium instance as SND1 from Amazon EC2. The two instances are in different zones, so they are not in the same physical machine and the path SND1$\rightarrow$ RCV goes through multiple links.

We use iperf to measure the capacity of the path. We found that EC2 uses a token bucket shaper. The peak rate is 1Gbps, and token rate is about 300Mbps. So we change the crossing traffic from 0 to 300 Mbps. The accurate AB is 1Gbps minus the crossing traffic we add from SND1 to RCV. The crossing traffic is generated by a light weight Poisson crossing traffic generator we developed.

Figure 5.19 shows the result of AB estimation from Pathload and PackeTick. The results of PacketTick are a little lower than the accurate AB. This may be caused by the

real crossing traffic besides the crossing traffic we probe in the network. Compared with Pathload, PacketTick's estimated AB changes linearly with the crossing traffic, and reflects the load of the crossing traffic we send. Pathload cannot be used to estimate AB in EC2.

5.8 Conclusions

In this chapter, we propose a novel way to use packets to convey the time information. We call these packets ticking packets, and they work the same as clock ticks. We designed a new available bandwidth estimation software, PacketTick, based on the packet ticks. Since PacketTick uses the time information in the network as estimated from packet ticks, it can correctly estimate the AB. The advantage is that the time information is saved in packet sequence, so it is not affected by receiver's interrupt delay or virtual machine scheduling. From our testbed study, we find that PacketTick can estimate AB more correctly than Pathload, though it requires a few more packets than Pathload. We also show that PacketTick can be used to estimate AB in the Amazon EC2 network. In the future, we implement the other algorithms such as Pathchirp to estimate AB.

6 Conclusion and Future Work

6.1 Conclusion

In this dissertation, we studied bandwidth estimation in virtual networks. The challenges of bandwidth estimation are multifold because of the properties of virtual networks. As per our analysis in this dissertation, the two main problems are rate limiters used in virtual networks and incorrect time information used in bandwidth-estimation software.

For the first problem, we designed a new software tool to estimate bandwidth in a virtual network with a tbf-like rate limiter. This case involves the most popular ones used, such as Amazon EC2 and Linux systems. The bandwidth is separated into two parts, peak rate and token rate. Our algorithms and software can estimate peak rates and token rates successfully. We tested this in different networks, including Amazon EC2.

For the second problem, we designed novel algorithms to use packet-sequence information to estimate bandwidth. Our idea was to compare two paths and determine which path is faster at transferring packets from a packet sequence. We cannot only determine which path is faster, but also get an accurate capacity ratio. Based on this idea, we designed two software tools to estimate capacity and available bandwidth separately. Through our experiments in different environments, we found the new software tools can estimate the bandwidth successfully.

6.2 Future Work

With the rapid development of cloud technologies, there are now many cloud vendors in the industry; these include Amazon Web Services, Cloudera, Hortonworks, IBM, Intel, MapR Technologies, Microsoft, Pivotal Software, and Teradata. They use very different technologies at various levels: hardware, software drivers, operating system, and management. The vendors are trying their best to provide a good service to customers. It is still very difficult to check whether there is a networking problem in the cloud networks even if the customer has good networking knowledge. A powerful networking tool is needed for customers to detect networking issues. The new tools proposed in this dissertation aim to detect networking problems in clouds. However, there are still some improvements and enhancements needed for these tools, which are construed as future work.

The first improvement is PacketTick. In PacketTick, SND2 is required to send packets at a fixed time step. We used a very slow traffic rate to ensure that the traffic is not affected by VM scheduling. However, there is no 100% guarantee that SND2 is scheduled out during the measurement. For our future network, we can implement a specific packet for VM that can go through DOM0 at a fixed time step. This packet will be sent out at real time and not affected by VM scheduling, so it can further improve our available bandwidth estimation. Moreover, we can implement the SND2 as a special server in a cloud that sends ticking packets to RCV when a cloud user requests an estimation of available bandwidth

We provide efficient bandwidth-estimation tools both for cloud users and cloud vendors. However, cloud vendors still cannot provide the best VM allocation for cloud users because the bandwidth usage is unknown when the vendors try to allocate VMs to cloud users. For our future work, we can use our current bandwidth-estimation techniques to estimate AB. Then we can manage the network more efficiently by various techniques, such as changing the routing to schedule traffic to idle links from busy links.

Bibliography

[1] B. Adamson. The mgen toolset, 2015. `http://www.nrl.navy.mil/itd/ncs/products/mgen`.

[2] S. Akhshabi, L. Anantakrishnan, A. Begen, and C. Dovrolis. What happens when http adaptive streaming players compete for bandwidth? In *Proceedings of NOSSDAV*, Toronto, Canada, June 2012.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM Computer Communication Review*, 38(4):63–74, 2008.

[4] M. Alicherry and T. Lakshman. Network aware resource allocation in distributed clouds. In *Proceedings of IEEE INFOCOM*, Orlando, FL, March 2012.

