2005

PoissonProb: A new rate-based available bandwidth measurement algorithm.

Lu Xin
University of Windsor

PoissonProb: A New Rate-based Available Bandwidth Measurement Algorithm

By

Lu Xin

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada
2005

© 2005 Lu Xin

Your file Votre référence
ISBN: 0-494-09731-0
Our file Notre référence
ISBN: 0-494-09731-0

1032656

Abstract

Accurate available bandwidth measurement is important for network protocols and distributed programs design, traffic optimization, capacity planning, and service verification. Research on measuring available bandwidth falls into two basic classes: the network traffic modeling algorithms and the self-induced algorithms. The self-induced algorithms are based on packet dispersion techniques. The currently available bandwidth measurement algorithms face the problems of distortion of measurement on multi-hop paths, system resource limitations, probe traffic intrusiveness and measurement accuracy. We have developed a new rate-based self-induced algorithm --- PoissonProb. The intervals between probe packets of this algorithm are in Poisson distribution format and the algorithm infers the available bandwidth according to the average of probe packets rate. The algorithm has been implemented as the PoissonProb Available Bandwidth (PAB) measurement tool. The PAB tool can be operated in either sender-based or receiver-based mode. We have been able to test for the available bandwidth at Gbps networks on NS2. We have also tested it, in the range from several Mbps to 400Mbps, on common desktops on a grid test-bed. Another feature of this algorithm is that it can be operated under both Windows and UNIX environments. We have compared the PAB tool with C-Probe, PathChirp and IGI, the three algorithms, which are normally used today. The three algorithms can work only on UNIX environment. Our tests show that, even in the Windows environment, we are able to obtain the same or even better accuracy and efficiency as the other three algorithms. Lastly, we have done extensive testing on an ISP's network and compared the results with the data from the ISP.

Keywords: available bandwidth, bandwidth measurement, bottleneck bandwidth, packet train, Poisson distribution, PoissonProb

Acknowledgements

I would like to give special thanks to Dr. A. K. Aggarwal. His excellent teaching as well as the guidance is the basis of this research work. He is the best advisor I have met in the network field.

I would also like to thank and acknowledge Dr. Kemal Tepe, Dr. Arunita Jaekel and Dr. Jianguo Lu for their valuable comments.

Finally, I would like to thank my wife, Haiyan. Without her support and encouragement, this thesis would not have been finished.

Table of Contents

Abstract .... **III**

Acknowledgements .... **IV**

List of Tables .... **VIII**

List of Figures .... **IX**

1. Introduction

1.1 Bandwidth Metrics 1

1.2 Stationary Nature of Available Bandwidth 1

1.3 Available Bandwidth Measurement Algorithms 2

1.4 Some Clarification of Bandwidth Metrics 4

1.4.1 Which Layer is the Metrics related to? 4

1.4.2 Available Bandwidth vs. Achievable Bandwidth 4

1.4.3 BTC vs. Available Bandwidth 5

1.5 Where Is Available Bandwidth Measurement Used? 5

1.6 The Need for Available Bandwidth Measurement Algorithms 6

1.7 Organization of the Thesis Documents 6

2. Background and Related Work

2.1 Passive vs. Active Measurement 8

2.2 Receiver-based vs. Sender-based Measurement 8

2.3 Related Works 9

2.3.1 Model –based Algorithms 9

2.3.2 Gap-based Algorithms 10

2.3.2.1 Cprobe and pipechar 10

2.3.2.2 IGI 11

2.3.2.3 SPRUCE 11

2.3.3 Rate-based Algorithms 12

2.3.3.1 SloPS 12

2.3.3.2 TOPP 12

2.3.3.3 PTR 14

2.3.3.4 PathChirp. 14

3. Measurement Challenges

3.1 Rate-based Algorithms and Gap-based Algorithms 17

3.2 Packet Pair vs. Packet Train 18

3.3 Convergence Time 22

3.4 Intrusion Consideration 23

3.5 Tight Link 23

3.6 System Resources 24

3.6.1 Resolution of System Time 25

3.6.2 Context Switch 27

3.6.3 System Call 27

3.6.4 Interruption Delay 27

3.6.5 System I/O Bandwidth 30

4. PoissonProb Details

4.1 The Client-server Mode Algorithm 31

4.2 The Pseudo-Code for the C/S Mode Algorithm 33

4.3 The Standalone Mode Algorithm 34

4.4 The Pseudo-Code for the Standalone Mode Algorithm 35

5. Experiments

5.1 NS2 Testbed Description 36

5.2 Results of Experiments using NS2 37

5.3 Description of the Network Testbed 40

5.4 Results from the Experiments on the Test Bed 42

5.5 Measurement Results Between Gateways 47

5.6 Tests of the Sender-based Algorithm on NS2 49

6. Conclusion and Future Work 53

Reference 55

Appendix 58

A.1 Poisson Processes and Poisson Traffic 58

A.2 Using Poisson Processes to Model Internet Traffic 60

VITA AUCTORIS 69

List of Tables

Table 3-1 Different PoissonProb Train Lengths and Available Bandwidth Values 19

Table 3-2 Different PoissonProb Packet Size and Available Bandwidth Values 21

Table 5-1 Pre-Bottleneck Measurements Comparing on NS2 37

Table 5-2 Post-Bottleneck Measurements Comparing on NS2 39

Table 5-3 Link Capacity Verification 41

Table 5-4 Result of Experiments for Verification of the Poisson Cross-traffic Generator 42

Table 5-5 IGI Pre-bottleneck Measurements 43

Table 5-6 PoissonProb Pre-bottleneck Measurements 43

Table 5-7 PathChirp Pre-bottleneck Measurements 43

Table 5-8 IGI Post-Bottleneck Measurements 45

Table 5-9 PoissonProb Post-bottleneck Measurements 45

Table 5-10 PathChirp Post-bottleneck Measurements 45

Table 5-11 Sender-based Pre-Bottleneck Measurement 49

Table 5-12 Sender-based Post-Bottleneck Measurement 50

List of Figures

Figure 1-1 Gap-based Measurement 3
Figure 2-1 Exponentially Spaced Probe Packets 15
Figure 2-2 pathChirp Probe Packets Trace 15
Figure 3-1 Single Hop Test Bed 19
Figure 3-2 Error Rate for Different PoissonProb Train Lengths 20
Figure 3-3 Error Rate for Different PoissonProb Packet Sizes 22
Figure 3-4 Context Switch and Interruption Coalescence 29
Figure 5-1 Multi-hop Network Topology on NS2 36
Figure 5-2 Pre-Bottleneck Measurements Comparing on NS2 38
Figure 5-3 Post-Bottleneck Measurements Comparing on NS2 40
Figure 5-4 Multi-hop Network Topology 41
Figure 5-5 Pre-Bottleneck Measurements Comparing I 44
Figure 5-6 Pre-Bottleneck Measurements Error Rate I 44
Figure 5-7 Post-Bottleneck Measurements Comparing II 46
Figure 5-8 Post-Bottleneck Measurements Error Rate II 46
Figure 5-9 ISP Topology 47
Figure 5-10 Measurement Result Between Gateways (Wed) 48
Figure 5-11 Measurement Result Between Gateways (Thur)..............................48
Figure 5-12 Measurement Result Between Gateways (Fri) 49
Figure 5-13 Measurement Error Rate between Gateways 49
Figure 5-14 Sender-based Pre-Bottleneck Measurement Result 50
Figure 5-15 Sender-based Post-Bottleneck Measurement Result 51

1. Introduction

1.1 Bandwidth Metrics

Bandwidth is defined most generally as the amount of data the network can transfer per unit time. Five common metrics are summarized here:

- Bandwidth Capacity: The maximum amount of data per time unit that a hop or path can carry when there is no competing traffic.
- Utilization: The aggregate capacity currently being consumed on a hop or path.
- Available Bandwidth: The maximum amount of data per time unit that a hop or path can provide given the current utilization.
- Achievable Bandwidth (also referred as throughput): The maximum amount of data per time unit that a hop or path can offer to an application, given the current utilization, the protocol and operating system used, and the end-host performance capability and load.
- Bulk-transfer Capacity (BTC): [17] "*The intuitive definition of BTC is the expected long term average data rate (bits per second) of a single ideal TCP implementation over the path in question*".

Bandwidth measurement algorithms can be classified into several categories, depending on the specific metric of interest. For the end-to-end measurement, from the concepts described above, one can easily infer that if a path consists of several links, the link with the minimum transmission rate determines the capacity of the path while the link with the minimum unused capacity limits the available bandwidth. For the case of a multi-hop path, using end-to-end measurements, the capacity-limited link is called the bottleneck link and the available bandwidth-limited link is called the tight link.

1.2 Stationary Nature of Available Bandwidth

The available bandwidth measurement usually means measuring available bandwidth over a certain period of time. Assume the capacity of the tight link is C_t. The average utilization of the link during the measurement time Δ can be expressed as

$$u(t,t+\Delta) = \frac{1}{\Delta}\int_{t}^{t+\Delta} u(t)\,d(t) \qquad (1)$$

where $\mu(t)$ is the utilization of a link at time t. $\mu(t)$ is expressed as a fraction. Its value is 1 for full utilization and its value is 0, when there is no traffic on the link.
Then the available bandwidth C_a in time interval $[t, t+\Delta]$ can be expressed as

$$Ca(t,t+\Delta) = Ct(1-u(t,t+\Delta)) \qquad (2)$$

Actually, as a straightforward relationship with the bandwidth utilization, some of algorithms measure the network utilization first, and then get the available bandwidth by subtracting utilization from the capacity of the tight link.

The available bandwidth may change at a very fast rate due to the burstiness of some of the network traffic. Under such conditions, researchers have studied the issue of predicting future available bandwidth by using the past measurements of available bandwidth. Research done through large scale experiments on Internet infrastructure, in [6][27] has shown the stationary nature of available bandwidth. Their conclusion is that "*there is no simple relationship between the mathematics and operational constancy of throughput*" and "*when indicating throughput, remembering observations from a number of minutes in the past is fine, but remembering for more than an hour can mislead the estimator*" [27]. The same conclusion was proven by network traffic modeling research. He and Hou [6] experimentally proved that the available bandwidth could be predicted ahead for 5 times the measurement time interval. So, it is meaningful to predicate the available bandwidth through the available bandwidth measurement algorithms; the key issue here is the time scale. The above observations are so important that most modern available bandwidth measurement algorithms were designed based on them.

1.3 Available Bandwidth Measurement Algorithms

The research on measuring available bandwidth falls into two basic classes: the development from the network traffic modeling research, and research on self-induced algorithms. Self-induced algorithms can be further divided into rate-based algorithm and gap-based algorithm [15]. The idea of rate-based algorithm is that, if the rate of the probe packets pair or train sent from the sender is larger than the available bandwidth, the stream will produce a short-term overload at the tight link. Then at the receiver side, an increasing trend of one-way-delay of probing packets should be observed. The goal of the rate-based algorithm is to find the curve where there is available bandwidth along the path. Gap-based algorithm shown in as figure 1-1, are quite different.

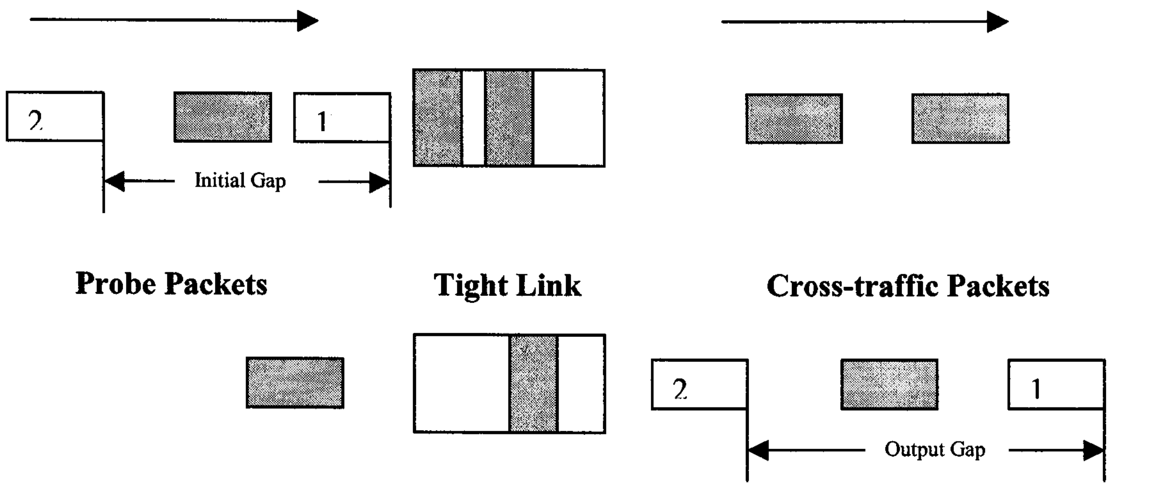


Figure 1-1 Gap-based Measurement

Gap-based algorithms are usually facilitated by the packet pair or packet train properties, by which the change of intervals of probing packets is observed (*Output Gap – Initial Gap*) as shown by above graph. The interval between probing packet 1 and 2 is changed by the tight link. The algorithm assumes the intervals between probe packets are affected only by cross-traffic on tight link, and the bottleneck link is the tight link. Then, they can infer the available bandwidth. The original self-induced algorithm can be traced back to the Cprobe [20], which is a gap-based algorithm, designed for prioritizing dynamic servers' selection. Gap-based algorithms such as IGI [18], and spruce [10] adopt different strategies, such as adjusting the probe packets rate and applied the statistical techniques to improve accuracy. The rate-based algorithm is focused on finding the transmission rate of probe packets while the probe packets just saturate the tight link, attempting to minimize the number of probe packets and filtering out the downstream's cross-traffic effect. Such algorithm includes SLoPS [4], PTR [18], pathChirp [24] and TOPP [3].

1.4 Some Clarification of Bandwidth Metrics

1.4.1 Which Layer is the Metrics related to?

Before the measurement tools can be applied in various applications, it is important to specify which network layer the metric is concerned with, when measuring bandwidth. For instance, the actual bandwidth that the higher layer sees may be very different from what the lower layer sees. The bandwidth capacity reflects physical bit rate capacity at layer-1. But if one uses the upper layers for measurement of capacity, the measured value of capacity decreases, since the framing and the protocol overheads for upper layers have to be taken into account. The algorithms discussed here are software-based. Thus, the network layer is considered for bandwidth capacity and available bandwidth, while the transport layer information is considered for achievable bandwidth and BTC.

1.4.2 Available Bandwidth vs. Achievable Bandwidth

By definition, available bandwidth is a physical-layer parameter. At any point of time, available bandwidth is the bandwidth the links can offer at current cross-traffic situation. On the other hand, the achievable bandwidth is an application-layer metric. It considers a number of factors such as network protocol, host speed, network path, and TCP buffer space, whereas available bandwidth only considers the network path. When one application wants to send data from one host to another through the network. The achievable bandwidth may be limited by every component along the path from the source host to the destination host, including all hardware and software. In fact, achievable bandwidth is often a measurement of the capacity of the end host, rather than being a measurement of the capability of the network. Hence the achievable bandwidth may or may not correlate with the available bandwidth.

1.4.3 BTC vs. Available Bandwidth

One fundamental difference between the available bandwidth and BTC relates to the cross traffic on the network. The available bandwidth is the amount of usable bandwidth without affecting the current cross-traffic. BTC, on the other hand, is measured by sending out data as fast as possible, thus grabbing as much of bandwidth as possible. Thus measurement of BTC definitely influences other traffic. Experiments have shown that the traffic generated for a BTC measurement may grab 20%-30% more bandwidth than that for an available bandwidth measurement. Secondly, BTC is a metric to present long-term TCP traffic bandwidth, whereas, the available bandwidth reflects the instantly usable bandwidth. Reflected into the measurement tools, the measurement tools for BTC usually take longer time and are more intrusive than those for available bandwidth. Another subtle difference between BTC, achievable bandwidth and available bandwidth, is that the BTC and achievable bandwidth also rely on the reaction of the other traffic to the resource competition. When the transmission rate of the sender reaches the available bandwidth and beyond, the other traffic will be affected and may react accordingly to accommodate the new traffic by lowering its rates or it may not react at all. Thus, the protocol dependent bandwidth is hard to predict from probing unless the probe behaviour is exactly like that of the application.

1.5 Where Is Available Bandwidth Measurement Used?

Available Bandwidth information is so critical to various network applications that it has attracted much of the recent research. The applications can be summarized as following.

- Information for network protocols and application development. Developers need to know the available bandwidth to judge the efficiency of their protocols and applications.
- Dynamic server selection and adapting content. Network clients could dynamically choose the server with highest available bandwidth. Meanwhile, a

server could scale the size and quality of its content (e.g. pictures, sound and video) depending on the available bandwidth of the path to the client.

- Network management, like end-to-end admission control, congestion control, TCP windows control, and network-aware cache or replica placement policy.
- Bandwidth-Aware Routing. Information about the bandwidth of links also allows building efficient overlay networks.
- Benchmarking equipment and service level agreement verification.

1.6 The Need for Available Bandwidth Measurement Algorithms

Routers or switches along the path sometimes can offer the link capacity information via SNMP query or the tools, like MRTG or HP Openview can be used to monitor the utilization of links. This approach usually can get more accurate information than other methods. However, in general, it is not possible to get access to this information because network switches and routers may be in different administrative domains. For example, an end-user is unlikely to be able to access SNMP information in a commercial ISP's network. Also, most router software has been optimized for routing speed. Hence, the routers usually ignore the queries or other activities that influence their speed. Another drawback of this approach is insufficient level of measurement resolution. For example, the MRTG reports packet statistics every 5 minutes. This is not adequate for some applications that need the instant available bandwidth information to make decisions.

