List of Publications

Journal Papers

1. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "ImTCP: TCP with an inline measurement mechanism for available bandwidth," *Computer Communications Special Issue: Monitoring and Measurements of IP Networks*, Vol. 29, No. 10, pp. 1614–1626, June 2006.

2. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "A simultaneous inline measurement mechanism for capacity and available bandwidth of end-to-end network path," *IEICE Transactions on Communications*, Vol. E89-B, No. 9, pp. 2469–2479, September 2006.

3. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "Inline bandwidth measurement techniques for gigabit networks," submitted to *International Journal of Internet Protocol Technology (IJIPT)*, September 2006.

Refereed Conference Papers

1. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "A new available bandwidth measurement technique for service overlay networks," in *Proceedings of Workshop on End-to End Monitoring Techniques and Services (E2EMON 2003)*, pp. 436–448, September 2003.

2. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "Available bandwidth measurement via TCP connection," in *Proceedings of the 2nd IEEE Workshop on End-to-End Monitoring Techniques and Services (E2EMON 2004)*, pp. 38–44, October 2004.

3. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "A merged inline measurement method for capacity and available bandwidth," in *Proceedings of the 6th Passive and Active Measurement Workshop (PAM 2005)*, pp. 341–344, March 2005.

4. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "An inline measurement method for capacity of end-to-end network path," in *Proceedings of the 3rd IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (E2EMON 2005)*, pp. 56–70, May 2005.

5. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "ImTCP HighSpeed: Inline network measurement for high-speed networks," in *Proceedings of the 7th Passive and Active Measurement Workshop (PAM 2006)*, March 2006.

6. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "ICIM: An inline network measurement mechanism for highspeed networks," in *Proceedings of the 4th IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (E2EMON 2006)*, pp. 67–74, April 2006.

Non-Refereed Technical Papers

1. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "A study on inline network measurement mechanism for service overlay networks," *IEICE Technical Report IN2003-176 (in Japanese)*, Vol.102, No.561, pp. 53–58, January 2003.

2. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "Inline network measurement with active TCP connections," *2003 IEICE General Conference (in Japanese)*, SB-4-6, March 2003.

3. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "ImTCP: TCP with an inline network measurement mechanism," *IEICE Technical Report IN2004-7 (in Japanese)*, Vol. 104, No. 73, pp. 37–42, May 2004.

4. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "Inline network measurement: TCP with a built-in measurement technique," *2004 IEICE Communications Society Conference*, BS-10-1, September 2004.

5. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "Inline measurement method for capacity bandwidth of end-to-end network path," *IEICE Technical Report IN2005-17 (in Japanese)*, Vol. 105, No. 113, pp. 5–10, June 2005.

6. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "An inline network measurement mechanism for high-speed networks," *IEICE Technical Report IN2005-123*, Vol. 105, No. 472, pp. 79–84, December 2005.

7. Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata, "A packet burst-based inline network measurement mechanism," *2006 IEICE General Conference*, BS-5-4, March 2006.

Preface

End-to-end bandwidth is a metric that demonstrates how fast data is transmitted over a path connecting two separated end hosts of a network. The bandwidth information is particularly important in adaptive control and optimal resource provision of many emerging network technologies: content delivery networks, storage area networks, peer-to-peer networks, overlay networks, multicasting, and grid networks. End-to-end bandwidth is also deployed in network management such as bandwidth prediction, network topology design, enforcement of service level agreement, and detecting distributed intrusion/attacks or isolating fault locations. In addition, end-to-end bandwidth is important for developing adaptive control for network protocols such as routing, congestion/rate control and adaptive multimedia error concealment.

Because of the importance of bandwidth information, methodologies for monitoring bandwidth have attracted a great deal of attention. Network operators normally collect traffic information from routers to infer the bandwidth. For the majority of users, who do not have access authority to the inside of networks, monitoring bandwidth at the end hosts is the only alternative. Bandwidth measurement tools at the end hosts can be divided into two groups: active and passive. Passive measurement tools can collect traffic information at some end hosts for measurements without creating or modifying any traffic on the network, but this approach requires a relatively long time for data collection and bandwidth estimation. Active tools actively exchange probe traffic between two end hosts to perform the bandwidth measurement, and appear more accurate and sensitive to changes in bandwidth. However, their common weakness is causing an additional load on the network due to probe traffic.

This thesis presents a novel approach to end-to-end bandwidth measurement. We propose an active measurement method that does not add probe traffic to the network: *inline measurement*, with the concept of plugging a measurement mechanism into an active Transmission Control Protocol (TCP) connection. TCP is currently the most popular

transport protocol of the Internet. During data transmission, the TCP sender continuously sends data packets to the TCP receiver, which replies to the data packets with ACK (acknowledgement) packets. Using this mechanism, the proposed measurement approach periodically adjusts the transmission interval of some data packets, and we then examine the arrival-intervals of the corresponding ACK packets to infer the end-to-end bandwidth of the network path connecting TCP sender and TCP receiver. Thus, data transmitted in TCP connection is utilized for active measurement, rather than injecting probe traffic into the network.

We first introduce a new version of TCP, called Inline measurement TCP (ImTCP). ImTCP can transmit data as any other previous TCP version, and can also measure the unused bandwidth (called *available bandwidth*) of the path that the TCP connection is passing through. We introduce a measurement algorithm suitable for inline network measurement and that generates periodic measurement results at short intervals. The key idea in rapid measurement is to limit the bandwidth measurement range using statistical information from previous measurement results. When the ImTCP sender transmits data packets, it first stores a group of up to several packets in a queue and subsequently forwards them at the transmission rate determined by the measurement algorithm. Then, considering the arrival intervals of the ACK packets, the ImTCP sender performs the bandwidth calculation. The simulation experiments show that ImTCP can yield measurement results within 20% of the actual available bandwidth every few RTTs, without degrading transmission throughput. We also present two examples in which ImTCP uses measured bandwidth information to optimize link utilization or improve the transmission performance of TCP itself.

We then consider inline measurement of the maximum bandwidth (called the *physical bandwidth* or *capacity*) of an end-to-end network path. We develop a new capacity measurement function and combine it with ImTCP in order to enable simultaneous measurement of both capacity and available bandwidth in ImTCP. The capacity measurement algorithm is a new packet-pair-based measurement technique that utilizes the estimated available bandwidth values for capacity calculation. This new algorithm promises faster measurement than current packet-pair-based measurement algorithms for various situations and works well for high-load networks, in which current algorithms do not work properly. Simulation results indicate that the capacity measurement algorithm can deliver results with small errors when the network load is as high as 93% of capacity. Moreover, the new algorithm provides a confidence interval for the measurement results.

We also perform measurement tasks for 1 Gbps or higher networks. In such high-speed networks, current bandwidth measurement algorithms that utilize packet transmission/arrival intervals are faced with two main problems. First, network measurement for large bandwidth requires short packet transmission intervals, which causes a heavy load on the CPU. Second, network interface cards for high-speed networks usually employ Interrupt Coalescence, which rearranges the inter-arrival times of packets and causes bursty transmission of packets. We introduce a new inline measurement method that overcomes these two problems. Measurement algorithms for both the available bandwidth and the capacity are proposed. Rather than adjusting packet transmission intervals, we adjust the number of packets involved in a packet burst and utilize the inter-intervals of the bursts of the corresponding ACK packets for bandwidth measurement. Simulation results show that the proposed method can measure bandwidth in the network path of at least 1 Gbps or faster. When measuring a 5-Gbps network path, 94% of the available bandwidth measurement results delivered by the proposed method have relative errors smaller than 20%. The measurement frequency is also approximately 60 times higher than that of an existing high speed network measurement tool. We also show an implementation result in a laboratory network environment to validate the proposed method.