[5] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *Proceedings of USENIX NSDI*, San Jose, CA, April 2012.

[6] Amazon Web Services, Inc. `http://aws.amazon.com/`.

[7] Amazon Web Services, Inc. Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2/`.

[8] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predicatable datacenter networks. In *Proceedings of ACM SIGCOMM*, Toronto, Canada, August 2011.

[9] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of ACM SOSP*, New York, October 2003.

[10] R. Carter and M. Crovella. Measuring bottleneck link speed in packet-switched networks. *Performance evaluation*, 27:297–318, October 1996.

[11] L. Chen, T. Sun, B. Wang, M. Sanadidi, and M. Gerla. PBProbe: A capacity estimation tool for high speed networks. *Computer Communications*, 31(17):pp. 3883–3893, November 2008.

[12] L. Cheng and C. Wang. Defeating network jitter for virtual machines. In *Proceedings of IEEE Conference on Utility and Cloud Computing (UCC)*, Melbourne, Australia, December 2011.

[13] L. Cherkasova, D. Gupta, and A. Vahdat. Comparison of the three cpu schedulers in xen. *SIGMETRICS Performance Evaluation Review*, 35(2):42–51, 2007.

[14] D. Croce, E. Leonardi, and M. Mellia. Large scale available bandwidth measuremens: interference in current techniques. *IEEE Transactions on Network and Service Management*, 8(4):pp 361–374, December 2011.

[15] C. Dovrolis, P. Ramanathan, and D. Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Transactions on Networking*, 12(6):963–977, December 2004.

[16] A. Downey. Using pathchar to estimate Internet link characteristics. In *Proceedings of ACM SIGCOMM*, pages 241–250, Cambridge, MA, August 1999.

[17] Google Inc. Google cloud platform. `https://cloud.google.com/`.

[18] C. Guo, G. Lu, H. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y Zhang. SecondNet: a data center network virtualization architecture with bandwidth guarantees. In *Proceedings of ACM Co-NEXT*, Philadelphia, USA, November 2010.

[19] W. Han, L. Ki Suh, L. Erluo, L. Chiun Lin, T. Ao, and W. Hakim. Timing is everything: Accurate, minimum overhead, available bandwidth estimation in high-speed wired network. In *Procedings of ACM IMC*, Vancouver, Canada, November 2014.

[20] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE Journal on Selected Areas in Communications*, 21(6):879–894, September 2006.

[21] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, and P. Larroy. Linux advanced routing & traffic control. In *Ottawa Linux Symposium*, page 213, 2002.

[22] Microsoft Inc. Windows azure. `http://www.windowsazure.com/`.

[23] M. Jain and C. Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput. *IEEE/ACM Transactions on Networking*, 11(4):537–549, August 2003.

[24] V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, C. Kim, and A. Greenberg. EyeQ: practical network performance isolation at the edge. In *Proceedings of USENIX NSDI*, Lombard, IL, April 2013.

[25] G. Jin and B. Tierney. System capability effects on algorithms for network bandwidth measurement. In *Proceedings of ACM IMC*, Miami Beach, FL, October 2003.

[26] S. Kang and D. Loguinov. Imr-pathload: Robust available bandwidth estimation under end-host interrupt delay. In *Proceedings of IEEE PAM*, Cleveland, Ohio, April 2008.

[27] S. Kang and D. Loguinov. Characterizing tight-link bandwidth of multi-hop paths using probing response curves. In *Proceedings of IEEE IWQoS*, Beijing, China, June 2010.

[28] P. Kanuparthy and C. Dovrolis. Shaperprobe: end-to-end detection of ISP traffic shaping using active methods. In *Proceedings of ACM IMC*, pages 473–482, Berlin, Germany, November 2011.

[29] R. Kapoor, L. Chen, L. Lao, M. Gerla, and M. Sanadidi. Capprobe: a simple and accurate capacity estimation technique. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 67–78. ACM, 2004.

[30] H. Khandelwal, R. Kompella, and R. Ramasubramanian. Cloud monitoring framework. Technical report, Purdue University, 2010. `https://www.cs.purdue.edu/homes/bb/cloud/h-report.pdf`.

[31] K. LaCurts, S. Deng, A. Goyal, and H. Balakrishnan. Choreo: Network-aware task placement for cloud applications. In *Proceedings of ACM IMC*, Barcelona, Spain, October 2013.

[32] K. Lai and M. Baker. Measuring link bandwidths using a deterministic model of packet delay. In *Proceedings of ACM SIGCOMM*, New York, NY, August 2000.

[33] K. Lakshminarayanan, V. Padmanabhan, and J. Padhye. Bandwidth estimation in broadband access networks. In *Proceedings of ACM IMC*, pages 314–321, New York, NY, October 2004.

[34] K. Lee, H. Wang, and H. Weatherspoon. Sonic: Precise realtime software access and control of wired networks. In *Proceedings of USENIX NSDI*, Lombard, IL, April 2013.