1.7 Organization of the Thesis Document

The rest of this thesis is organized as follows. The literature review of the current available bandwidth measurement algorithms and a general discussion of bandwidth measurement methods are presented in chapter 2. Chapter 3 is an analysis of difficulties faced when the available bandwidth measurement algorithms are used. The distortion of measurement on the multi-hops path, the system resource limitations, the probe traffic intrusiveness and the measurement accuracy and other such factors, which affect the accuracy of measurement, are discussed. The effect of these factors on the design process of PoissonProb algorithm is also described in this chapter. Chapter 4 gives a detailed description of the algorithm. The results of our experiments for testing the algorithm are shown in chapter 5. The conclusions of this study constitute the last chapter.

2. Background and Related Work

2.1 Passive vs. Active Measurement

The bandwidth measurement algorithms can be separated into two categories. One is algorithms, which use active measurements. The others use passive measurement. An active measurement algorithm sends packets along the path, for which the bandwidth is to be measured. A passive measurement algorithm monitors the passing packets without interfering. The active measurement algorithms can be further separated into sender-based or receiver-based algorithms. These will be discussed in the next sub section. Active measurement may be intrusive to the other traffics on the network. Some active measurement algorithms send a large number of packets into the network to collect sufficient number of samples to filter out the effect of cross-traffic, random network behavior or the effect of the measuring host. Though many efforts have been made to use passive measurement, passive measurement algorithms are often less reliable than the active one. Claffy et al. [12] showed that from passive measurements, it might be impossible to extract any useful data at all in some cases. Due to the real time and accuracy requirements of most of the applications, available bandwidth algorithms are usually operated in the active mode.

2.2 Receiver-based vs. Sender-based Measurement

Receiver-based algorithms (also referred to as end-to-end ones) usually use the one-way UDP/TCP datagrams/streams to probe the bandwidth information. On the other hand, the sender-based algorithms (also referred to as echo-based ones) force the receiver to reply to the ICMP query, UDP echo or TCP-RST packets by TCP-FIN. The choice of sender-based or receiver-based algorithms affects ease of deployment and accuracy of measurement. Obviously, the sender-based algorithms are easier to deploy than receiver-based algorithms. It is impossible to deploy the receiver-based algorithms without destination domains' cooperation. Unfortunately, accuracy suffers when measuring with sender-based algorithms. First, the effect of cross traffic on measurement of delay of the probe packets may be more for a round-trip traversal of a path than it would be on the delay of a one-way transversal of the path. Secondly, the response packets may go back through a different path. Moreover, the most important point is that sender-based algorithms assume that "*the sender can precisely match acknowledgements to packets, the receiver acknowledges each packet immediately and consistently, the acknowledgements are all of the same size, and the acknowledgements experience no queuing in the reverse path*" [13]. However, many network environments cannot meet these strict conditions. For example, every other TCP packet may be acknowledged at the receiver side. ICMP and UDP echo packets are either rate-limited or filtered out at some routers. Different TCP/IP implementations may respond in different ways to these probe packets. Finally, the probe packets of the sender-based algorithms may trigger the alarm of an intrusion detection system or may be blocked by firewall, and thus making the measurement impossible. There is really a trade-off between the easy-of-deployment and accuracy. All the current available bandwidth algorithms are receiver-based except the Cprobe which is a sender-based algorithm. PoissonProb can be operated in either receiver-based or sender-based mode. PoissonProb sends out UDP packets and waits for echoes from the destination when it is impossible to deploy the software at the receiver side, no strict network security policy is present, and the accuracy requirement is low.

2.3 Related Works

2.3.1 Model –based Algorithms

As mentioned in section 1.3, one class of the available bandwidth measurement algorithms has been developed on the basis of the network traffic modeling research. The foundation of the Delphi algorithm [25] is based on the Multifractal Wavelet Model (MWM). The core idea of the MWM is that the cross-traffic stream is a superposition of many data flows that share common link resources with the probe connections. The statistical analysis showed such superposition has the characteristics of self-similarity, burstiness, long-range dependence (LRD) and even multifractal behaviour (non-Gaussianity) [23]. This multifractal behaviour makes it possible to present aggregated cross-traffic as a binary tree structure. In this structure, the β multiplier split parent aggregate into two child aggregates at the next scale which increases or decreases β flow of traffic. The MWM also provides means to estimate the queuing behaviour of a synthetic trace through the Multiscale Queuing Formula (MSQ). Then Delphi uses the derivative of the MSQ to obtain a probability density function for queue size.

Following this model, the Delphi algorithm sends out a chirp of probe packets. The initial interval between the packets is partitioned according to the exponential spacing and the interval is adjusted with the estimate of the previous result. Then the gap change of the two probing packets at the receiver can provide an estimate of the amount of traffic at a link. Delphi assumes that the path can be well modeled by a single queue (single-hop model), However, this assumption is not applicable when the tight and bottleneck links are different. It also looks upon all the queuing delays in the path as delay at the tight link. This assumption, in some situations, leads to wrong estimation of the cross-traffic. Actually, the implementation of Delphi is similar to that of gap-based algorithms. But the two have different theoretical foundations.

2.3.2 Gap-based Algorithms

2.3.2.1 Cprobe and pipechar

Cprobe [20] is the first algorithm, which claimed to measure the available bandwidth through gap-based approach. By sending out a short packet train back-to-back and measuring the time interval of the first and the last ICMP echo packets, the available bandwidth can be calculated. The underlying assumption is that the dispersion of a packet train is inversely proportional to the available bandwidth. This idea is the same as the one used by pipechar [8], which was implemented in Network Characterization Service (NCS). The only difference is that the pipechar can also be operated in the passive mode through utilizing the deployment of the NCS infrastructure. Though these algorithms are straightforward, researches have shown that some assumptions in the approach are of doubtful validity. Dovrolis et al. [5] proved that the dispersion of long packet trains, sent out back-to-back, is not actually inversely proportional to the available bandwidth. Other research [3] has shown that the "*hidden bottleneck problem*" may also be a detriment to the accuracy of measurements.

2.3.2.2 IGI

Hu et al. [18] set up a single-hop gap model and then proposed two available bandwidth measurement algorithms based on this model: the gap-based algorithm *Initial Gap Increasing* (IGI) and the rate-based algorithm *Packet Transmission Rate* (PTR). The core task of IGI/PTR algorithm is to find the "*turning point*" at the point of probe packets transmission rate. The output interval of the whole packet train at destination is the same as the initial interval. Researchers claimed, "*at this point, the probing packets interleaves nicely with the competing traffic and the average rate of the packet train equals the available bandwidth on the bottleneck link.*" Hu et al. also summarized the gap-base algorithms as the following formula:

$$\frac{B_o \sum_{i=1}^{M} (g_i^+ - g_B)}{\sum_{i=1}^{M} g_i^+ + \sum_{i=1}^{K} g_i^= + \sum_{i=1}^{K} g_i^-} \qquad \text{(3) (Reference [18] page 882 equation 4, 5)}$$

In formula (3), $Bo\sum_{i=1}^{M}(g_i^+ - g_B)$ is the amount of competing traffic that arrives at bottleneck link during the probing period. Ideally $\sum_{i=1}^{M} g_i^+ + \sum_{i=1}^{K} g_i^= + \sum_{i=1}^{N} g_i^-$ is the total probing time.

The other contribution of this research is that they noticed the packet pairs could not measure the available bandwidth when continuous probe packets fall in a "*disjoint queuing region*" (DQR).

2.3.2.3 SPRUCE

Spruce [10] is a lightweight available bandwidth measurement tool designed by Strauss et al. The tool uses UDP packet pair as the probing packets. The packet size is fixed to 1500 bytes and the changes of gaps between the packet pairs are collected to calculate the available bandwidth. The formula is the same as that of IGI algorithm, which is described in section 2.3.2.2 equation 3, except that just one gap of packet pair is calculated. Spruce takes the average of a large number of samples of probe packet pairs. All the parameters in Spruce are fixed. The initial gap of packet pair is set to the bottleneck capacity along the path as the tool assumes the bottleneck link is the tight link. The characteristics of Spruce is that it sends two packets (one packet pair) back-to-back and sets the inter pack pair gaps as the Poisson distribution at a very low rate, as the researchers claimed, "*A sequence of measurements according to a Poisson sampling process sees the average cross-traffic rate*".

2.3.3 Rate-based Algorithms

2.3.3.1 SLoPS

Dovrolis et al [16] devised a Self-Loading Period Streams (SLoPS) algorithm to measure the end-to-end available bandwidth. The implementation tool is Pathload. The Pathload sends out a fleet of probe packet streams at the rates from low to high. So it measures the "*fleet duration*" instead of "*stream duration*" to address the volatile characteristics of short-term cross-traffic. The sender's sending rate is self-adaptive with the delay treads observed at the receiver. Finally this rate converges to the range of R_{max} and R_{min} (Pathload reports the range of the estimated available bandwidth) through a binary tree search algorithm. Pathload uses "*Pairwise Comparison Test*" (PCT) and "*Pairwise Difference Test*" (PDT) to decide whether the measurements fall into the available bandwidth range. The limitation of SLoPS is that it can succeed in estimating a range that includes the actual available bandwidth when there is only one tight link along the path; but it underestimates the available bandwidth when there are multiple links, as the congestion links exist along the path. This is also the common limitation of the present algorithms. The other drawbacks of SLoPS are slow-convergence and intrusiveness to the current traffic.

2.3.3.2 TOPP

TOPP [3] and SLoPS are based on the same principle that the queuing delays of successive periodic probing packets increase when the probing rate is higher than the available bandwidth along the path. But TOPP has a different analysis model to the samples at the receiver side. The contribution of the TOPP algorithm is that it adopts a mathematical model that can catch the effect of different congestion links on the probe packets, if the congestion links are in "*Smallest Surplus First (SSF)*"order, i.e. if the congestion links capacities are in order, they can be detected when raising the probe load.

TOPP uses packet trains with different transmission rates and estimates available bandwidth from the time average spacing of the packet pairs in the packet train at the receiver. The transmission rate increases from O_{min} to O_{max} with the increment δ. With the increase of the probe train's transmission rate, the links with lower available bandwidth become congested. If the router adopts FCFS policy, the probe traffic is proportional sharing the bandwidth with the cross traffic. The partitioned probe traffic will share the bandwidth with the cross traffic again in the following congested link. Refer to equation (4).

$$o_j = o_{j-1} \text{ if } o_{j-1} < s_j \text{ or } o_j = \frac{o_{j-1}}{m_j + o_{j-1}} l_j \text{ if } o_{j-1} \geq s_j \quad (4)$$

(Reference [3] page 416 equation 2)

Equation (4) shows that when probe traffic arrives at link j, if sustained bandwidth s_j is higher than probe traffic bandwidth o_{j-1}, the probe traffic keeps original bandwidth. Otherwise, probe traffic shares the bandwidth with cross traffic at the jth link m_j

$$o_j = \frac{o}{m_j + o} l_j \quad \text{for } o > s_j \qquad (5)$$

$$o_{j+p} = \frac{\frac{o}{m_j + o} l_j}{m_{j+p} + \frac{o}{m_j + o} l_j} \quad \text{for } o_j > s_{j+p} \qquad (6)$$

(Reference [3] page 416 equation 3)

Equation (5) and (6) show the proportional sharing of bandwidth between cross-traffic and probe-traffic at the congested link j (5) and the following congested link $j + p$ (6).

$$\frac{o}{f} = (1 - \frac{l-m}{l}) + \frac{1}{l}o = (1 - \frac{s}{l}) + \frac{1}{l}o = \alpha + \beta o \qquad (7)$$

(Reference [3] page 416 equation 5)

When probe packets arrive at the destination, TOPP calculates the effect of the cross-traffic on the packet pair f and linearizes the congested link bandwidth o with f by equation (7). On plotting the function as continued curves, the curves represent the segment regression function of equation (7). Vertices correspond to the available bandwidth of the congested links.

2.3.3.3 PTR

The PTR algorithm [18] is focused on finding the "*turning point*". It calculates available bandwidth according to equation (8):

$$\frac{(M+K+N)L}{\sum_{i=1}^{M} g_i^{+} + \sum_{i=1}^{K} g_i^{=} + \sum_{i=1}^{N} g_i^{-}}$$ (8) (Reference [18] page 882 equation 4)

Here, the gap values $G^{+} = \{gi|\ i= 1...M\}$, $G^{=}=\{g_i^{=}|=1...K\}$, and $G^{-}=\{g_i^{-}|I=1...N\}$ denote the gaps that are increased, unchanged, and decreased respectively.

2.3.3.4 pathChirp

Another rate-based algorithm is pathChirp [24]. PathChirp is more efficient, in that, it sends less number of probe packets. The probe packets are exponentially spaced by space factor γ as shown in figure 2-1.

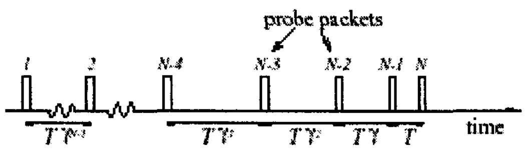


Figure 2-1 Exponentially Spaced Probe Packets (reference [24] page 2, figure 1)

Then to measure the range of available bandwidth *[G1, G2]*, it just needs $log_2G2- log_2G1$ packets to cover the bandwidth range. The method needs the lowest number of probe packets because it uses the correlation of the interval between these packets. For example, to cover the bandwidth from 1-100 Mbps, the pathChirp algorithm just needs 13 packets if the space factor $\gamma=1.4$. At the receiver side, a typical trace of probe packets affected by the burst of cross-traffic is observed and analyzed. The typical trace is usually presented as in figure 2-2.

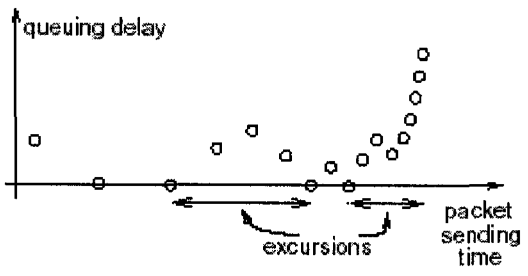


Figure 2-2 pathChirp Probe Packets Trace (reference [24] page 3, figure 2)

The trace of the probe packets received at the receiver side provides abundant information about the network cross-traffic. By carefully discriminating the different queue situation the probe packets meet, pathChirp makes an estimate of per-packet available bandwidth E_k of the k*th* packet estimated at time interval of continuous k and $k+1$ probe packets $B\ [t_k,\ t_{k+1}]$. It then takes the weighted average of all estimates corresponding to each chirp with N probe packets to obtain estimated per-chirp available bandwidth $B\ [t_1,\ t_{N+1}]$. Here, the weighted average of E_k means that not all the packets take their individual estimates as valid. The probe packet's queuing delays falling uphill of the excursion range (refer to figure 2-2) are processed as valid samples because these packets are believed to have experienced cross-traffic interference. Other invalid samples' estimates are represented by E_l which is the last start of excursion. For the time interval of measurement T (default is 3 seconds), pathChirp algorithm takes the average of per-chirp estimates to get the time interval T's available bandwidth.

3. Measurement Challenges

3.1 Rate-based Algorithms and Gap-based Algorithms

Gap-based algorithms are usually facilitated by the packet pair/train properties. The gaps between probing packets are observed. The advantage of this kind of algorithms is that they are very sensitive to the burstiness of cross-traffic because of fine-grained interaction between the probing packets and cross-traffic packets. The algorithms are based on a single hop model where the bottleneck link and tight link is the same one. In other words, the whole path can be modeled as one queue. The formula can be summarized as equation (3). Though such algorithms are based on a single-hop model, some papers have described the measurement results as very accurate. The reasons can be analyzed as follows:

- Tight link in most situations is the bottleneck link.
- Tight link is usually located at the edge of the network.

Rate-based algorithms try to find a "*turning point*" where the rate of the probing packet is around the available bandwidth. The advantage of this type of algorithms is that they adapt widely to most of network situations. They have better resistance to the cross-traffic effect and they can always report reasonable results. In comparison with the rate-based algorithms, the gap-based algorithms may deviate largely from the correct value because of errors in estimating either the bottleneck capacity or the cross-traffic rate. The shortcoming of rate-based algorithm is that the overhead to converge to the turning point is too high. The rate-based algorithms formula can be summarized as equation (8).

PoissonProb is a rate-based algorithm. The sending rate of probe packets is adjusted according to the accumulated gaps between probe packets at the destination. The gaps between the probe packets are separated in Poisson distribution. The mean inter-packets interval λ is increased step-by-step till it reaches the stop point when the probe packets are interweaved with cross-traffic perfectly. Then the algorithm calculates the utilization according to equation (8). The bottleneck capacity is inferred through the simple version of histogram approach of bandwidth capacity measurement algorithm that is used to reduce the overhead of convergence to the turning point. The detailed description of the algorithm is presented section 4.