The increasing demand for end-to-end bandwidth monitoring has led to extensive development and deployment of several measurement tools. TCP with built-in measurement technique is an effective way of overcoming the limitations of the two current measurement approaches. Moreover, by using the mechanism of TCP, inline measurement can work even better than stand-alone tools in some network scenarios, such as heavily loaded networks and high-speed networks. Until recently, TCP traffic has accounted for a large proportion of current Internet traffic. Therefore, we expect that we can monitor end-to-end bandwidth with the proposed approach at several locations, in real-time, with high accuracy, while having no effect on the network.

The future holds several challenging tasks. First, that is the improvement of TCP performance using the bandwidth information inferred by inline network measurements. Second, as streaming protocols, such as the Real Time Streaming Protocol, are emerging, inline measurement techniques for these protocols are also of interest.

Acknowledgements

This thesis would be impossible without people who supported me and believed in me. Thanks to all of the wonderful people.

First and foremost, I would like to express my sincere appreciation to my research advisor, Prof. Masayuki Murata of Osaka University. His guidance and inspiration have provided an invaluable experience that has helped me to fulfill this research.

I am also thankful to my thesis committee members, Prof. Koso Murakami, Prof. Makoto Imase, Prof. Teruo Higashino and Prof. Hirotaka Nakano for their careful reading and constructive comments in completing this thesis.

I would like to express my deepest gratitude to Associate Prof. Go Hasegawa of Osaka University. He has been a constant source of encouragement and advice throughout my studies and in the preparation of this manuscript.

I would like to thank Associate Prof.s Naoki Wakamiya, Kenji Leibnitz and Research associates Shinichi Arakawa, Masahiro Sasabe of Osaka University who gave me helpful comments and feedback.

I want to thank secretaries, my many friends and colleagues in the Graduate School of Information Science and Technology of Osaka University for assisting me in many different ways.

Finally, I am forever indebted to my father Cao Xuan Thanh. His strong will power has led me to the goal. I am obliged to my mother, my two sisters for their understanding, endless patience and encouragement. Especially, I would like to give my special thanks to my wife Thien Trang whose patient love enabled me to complete this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Significance of bandwidth measurement

The Internet is a large network of interconnected computer networks that transmit data by packet switching using the standard Internet Protocol (IP). The Internet has become an indispensable part of daily life for many people, and an increasing number of services have been made available via the Internet. IP networks in the Internet make best-effort attempts to deliver data to their destination. However, no special guarantees are given for the transmission quality. Meanwhile, how fast the data is carried over the separated locations of the network greatly affects the performance of networked services. Therefore, monitoring IP network transmission quality and behaving adaptively has become important in many network technologies.

The end-to-end bandwidth of a network path is an important metric (together with other metrics such as latency and packet loss ratio) that expresses directly the data transmission quality of that path. There are two major bandwidth-related definitions: *capacity* and *available bandwidth*. The capacity is the maximum speed at which the data can be transmitted on the path when there is no competing traffic load (cross traffic). The capacity of a path is determined by the link with the minimum capacity (bottleneck link). The available bandwidth, on the other hand, indicates the maximum speed at which the data can be transmitted at the present time, given the current cross traffic load of the path. The available bandwidth of a path is determined by the link with the minimum unused bandwidth (tight link). Figure 1.1 illustrates the difference between capacity and available bandwidth.

Bandwidth information is important in adaptive control and optimal resource provision

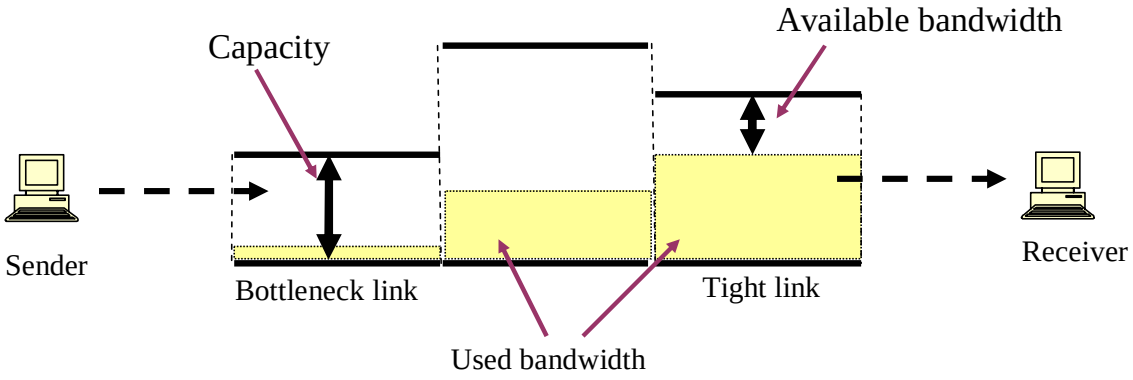


Figure 1.1: Difference between capacity and available bandwidth

of several emerging network technologies: content delivery networks (CDNs) [1], storage area networks (SANs) [2], peer-to-peer (P2P) networks [3, 4], multicasting [5], grid networks [6, 7], and multihoming [8]. For example, in a CDN service such as Akamai [9] or Exodus [10], measurement techniques can be used to estimate the available bandwidth and regulate the sending rate of transmissions for Web prefetching [11] in order to avoid degrading the performance of other traffic. Bandwidth information can also be used to realize adaptive control mechanisms in various service overlay networks, including the following examples. In P2P networks, when the resource discovery mechanism finds multiple peers having the requested contents, the measurement results help to determine from which peer the contents are transmitted. In data grid networks, when multiple sites have the same data, the measurement results help to determine from which site data will be copied or read.

End-to-end bandwidth is also deployed in network management such as bandwidth prediction, network topology design, enforcement of service level agreement, detecting distributed intrusion/attacks or isolating fault locations [12, 13]. End-to-end bandwidth is also important for developing adaptive control for network protocols such as routing, congestion/rate control [14–17], and adaptive multimedia error concealment [18]. In many cases, both capacity and available bandwidth information are required at the same time. For example, network transport protocols should optimize link utilization according to available bandwidth. However, if a connection tends to use all of the available bandwidth, other connections that join the network later will find it difficult to obtain bandwidth to fairly share the bandwidth. In this case, if the connections are aware of the capacity of the network path, they can quickly change the used bandwidth so that the fairness with

newly attended connections is maintained. One method of using capacity and available bandwidth information to optimize both bandwidth utilization and connection fairness for TCP is proposed in [14]. Another example is streaming multimedia data. Capacity information can be used for the decision of the size and the quality of the multimedia data. Because available bandwidth is normally highly variable, it is better to use capacity information in this case. Available bandwidth information is then used to improve the performance of the transmission of the data. In addition, the billing policy of the Internet service provider may be based on both the capacity and the available bandwidth of the access link provided to the customer.

1.2 Limitations in existing bandwidth measurement tools

Recent years have witnessed extensive development of bandwidth measurement tools. Network operators commonly use tools such as NetFlow [19] and MRTG [20, 21] to collect traffic information from routers and infer the bandwidth information from this information. Although this approach normally delivers accurate results, it requires router access authorities. As such, its usage is limited to network administrators. Instead, monitoring end-to-end bandwidth at the end hosts without any support from the network appears to be a more realistic approach and has recently attracted a great deal of attention.

Bandwidth measurement tools at the end hosts can be divided into two groups: *active* and *passive* approaches. Passive measurement refers to the process of measuring a network without creating or modifying any traffic on the network. Representative methods include Nettimer [22] and PPrate [23]. Passive measurement is implemented by incorporating some additional devices into the end hosts to enable them to identify and record the characteristics and quantity of the packets as they pass. Packet statistics can then be collected for bandwidth estimation. The limitation of passive approaches is that they require a relatively long time for data collection before starting bandwidth estimation.