[35] J. Liebeherr, M. Fidler, and S. Valaee. A system theoretic approach to bandwidth estimation. *IEEE/ACM Trans. Networking*, 18(4):pp 1040–1053, 2010.

[36] Y. Liu, Y. Guo, and C. Liang. A survey on peer-to-peer video streaming systems. *Journal of Peer-to-Peer Networking and Applications*, 1(1):18–28, March 2008.

[37] J. Martins, M. Ahmed, C. Raiciu, and F. Huici. Enabling fast, dynamic network processing with ClickOS. In *Proceedings of ACM HotSDN*, Hong Kong, China, August 2013.

[38] Microsoft. Introduction to receive side scaling. `http://msdn.microsoft.com/en-us/library/windows/hardware/ff556942%28v=vs.85%29.aspx`.

[39] Open VSwitch. An open virtual switch. `http://openvswitch.org/`.

[40] V. Paxson. End-to-end internet packet dynamics. In *Proceedings of ACM SIGCOMM*, France, September 1997.

[41] V. Paxson. On calibrating measurements of packet transit times. *ACM SIGMETRICS Performance Evaluation Review*, 26(1):pp 11–21, June 1998.

[42] N. Piratla, A. Jayasumana, and H. Smith. Overcoming the effects of correlation in packet delay measurements using inter-packet gaps. In *Proceedings of IEEE International Conference on Networks*, Nov 2004.

[43] L. Popa, P. Yalagandula, S. Banerjee, J. Mogul, Y. Turner, and J. Santos. ElasticSwitch: Practical work-conserving bandwidth guarantees for cloud computing. In *Proceedings of ACM SIGCOMM*, Hongkong, August 2013.

[44] R. Prasad, M. Jain, and C. Dovrolis. Effects of interrupt coalescence on network measurements. In *Proceedings of PAM*, France, April 2004.

[45] R. Prasad, M. Murray, C. Dovrolis, and K. Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE Network*, 17(6):27–35, November-December 2003.

[46] Rackspace Inc. Rackspace public cloud. `http://www.rackspace.com/cloud/`.

[47] S. Radhakrishnan, R. Pan, A. Vahdat, G. Varghese, and etc. Netshare and stochastic netshare: predictable bandwidth allocation for data centers. *ACM SIGCOMM Computer Communication Review*, 42(3):5–11, 2012.

[48] Ericsson Research. OpenFlow software switch. `http://cpqd.github.io/ofsoftswitch13/`.

[49] V. Ribeiro, J. Navratil, R. Riedi, R. Baraniuk, and L. Cottrell. Pathchirp: Efficient available bandwidth estimation for network paths. In *Proceedings of ACM SIGCOMM*, 2003.

[50] H. Rodrigues, J. Santos, Y. Turner, P. Soares, and D. Guedes. GateKeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. In *Proceedings of Workshop on I/O virtualization*, Portland,OR, June 2011.

[51] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *Proceedings of NSDI*, Boston, MA, March 2011.

[52] N. Spring, L. Peterson, A. Bavier, and V. Pai. Using PlanetLab for network research: myths, realities, and best practices. *ACM SIGOPS Operating Systems Review*, 40(1):pp 17–24, January 2006.

[53] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proceedings of ACM IMC*, pages 39–44, Miami, FL, October 2003.

[54] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). *ACM SIGCOMM Computer Communication Review*, 42(4):115–126, 2012.

[55] VMware Inc. VMware vSphere server virtualization. `http://www.vmware.com/products/vsphere/`.

[56] G. Wang and T. Ng. The impact of virtualization on network performance of amazon EC2 data center. In *Proceedings of IEEE INFOCOM*, San Diego, CA, March 2010.

[57] H. Wang, R. Shea, F. Wang, and J. Liu. On the impact of virtualization on dropbox-like cloud file storage/synchronization services. In *Proceedings of IEEE IWQoS*, Coimbra, Portugal, 2012.

[58] J. Whiteaker, F. Schneider, and R. Teixeira. Explaining packet delays under virtualization. *ACM SIGCOMM Computer Communication Review*, 41(1):pp 39–44, January 2011.

[59] H. Wu, Z. Feng, C. Guo, and Y. Zhang. Ictcp: incast congestion control for tcp in data-center networks. *IEEE/ACM Transactions on Networking (TON)*, 21(2):345–358, 2013.

[60] Q. Yan, J. Kaur, and F. D. Smith. Can bandwidth estimation tackle noise at ultra-high speeds? In *Proceedings of IEEE ICNP*, Raleigh, NC, October 2014.

[61] P. Yi, B. Ramamurthy, and H. Ding. Cost-optimized joint resource allocation in grids/clouds with multi-layer optical network architecture. In *Journal of Optical Communications and Networking*, pages 911–924, 2014.

[62] E. Zhang and L. Xu. Network path capacity comparison without accurate packet time information. In *Proceedings of IEEE ICNP*, Raleigh, NC, October 2014.