3.2 Packet Pair vs. Packet Train

Characterizing available bandwidth is more difficult since it is a dynamic property of the network and it depends on many factors. Available bandwidth measurement algorithms try to take a snapshot of the cross-traffic effect. Because of the dynamic nature of the available bandwidth, it must be averaged over a time interval; therefore, active measurement techniques often use packet trains instead of packet pairs as the packet pair cannot catch the bursty type of traffic. Spruce uses only packet pair gaps to infer the utilization. Though the Cprobe, pipechar, and TOPP use packet trains as the probe type they simply take the total interval of the packet train or packet pair intervals to infer the available bandwidth, ignoring the correlation of the effect of the bursty cross-traffic on packet trains. This yields less accurate results than the tools, like Pathload, IGI/PRT and pathchirp, which take the delay correlation of each probe packet into consideration. The other shortcoming of the packet pair technique is that when applied in the TCP stream to measure available bandwidth, it may fail to saturate the pipe, thus underestimating the available bandwidth.

The next question is what should be an adequate length of the packet train. Intuitively, a shorter packet train provides less accurate information. This has been proven in [18] as well as [24] which claimed "*The shorter chirps will exhibit more erratic signatures and give less accurate estimates.*" To get a snapshot of the cross-traffic, a longer packet train is certainly preferred. However, a longer packet train may result in a higher network overload, lost packets, or a congested network.

To better understand the packet train properties, we set up a single hop test bed based on NS2. Please refer the figure 3-1 Single Hop Test Bed. C1, C2, C3, C4 and C5 are agents to send out CBR type traffics to the sinks S1, S2, S3, S4 and S5 respectively. The capacity of links C [1-5] A, S [1-5] B and AB is 100Mbps and the agents start in sequence at the rate 20Mbps except C4 and C5 at 10Mbps. Ps and Pr are PoissonProb client and server. Then we reduce the transmission rate of agents gradually to 10Mbps at the end. In this experiment, we set the length of packet train from 10 to 60 packets. The probe packet size is 1000 bytes. The result is shown in Table 3-1.

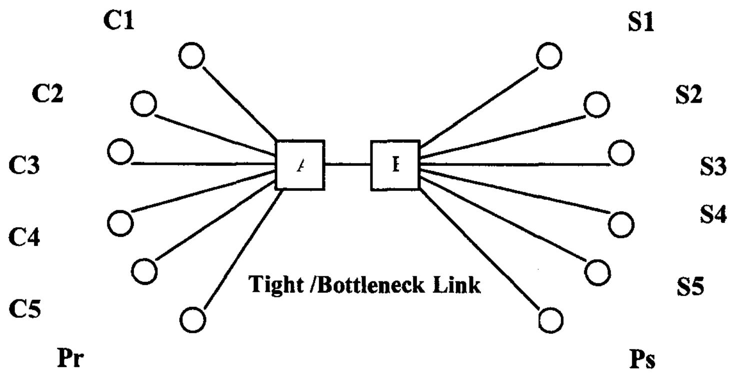


Figure 3-1 Single Hop Test Bed

Table 3-1 Different PoissonProb Train Lengths and Available Bandwidth Values

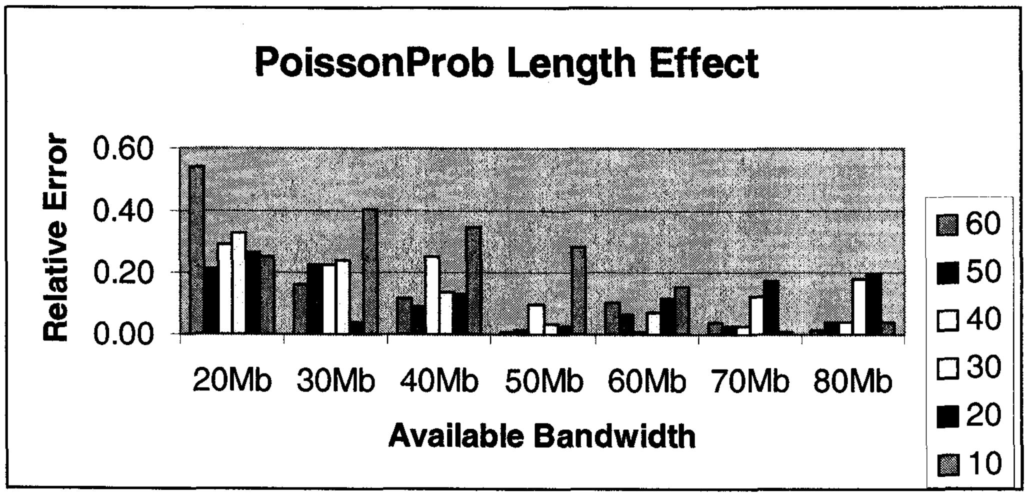
	60	50	40	30	20	10
20Mbps	30.89	24.25	25.82	26.61	25.33	25.00
30Mbps	34.91	36.84	36.79	37.18	28.79	42.21
40Mbps	44.70	43.59	50.00	45.31	34.87	26.00
50Mbps	50.29	50.79	54.75	51.48	51.35	64.29
60Mbps	66.29	56.30	60.22	55.63	53.14	69.30
70Mbps	72.65	71.68	68.19	61.59	57.61	70.41
80Mbps	78.90	83.06	82.97	65.56	64.74	76.68
90Mbps	88.08	90.57	90.25	72.49	72.29	81.96

The first row in the table is the PoissonProb train length that is changed from 10 to 60 packets. Each column is the measurement result and the most left-most column is the real available bandwidth in Mbps. We define the relative error as:

$$\delta = \frac{|MeasurementAB - \mathrm{Re}\,alAB|}{\mathrm{Re}\,alAB}$$

Then, the relative error of each length of PoissonProb can be shown as in the figure 3-2:

Figure 3-2 Error Rate for Different PoissonProb Train Lengths



From the chart, we can conclude as following:

- The shorter length probe packet trains do show less accuracy than the trains with longer length. (With the exception of length = 60 packets at AB= 20Mbps).
- The short train (length = 10 packets) cannot be used, as it cannot snapshot cross-traffic.
- Commonly, a train length more than 20 packets can give a reasonable result. In PoissonProb NS2 implementation, the train length is 20 packets and in its implementation on the test-bed and on the Internet, the train length is 60 packets to catch complex Internet traffic.

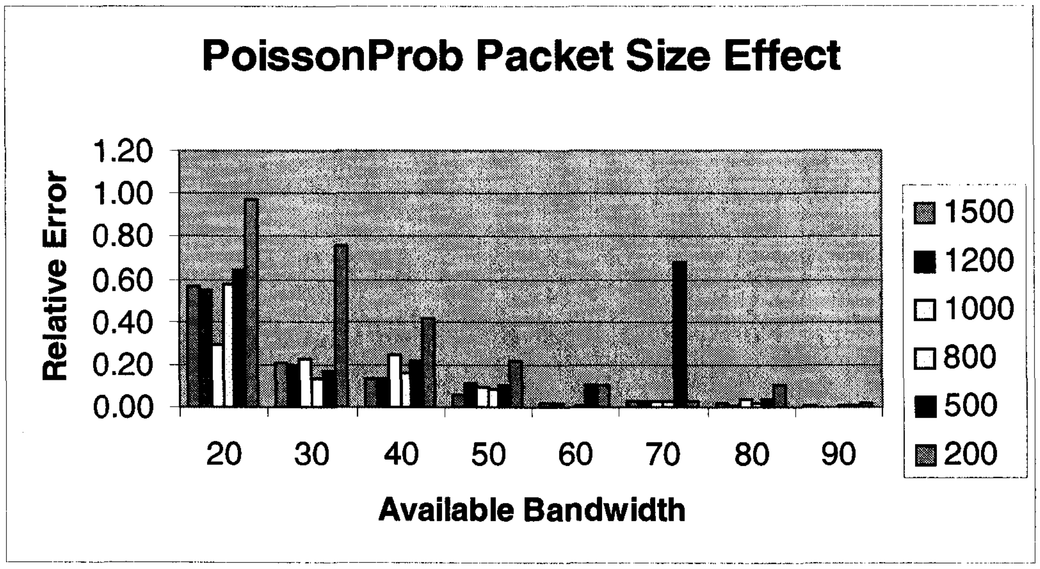
The next concern is the size of probe packet. The packets with smaller size are more sensitive to the cross-traffic than the packets with bigger size. On the other hand, the packets with bigger size are more intrusive to the network, i.e. they occupy more bandwidth resource. The other drawback of small packets is that it is difficult to measure the small time interval between two probe packets because of system resource limitations. All the published available bandwidth measurement algorithms choose the probe packet size higher than 500 bytes (Internet traffic analysis [11] showed the average packet' size is around 500-750 and the trend is becoming larger because of the increasing popularity of multimedia traffic).

Again, based on the same single-hop test bed, we tested our PoissonProb algorithm in different packet size. The first row in Table 3-3 showed the packet size from 200 bytes to 1500 bytes. The left-most column is the real bandwidth.

Table 3-2 Different PoissonProb Packet Size and Available Bandwidth Values

	1500	1200	1000	800	500	200
20Mbps	31.37	31.03	25.82	31.45	32.77	39.39
30Mbps	36.22	36.06	36.79	33.85	35.22	52.70
40Mbps	45.35	45.38	50.00	46.43	48.78	56.52
50Mbps	52.94	55.68	54.76	54.36	55.31	60.94
60Mbps	61.24	61.17	60.22	59.32	66.10	66.10
70Mbps	67.88	67.81	68.19	72.22	117.74	72.22
80Mbps	81.82	80.97	82.98	81.68	82.98	88.64
90Mbps	90.70	90.35	90.26	89.32	90.70	88.63

Figure 3-3 Error Rate for Different PoissonProb Packet Sizes



From the figure 3-3, we can conclude as follows:

- The small packet size (size = 200,500 bytes) is sensitive to the cross-traffic and reports more available bandwidth because of queuing behind the big packets. The intervals between the probe packets are decreased.
- Commonly, a packet size bigger than 500 bytes is enough for probing.
- PoissonProb NS2 implementation uses 800 bytes as the probe packets size and the implementation on the test-bed and the Internet uses 1000 bytes to address the high speed network time resolution issue.

3.3 Convergence Time

Cprobe, IGI/PTR, Spruce and TOPP algorithms use the bottleneck capacity information to infer the available bandwidth. Furthermore, the IGI/PTR algorithm uses this value to decide its *init_gap* and *gap_step* separately. There is no doubt this method is very useful to optimize the convergence time. The bottleneck capacity measurement is efficient, i.e. the result of the measurement is accurate and the convergence time is short. Because of the advanced structure design of probe packets, PathChirp has a short convergence time without knowing the network property in advance, as discussed in section 2.3.3.4. The

Pathload has the longest convergence time among these algorithms due to its convergence algorithm. Pathload monitors changes in the one-way delay of the probing packets in order to determine the relationship between probing speed and available bandwidth. This can be difficult if probing packets experience different levels of congestion. This can slow down the convergence process and can result in a high value of probing time. In contrast, the convergence of IGI/PTR is determined directly by the single packet train dispersion at the source and destination. PoissonProb takes advantage of IGI/PTR approach and infers the bottleneck bandwidth through histogram analysis. Then it decides the initial probing rate based on the bottleneck bandwidth.

3.4 Intrusion Consideration

Active measurements are all intrusive as they place additional load on the network, delaying other flows. Though many efforts have been made to use passive measurements, passive measurements are inefficient in most situations. To design measurement algorithms with minimal additional load to the network have become the challenge. However, to judge an algorithm's intrusiveness is difficult because it hasn't been standardized. Some researchers proposed average probing traffic rate as the standard. Others evaluated the intrusiveness by comparing this rate with the available bandwidth in the path. The following comparison is based on the analysis of the algorithms, without formulary calculation or measurements. Through analysis of these algorithms, the PathChirp algorithm clearly shows its advantage over other algorithms. At the same time, Cprobe and pipechar may be the most intrusive algorithms, because they send out their probe packets back-to-back (as fast as possible), and can easily induce overload on the network. Based on the self-induced algorithm characteristics, the rate-based available bandwidth measurement tools are more intrusive than the gap-based tools because of the coverage overhead.

3.5 Tight Links

The critical issue that influences the accuracy of available bandwidth measurement occurs when there are several tight links (or congested link) existing along the path or the tight link is not the bottleneck link. The algorithms, like PoissonProb, Cprobe, pipechar, IGI/PTR, pathChirp and Pathload were designed based on the assumption that the tight link is the bottleneck link. All these algorithms assume a single-hop model. If the first assumption fails to be satisfied, it may result in reporting wrong answer for Cprobe and IGI like algorithms because they report the available bandwidth by directly involving the bottleneck link capacity. If there are several congested links along the path, all algorithms may suffer inaccuracy, which depends on the smallest tight link location. It is easy to prove that both the upstream and the downstream of tight links will reshape the probe packet's interval and the competing traffic on the ending up tight link has dominating impact on the probe packet's interval. If the smallest tight link is at the end of path, then the probe packets can take the cross-traffic snapshot of that link and get the right result; otherwise, the result deviates from the correct value. Out of these algorithms, only TOPP algorithm addressed a limited solution of this issue. The solution is valid only when the congested links are compliant with Smallest Surplus First order. Fortunately, investigations on today's Internet shows that most congested links are usually located at the edge link [11] [14]. This discovery alleviates most of algorithms' problems. However, this issue may still be a future research topic. PoissonProb separates the probe packets in Poisson distribution and probes the tight link at the base rate of that tight link. It doesn't accumulate the several tight links effect to the sink. It shows better resistance to the interference of cross-traffics before the tight link as well as after tight link. We'll prove this theoretically in chapter 4 and compare the PoissonProb, IGI/PTR and pathChirp on a multi-hop testbed.

3.6 System Resources

Recently, the bandwidth measurement tools have been facing another significant challenge as the network speed has been increasing. As shown at [7], (page 8, paragraph 5):

"In the past 10 years, network speed has increased by a factor of 100, CPU clock speed has increase by more than a factor of 30; memory clock speed has increased by almost a factor of 20. Memory bandwidth, however, has increased by only a factor of 10, and PCI I/O bus bandwidth has increased by only a factor of 8."

It apparently shows the discord of increasing speed between the network and the end host system. Sometimes, the end host is the real bottleneck along the path and network bandwidth measurement falls, when estimate at the end host. Most of the available bandwidth measurement algorithms require the end hosts to send the probe packets at a speed faster than the possible available bandwidth. For some high-speed networks (>= 1Gbps), end-hosts may fail to meet this condition. (We used a desktop with 1837MHz CPU and 133 PCI bus. The fastest rate the machine can send is about 550 Mbps) Thus, when designing and implementing the bandwidth measurement tools, the system resource issues have to be taken into consideration. The system resources that may affect bandwidth measurement are summarized in the following subsections.

3.6.1 Resolution of System Timer

There are two system functions, which are commonly used to get system time in the Windows system: *GetSystemTime* and *GetTickCount*. *GetSystemTime* returns current time in millisecond, *GetTickCount* is used to get the elapsed time, and is limited by system timer resolution. The system timer runs at approximately 10 ms in Windows. For the higher resolution, Windows supports hardware devices and network protocols. The highest resolution is the 100-nanosecond interval (10^-7 second) [1]. The UNIX system offers a better time resolution at 1μs through a system call (*gettimeofday*). Even at this resolution, it is impossible to measure any incoming packet over 3Gbps due to the additional overhead of system calls. For example, the biggest packet size is $L = 1500$ bytes for most networks. If two probe packets go through the OC-48 (2488.32Mbps) link, the interval between the two probe packets is about 4.8μs when they reach destination. However, most of the time resolution of workstations falls in range of 1-10 μs. Thus, it is futile to try to accurately measure packet delays on OC-48 or higher bandwidth links.

An experiment was performed on Linux RedHat 9 (P4 PC) boxes and the Solaris 9 system (Sun Microsystems V880 systems) to find timer resolution through the Milliken Oil Drop Experiment to time an operation that takes some tiny amount of time, and does it several times. If that operation is close to the resolution, some timings will be zero while others will show up as taking a small amount of time, which is an integer multiple of the clock resolution. When the resolution was approached, Solaris reported 0-1 microseconds and Linux reported 1 microsecond constantly. Linux will not return the same *gettimeofday* values twice in succession. Since it provably takes less than 1 microsecond to make a system call on a modern machine, Linux must be waiting within the *gettimeofday* procedure long enough to make certain that the time has changed. This may be screwing up any available bandwidth measurements made with *gettimeofday*. However, we can believe the time resolution in UNIX system is 1 microsecond through the system call. Due to the poor time resolution offered by the Windows system (10ms) through system call, several APIs (non-Microsoft software) have been developed to give high-resolution time stamp in windows, the resolution is as that in UNIX – in microseconds. For example, the IBM High Resolution Time Stamp Facility (IBMTS) is a library of functions that can be used to measure activities of less than one millisecond's duration using highly accurate timestamps. The API returns two values [2]:

- Seconds since the midnight 1/1/70 CUT epoch
- microseconds

IBMTS uses the multimedia timer routines QueryPerformanceCounter and QueryPerformanceFrequency at Windows system. The QueryPerformanceCounter function can be used to express the frequency, in counts per second. The value of the count is processor dependent. On some processors, for example, the count might be the cycle rate of the processor clock. The QueryPerformanceFrequency function retrieves the current value of the high-resolution performance counter. By calling this function at the

1. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sysinfo/base/time.asp
2. http://www.alphaworks.ibm.com/tech/ibmts

beginning and end of a section of code, an application essentially uses the counter as a high-resolution timer. We tested most of desktops with 700-1837MHz CPUs and IBM *i* series servers; all the QueryPerformanceFrequency return 3579545 ticks per second. So, the time resolution is even better than a microsecond. When we repeat calling this API, the systems returned at the 2-5 microseconds later. PoissonProb is facilitated by the IBMTS to get the time resolution comparable to those algorithms which run on Unix machines.