Active measurement tools, on the other hand, actively exchange probe traffic between two end hosts to perform the bandwidth measurement. Table 1.1 shows some active measurement tools that measure end-to-end capacity or available bandwidth. Measurement tools for per-link capacity, such as Pathchar [24] and Pchar [25] are also included because they can also infer the end-to-end capacity, due to the fact that the minimum capacity of

Table 1.1: Active measurement tools for end-to-end bandwidth

Tool	Author	Metric	Probe model	Methodology
Pathchar	Jacobson	Per-link capacity	Variable-size packets	VPS
Clink	Downey	Per-link capacity	Variable-size packets	VPS
Pchar	Mah	Per-link capacity	Variable-size packets	VPS
Bprobe	Carter	Capacity	Packet pairs	Packet pair dispersion
Pathrate	Dovrolis	Capacity	Packet pairs	Packet pair dispersion
CapProbe	Kapoor	Capacity	Packet pairs & Packet trains	Packet pair dispersion
Cprobe	Carter	A-bw	Packet trains	Packet train dispersion
TOPP	Melander	A-bw& Capacity	Packet pairs	Self-induced congestion
Delphi	Ribeiro	A-bw	Packet streams	Packet pair dispersion
Pathload	Jain	A-bw	Packet streams	Self-induced congestion
Spruce	Strauss	A-bw	Packet pairs	Self-induced congestion
IGI	Hu	A-bw	Packet streams	Packet pair dispersion
PTR	Hu	A-bw	Packet streams	Self-induced congestion
ABwE	Navratil	A-bw	Packet pairs	Packet pair dispersion
PathChirp	Ribeiro	A-bw	Packet streams	Self-induced congestion

all link is the end-to-end capacity.

The method used for measuring the per-link capacity is called Variable Packet Size (VPS). This method probes the network by packets in various sizes. VPS is based on round trip time (RTT) measurements as well as on the Internet Control Message Protocol, ICMP. VPS adjusts the Time-to-Live (TTL) to force probing packets to expire at a particular router. The router at that hop discards the probing packets, returning ICMP messages back to the sender. By sending packets in various sizes, the sender can estimate the minimum RTT to the router. From the minimum RTT values to all of the routers, the propagation of packets on each link is calculated. The capacity of the link is then calculated from the packet size and the propagation delay of the packet on the link.

End-to-end capacity measurement tools utilize two packets sent back-to-back (packet pair) for network probing. The intuitive rationale of capacity measurement using packet pairs is that if two packets are sent close enough together in time to cause the packets to queue back-to-back at the bottleneck link, then the packets will arrive at the destination

with the same spacing as when they left the bottleneck link. Thus, these tools send numerous packet pairs and observe the packet pair dispersion. They filter the arrival intervals of the pairs to discover what interval is created by the bottleneck link. The capacity is then calculated from this interval and the packet size.

For available bandwidth measurement, both packet pair and packet stream can be used for probing the network. Packet stream refers to a group of packets transmitted at a determined rate. In addition to Bprobe [26], these measurement tools have two main measurement approaches. The first approach exploits the information in the time gap between the arrivals of two successive probe packets, similar to the capacity measurement tools. Here, the two successive packets can be either a packet pair or two successive packets in a stream. Packet dispersion is used for inferring the cross traffic at the tight link, from which the available bandwidth is calculated. The second approach is based on the concept of self-induced congestion. That is, if the transmission rate of the probe traffic is lower than the available bandwidth along the path, then the probe traffic will pass the network with the rate unchanged. In contrast, if the transmission rate is higher than the available bandwidth of the network path, the probe traffic will be delayed and will arrive at the receiver at a lower rate. Thus, by searching for the change point in the arrival rate of probe traffic, the available bandwidth of the network path can be inferred. Bprobe sends trains of packets at a high rate and considers the dispersion rate of the trains as the available bandwidth.

Although active measurement tools seem accurate and sensitive to changes in bandwidth, sending extra traffic into the network is their common drawback. The amount of required probe traffic differs depending on the algorithm. According to one study [27], Pathload [28] generated between 2.5 to 10 MB of probe traffic per measurement. Newer tools have succeeded in reducing the amount of probe traffic per measurement. The average per-measurement probe traffic generated by IGI [29] is 130 KB, and that generated by Spruce [27] is 300 KB. A few KB of probe traffic for a single measurement is a negligible load on the network. However, for routing in overlay networks, or adaptive control in transmission protocols, these measurements may be repeated continuously and simultaneously from numerous end hosts. In such cases, a few KB of per-measurement probes will create a large amount of traffic that may damage other data transmission in the network as well as degrade the measurement accuracy itself.

1.3 Inline bandwidth measurement and its related studies

The present thesis proposes a novel approach for end-to-end bandwidth measurement. We propose an active measurement method that does not add probe traffic to the network: *inline network measurement*, in which the packets transmitted in a data flow are used for the purpose of measurement. Inline network measurement has the advantages of both active and passive measurement approaches. That is, the accuracy of the measurement is as good as that of active measurement, other traffic in the network is not affected. However, inline measurement algorithms must be adaptive to the environment of the data flow. Otherwise, the performance of the data transmission will be degraded.

We propose measurement algorithms that work in an active Transmission Control Protocol (TCP) connection. TCP is currently the most popular transport protocol of the Internet. During data transmission, the TCP sender continuously sends data packets to the TCP receiver, which replies to the data packets with ACK (acknowledgement) packets. Using this mechanism, the proposed measurement approach periodically adjusts the transmission interval of a number of data packets. The proposed approach then examines the arrival-intervals of the corresponding ACK packets to infer the end-to-end bandwidth of the network path connecting the TCP sender and the TCP receiver. Thus, data transmitted in a TCP connection is used for active measurement, rather than injecting probe traffic into the network. Figure 1.2 illustrates the key concept of inline network measurement.

The concept of inline measurement has previously appeared in traditional TCP. To a certain extent, traditional TCP can be considered to be a tool for measuring available bandwidth because of its ability to adjust the congestion window size to achieve a transmission rate that is appropriate for the available bandwidth. One version of TCP, TCP Vegas [30], also measures the packet transmission delay. There are, in addition, other tools that convert the TCP data transmission stack into network measurement tools. These tools include Sting [31] (measuring packet loss), Sprobe [32] (measuring capacity), and abget [33] (measuring available bandwidth). Note that these tools can only be used for the purpose of measurement and can no longer transmit data as a transport protocol.

A number of studies have considered bandwidth measurement in an active TCP connection. Bandwidth estimation in traditional TCP (Reno TCP) is insufficient and inaccurate