3.6.2 Context Switch

Context switch is another system resource issue. If the measurement period spans a context switch, then the measurement will include this time. Generally, a process gets 10ms execution time between context switches. So the context switch is likely to occur and introduce significant errors in the measurement of a long packet train or delay (>10ms). Some researchers noticed this problem. In [24], the pathChirp tool hard coded a threshold value; if the interval of two packets is less than this threshold, the sample is ignored. PoissonProb took a similar approach. It finds deviant measurements and filters them out before the available bandwidth analysis. Some measurement algorithms improve the measurement process priority, but this approach is platform dependent and is not an appropriate solution.

3.6.3 System Call

The time to perform system calls influences two aspects of measurement tools: both the outgoing packet spacing and get system time for the incoming packets. In [7], Jin et al. showed "the *system internal timer resolution is often at 1 nanosecond in modern UNIX systems. However, the time to perform a system call limits the user timer resolution to 1.9μs on most systems with x86-based CPU running Linux*". The reason is that the system has to access the clock time counter (CTC) via the low-speed I/O bus.

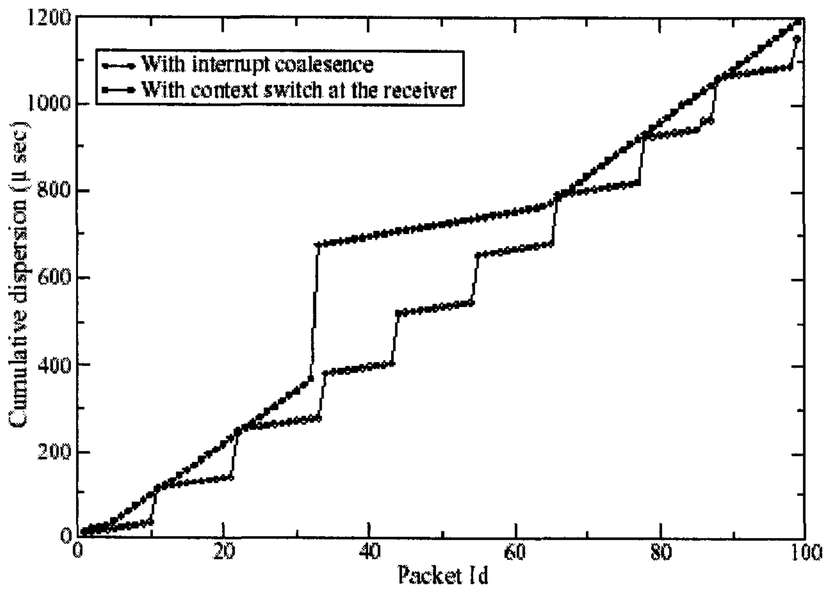
3.6.4 Interruption Delay

The core reason of interruption delay is that the efficiency of the CPU decreases as the number of interruption requests becomes higher. The experiments [7] show that the CPU efficiency might decrease by 40% through response to the interruption request. Interruption delay (or interruption coalescence) is a common technique implemented in high-speed network interface cards (NIC) that helps to reduce the CPU load. Currently, almost all the NICs with bandwidth more than 1Gbps have this function; it's also very common for the IBM NIC drivers as well as drivers in the Linux world. The NIC driver in Linux is an adjustable kernel parameter even with NICs working at 100Mbps. The advantage of this approach is that the CPU doesn't need to respond to every arrived packet interruption request. The arriving packets are saved at the buffer and processed by the CPU as a batch. However, this may influence the available bandwidth measurement algorithm design. For the available bandwidth measurement algorithms, one solution is to tune/cancel interruption coalescence through driver software like Intel 82540EM Gigabit network controllers or giving timestamps at the NIC when measurement is proceeding. Apparently, this approach is not practicable in most situations in the real world. Some modern bandwidth measurement algorithms have functions to detect the interruption coalescence and then adjust their probe packets accordingly. Certainly, applying the packet train to detect the interruption coalescence is a better choice than packet pair.

The PoissonProb algorithm determines whether the sink with interruption coalescence is set or not by observing the timestamps from returning packets at the first probe round. At the first round, PoissonProb sends out the probe packets back-to-back to the sink. The purpose is to infer the bottleneck bandwidth along the path and interruption coalescence. If approximately same timestamps (within 5 μs) are found for continuous probe packets, then it determines that the other end NIC is working with interruption coalescence and the number of packets with the same timestamps may be causing one interruption. However, not all interrupt coalescence is implemented with hard coded number of packets, which may cause the interruption. The possible implementations which determine how to generate the interrupt request also include the maximum number of interrupts per second, the delay between the arrivals of the first packet after the last interrupt or the delay between the last arrival of a packet and the generation of a new interrupt. To cope with the variety of implementations of interruption coalescence, PoissonProb sends several packets trains back-to-back to the other end and then determines the coalescence number from the average. PoissonProb also has one sanity check process to trace the gap changes of arriving probe packets to distinguish the context switch and interruption coalescence. The typical signature of context switch and interruption coalescence is as follows:

In the PoissonProb implementation, the length of probe packets train is 60. This is enough for samples spanning one context switch and several interruption coalescence of preferred measuring links with bandwidth less than 400Mbps. PoissonProb also can automatically adjust the train length to guarantee that there are at least 10 valid samples in one probe attempt, if the interruption coalescence number is larger than 6. After each round of measurement, the sanity check process will filter out the context switch, interruption moderation and the random network behaviour effects.

Figure 3-4 Context Switch and Interrupt Coalescence (reference [19] page 5, figure 2)



3.6.5 System I/O Bandwidth

System I/O and the memory bandwidth are real bottlenecks of an end host system [7]. The speed of either PCI bus or ISA bus is increasingly slower than that of the other parts of the end host system. Though some techniques have improved speed of memory bandwidth in a faster track, it is a trivial improvement for the processes, as they have to access memory through the system bus. Some solutions have been proposed and implemented, as zero-copy or symmetric multiple processors (SMP) which partially improve performance. For most of the modern measurement tools, the ability of available bandwidth measurement algorithms is limited by the end system and they end up measuring the capacity of the end host system instead of measuring the capacity of the network.

4. PoissonProb Details

PossionProb is a rate-based algorithm. The current version was implemented in JAVA (PoissonProb UNIX version is only supported by JAVA 1.5 and above). It operates at either the client-server mode (receiver-based) or the standalone mode (sender-based). We describe each of modes separately.

4.1 The Client-server Mode Algorithm

Under the client-server mode, PoissonProb opens two connections for the available bandwidth measurement between the server and the client. One connection is the TCP session which is used for transferring the control information and the other is the UDP connection, which is used for the probe packets transmission. As we mentioned before, we are using the IBMTS under Windows environment and the JAVA system call System.nanoTime() under UNIX environment to get the timestamps for packets at user space. This makes it possible to run the program without the need for administrative privilege. When the measurement starts, PoissonProb client sends the measurement request to the server through the TCP connection. Once the server receives the request, it checks for the available resource ---UDP ports. The server maintains a client connection table for the running client because of the stateless property of UDP connections. If the UDP port is still available, the server informs the client to start measurement at the available UDP port. In the first round, PoissonProb client sends out the probe packets back-to-back to the server. The goals are to find out the interruption coalescence at the receiver and the bottleneck information along the path. The server timestamps each arrived packet and sends back the timestamps information to the client. We are using a histogram method to estimate the bottleneck capacity along the path. This is similar to the idea in the early version of pathrate [5]. Then the PoissonProb separates the first round Poisson distribution packets in the mean inter-packets interval λ of the 1/3 of the bottleneck separation gap. The accurate bottleneck measurement is not so critical for the PoissonProb as for those gap-based algorithms, such as IGI. PoissonProb only uses the bottleneck bandwidth to decrease the convergence time. The other goal of the first round probe is to find out interrupt coalescence information at the server. The method was already discussed in section 3.6.4. At least 10 valid samples for each measurement round are retained so that the results are reliable. In the following measurement, the PoissonProb client sends the Poisson distributed packet train in which each packet has a separate timestamp to the server. The packets interval is determined by the mean inter-packets interval λ and the logarithm (base *e*) of the math random number between 0 to1. The server gives each arrived probe packet a timestamp accordingly. Then the total source gaps and destination gaps are accumulated. The clock skew doesn't influence accuracy of PoissonProb algorithm, since only the intervals between the contiguous probe packets are taken into consideration. If the gaps are the same (*(source gap – destination gap)/ destination gap* <= *0.15*), the server informs the client to stop the measurement and send the timestamps back to the client. Otherwise, the server prompts the client to increase or decrease the value of λ through the TCP connection. If the total destination gap is increased compared with the total source gap, then the λ will be increased by 1/5. On the other hand, it is decreased by 1/5 accordingly if the destination gap is decreased or unchanged. The PoissonProb algorithm gives out the results very fast, usually within 1-10 seconds. The convergence tie depends on the propagation delay, transmission delay and the queuing situation along the path.

There are three mechanisms in PoissonProb to cope with normally unexpected situations of the measured network, which may arise because of the complexity and variety of Internet. One is the maximum number of rounds. Once the maximum of rounds is reached, the server will delete the client connection registration from the table and inform the client about the failure of measurement. Such a case may arise when there are overloaded links along the path and/or the bottleneck capacity estimation deviates too much from the correct value. The client has to re-estimate the bottleneck again, by restarting the process of measurement. The second mechanism is used to take care of packets losses. The packet loss is observed at the server side through the socket time out threshold. If the packet loss is more than 2/3 of the total probe packets, the server will inform the client to stop measurement, as there may be one link that is so heavily overloaded that the incoming packets are dropped. To continue the measurement may make the situation worse. The client may start measurement later. There is the other possibility that the packets are dropped at the end host because of the small receiver buffer or system resources limitation. The small receiver buffer size is a very rare situation as most of modern operating systems offer enough space for the network operation. The system resources limitation has been discussed in section 3.6. Under this situation, the algorithm starts measuring the end host network transmission capacity instead of the link available bandwidth. The third mechanism is that of the regular table check. The server maintains a client connection table. The connection, which is inactive for 5 minutes, is deleted, and the occupied resources (UDP ports, TCP connection and measurements states) are released.

4.2 The Pseudo-Code for the C/S Mode Algorithm

```
{
        // Initialization
        train_length = TrainLength;
        packet_size = PacketSize;
        round =0;
        sendMeasurementRequest ();
        if (Valide() && Server_UDP_port ) {
                sendPackets (0,0); // round 0; probe packet interval 0;
                receiveProbInfor();
                bottleneck_capcity = calculateBottleneckCapctiy();
                initial_gap = bottleneck_capacity/packet_size/3;
                while ((round < MAX_ROUND) && phaseValide()
                        (!(source_ gap – destination_ gap)/ destination gap <= 0.15)) {
                                sendPackets (round++, initial_gap);
                                if ( total_destination_ gap> total_source_gap)
                                        initial_gap += initialgap*1/5;
                                else initial_gap -= initial_gap*1/5;
                }
        }
        sendPackets (round, initial_gap) {
                for ( int i=0; i<train_length; i++) {
                        sendPacket(index, timestamp(), Server_UDP_Port);
                        /* delay logarithm (base e) of the math random number between 0
                        to1* initial_gap */
                        delay(initial_gap); }
        }
}
```

4.3 The Standalone Mode Algorithm

The PoissonProb sender-based or standalone mode is much simpler compared with the client-server mode. The only requirement of the other end host is to open the UDP echo (port 7) facility. As the sending host maintains all the states of the measurement, the load on the other end is reduced to the minimum. However, as we discussed in section 2.2, the sender-based algorithm may have a lower accuracy than the client-server algorithm. The PoissonProb standalone mode may be used in a situation where the software deployment at other end is difficult. During the measurement, the PoissonProb bounce the probing packets to the target host UDP port 7 in the hope that the target may echo the packets back. We assume the probing packets are echoed back through the same route and without being interfered by the cross-traffic. We may apply the statistical algorithm to filter out the cross-traffic and random network effects. The measurement procedure is the same as for the client-server mode. In the first round, the initial or the sending host sends out back-to-back packets to detect the bottleneck information and in the following round, the probe packets distributed in Poisson are bounced out. The measurement host (which is the sending host, in this case) observes the total initial gaps and the total gaps of the coming back packets. On reaching the turning point, the measurement stops. PoissonProb standalone mode requires more measurement samples than the client-server.mode.

4.4 The Pseudo-Code for the Standalone Mode Algorithm

```
{
        // Initialization
        train_length = TrainLength;
        packet_size = PacketSize;
        round =0;
        sendPackets (0,0); // round 0; probe packet interval 0;
        receiveProbPackets();
        bottleneck_capcity = calculateBottleneckCapctiy();
        initial_gap = bottleneck_capacity/packet_size/3;
        while ((round < MAX_ROUND) && phaseValid() &&
                (!(source_ gap – destination_ gap)/ destination gap <= 0.15)) {
                        sendPackets (round++, initial_gap);
                        if ( total_destination_ gap> total_source_gap)
                                initial_gap += initialgap*1/5;
                                else initial_gap -= initial_gap*1/5;
                }

        sendPackets (round, initial_gap) {
                for ( int i=0; i<train_length; i++) {
                        sendPacket(index, timestamp(), 7);
                        /* delay logarithm (base e) of the math random number between 0
                        to1* initial_gap */
                        delay(initial_gap); }
        }
}
```

The following chapter gives the experimental result of the PoissonProb at the network simulator and the network testbed.

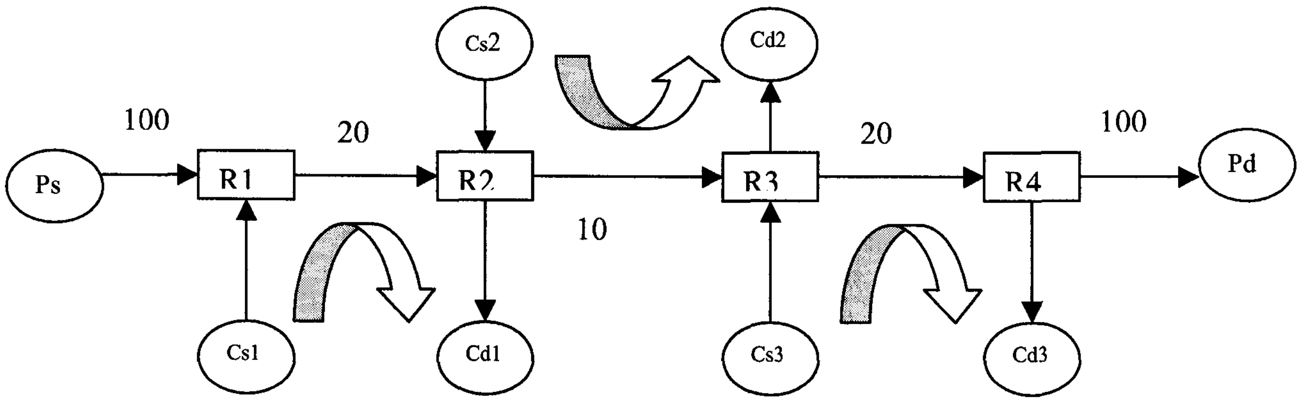
5. Experiments

5.1 NS2 TestBed Description

First, we compared the PoissonProb with the IGI/PTR and the pathChirp on the NS2 (ns-2.26). The reason is that the experiments are repeatable and we may control the network traffic flow so that various network conditions can be simulated. Furthermore, the accurate timestamps for packets at each queue would give us the insights into the interactions between the probing packets and the cross-traffics. The goals of the experiments can be summarized as follows:

- Comparing the accuracy of the measurement algorithms, under the extreme network conditions, such as the strict pre-bottleneck and post-bottleneck at the network edge.
- Comparing the algorithms' convergence time for a multi-hop network.
- Comparing the sensitivity to the network traffic change, when the cross-traffic varies at different links along the path.

Figure 5-1. Multi-hop Network Topology on NS2



The testbed can be described as follows:

This is a four-hop network. The probing packets are sent from Ps to the Pd. Cross-traffics are created from Cs to Cd. As most of the links are in duplex mode nowadays, considering the one-direction measurements does not detract from the generality of the conclusions. All the links' capacity where the packets enter the network is 100Mbps. The links R1-R2 and R3-R4 bandwidth capacity is 20Mbps. Meanwhile, link R2-R3 capacity is 10Mbps. Then the link R2-R3 is the bottleneck link along the path. In the test scenario one, cross-traffic from Cs2 to Cd2 keeps 3 Mbps passing through link R2-R3. When there is no other traffic, link R2-R3 is both the bottleneck link and the tight link. We increase the cross-traffic from Cs1 to Cd1 from 0 to 19Mbps to observe the pre-bottleneck effect, and obviously, after the rate increased to 17Mbps, the link R1R2 becomes the tight link. In test scenario two, there is no cross traffic crossing link R1R2, and the traffic from Cs3 to Cd3 is increased from 0 to 19Mbps to observe the post-bottleneck effect. As in the pre-bottleneck case after the rate increased to 17Mbps, R2R3 becomes the tight kink.