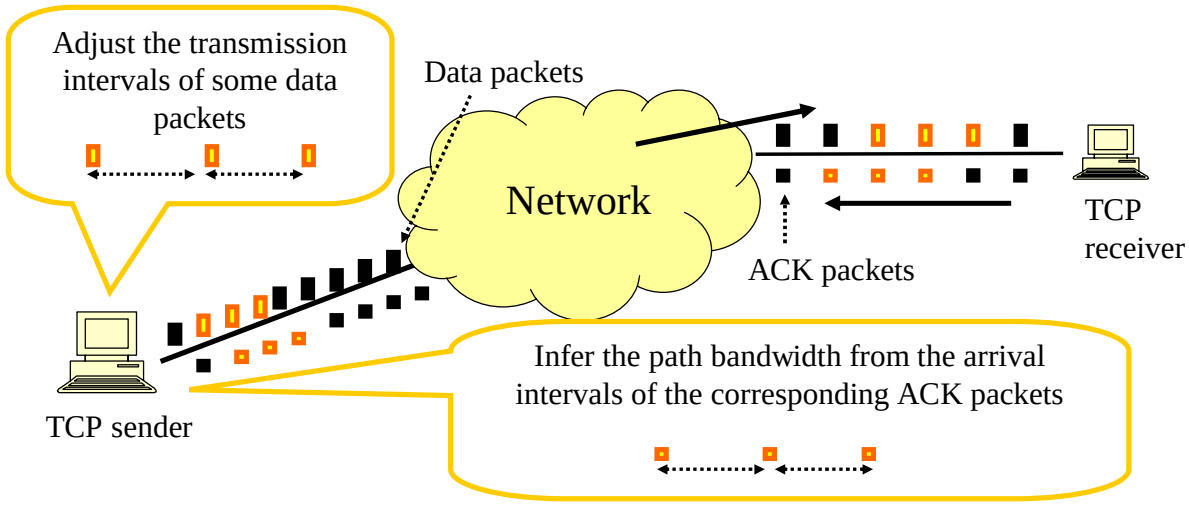


Figure 1.2: Key idea of inline network measurement

because it is a measure of used bandwidth, not available bandwidth. In particular, in networks in which the packet loss probability is relatively high, TCP tends to fail at estimating available bandwidth. Moreover, the TCP sender window size often does not accurately represent the available bandwidth due to the nature of the TCP congestion control mechanism. The first TCP measurement algorithm to improve accuracy used a passive method, in which the sender checks ACK arrival intervals to infer the available bandwidth [34]. This method is a simple approach based on the Cprobe [26] algorithm. A similar technique is used in TCP Westwood [16], in which the sender also passively observes ACK packet arrival intervals to estimate bandwidth, but the results are more accurate due to a robust calculation. Another study proposed TCP-Rab [35], which is a TCP with an inline measurement method based on the receiver. The receiver calculates the available bandwidth from the arrival rate of TCP segments and informs the sender, so that the sender can perform a measurement-based congestion window control mechanism. The approach estimates the bandwidth better than Westwood, because it can eliminate noise caused by the fluctuation of transmission delay of ACK packets. However, because these methods are all passive measurements, changes in available bandwidth cannot be detected quickly. In particular, when the available bandwidth increases suddenly, the TCP data transmission rate cannot adjust as rapidly and requires time to ramp up because of the self-clocking behavior of TCP. Meanwhile, as transmission proceeds at a rate lower than the available bandwidth, the measurement algorithm yields results lower than the true value. TCP Probe [36] employs the CapProbe [37] algorithm to perform inline measurement which can only measure

capacity. The proposed algorithm uses an active approach for inline measurement, for both available bandwidth and capacity. That is, the sender TCP not only observes ACK packet arrival intervals, but also actively adjusts the transmission interval of data packets. The sender thus collects more information for a measurement, and improved accuracy can be expected. Moreover, the proposed mechanism requires a modification of the TCP sender only, incurring the same deployment cost as the approaches in described [16, 34–36].

1.4 Organization of the present thesis

The present thesis consists of three inline measurement algorithms. Each of these algorithms is described together with the experiment results for evaluations and comparison with related methods.

In Chapter 2, we introduce a new version of TCP, called Inline measurement TCP (ImTCP). Like previous TCP versions, ImTCP can transmit data; however, ImTCP can also measure the available bandwidth of the path followed by TCP packets. We first introduce a measurement algorithm suitable for inline network measurement that generates periodic measurement results at short intervals, on the order of several RTTs. The key idea in rapid measurement to limit the bandwidth measurement range using statistical information from previous measurement results, unlike existing algorithms, which search from 0 bps to the upper limit of the physical bandwidth with every measurement [28, 38]. By limiting the measurement range, we can avoid adjusting TCP packets to an extremely high rate and keep the number of packets required for measurement small.

We then build the algorithm into the most popular TCP version, Reno TCP, to create ImTCP. When a sender transmits data packets, ImTCP first stores a group of up to several packets in a queue and then subsequently forwards them at a transmission rate determined by the measurement algorithm. Each group of packets corresponds to a probe stream. Then, considering ACK packets as echoed packets, the ImTCP sender estimates available bandwidth according to the algorithm. To minimize the transmission delay caused by the packet store-and-forward process, we introduce an algorithm using the round trip timeout (RTO) calculation in TCP to regulate the packet storage time in the queue. We evaluate the inline measurement system using simulation experiments. The results show that ImTCP can yield measurement results within 20% of the actual available bandwidth every few RTTs without degrading transmission throughput. We also present two examples in which ImTCP uses bandwidth information to optimize link utilization or improve the

transmission performance of TCP itself.

In Chapter 3, we focus on inline measurement of the capacity of the end-to-end network path. We develop a new capacity measurement function and combine it with ImTCP in order to enable simultaneous measurement of both capacity and available bandwidth in ImTCP. The proposed algorithm utilizes the arrival intervals of the ACK packets of packet pairs (PPs) that are sent back-to-back. Due to the characteristic of ImTCP whereby PPs are available after the transmission of each measurement stream, the capacity measurements do not require any further changes in ImTCP. Using the proposed method, ImTCP measures the capacity at the early stage of the connection and continues to collect data to improve the measurement accuracy during the transmission. We do not intend to develop a new capacity measurement tool that is more accurate than the existing tools [22, 37, 39]. Rather, in an attempt to reduce the load over the network caused by probe traffic, our primary focus is how to extract capacity information from a TCP connection with the smallest change in TCP. The main concept of the proposed capacity measurement algorithm in ImTCP is that the available bandwidth information, which can be yielded periodically due to the deployed available bandwidth measurement mechanism, is exploited. In the existing PP-based capacity measurement algorithms, the PPs that are cut into by other packets from cross traffic at the bottleneck link cause incorrect capacity estimation and are therefore eliminated from the data used in the calculation. However, in the proposed method, the available bandwidth information is used for estimation of the quantity of the cross traffic that cuts into PPs at the bottleneck link. The interval of the PPs becomes usable for the capacity measurement, which enables ImTCP to collect more information from PPs so that faster and more accurate measurements can be expected. The proposed algorithm also uses statistical analysis to calculate the confidence interval of the delivered results. Through simulation validations, we show that ImTCP can deliver capacity measurement results quickly, independent of the characteristics of the network. In addition, we find that the capacity measurement algorithm can deliver accurate results even when the network load is as high as 93% of the capacity, while current measurement algorithms do not work well.

Chapter 4 describes the challenges involved in bandwidth measurement for high-speed (1 Gbps or higher) networks. For most active bandwidth measurement tools, bandwidths of 1 Gbps or higher are difficult to measure. There are two reasons for this. First, measurement in fast networks requires short transmission intervals of the probe packets (for example, 12 μs for a 1-Gbps link and a 1,500-byte packet). However, regulating such short

intervals causes a heavy CPU load. Second, network cards for high-speed networks usually employ Interrupt Coalescence (IC)) [40, 41], which rearranges the arrival intervals of packets and causes bursty transmission, and, therefore, algorithms utilizing packet arrival intervals do not work properly. We then introduce a new inline measurement mechanism that works well in high-speed networks. Measurement algorithms for both the available bandwidth and capacity are proposed. We refer to these algorithms collectively as Interrupt Coalescence-aware Inline Measurement (ICIM). Unlike other active measurement tools, which observe the inter-intervals of the packets, ICIM adjusts the number of packets that are transmitted in a burst caused by IC and estimates the capacity and available bandwidth by checking whether the inter-intervals of the bursts of corresponding ACK packets are increased as they pass through the network. ICIM does not set the sending interval of the packets, so the overhead for packet spacing at the sender is eliminated. The simulation results show that ICIM-abw performs accurate measurements in networks of 1 Gbps and faster. More than 94% of the results delivered by ICIM-abw have relative errors smaller than 20% for measurement on a 5-Gbps network path. Moreover, TCP with ICIM can transmit data with the same performance as Reno TCP.

In the final chapter of this thesis, we present a summary of the three proposed inline measurement approaches. We also discuss future challenges concerning the present research.

Chapter 2

ImTCP: TCP with an Inline Measurement Mechanism for Available Bandwidth

2.1 Introduction

Information concerning bandwidth availability in a network path plays an important role in adaptive control of the network. Many researches on measuring available bandwidth have been done so far. Available bandwidth can be measured at routers within a network [19–21]. This approach may require a considerable change to network hardware and is suitable for network administrators only. Some *passive* measurement tools can collect traffic information at some end hosts for performance measurements [22, 23], but this approach requires a relatively long time for data collection and bandwidth estimation. Exchanging probe traffic between two end hosts to find the available bandwidth along a path (an *active* measurement) seems the more realistic approach and has attracted much recent research. They are Cprobe [26], TOPP [42], Pathload [28], IGI/PTR [29], CapProbe [37] and many others [27, 33, 38, 39, 43, 44]. However, sending extra traffic into the network is the common weakness in all active available bandwidth measurement tools. In some cases, probe traffic may damage other data transmission in the network as well as degrade the measurement itself.

In this thesis, we propose an active measurement method that does not add probe traffic to the network, named *inline network measurement*. This is the idea of *plugging* a measurement mechanism into an active TCP connection; data packets and ACK packets

of an TCP connection are utilized for the measurement, instead of probe packets. This method has the advantage of requiring no extra traffic to be sent to the network.

As for the measurement of available bandwidth in an active TCP connection, there is some related research. The first TCP measurement algorithm to improve accuracy used a passive method in which the sender checks ACK arrival intervals to infer available bandwidth proposed by Hoe [34]. It is a simple approach based on the Cprobe [26] algorithm. A similar technique is used in TCP Westwood [16] where the sender also passively observes ACK packet arrival intervals to estimate bandwidth, but the results are more accurate due to a robust calculation. Another study in [35] proposes TCP-Rab, a TCP with an inline measurement method that based on the receiver. The receiver calculates the available bandwidth from the arrival rate of TCP segments and informs the sender, so that the sender can perform a measurement-based congestion window control mechanism. The approach estimates the bandwidth better than Westwood, because it can eliminate noise caused by the fluctuation of ACK packets' transmission delay. However, because these methods are all passive measurements, changes in available bandwidth cannot be detected quickly. Especially when the available bandwidth increases suddenly, the TCP data transmission rate cannot adjust as rapidly and needs time to ramp up because of the self-clocking behavior of TCP. Meanwhile, as transmission proceeds at a rate lower than the available bandwidth, the measurement algorithm yields results lower than the true value.

In this chapter, we first introduce an active measurement algorithm suitable for inline network measurement that generates periodic measurement results at short intervals. The key idea in measuring rapidly is to limit the bandwidth measurement range using statistical information from previous measurement results. This is done rather than searching from 0 bps to the upper limit of the physical bandwidth with every measurement as existing algorithms do [28, 38]. By limiting the measurement range, we can avoid sending probe packets at an extremely high rate and keep the number of probe packets small.

We then introduce ImTCP (Inline measurement TCP), a Reno-based TCP that includes the proposed algorithm for inline network measurement described above. When a sender transmits data packets, ImTCP first stores a group up to several packets in a queue and subsequently forwards them at a transmission rate determined by the measurement algorithm. Each group of packets corresponds to a probe stream. Then, considering ACK packets as echoed packets, the ImTCP sender estimates available bandwidth according to the algorithm. To minimize transmission delay caused by the packet store-and-forward process, we introduce an algorithm using the RTO (round trip timeout) calculation in

TCP to regulate packet storage time in the queue. We evaluate the inline measurement system using simulation experiments. The results show that ImTCP can yield measurement results within 20.8% of the actual available bandwidth in every some RTTs without degrading transmission throughput.

Measurement results of ImTCP can be passed to higher network layer and used for optimal route selection [45] in service overlay networks, in network topology design or in isolating fault locations [13]. Besides, ImTCP can use such bandwidth information to optimize link utilization or improve transmission performance of TCP itself. We present two examples of the second usage. In *background mode*, ImTCP uses the results of bandwidth availability measurements to prevent its own traffic from degrading the throughput of other traffic. This allows a prioritization of other traffic sharing the network bandwidth. In *full-speed mode*, ImTCP uses measurement results to keep its transmission rate close to the measured value necessary for optimum utilization of the available network bandwidth. This mode is expected to be used in wireless and high-speed networks where traditional TCP cannot use the available bandwidth effectively.

The remainder of this chapter is organized as follows. In Section 2.2, we introduce our proposed algorithm for inline available bandwidth measurement. In Section 2.3 we introduce ImTCP and evaluate its performance in Section 2.4. In Section 2.5, we introduce two examples of congestion window control mechanisms for ImTCP. Finally, in Section 2.6, we present concluding remarks.

2.2 Algorithm for inline available bandwidth measurement

2.2.1 Requirements

We consider the following factors to be the requirements for the algorithm of inline available bandwidth measurement:

- Small number of packets used

  Because our method uses TCP packets for the measurement, there is a limitation on the number of packets available for transmission at any one time because of the TCP

window size. Since the TCP window size is relatively small and changes dynamically, the measurement algorithm should use as small a number of packets as possible.

- Little effect on other traffic on the network

  The measurement should not affect either the cross traffic the external TCP traffic. The measurement may adversely affect the network in two ways: by sending numerous probe packets and by sending probe packets at a high rate.

- Providing results continuously

  Since the characteristics of the IP network changes constantly and dynamically, measurement should provide periodic estimation results. Furthermore, the interval should be as small as possible in order to provide an accurate depiction of the rapid network change.

- Providing results quickly

  The measurement should be performed quickly in order to obtain up-to-date information of the IP network. In the proposed method, we therefore assign a higher priority to measurement speed than to measurement accuracy.

As mentioned in Chapter 1, the existing measurement methods can be divided into two groups: passive measurement methods and active measurement methods. The active measurement methods inject probe packets into the network and collect the feedback information from monitored results including transmission delay, packet arrival-interval time, packet loss ratio, and so on. Therefore, we can expect a higher accuracy of measurement results in an end-to-end fashion than what is possible by passive methods.

However, considering the requirements mentioned above, existing active measurement algorithms for available bandwidth have fundamental disadvantages. One is that many probe packets are sent at a high transmission rate. For instance, TOPP [42] sends 5000 packets to obtain only one measurement, and Cprobe [26] injects 100–200 probe packets at the physical bandwidth speed of the link connected to the sender host. PathLoad sends several 100-packet measurement streams for a measurement. PathChirp [38] is a modification of PathLoad on the purpose of decreasing the number of probe packets. But the required number of packets to be sent at one time in PathChirp is still large. The probe traffic can affect other traffic along the path, for example by degrading traffic throughput and increasing the packet loss ratio and packet transmission delay. Existing active measurement

algorithms also require a long time to obtain one measurement result (for example, 50–100 RTTs are necessary to obtain one estimation value in TOPP and Pathload). Long-term measurement can provide an accurate result but cannot follow the dynamic changes on the IP network.

Thus, the existing active measurement algorithms do not satisfy the requirements mentioned. In the next subsection, we introduce a measurement algorithm which satisfies the requirements. Note that we do not attempt to replace the existing active measurement approaches by the proposed measurement algorithm. Rather, the proposed measurement algorithm is useful in the inline network measurement.

2.2.2 Proposed measurement algorithm

The proposed measurement algorithm is based on the concept of self-induced congestion, which is introduced in [28]. It utilizes packet streams sent in various transmission rates for network probing. It infers the available bandwidth of the network path by searching for the change point in the arrival rate of the probe packet streams. The deployment of MPLS may impact the measurement algorithm performance because MPLS can hide IP-level routes.