5.2 Results of Experiments using NS2

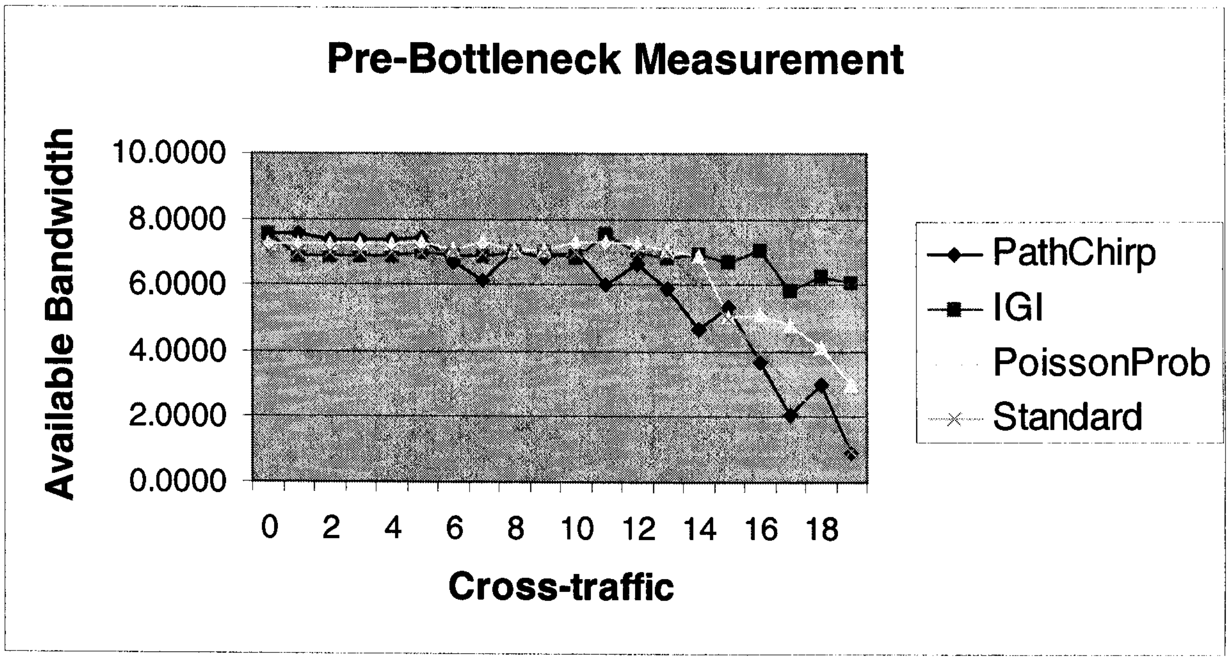
The implementation of pathChirp (pathChirp_ns_2.26) was downloaded from the researchers' website (http://www.spin.rice.edu/Software/pathChirp/). IGI implementation was coded according to the paper's description and validated through original paper's testbed. During the test, the parameters are set by its default. The cross-traffic at NS2 testbed was set as CBR traffic and the packet size was set to 800 bytes. However, when the original parameters setting (*lowrate_, highrate_ and avgrate_*) were kept, we observed ridiculous results from pathChirp. After adjusted the parameters according to the links capacity, the results are reasonable. The same results were observed at the measurement on the test-bed. This may cause pathChirp failing to measure a wide range of bandwidth where we don't know the network properties in advance. The pre-bottleneck measurement results are shown in table 5-1 and the figure 5-2.

Table 5-1. Pre-Bottleneck Measurement on NS2

CrossTraffic (Mbps)	PathChirp (Mbps)	IGI (Mbps)	PoissonProb (Mbps)	Standard (Mbps)
0	7.5332	7.5758	7.2152	7.0000
1	7.5332	6.9018	7.2152	7.0000
2	7.3816	6.9018	7.2152	7.0000
3	7.3816	6.9018	7.2152	7.0000
4	7.3816	6.9018	7.2152	7.0000
5	7.4119	7.0093	7.2785	7.0000
6	6.6647	6.9018	7.0678	7.0000
7	6.1086	6.9018	7.3077	7.0000

8	7.0597	7.0093	7.0370	7.0000
9	6.7905	7.0093	7.0370	7.0000
10	7.0692	6.8389	7.3077	7.0000
11	6.0338	7.5758	7.3077	7.0000
12	6.6315	7.0093	7.2152	7.0000
13	5.8740	6.8182	7.0370	7.0000
14	4.6498	6.9207	6.8771	6.0000
15	5.3180	6.7164	5.0593	5.0000
16	3.7097	7.0355	5.0974	4.0000
17	2.0736	5.8313	4.8148	3.0000
18	2.9860	6.2654	4.0882	2.0000
19	0.8958	6.0497	3.0045	1.0000

Figure 5-2 Comparison of Pre-Bottleneck Measurements on NS2

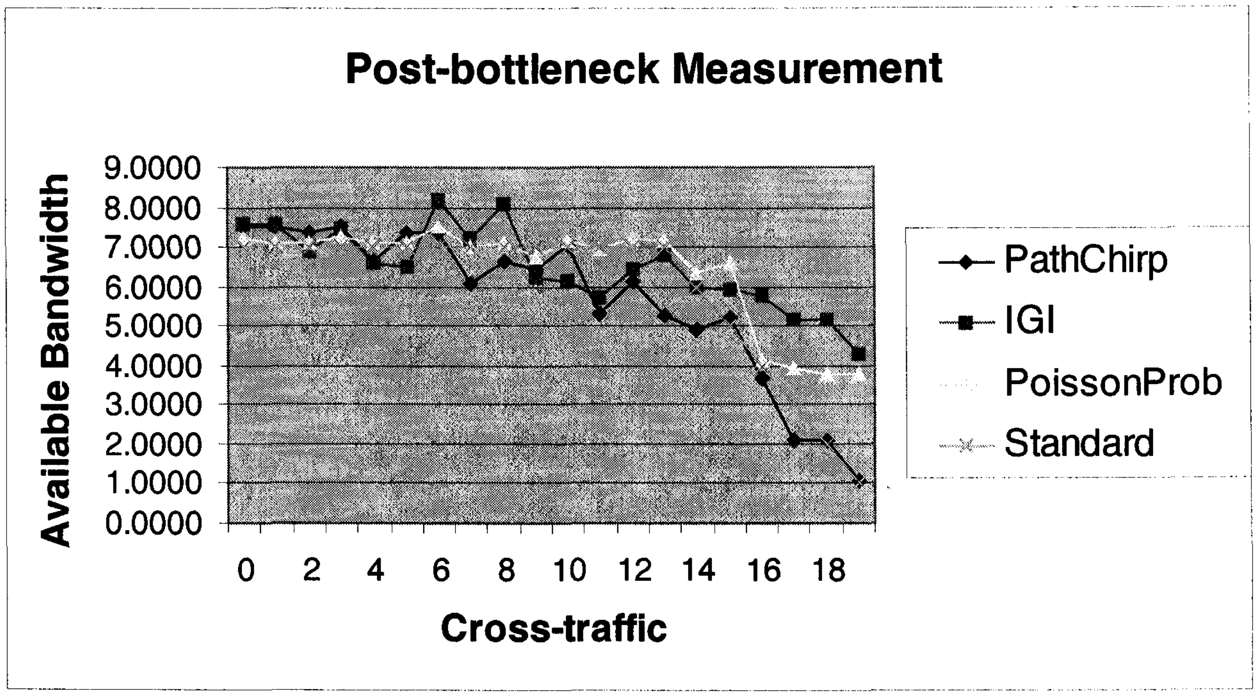


We found that under most of the situation, IGI and PoissonProb may give the results within 5 seconds. In our experiments, pathChirp usually converged to the stable states within 16 seconds. We have shown the first stable result from pathChirp result set for comparison in the Table 5-1 and the Figure 5-2. From the above graph, we may see that the results from PoissonProb are closer to the standard line than other algorithms, except when the available bandwidth below 2Mbps. Apparently, IGI failed to reflect the bandwidth change under current network conditions. Many reasons may contribute to the IGI results. As we discussed in the early section, the measurements exhibited more deviation around the standard curve under the pre-bottleneck conditions. The cross traffic on the ending up tight link has the dominating impact on the probe packet's intervals. IGI focuses on the every packet's interval changes at the receiver side. The bottleneck link behind the tight link distorts the snapshot of cross-traffic effect on probe packets at the tight link so that it displayed more deviations than other two algorithms that took the accumulated gaps between the probe packets as the reference. Researchers in [15] got the same conclusion and provided their analysis as well.

Table 5-2. Post-Bottleneck Measurement on NS2

CrossTraffic (Mbps)	PathChirp (Mbps)	IGI (Mbps)	PoissonProb (Mbps)	Standard (Mbps)
0	7.5332	7.5758	7.2152	7.0000
1	7.5332	7.5758	7.1123	7.0000
2	7.3816	6.9046	7.1123	7.0000
3	7.5332	7.4469	7.2851	7.0000
4	6.6729	6.5721	7.1123	7.0000
5	7.3816	6.5127	7.1123	7.0000
6	7.3816	8.1579	7.4952	7.0000
7	6.0742	7.2039	7.0051	7.0000
8	6.6729	8.0727	7.1123	7.0000
9	6.4346	6.2238	6.7401	7.0000
10	7.0626	6.1123	7.1417	7.0000
11	5.2968	5.7380	6.9617	7.0000
12	6.1532	6.4480	7.2152	7.0000
13	5.2448	6.7829	7.1342	7.0000
14	4.8961	6.0063	6.3333	6.0000
15	5.2249	5.9196	6.5906	5.0000
16	3.6922	5.7915	4.1317	4.0000
17	2.0736	5.1546	3.9455	3.0000
18	2.0736	5.1546	3.8000	2.0000
19	1.0750	4.3176	3.7669	1.0000

Figure 5-3 Post-Bottleneck Measurements on NS2 Comparing

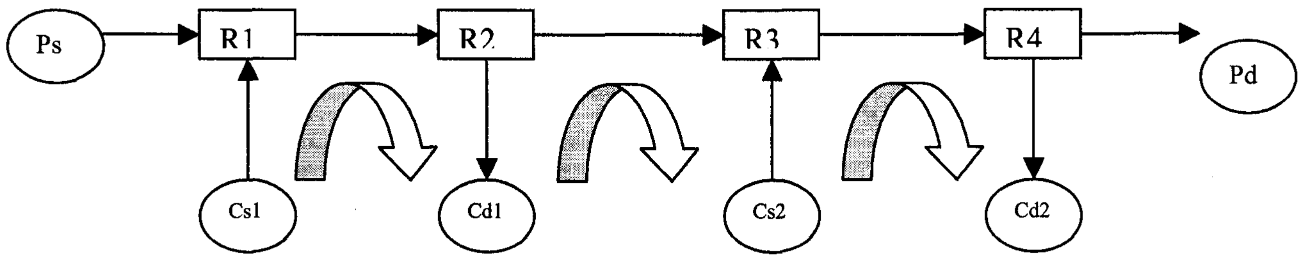


As we expected, all three algorithms presented the better performance at the post-bottleneck situation as shown in table 5-2 and figure 5-3. The cross-traffic on the link R3-R4 dominated the intervals between the probe packets. The snapshots are what the algorithms want to keep to the sink. Though the PoissonProb measurement results are very close to the standard line at the first several rounds. It deviated when the available bandwidth decreased below the 3 Mbps. We'll address this issue in our future works.

5.3 Description of the Network Testbed

In this experiment, we set up a testbed to compare the above three algorithms. The goals of the experiment are the same: comparing the accuracy of the measurement results at pre-bottleneck and post-bottleneck conditions. When the bottleneck link is not the tight link, we observe the performance of the algorithms to the change and their convergence time. The network topology is the same as that applied in NS2. The testbed is comprised of the four Linux machines that work as the routers. The probing packets are sent from Ps to the Pd. Cross-traffics are generated from Cs to Cd as before. Figure 5-4 is the network topology graph.

Figure 5-4. Multi-hop Network Topology



To verify the links' capacity, we use the well-known bandwidth measurement algorithm pathrate-2.3 (http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/) to measure the capacity of each link. Table 5-3 is the links' capacity in Mbps.

Table5-3. Link Capacity Verification

Node	Inbound (Mbps)	Outbound (Mbps)
Ps - R1	97-98	97-98
Cs1 - R1	98-98	97-105*
R1 - R2	98-98	97-98
Cd1 - R2	97-98	98-98
R2 - R3	86-87	98-98
Cs2 - R3	97-98	97-105*
R3 - R4	98-98	98-98
Cd2 - R4	98-98	97-105*
Pd - R4	98-98	97-105*

From above table, we may see that the capacity of all the links is approximately 97-98Mbps (we only consider the direction Ps→Pd only, for reasons stated in section 5.2). In test scenario one, the link between R2 and R3 has a cross-traffic of 30 Mbps from Cd1 to Cs2. When there is no other traffic, link R2-R3 is both the bottleneck link and tight link. We increase the traffic from Cs1 to Cd1 from 0 to 90Mbps to observe the pre-bottleneck effect. Obviously, after the rate of cross-traffic becomes higher than 30Mbps, the link R1-R2 becomes the tight link. In test scenario two, there is no cross-traffic crossing link R1-R2, and the cross-traffic from Cs2 to Cd2 is increased from 0 to 90Mbps to observe the post-bottleneck effect. Again, when the rate of cross-traffic increases above 30Mbps, the R2-R3 link becomes the tight kink.

To simulate the Internet traffic, we use Poisson traffic generator and the packet size is chosen as 1000Bytes. Table 5-4 shows the verification of the traffic generator. We verified the generator at three time scales, 10 seconds, 60 seconds and 3600 seconds respectively according to the requirement of our measurement algorithm.

Table 5-4 Result of Experiments for Verification of the Poisson Cross-traffic Generator

Average Traffic (Mbps)	Time Scale (10 secs)	Time Scales (60 secs)	Time Scales (3600 secs)
10	10.059786	10.060956	10.063651
20	20.032575	20.031342	20.026537
30	30.060581	30.066570	30.002312
40	39.967955	39.972571	39.082841
50	49.758193	49.779028	49.763211
60	59.304507	59.269797	59.268689
70	68.937374	68.964083	68.907604
80	78.234092	78.258344	78.214785
90	87.356155	87.313973	87.316224

5.4 Results from the Experiments on the Test Bed

The version of pathChirp is 2.4.1 which was downloaded from the researchers' website (http://www.spin.rice.edu/Software/pathChirp/). IGI/PTR source code was downloaded from the researchers' website (http://gs274.sp.cs.cmu.edu/www/igi/). The tables below present the pre-bottleneck measurements results. We adjusted the pathChirp parameters to adapt to the links' bandwidth properties since these were known to us for the test-bed. In this experiment, we gave each algorithm one minute to measure the available bandwidth that the networks are keeping for one minute accordingly, and then took the average of the measurement results as the final result.

Table 5-5 IGI Pre-bottleneck Measurements

MAX	MIN	AVER	CT	EBW
82.07	62.86	70.51	0.00	67-68
75.88	61.95	70.53	10.00	67-68
72.32	59.27	66.05	20.00	67-68
73.70	49.82	63.22	30.00	67-68
68.41	48.66	58.96	40.00	57-58
61.31	53.16	56.75	50.00	48-49
59.55	41.66	51.30	60.00	38-39
59.27	40.57	50.68	70.00	29-30
62.98	17.93	36.64	80.00	19-20
36.43	17.93	27.01	90.00	10-11

Table 5-6 PoissonProb Pre-bottleneck Measurements

MAX	MIN	AVER	CT	EBW
88.37	59.19	72.82	0.00	67-68
80.58	62.55	70.28	10.00	67-68
81.98	53.39	70.37	20.00	67-68
80.38	58.55	68.87	30.00	67-68
66.87	49.20	57.57	40.00	57-58
63.23	35.82	47.93	50.00	48-49
53.62	17.90	39.32	60.00	38-39
42.10	17.09	25.94	70.00	29-30
34.42	15.57	19.48	80.00	19-20
17.53	8.23	12.65	90.00	10-11

Table 5-7 PathChirp Pre-bottleneck Measurements

MAX	MIN	AVER	CT	EBW
76.97	58.36	67.49	0.00	67-68
72.47	55.01	66.34	10.00	67-68
78.53	63.89	70.11	20.00	67-68
77.39	62.53	69.61	30.00	67-68
70.01	55.71	63.82	40.00	57-58
66.78	46.79	55.48	50.00	48-49
49.73	42.27	45.29	60.00	38-39
44.79	25.02	33.73	70.00	29-30
21.00	16.54	18.85	80.00	19-20
20.71	12.38	16.23	90.00	10-11