The algorithm requires a sender host transmit measurement packets to a receiver host, which immediately sends received packets back to the sender host. The sender host adjusts the transmission intervals of packets to form packet streams, that are group packets sent at one time, for the available bandwidth measurements. The measurements are performed repeatedly.

In every measurement, a *search range* is introduced for searching the value of the available bandwidth. Search range I= (B_l, B_u) is a range of bandwidth which is expected to include the current value of the available bandwidth. The proposed measurement algorithm searches for the available bandwidth only within the given search range. The minimum value of B_l, the lower bound of the search range, is 0, and the maximum value of B_u, the upper bound, is equal to the physical bandwidth of the link directly connected to the sender host. By introducing the search range, sending probe packets at an extremely high rate, which seriously affects other traffic, can be avoided. The number of probe packets for the measurement can also be kept quite small. As discussed later herein, even when the value of the available bandwidth does not exist within the search range, the correct value can be found in a few measurements. The following are the steps of the proposed algorithm for one measurement of the available bandwidth A:

1. Set initial search range

   First, the program sends a packet stream according to the Cprobe algorithm [26] to find a very rough estimation of the available bandwidth. We set the search range to $(A_{cprobe}/2, A_{cprobe})$, where A_{cprobe} is the result of the Cprobe test.

2. Divide the search range

   The search range is divided into k sub-ranges $\mathrm{I}_i = (B_{i+1}, B_i)$ $(i = 1, 2..k)$. All sub-ranges have the identical width of the bandwidth. That is,

   $$B_i = B_u - \frac{B_u - B_l}{k}(i-1),\ i = 1, ..., k+1. \tag{2.1}$$

   When k increases, the results of Steps 4 and 6 become more accurate, because the width of each sub-range becomes smaller. However, a larger number of packet streams is required, which results in an increase in the number of used packets and the measurement time.

3. Send packet streams and check increasing trend

   For each of k sub-ranges, a packet stream i $(i = 1...k)$ is sent. The transmission rates of the stream's packets vary to cover the bandwidth range of the sub-range. We denote the j-th packet of the packet stream i as $\mathrm{P}_{i,j}$ $(1 \leq j \leq N$, where N is the number of packets in a stream) and the time at which $\mathrm{P}_{i,j}$ is sent from the sender host as $S_{i,j}$, where $S_{i,1} = 0$. Then $S_{i,j}$ $(j = 2..N)$ is set so that the following equation is satisfied:

   $$\frac{M}{S_{i,j} - S_{i,j-1}} = B_{i+1} + \frac{B_i - B_{i+1}}{N-1}(j-1), \tag{2.2}$$

   where M is the size of the probe packet. Figure 2.1 shows the relationship between the search range, the sub-ranges and the packet streams. In the proposed algorithm, packets in a stream are transmitted with different intervals, for this reason the measurement result may not be as accurate as the Pathload algorithm [28], in which all packets in a stream are sent with identical intervals. However, the proposed algorithm can check a wide range of bandwidth with one stream, whereas the Pathload checks only one value of the bandwidth with one stream. This reduces the number

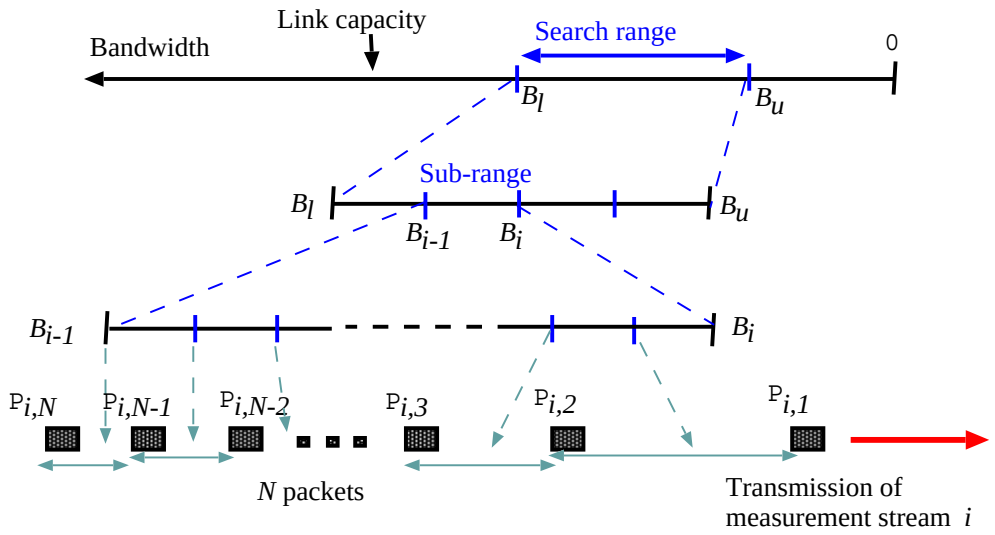


Figure 2.1: Relationship of search range, sub-ranges, streams, and probe packets

of probe packets and the time required for measurement. By this mechanism, the measurement speed is improved at the expense of measurement accuracy.

The program then observes $R_{i,j}$, the time the ACK of packet $P_{i,j}$ arrives at the sender host, where $R_{i,1} = 0$. We calculate the transmission delay $D_{i,j}$ of $P_{i,j}$ using the function $D_{i,j} = R_{i,j} - S_{i,j}$. We then check if an increasing trend exists in the transmission delay $(D_{i,j} - D_{i,j-1})$ $(2 \leq j \leq N)$ according to the algorithm used in [28]. As explained in [28], the increasing trend of transmission delay in a stream indicates that the transmission rate of the stream is larger than the current available bandwidth of the network path.

Let T_i be the increasing trend of stream i as follows:

$$T_i = \begin{cases} 1, & \text{increasing trend in stream } i, \\ -1, & \text{no increasing trend in stream } i, \\ 0, & \text{unable to determine.} \end{cases}$$

As i increases, the rate of stream i decreases. Therefore, T_i is expected to be 1 when i is sufficiently small. On the other hand, when i becomes large, T_i is expected to

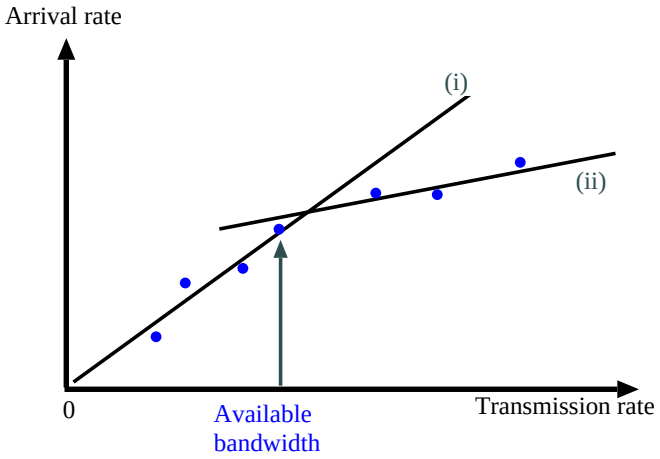


Figure 2.2: Finding the available bandwidth within a sub-range

become -1. Therefore, when neither of the successive streams m or $m+1$ have an increasing trend ($T_m = T_{m+1} = -1$), the remaining streams are expected not to have increasing trends ($T_i = -1$ for $m+2 \leq i \leq k$). Therefore, the program stops sending the remaining streams in order to speed up the measurement.

4. Choose a sub-range

   Based on the increasing trends of all streams, the algorithm chooses a sub-range which is most likely to include the correct value of the available bandwidth. First, it finds the value of a ($0 \leq a \leq k+1$), which maximizes $(\sum_{j=0}^{a} T_j - \sum_{j=a+1}^{k} T_j)$. If $1 \leq a \leq k$, it determines the sub-range I_a is the most likely candidate of the sub-range which includes the available bandwidth value. That is, as a result of the above calculation, I_a indicates the middle of streams which have increasing trends and those which do not. If $a = 0$ or $a = k+1$, on the other hand, the algorithm decides that the available bandwidth does not exist in the search range (B_l, B_u). The algorithm determines that the available bandwidth is larger than the upper bound of the search range when $a = 0$, and that when $a = k+1$ the available bandwidth is smaller than the lower bound of the search range.

   In this way, the algorithm finds the sub-range which is expected to include the available bandwidth according to the increasing trends of the packet streams.

5. Calculate the available bandwidth

   The algorithm then derives the available bandwidth A from the sub-range I_a chosen

by Step 4. It first determines the transmission rate and the arrival rate of the packet $P_{a,j}$ $(j = 2...N)$ as $\frac{M}{S_{a,j}-S_{a,j-1}}$, $\frac{M}{R_{a,j}-R_{a,j-1}}$, respectively. It then approximates the relationship between the transmission rate and the arrival rate as two straight lines using the linear regression method, as shown in Figure 2.2. Since it determines that the sub-range I_a includes the available bandwidth, the slope of line (i) which consists of small transmission rates is nearly 1 (the transmission rate and the arrival rate are almost equal), and the slope of line (ii) which consists of larger transmission rates is smaller than 1 (the arrival rate is smaller than the transmission rate). Therefore, it determines that the highest transmission rate in line (i) is the value of the available bandwidth.

On the other hand, when the algorithm has determined that the available bandwidth value does not exist in the search range (B_l, B_u) in Step 4, it temporarily sets the value of available bandwidth as follows:

$$A = \begin{cases} B_l, & a = 0, \\ B_u, & a = k + 1. \end{cases}$$

6. Create a new search range

When the program have found the value of the available bandwidth from a sub-range I_a in Step 5, we accumulate the value as the latest statistical data of the available bandwidth. The next search range (B'_l, B'_u) is calculated as follows:

$$B'_l = A - \max\left(1.96\frac{S}{\sqrt{q}}, \frac{B_m}{2}\right),$$

$$B'_u = A + \max\left(1.96\frac{S}{\sqrt{q}}, \frac{B_m}{2}\right),$$

where S is the variance of stored values of the available bandwidth and q is the number of stored values. Thus, we use the 95% confidential interval of the stored data as the width of the next search range, and the current available bandwidth is used as the center of the search range. B_m is the lower bound of the width of the search range, which is used to prevent the range from being too small. When no accumulated data exists (when the measurement has just started or just after the accumulated data is discarded), we use the same search range as that of the previous

measurement.

On the other hand, when we can not find the available bandwidth within the search range, it is possible to consider that the network status has changed greatly. Therefore, we discard the accumulated data because this data becomes unreliable as statistical data. In this case, the next search range (B'_l, B'_u) is set as follows:

$$B'_l = \begin{cases} B_l, & a = 0, \\ B_l - \frac{B_u - B_l}{2}, & a = k+1, \end{cases}$$

$$B'_u = \begin{cases} B_u + \frac{B_u - B_l}{2}, & a = 0, \\ B_u, & a = k+1. \end{cases}$$

This modification of the search range is performed in an attempt to widen the search range in the possible direction of the change of the available bandwidth.

By this statistical mechanism, we expect the measurement algorithm to behave as follows: when the available bandwidth does not change greatly over a period of time, the search range becomes smaller and more accurate measurement results can be obtained. On the other hand, when the available bandwidth varies greatly, the search range becomes large and the measurement can be restarted from the rough estimation. That is, the proposed algorithm can give a very accurate estimation of the available bandwidth when the network is stable, and a rough but rapid estimate can be obtained when the network status changes.

2.3 ImTCP: TCP with inline network measurement

2.3.1 Overview

The deployment of the proposed measurement algorithm requires a modification of the TCP sender only, incurring a low deployment cost. We implement a program for inline network measurement in the sender program of RenoTCP to create ImTCP. The program locates at the bottom of TCP layer, as shown in Figure 2.3. When a new TCP data packet is generated at the TCP layer and is ready to be transmitted, it is stored in an intermediate FIFO buffer (hereafter called the *ImTCP buffer*) before being passed to the IP layer. The timing at that the packets are passed to the IP layer is controlled by the program.

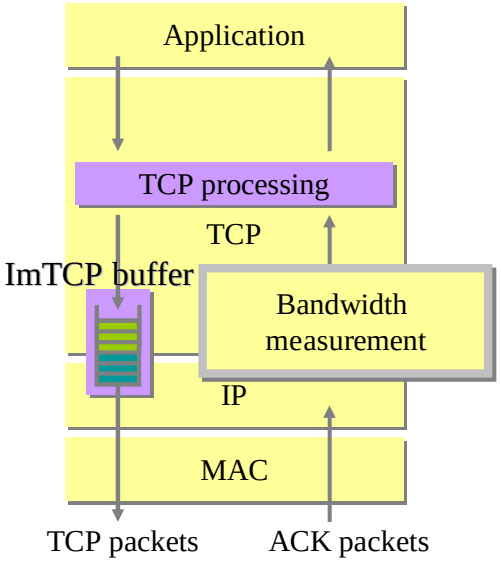


Figure 2.3: Placement of measurement program at ImTCP sender

When ImTCP performs a measurement, the program adjusts the transmission intervals of some packets according to the measurement algorithm. When ImTCP is not performing a measurement, it passes all TCP data packets arriving at the buffer immediately to the IP layer. On the other hand, when an ACK packet arrives at the sender host, the measurement program records the arrival time for measurement then passes the ACK packets to the TCP layer for TCP protocol processing.

2.3.2 Packet storing mechanism

Available bandwidth fluctuations occur in different time scales, whose durations range from the order of minutes to less than a microsecond. Therefore, we enable ImTCP to perform measurements in as small intervals as possible. The measurement algorithm uses previous measurement results to determine a search range for the next measurement. Therefore, it is natural that only one measurement operation should be performed for one RTT. If the TCP window size is sufficiently large, we can perform multiple measurements for one RTT by introducing a quite complex mechanism. However, many difficulties must be overcome to accomplish this, including interaction of measurement tasks, delays caused by multiple streams. We therefore decided that ImTCP should perform at most one measurement operation per RTT. One RTT is long enough for ImTCP to recover the transmission rate after a measurement.

The measurement program dynamically adapts to changes in the TCP window size. It stores no data packets when the current window size is smaller than the number of packets required for a measurement stream. This is because the TCP sender cannot transmit a number of data packets larger than the window size. On the other hand, when the window size is sufficiently large, the program creates all streams required for a measurement in each RTT.

The measurement program consists of three units. The *ImTCP Buffer unit* stores TCP data packets and passes each packet to the IP layer under control of the *Control unit*. It informs the Control unit when a new TCP packet arrives. The Control unit determines when to send the packets stored in the buffer. Details of the Measurement unit were introduced in Section 2.2.

Here, we explain the operation of the Control unit. The Control unit has four functional states, STORE PACKET, PASS PACKET, SEND STREAM and EMPTY BUFFER, as shown in Figure 2.4. The Control unit is initially in the STORE PACKET state. In what follows, we describe the detailed behaviors of the Control unit in each state;

- *STORE PACKET* state
  - Start storing packets for the creation of measurement streams. Set the packet storing timer to end packet storing after certain length of time T. The timer value T is discussed in Subsection 2.3.3.
  - Go to the SEND STREAM state if the number of stored packets equals to m. The value of m is discussed in Subsection 2.3.3.
  - Go to the EMPTY BUFFER state if the current TCP window size becomes smaller than N or the packet storing timer expires. N is the number of packets needed to create a measurement stream.
- *EMPTY BUFFER* state
  - Pass currently stored packets to the IP layer until the buffer becomes empty.
  - Return to the STORE PACKET state.
- *SEND STREAM* state
  - Send a measurement stream. The transmission rate of the stream is determined according to the measurement algorithm. During stream transmission, packets arriving at the buffer are stored in the ImTCP buffer.