Figure 5-5 Pre-Bottleneck Measurements Comparing

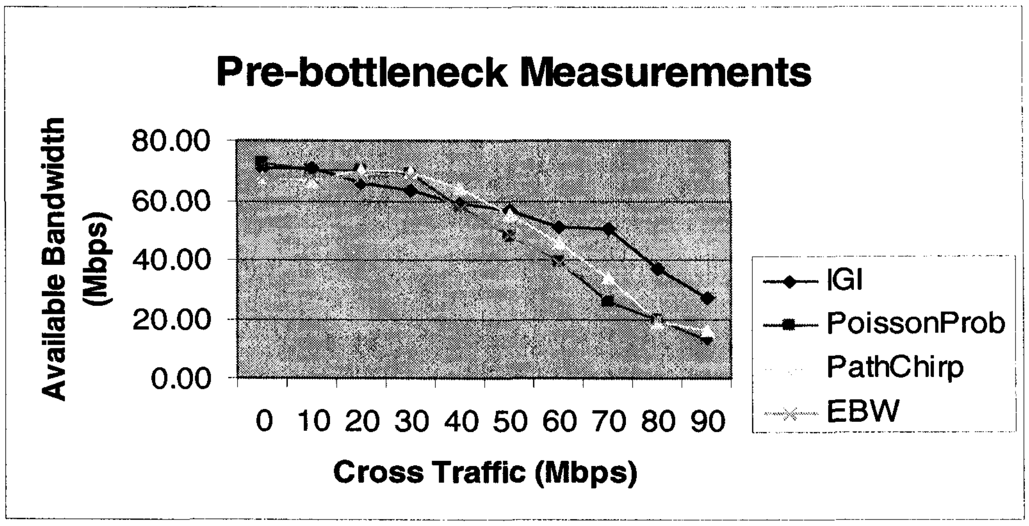


Figure 5-6 Pre-Bottleneck Measurements Error Rate

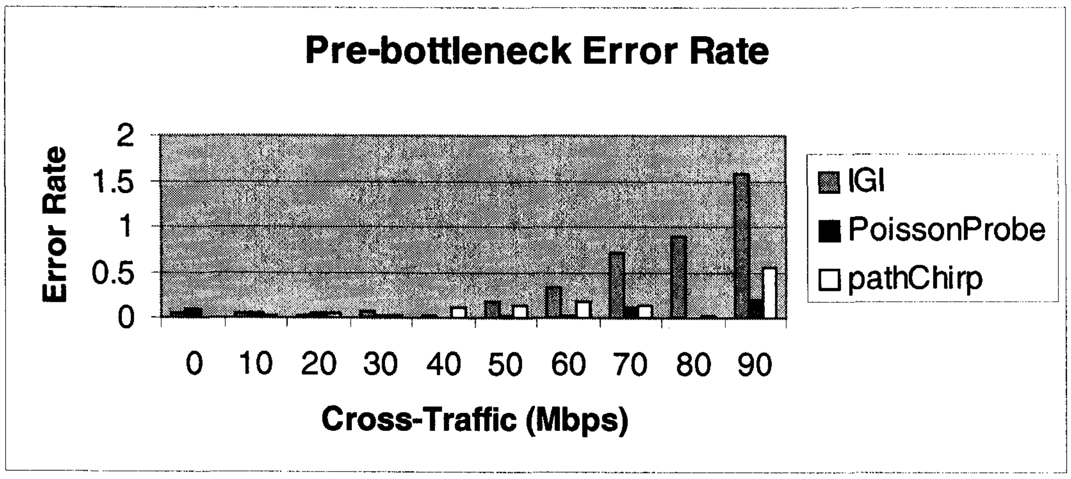


Table 5-5 – 5-6 are measurement results of three algorithms. The MAX and MIN columns showed the maximum and minimum measurement results algorithms reported. We took the average (AVER) measurement results compared with the estimated available bandwidth (EBW). The figure 5-5 is the graph shown the pre-bottleneck comparing and the figure 5-6 shown the error rate of the three algorithms. Here, the error rate δ is defined as

$$\delta = \frac{|MeasurementAB - \mathrm{Re}\,alAB|}{\mathrm{Re}\,alAB}$$

From the figure 5-6, we may get the same conclusion as that got from the NS2 experiment. However, as all the three algorithms applied the statistical techniques to filter out the probe deviations in their implementation. They showed better performance than that on NS2 testbed.

Table 5-8 IGI Post-Bottleneck Measurements

MAX	MIN	AVER	CT	EBW
57.09	43.74	48.14	10.00	67-68
69.14	44.30	49.80	20.00	67-68
48.05	41.57	45.19	30.00	67-68
69.93	56.74	61.94	40.00	57-58
63.44	45.94	56.77	50.00	48-49
42.28	31.22	38.01	60.00	38-39
47.00	26.71	38.34	70.00	29-30
39.57	18.83	26.76	80.00	19-20
30.98	16.00	22.55	90.00	10-11

Table 5-9 PoissonProb Post-Bottleneck Measurements

MAX	MIN	AVER	CT	EBW
78.75	57.79	71.34	10.00	67-68
79.71	64.74	72.60	20.00	67-68
77.90	52.40	67.79	30.00	67-68
70.94	44.88	57.41	40.00	57-58
55.31	39.62	48.67	50.00	48-49
50.54	19.99	34.24	60.00	38-39
38.56	15.99	22.44	70.00	29-30
31.16	17.54	20.16	80.00	19-20
15.98	7.03	11.76	90.00	10-11

Table 5-10 PathChirp Post-Bottleneck Measurements

MAX	MIN	AVER	CT	EBW
94.18	69.34	79.83	10.00	67-68
83.84	76.19	80.71	20.00	67-68
89.42	59.10	71.31	30.00	67-68
71.04	56.31	64.40	40.00	57-58
53.40	45.43	49.01	50.00	48-49

55.85	42.52	46.29	60.00	38-39
40.00	28.92	38.62	70.00	29-30
21.97	14.44	16.21	80.00	19-20
18.73	9.62	15.48	90.00	10-11

Figure 5-7 Post-Bottleneck Measurements Comparing

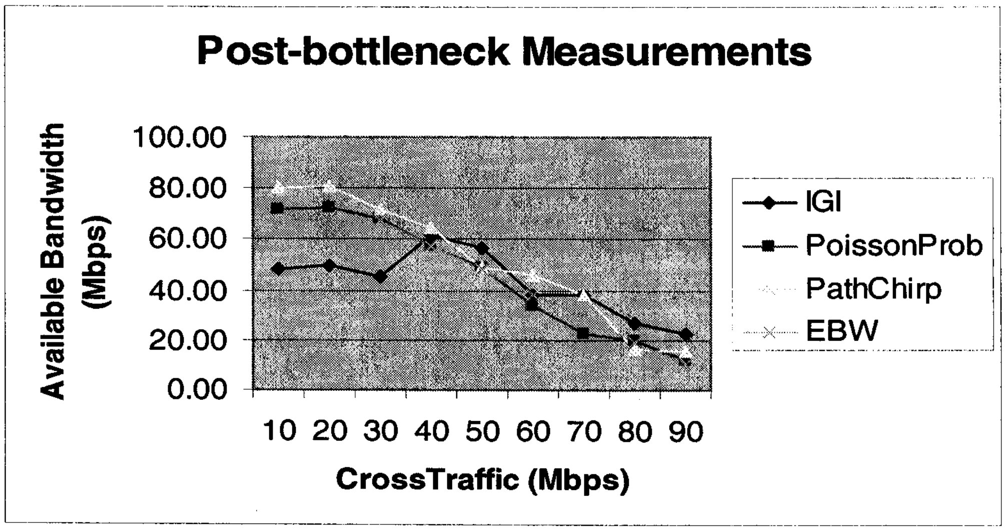


Figure 5-8 Post-Bottleneck Measurements Error Rate

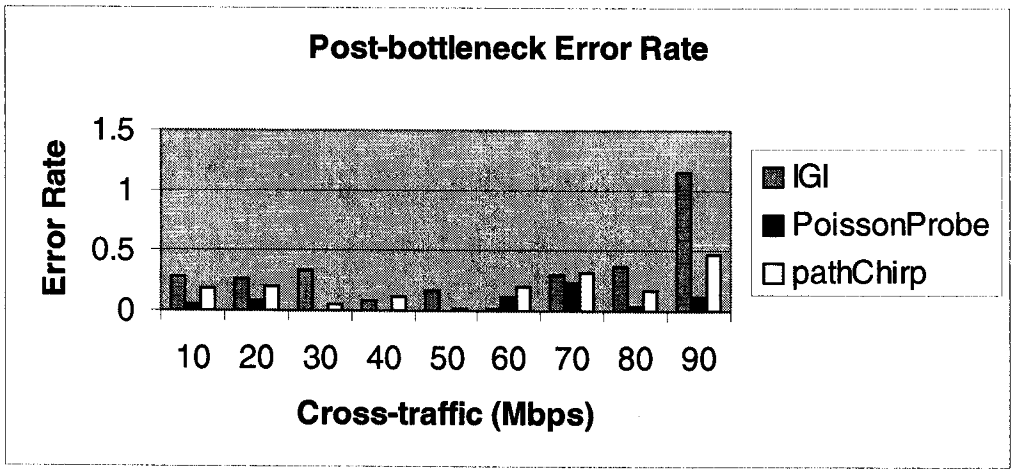
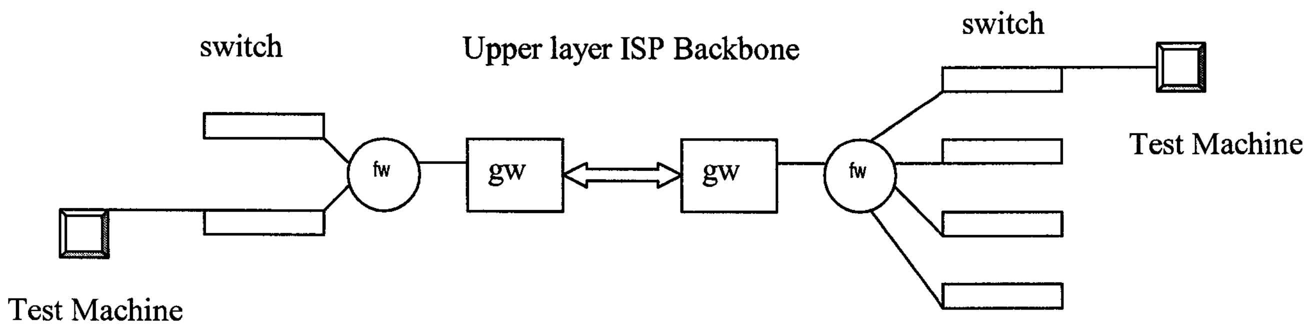


Table 5-7 – 5-8 are measurement results of three algorithms under the post-bottleneck conditions. Compared figure 5-8 with figure 5-6, again, we got the lower error rate of all three algorithms under post-bottleneck conditions than that got under the pre-bottleneck conditions. When the cross-traffic under 30Mbps on the ending up tight link, the

bottleneck link is also the tight link. However, the final tight link distorted the intervals between the probe packets. Then the algorithms produced higher error rate than those produced at last several rounds.

5.5 Measurement Results Between Gateways

Figure 5-9 ISP Network Topology



To further evaluate three algorithms, we deployed programs behind two firewalls within a small ISP network whose gateways are connected through the upper layer ISP backbones. Figure 5-9 is the network topology we used for measurement. As one of the firewall is configured with TC rules (Traffic Shaping/Control) that limit the ceiling rate of egress port to 15Mbps, this link, between the firewall (fw) and the gateway (gw) is apparently both bottleneck link and the tight link. The other links are only limited by the gateway capacity, which is 100Mbps. The ISP authorities permitted us to run the programs for three days (Wednesday, Thursday and Friday) from 0:00AM to 17:00PM. Each program was scheduled to run one minute during the 5 minutes intervals. We took the average output of measurement and compared it with the SNMP results (layer 2 link utilization) that were read from the network card every 5 minutes by using the monitoring server. The results are shown as figure 5-10 to 5-12. We also compared the error rate which defined as before, for the three algorithms, as shown in figure 5-13. PoissonProb showed the superior performance (average error rate: PoissonProb 0.17; IGI: 0.26; pathChirp: 0.20).

Figure 5-10 Measurement Result Between Gateways (Wed)

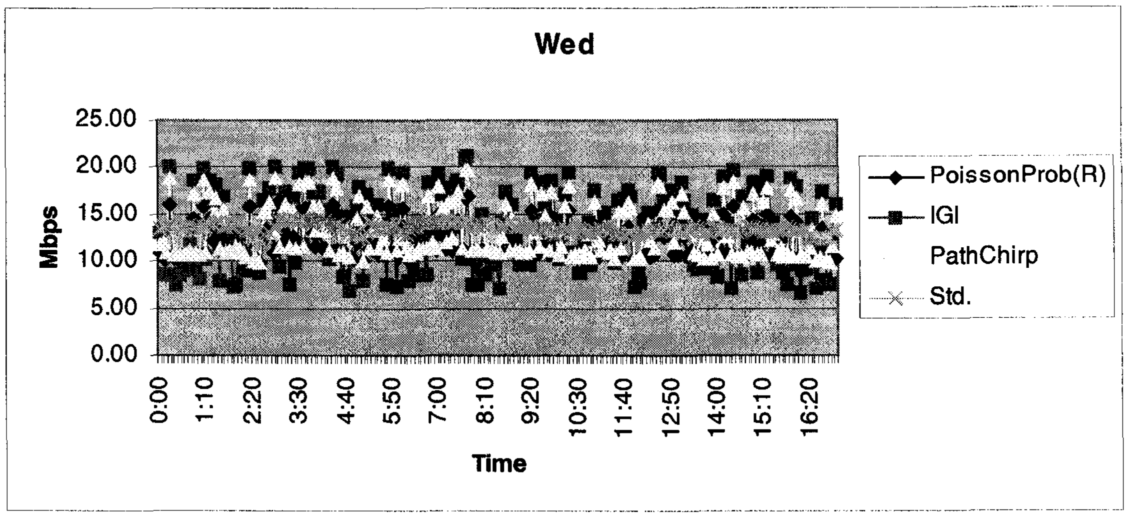


Figure 5-11 Measurement Result Between Gateways (Thur)

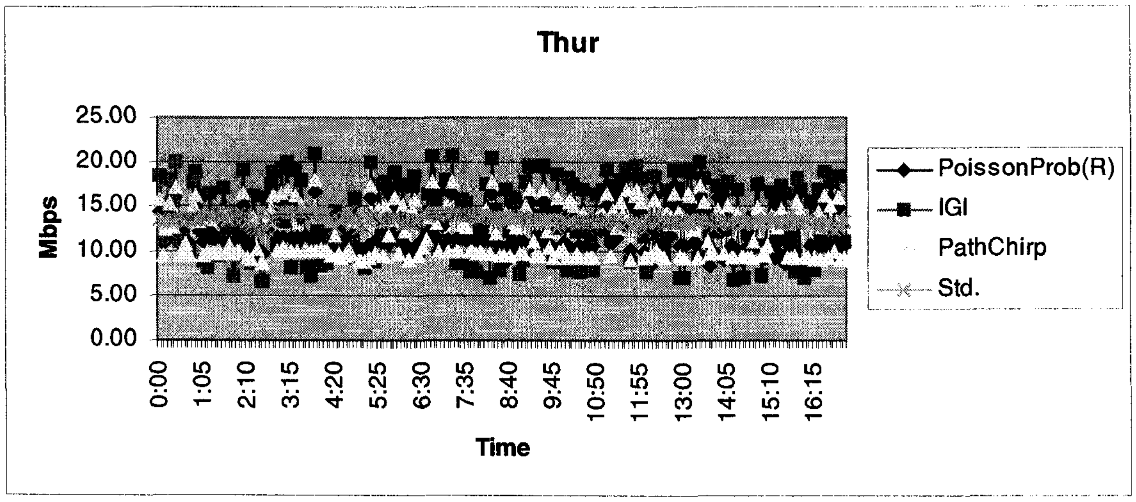


Figure 5-12 Measurement Result Between Gateways (Fri)

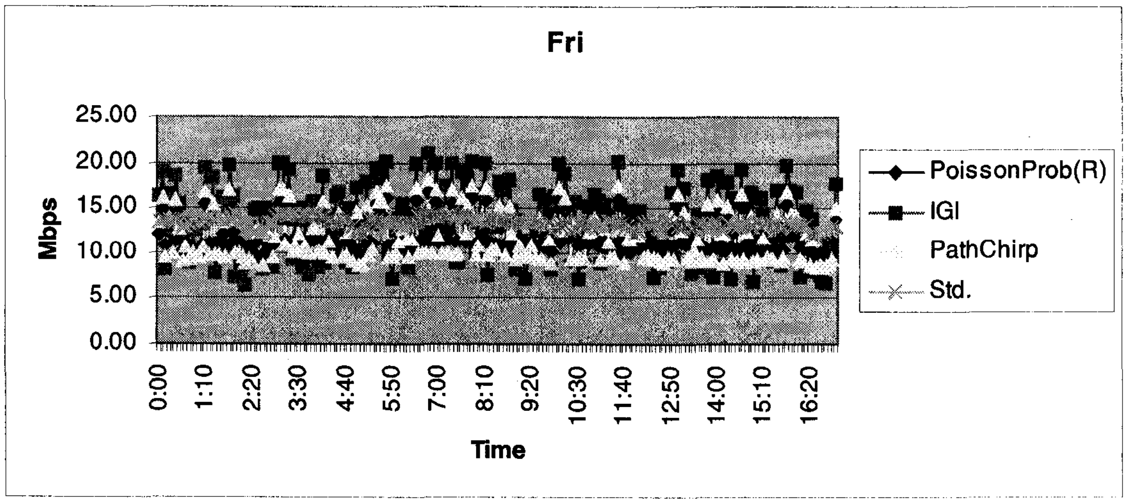
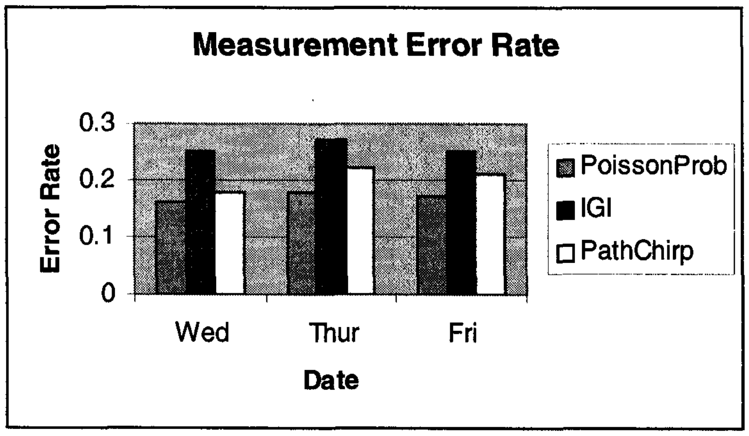


Figure 5-13 Measurement Error Rate between Gateways



5.6 Tests of the Sender-based Algorithm on NS2

PoissonProb can also be operated in the standalone mode (sender-based mode), which decreases the load at the other end. Here, we compare the PoissonProb with Cprobe which is the well-known sender-based available bandwidth measurement algorithm. Both the sender-based algorithms were tested on the NS2 testbed under the same conditions that were used for measurements of the receiver-based algorithms. The following are the measurement results.

Table 5-11 Sender-based Pre-Bottleneck Measurement

CrossTraffic (Mbps)	PoissonProb (Mbps)	Cprob (Mbps)	Standard (Mbps)
0	7.2152	7.3567	7.0000
1	7.2152	6.7355	7.0000
2	7.2152	6.7355	7.0000
3	7.2152	6.7355	7.0000
4	7.2152	6.7355	7.0000
5	7.2785	6.8236	7.0000
6	7.0678	6.7355	7.0000
7	7.5523	6.7355	7.0000
8	7.5248	6.7355	7.0000
9	7.4643	6.8236	7.0000
10	7.3077	6.6576	7.0000
11	7.3077	7.3567	7.0000
12	7.2152	6.8236	7.0000

13	7.0370	6.6703	7.0000
14	6.8771	6.7532	6.0000
15	5.0593	6.5708	5.0000
16	5.0974	6.8427	4.0000
17	4.8148	5.7282	3.0000
18	3.9649	6.1428	2.0000
19	4.4048	5.9403	1.0000

Figure 5-14 Sender-based Pre-Bottleneck Measurement Result

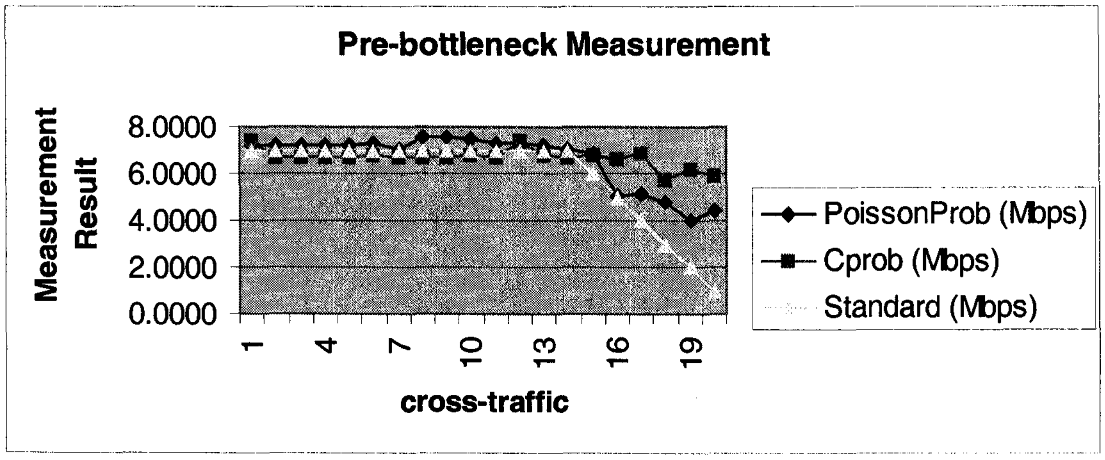
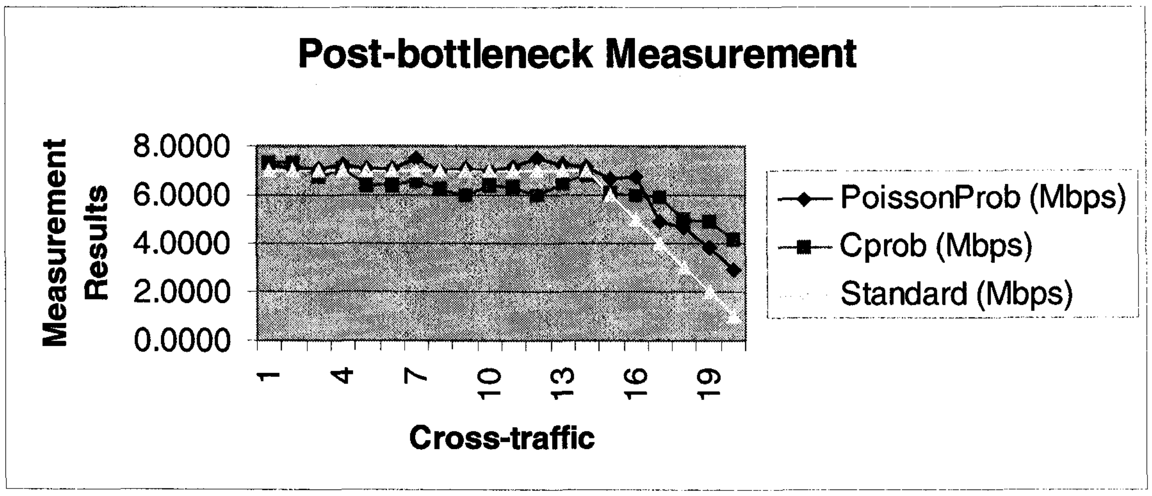


Table 5-12 Sender-based Post-Bottleneck Measurement

CrossTraffic (Mbps)	PoissonProb (Mbps)	Cprob (Mbps)	Standard (Mbps)
0.0000	7.2152	7.3567	7.0000
1.0000	7.1123	7.3567	7.0000
2.0000	7.1123	6.7385	7.0000
3.0000	7.2851	7.0858	7.0000
4.0000	7.1123	6.4485	7.0000
5.0000	7.1123	6.3976	7.0000
6.0000	7.4952	6.6170	7.0000
7.0000	7.0051	6.2664	7.0000
8.0000	7.1123	6.0099	7.0000
9.0000	7.0185	6.4311	7.0000
10.0000	7.1417	6.3555	7.0000
11.0000	7.4737	5.9704	7.0000
12.0000	7.2152	6.5366	7.0000
13.0000	7.1342	6.8443	7.0000
14.0000	6.6334	6.0655	6.0000
15.0000	6.7692	6.0147	5.0000

16.0000	4.9524	5.8961	4.0000
17.0000	4.6847	5.0166	3.0000
18.0000	3.8000	4.9533	2.0000
19.0000	2.9420	4.1992	1.0000

Figure 5-15 Sender-based Post-Bottleneck Measurement Result



PoissonProb algorithm uses the UDP echo while the Cprobe uses the Ping echo packets. Cprobe is the gap-based algorithm, which depends on Bprobe to infer the bottleneck capacity first, and then the available bandwidth is calculated through bottleneck capacity minus the utilization of the bottleneck link. PoissonProb is a rate-based algorithm which compares the accumulated source and destination gaps. Then it infers the available bandwidth through the average probe packets rate when they are bounced back. Another difference between the two is that Cprobe send out the probe packets back-to-back, in other words, as fast as possible. This can easily congest the measured network while the PoissonProb algorithm sends out the probe packets in Poisson distribution and the rate measurement is started at the one third of the bottleneck capacity. From the Figure 5-14 and 5-15, PoissonProb apparently showed better performance than that of Cprobe.

6 Conclusion and Future Work

To design an algorithm, which measures the network characteristics accurately, is a challenging work, with the Internet growing exponentially in both complexity and scale. However, it is the critical requirement for many network engineering aspects, such as the network protocols and distributed programs design, traffic optimization, capacity planning, and service verification. The currently available bandwidth measurement algorithms face the problems of distortion of measurement on multi-hop paths. Especially when the path consists of more than ten hops the accuracy of measurement deceases sharply. System resource limitations on high-speed networks can lead the algorithms to measure the end host capacity instead of a link's available bandwidth. Probe traffic intrusiveness and the measurement accuracy are other issue of concern. We have developed a new rate-based algorithm --- PoissonProb. The intervals between probe packets of this algorithm are in Poisson distribution format and the algorithm infers the available bandwidth according to the measured change of intervals. The algorithm has been implemented as the PoissonProb Available Bandwidth (PAB) measurement tool. The PAB tool can be operated in either sender-based or receiver-based mode. We have been able to test for the available bandwidth at Gbps networks in NS2 and in the range from several Mbps to 400Mbps on a test-bed consisting of common desktops. The measurement of available bandwidth on the test bed is limited by the end host bottleneck. Another feature of this algorithm is that it can be operated under both Windows and UNIX environments.

We have compared the PAB tool with C-Probe, PathChirp and IGI, the three algorithms, which are normally used today. However, the three algorithms can work only on UNIX environment. We present the measurement results of the three algorithms on NS2, lab testbed and an ISP network. Even with Windows environment, we are able to obtain the same or even better accuracy and efficiency as for other three algorithms in the Linux environment.

It is hard to thoroughly evaluate the algorithms through the network simulator and the small-scale network where the links' variety and the cross-traffic complexity may not appear. The production tests on the backbone networks and the edge networks with multiple links may gain us an insight into the available bandwidth properties so that further tuning of PoissonProb may become possible.

Reference:

[1] Andre Broido, Young Hyun and K. C. claffy, "*Their share: diversity and disparity in IP traffic*", to appear at the PAM conference in 2004.

[2] Athanasios Papoulis and S. Unnikrishna Pillai, "*Probability, random variables, and stochastic processes*" McGraw-Hill, 2002.

[3] Bob Melander, Mats Bjorkman and Per Gunningberg, "*A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks*", in IEEE Global Internet Symposium, 2000.

[4] Constantinos Dovrolis and Manish Jain, "*End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput*", IEEE/ACM Transactions in Networking, Aug. 2003.

[5] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore, "*Packet Dispersion Techniques and Capacity Estimation*", IEEE/ACM Transactions in Networking - Revised: March 2004.

[6] Guanghui He and Jennifer C. Hou, "*On Exploiting Long Range Dependence of Network Traffic in Measuring Cross Traffic on an End-to-end Basis*", INFOCOM 2003. Twenty-Second Annual Joint Conferences of the IEEE Computer and Communications Societies. IEEE, Volume: 3, 30 March - 3 April 2003, Pages: 1858 – 1868.

[7] Guojun Jin and Brian L. Tierney, "*System Capability Effects on Algorithms for Network Bandwidth Measurement*", Proceedings of the Internet Measurement Conference Oct. 27-29, 2003, Miami, Florida, LBNL-48556.

[8] Guojun Jin, George Yang, Brian R. Crowley, and Deborah A. Agarwal, "*Network Characterization Service*", in Proceedings of 10th IEEE Symposium on High Performance Distribute Computing, August 2001.

[9] J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun, "*Internet Traffic Tends Toward Poisson and Independent as the Load Increases*," in Nonlinear Estimation and Classification, New York: Springer, 2002, pp. 83--109.

[10] Jacob Strauss, Dina Katabi and Frans Kaashoek, "*A Measurement Study of Available Bandwidth Estimation Tools*", Internet Measurement Workshop Proceedings of The Conference on Internet Measurement Conference, Miami Beach, FL, USA, Oct., 2003

[11] K.C. Claffy and S. McCreary, "*Trends in Wide Area IP Traffic Patterns*", technical report, CAIDA, Feb. 2000.

[12] K.C. Claffy and S. McCreary, "*Internet Measurement and Analysis: Passive and Active Measurement*", American Statistical Association, 1999.

[13] Kevin Lai, "*Measuring the Bandwidth of Packet Switched Networks*", Ph.D. thesis, Department of Computer Science, Stanford University, October 2002.

[14] Konstantina Papagiannaki, N. Taft, Z. Zhang, C. Diot, "*Long-Term Forecasting of Internet Backbone Traffic: Observations and Initial Models*", in IEEE INFOCOM, San Francisco, U.S.A., April 2003.

[15] Manish Jain and Constantinos Dovrolis, "*Ten Fallacies and Pitfalls in End-to-End Available Bandwidth Estimation*", In the Proceedings of the ACM Internet Measurements Conference (IMC), Sicily Italy, October 2004.

[16] Manish Jain and Constantinos Dovrolis, "*Pathload: A Measurement Tool for End-to-end Available Bandwidth*", in Proceedings of the 3rd Passive and Active Measurements Workshop, Mar. 2002.

[17] Matthew Mathis and Mark Allman, "*A Framework for Defining Empirical Bulk Transfer Capacity Metrics*", RFC 3148, Jul. 2001.

[18] Ningning Hu and Peter Steenkiste, "*Evaluation and Characterization of Available Bandwidth Probing Techniques*", IEEE Journal on Selected Areas in Communications, 2003.

[19] Ravi Prasad, Manish Jain and Constantinos Dovrolis, "*Effects of Interrupt Coalescence on Network Measurements*", Passive and Active Measurements (PAM) conference, April 2004.

[20] Robert L. Carter and Mark E. Crovella, "*Dynamic Server Selection Using Bandwidth Probing in Wide-area Networks*", Technical Report BU-CS-96-006, Boston University, 1996.

[21] Sally Floyd and Vern Paxson, "*Wide-Area Traffic: The Failure of Poisson Modeling*", IEEE/ACM Transactions on Networking, Vol. 3, No. 3, pp. 226-244, June 1995.

[22] Thomas Karagiannis, Mart Molle and Michalis Faloutsos, "*A Nonstationary Poisson View of Internet Traffic*", Presented at the Infocom in 2004.

[23] Vinay J. Ribeiro, Rudolf H. Riedi, Matthew Crouse and Richard G. Baraniuk, "*Multiscale Queuing Analysis of Long-Range-Dependent Network Traffic*", INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE, Volume: 2, 26-30 March 2000, Pages: 1026 – 1035.

[24] Vinay J. Ribeiro, Rudolf H. Riedi, Richard Baraniuk, Jiri Navratil, and Les Cottrell, "pathChirp*: Efficient Available Bandwidth Estimation for Network Paths*", in Proceedings of Passive and Active Measurements workshop, Apr. 2003.

[25] Vinay J. Riberio, Mark Coates, Rudolf H. Riedi, Shriram Sarvotham, Brent Hendricks, and Richard Baraniuk, "*Multifractal Cross-Traffic Estimation*", in Proceedings ITC Specialist Seminar on IP Traffic Measurement, Modeling, and Management, Sep. 2000.

[26] W. Leland, M. Taqqu, W.Willinger, and D.Wilson, "*On the Self-Similar Nature of Ethernet Traffic*," IEEE/ACM Transactions on Networking, vol. 2, pp. 1–15, 1994.

[27] Yin Zhang, Nick Duffiled, Vern Paxson, and Scott Shenker, "*On the Constancy of Internet Path Properties*", in Proceedings of ACM SIGCOMM Internet Measurement Workshop, page 197-211, Nov. 2001.

Appendix

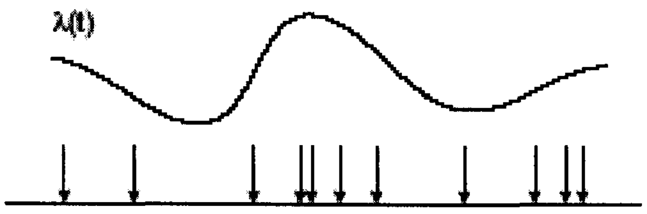
A.1 Poisson Processes and Poisson Traffic

Poisson processes have been historically used for modeling the packet arrivals and packets queuing time for a system. The formula for the Poisson probability mass function is:

$$p(x,\lambda)=\frac{e^{-\lambda}\lambda^{x}}{x!} \quad \text{for } x=0,1,2\ldots$$

λ is the shape parameter which indicates the average number of packets arrival in a given time interval. This is also referred to as the intensity rate. There are two key characteristics of the Poisson distribution to describe the packet arrivals: the inter-arrival times are exponentially distributed and independent. For modeling a network, the Inhomogeneous Poisson Process is usually used to describe the packets arrival. The difference is that a Poisson process has a constant intensity λ whereas the inhomogeneous Poisson process can be generalized as with the intensity that varies with time $\lambda(t)$. Figure A-1 shows an example of the inhomogeneous Poisson process. (Note. $\lambda(t)$ is a deterministic function of time.)

Figure A-1 Inhomogeneous Poisson Process



Then, the probability of an arrival in a short interval of time $(t, t + dt)$ is now $\lambda(t)dt + o(dt)$.

This model was widely applied in network engineering till the early-90's. The important studies [21][26] at that time had shown that the LAN and WAN traffics deviate considerably from the Poisson process, as the exponential distribution underestimates the burstiness of traffic. The packet inter-arrivals had a marginal distribution that had a heavy longer tail than the exponential. Paxson and Sally in [21] concluded: "*wide-area traffic is much burstier than Poisson models predict, over many time scales.*" and "*in some cases commonly-used Poisson models seriously underestimate the burstiness of TCP traffic over a wide range of time scales (time scales of 0.1 seconds and larger).*" They showed that LAN and WAN packet arrival processes appear better modeled using self-similar processes and long-range dependence. This resulted in many other studies in this area and has greatly influenced the network modeling, protocols, algorithms, and network design for a decade.

Beginning in 2000, studies showed that the Internet had grown rapidly in diversity and disparity [1] and the nature of traffic had changed significantly. The speed of links has increased by several orders of magnitude and each link had greater connectivity. A new statistical phenomenon of Internet traffic has appeared to dominate packet arrival modeling. That is network multiplexing. A recent study has shown that the network traffic can again be modeled by the Poisson distribution. The reason is that the large number of simultaneous active connections cause a dramatic change in the statistical properties of packet traffic on an Internet link [9]. The long-range dependence is weakened with the standard deviation of the counts relative to the mean getting small; it is especially apparent for the backbone links with contemporary loads of thousands of connections. The other is that the high-speed link has the capacity to drain the packets so fast that " *the increasing connection load can bring the traffic to Poisson and independence before substantial upstream queuing occurs; the onset of queuing does not resurrect the long-range dependence* " [9]. The final result is that the bursty single network traffic cannot change the high degree of multiplexing links (connections), even though they are still busty as a single individual connection. This has been theoretically and empirically proven through packet analysis on the over provisioned links. Apparently, the packet arrival shows characteristics of Poisson distribution again. For a heavily loaded link, the packets arrive back-to-back, and then the distribution of the arrival depends on the packet size from the point of view of the transmitter. The packet size appears to be independent through the large-scale packets dataset analysis [22]. Certainly, for the edge links with limited connection load, the traffic is still showing the burstiness, self-similarity and long-range dependence characteristics. But on links with high speeds, towards the core of the internet, and carrying traffic made up of a large numbers of connections, the traffic is close to Poisson and independence.

The next important factor of network traffic modeling and measurement is the time scale. As we discussed in the earlier sections, we found distinct differences of network statistical properties when we observe the Internet traces at different time scales. There is no doubt the Internet traffic appears self-similar and long-range dependent at the large time scale. This can be explained in a simple example. No matter whether it is an ISP backbone link or the campus trunk, it could draw the regular load curves through the SNMP enquiry. These curves display similarity in the monthly, weekly or daily periods that can easily predict the average load of a dedicated link in the above time scales even in an hourly time scale. But for most applications, they may want the bandwidth information at the time scale of millisecond to minute level. At this time scale, the traffic is usually non-stationary and may show absolutely different properties compared with average properties of long-term time scales. Karagiannis et al [22] have shown *"packet arrivals appear Poisson at sub-second time scales; Internet traffic is nonstationary at multi-second time scales; Internet traffic exhibits long-range dependence (LRD) at large time-scales".* The above studies and findings are very important for the network measurement algorithms design that any algorithm has to take into considerations no matter what kind of network model it is based on. Meanwhile, we should not attempt to normalize available bandwidth measurements relative to the long-term average value. The measurement, prediction and normalization periods should fall into the same time scale. For the PoissonProb algorithm, it is best used to estimate the available bandwidth on high load links within seconds level time scales.

A.2 Using Poisson Processes to Model Internet Traffic

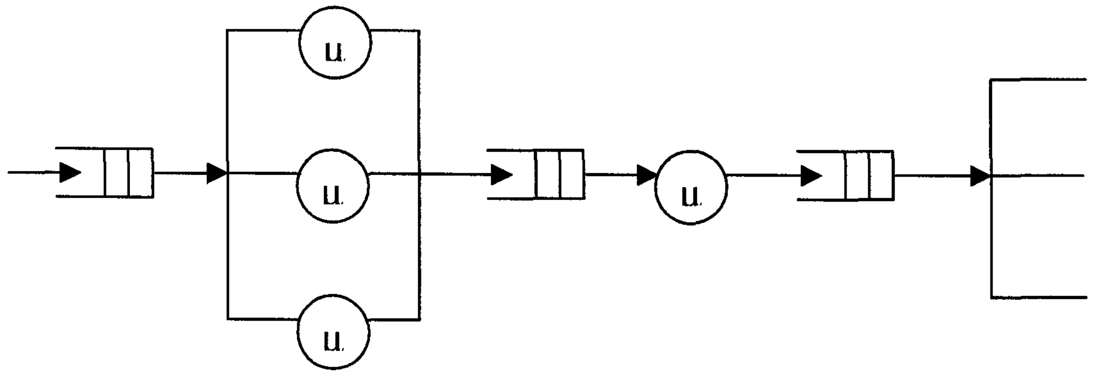
As discussed in chapter 3, the current available bandwidth measurement algorithms are facing some critical problems. Based on the single hop model, most algorithms are inaccurate when the algorithms are applied on the multi-link path. Most of them also have difficulties to quickly reflect the change of the available bandwidth along the path. We were inspired by the above findings and research results. The goal of the PoissonProb algorithm is to infer accurately the available bandwidth information based on a single Poisson distribution hop model in a faster way.

The algorithm design uses the following three properties of the Poisson process and queuing theory.

- Superposition and random split property: If a stream of Internet packets, called stream 1 arrive at the servers in the Poisson distribution with intensity rate λ_1 and if stream 2 of probing packets arrives in the Poisson distribution with intensity rate λ_2, and they are in the same probability space, then on merging the two Poisson streams, the resultant stream has a Poisson distribution with an intensity rate $\lambda_1+\lambda_2$. If a Poisson process with intensity λ is randomly split into two subprocesses with probabilities $p1$ and $p2$, where $p1+p2 = 1$, then the resulting processes are independent Poisson processes with intensities $p1\lambda$ and $p2\lambda$. This can also be generalized to a split into more than two subprocesses.
- PASTA (Poisson Arrivals See Time Averages) property. PASTA property is one of the central properties of queuing theory and the basis of our PoissonProb algorithm. Suppose packets arrive at the queue in Poisson process with intensity λ. These arrivals induce state transitions of the queue. If the queue length increases or the queue drains the packets immediately, then there are two different probabilities: The probability of the each state E_j as seen by an outside random observer, P1 is the probability that the queue is in the state E_j at a random instant. The probability of the state seen by an arriving probe packet P2 is the probability that the queue is in the state Ej just before a randomly chosen arrival. Here, the $P1 = P2$

- Consider the case of multiple servers in series and parallel queuing property. Assume the packets travel a complex network in which the path can be generalized, i.e. the series interconnection of single server queues may be generalized to a series interconnection of *m* phases, where the *i*th phase consists of r_i parallel channels. Assume that there are additional Poisson arrivals to each phase from outside of the network, and that there are feedbacks from various phases within the system as shown in Fig A-2. Then an interconnected feedback/feed-forward network with Poisson arrivals at various phases behaves like a cascade connection of independent queues with input rate λ_i with transmission rate μ_i at the *i*th phase.

Figure A-2 Multiple Servers in Series and Parallel Queuing



A. 2.1 Proofs of the Three Basic Properties

Most of the proofs are based on well-known probability theorems [2].

A.2.1.1 superposition property

The probability that an arrival occurs from process 1 in the interval *dt* is $\lambda_1 \bullet dt$. In Poisson process, this value is independent of the arrivals outside the interval. Similarly, the arrival probability from process 2 is $\lambda_2 \bullet dt$. Using superposition the probability of an arrival in the interval *dt* is $(\lambda_1 + \lambda_2) \bullet dt$ independent of arrivals outside the interval. The two streams, when combined together, yield a Poisson process with intensity $(\lambda_1 + \lambda_2)$.

A.2.1.2 Random Split Property

Using a direct intuition method of proof, the first step is the proof of random selection. If a random selection is made from a Poisson process with intensity λ such that each arrival is selected with probability *p*, independently of the others, the resulting process is a Poisson process with intensity $p\lambda$. The probability that an arrival occurs from the original process in the interval *dt* is $\lambda \bullet dt$ independent of the arrivals outside the interval. After the random selection the probability for an arrival in the interval *dt* is $p \bullet \lambda \bullet dt$ which is independent of the arrivals outside the interval. Then process of the selected arrivals is a Poisson process with intensity $p\lambda$. As both of the subprocesses resulting from the split represent a random selection of the original process, they are thus Poisson process with intensities $p_i\lambda$. So it remains to prove the independence of the processes. Let

- N_1 (I_1) = number of arrivals from subprocess 1 in the interval I_1
- N_2 (I_2) = number of arrivals from subprocess 2 in the interval I2

Denote $I = I_1 \cap I_2$

$$N_1(I_1) = N_1(I) + N_1(I_1 \cap \bar{I}_2) \qquad N_2(I_2) = N_2(I) + N_2(I_2 \cap \bar{I}_1)$$

Arrivals in non-overlapping intervals $I_1 \cap \bar{I}_2$ and $I_2 \cap \bar{I}_1$ are certainly independent. There may be dependence only between N_1 (I) and N_2 (I). But these represent the random split of the total number of arrivals from the original process, with Poisson distribution $(\lambda|I|)$, into two sets: The sizes of these sets were shown to be independent in considering the properties of the Poisson distribution.

The PoissonProb algorism is facilitated by the above properties. To measure the available bandwidth, PoissonProb sends Poisson distributed packets, in the expectation that if the probe packets and any session of cross-traffic enter or exit the path at any rate, the original traffic properties may not be influenced.

A.2.1.3 PASTA Property

To put it simply, the Poisson distributed packets can see the average cross-traffic along the path if the cross-traffic is also the Poisson distribution. To prove *P1=P2*, the method is as follows. The arrival histories before the instant of consideration, irrespective whether we are considering a random instant or an arrival instant, are stochastically the same: a sequence of arrivals with exponentially distributed interarrival times. This follows from the memoryless property of the exponential distribution. The remaining time to the next arrival has the same exponential distribution irrespective of the time that has already elapsed since the previous arrival, since the stochastic characterization of the arrival process before the instant of consideration is the same, irrespective of how the instant has been chosen. The state distributions of the system induced by the past arrivals processes at the instant of consideration must be the same in both the cases.

We can illustrate this further by the hitchhiker's paradox. The paradox is as follows:

- Cars are passing a point in a road according to the Poisson distribution.
- The mean interval between the cars is 10 minutes.
- A hitchhiker arrives at the roadside point at a random instant of time.
- What is the mean waiting time $\overline{w}$ until the next car?

The interarrival times in a Poisson process are exponentially distributed. From the memoryless property of the exponential distribution, it follows that the residual time to the next arrival has the same Exp (λ) distribution and the expected time is thus

$$\overline{w} = 10 \text{ min}$$

This conclusion appears paradoxical and most people may expect the $\overline{w} = 5$ min. However, the paradox lies in that the hitchhiker's probability to arrive during a long interarrival interval is greater than during a short interval. Given the interarrival interval, within that interval the arrival instant of the hitchhiker is uniformly distributed and the expected waiting time is one half of the total duration of the interval. The point is that in the selection by the random instant the long intervals are more frequently represented than the short ones (with a weight proportional to the length of the interval).

Consider a long period of time t. The waiting time to the next car arrival W (τ) as the function of the arrival instant of the hitchhiker τ can be represented by the sawtooth curve. The mean waiting time is the average value of the curve that is the sum of the triangles of the sawtooth. X_i is the interarrival time.

$$\overline{W} = \frac{1}{t}\int_0^t W(\tau)d\tau \approx \frac{1}{t}\sum_{t=1}^{n}\frac{1}{2}X_i^2$$

$$\overline{W} = \frac{1}{t}\int_0^t W(\tau)d\tau \approx \frac{1}{t}\sum_{t=1}^{n}\frac{1}{2}X_i^2$$

As $t \to \infty$ the number of the sawtooth triangles n tends to $t / \overline{X}$, then

$$\overline{W} = \frac{1}{\overline{X}}\frac{1}{n}\sum_{t=1}^{n}\frac{1}{2}X_i^2 = \frac{1}{2}\frac{\overline{X^2}}{\overline{X}}$$

For exponential distribution $\overline{X^2} = (\overline{X})^2 + V[X] = 2\overline{(X)^2}$, here, $V[X] = \overline{(X)^2}$ thus $\overline{W} = \overline{X}$.

As discussed in the earlier sections, both rate-based and gap-based algorithms are trying to snapshoot the cross-traffic at the tight link. PoissonProb algorithm is a rate-based algorithm; it sends probe packets in Poisson distribution and tries to saturate the available room at the tight link. The difference between the PoissonProb and the other rate-based algorithms is that most of the rate-based algorithm simply apply the packet train, packet pair or exponential distributed probe packets to detect the available bandwidth. These may not interweave with the Poisson-type cross-traffic as well as the Poisson distributed packets. So they may underestimate or overestimate the cross-traffic at the tight link. For the gap-based algorithms, which depend on the interval changes between the probe packets to infer the cross-traffic, failing to accurately estimate the cross-traffic may induce a large deviation of measurement.

To explain the third property is difficult, as the precise proof should start from *M/M/1* queue and Markovian Queues. Skipping the basic queuing theory, we start the proof from multiple servers in series and parallel directly. R.R.P. Jackson [2] has generalized the series interconnection of single server queues to a series interconnection of m phases, where the *i*th phase consists of r_i parallel channels, all with exponential service-rate μ_i. The input to the first phase is an unlimited Poisson input with parameter λ, and queuing is allowed before each phase. With n_i units in the *i*th phase, the probability that an item finishes service in Δt is given by $\mu_{ni}\, \Delta t + o(\Delta t)$, where $\mu_{ni} = n_i\mu_i$ ($n_i < r_i$) or $\mu_{ni} = r_i\mu_i$ ($n_i >= r_i$). In steady state, after substituting the steady state equations, it become,

$$(\lambda + \sum \mu_i) p(n_1, n_2, \ldots, n_m) = \sum_{i=1}^{m} \mu_{ni} p(n_1, n_2, \ldots, n_i + 1, n_{i+1} - 1, \ldots, n_m) + \lambda p(n_1 - 1, n_2, \ldots, n_m)$$

$n_i > 0$

Here, $p(n_1, n_2, \ldots, p_m)$ represents the probability that there are n_1 items in the first phase, n_2 items in the second phase, and so on. R.R. P. Jackson [2] has shown that the unique solution is give by the product form

$$p(n_1, n_2, \ldots, n_m) = p_1(n_1) p_2(n_2) \ldots p_i(n_i) \ldots p_m(n_m)$$

where

$$p_i(n_i) = \frac{(\lambda/\mu_i)^{ni}}{n_i!} p_{i,0} \qquad n_i < r_i \qquad \text{or}$$

$$p_i(n_i) = \frac{r_i^{ri} \rho_i^{ni}}{r_i!} p_{i,0} \qquad n_i >= r_i$$

Here

$$p_{i,0} = \frac{1}{\sum_{n=0}^{r_i - 1} \frac{(\lambda/\mu_i)^n}{n!} + \frac{(\lambda/\mu_i)^{ri}}{r_i!(1-\rho_i)}}$$

and $\rho_i = \lambda / r_i \mu_i$

The above equation represents an *M/M/r*$_i$ queue with n_i items, and from the product equation, it follows that in steady state a series-parallel network will behave like a cascade of independent *M/M/r*$_i$ queues, provided all servers in each parallel configuration have identical service rates.

Jackson has generalized this result by permitting additional Poisson arrivals to each phase from outside the system, and feedbacks from various phases within the system. Thus, a unit arrives at a phase with different probabilities. The services distributions are exponential, with the *i*th phase consisting of r_i parallel channels with identical service rate μ_i . Poisson arrivals from outside the system occur at the *i*th phase with rate γ_i , and after finishing service at *i*th phase, an item either leaves for the *j*th phase with probability q_{ij}, where it is served in the order of their arrival along with Poisson arrivals from outside, or it leaves the system with probability

$$q_{i,0} = 1 - \sum_{j=1}^{m} q_{ij}$$

Let λ_i represent the average arrival rate at the *i*th phase. Then λ_i satisfies

$$\lambda_i = \gamma_i + \sum_{K=1}^{m} q_{ki} \lambda_k \qquad i = 1,2,\ldots,m$$

Consider a network of *m* phases with *i*th phase consisting of r_i parallel servers, all with identical service rate μ_i. The network allows feedback and feed forward from phase *i* to *j* with probability q_{ij}, in addition to Poisson arrivals from outside to each phase at rate γ_i. Then the probability that there are n_i items in phase *i*, $i = 1,2,\ldots,m$ is given by

$$p(n_1, n_2, \ldots, n_m) = \prod_{i=1}^{m} p_i(n_i)$$

Also, the $\sum_{i=1}^{m} q_{i,0}\lambda_i = 1 - \sum_{i=1}^{m} \gamma_i$ and the total output from the system equals the total input into the system.

Thus, any complex network with external Poisson feeds behave like cascade connections of $M/M/r_i$ queues in steady state. Based on this property, PoissonProb algorithm can generalize the network as the single queue model and then the pre-bottleneck and post-bottleneck tight link problem which was discussed in the earlier section can be solved theoretically. The PoissonProb algorithm observes the probe packets gaps change at the receiver side. If the probe packets total destination gap falls within a small range around the total original gap, then PoissonProb algorithm believes the current rate of probe packets has saturated the available room at the tight link and they interfered with the cross-traffic at the tight link. Actually, the experimental results have shown that the PoissonProb algorithm has better performance as compared to other algorithm in the presence of the pre-bottleneck and post-bottleneck cross-traffic effects.

VITA AUCTORIS

Name:	Lu Xin
Place of Birth:	China
Year of Birth:	1972
Education:	QingDao Ocean University QingDao, ShanDong, China 1990 - 1994 B. Sc.
	University of Windsor Windsor, Ontario, Canada 2002 - 2005 M.Sc.