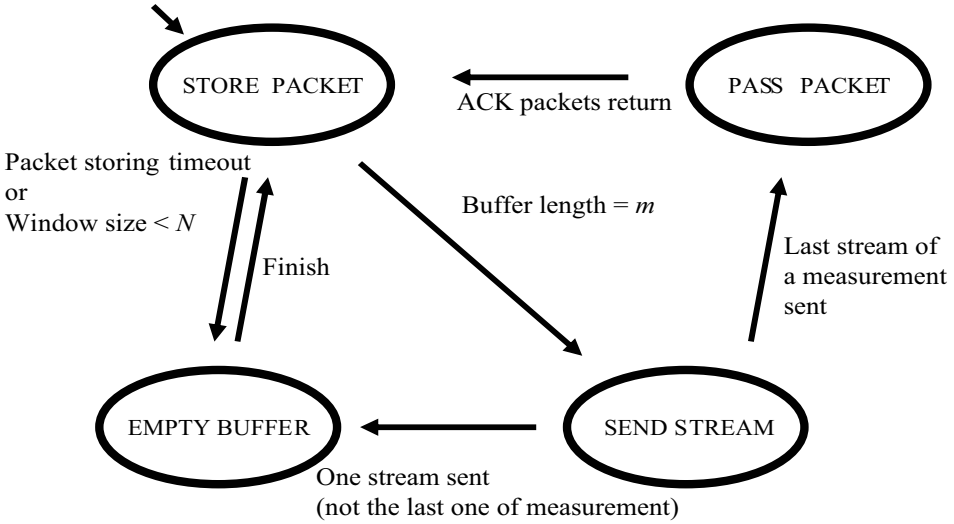


Figure 2.4: State transition in the Control unit

- After the transmission of the stream, if the stream is the last of a measurement, go to PASS PACKET state, if not, go to the EMPTY BUFFER state.

- *PASS PACKET* state

  - Pass every packet in the buffer immediately to the IP layer.
  - Go to the STORE PACKET state when all ACK packets of the transmitted measurement streams have arrived at the sender.

2.3.3 Parameter settings

Number of packets required to start a measurement stream (m)

The timing for sending packets in a measurement stream is determined by the measurement algorithm. If N packets were stored prior to the beginning of transmission, the long storage time would slow the TCP transmission speed. Instead, transmission begins when only a partial number of packets (m out of N packets) have arrived in the ImTCP buffer. The timing is such that the former part of the stream is being transmitted as the latter part of the stream is still arriving at the buffer, and the latter packets are expected to arrive in time for transmission. Thus, we reduce the effect of the packet storing mechanism on TCP transmission.

If we set m to a very small value, the latter part of the stream will not be available when the former part of the stream has already been transmitted, in which case the stream transmission fails. Therefore, m must be large enough to ensure successful transmission of the measurement stream, but no larger. The algorithm for determining m is given below. In the algorithm, m is adjusted according to whether or not transmission of the previous measurement streams was successful.

- Set $m = N$ initially. The minimum of m is 2, and the maximum of m is N.
- If F successive measurements are completed successfully, and m is greater than its minimum of 2, then decrease m by 1. We set F to 2.
- If a stream creation fails, and m is less than its maximum of N, then increase m by 1 and create the stream again.

Packet storing timer (T)

We avoid degrading the TCP transmission speed, caused by storing data packets before they are passed to the IP layer, by appropriately setting a timer to stop the creation of a stream. Obviously, there is a trade-off between measurement frequency and TCP transmission speed when choosing the timer value. That is, for large timer values, the program can create measurement streams frequently so measurement frequency increases. In this case, however, because TCP data packets may be stored in the intermediate buffer for a relatively long period of time, TCP transmission speed may deteriorate. Following is an example. An application temporarily stops sending data, but the measurement program is still waiting for more packets to form a measurement stream. There is no new data packet arriving at the ImTCP buffer so the packets currently in the buffer are delayed until the application sends new data. In this situation, if the application does not send data within 1 sec, the TCP timeout will occur. On the other hand, for small timer values, the program may frequently fail to create packet streams, leading a low frequency of measurement success. In the following discussion, we derive the appropriate value for the packet storing timer by applying an algorithm similar to the RTO calculation in TCP [46].

If we assume a normal distribution of packet RTTs with average A_{RTT} and variance D_{RTT}, A_{RTT} and D_{RTT} can be inferred from the TCP timeout function [46]. We use the following notation;

- X: RTT of a TCP data packet

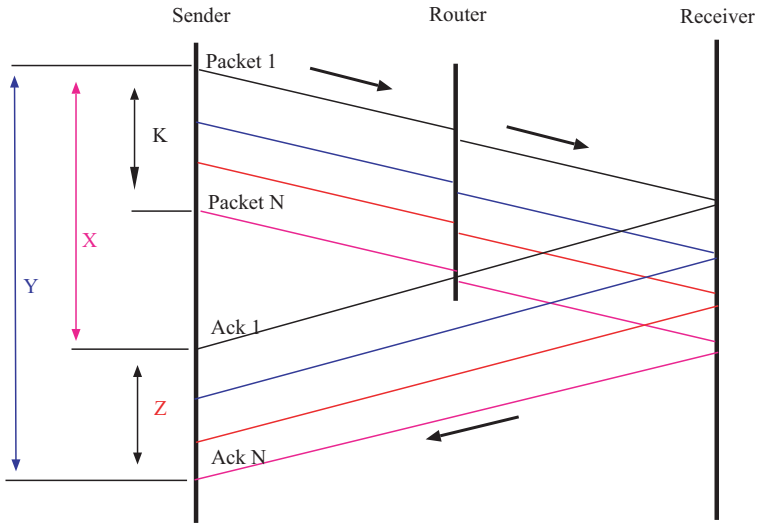


Figure 2.5: Packet transmission times in TCP

- Y: The time since the first of N successive data packets is sent until the ACK of the last packet arrives at the sender
- Z: The time necessary for N successive ACK packets to arrive at the sender

We illustrate X, Y and Z in Figure 2.5. We need to know the distribution of Z to determine the appropriate value for the packet storing timer. From Figure 2.5, we can see that:

$$Z = Y - X. \tag{2.3}$$

From the assumption mentioned above, X has a normal distribution $N(A_{RTT}, D_{RTT})$. Note that Y is the period of time from sending the first packet until the last packet is sent (we denote the length of this period as K) plus the RTT of the last packet. That is, we can conclude that the distribution of Y is $N(A_{RTT} + K, D_{RTT})$. From Equation (2.3) we then obtain the distribution of Z, as $N(K, 2 \cdot D_{RTT})$.

Here, we provide a simple estimate of K. In a TCP flow, due to the self-clocking phenomenon, the TCP packet transmission rate is a rough estimate of the available bandwidth of the network link. The average time needed to send N successive TCP data packets is

$$K = \frac{M}{A}(N - 1), \tag{2.4}$$

where M is the packet size and A is the value of available bandwidth which can obtain from

the measurement results. From the distribution of Z and Equation (2.4), we determine the waiting time for N ACK packets as below:

$$T = \frac{M}{A}(N-1) + 4 \cdot D_{RTT}.$$

Using this value for the timer, the probability of successfully collecting N packets reaches approximately 98% due to the characteristics of the normal distribution. Thus, we use a relatively short timer length that reduces additional processing delays caused by the measurement program but provides a high probability of collecting a sufficient number of packets for creating measurement streams.

2.3.4 Other issues

Effect of Delayed ACK option

When a TCP receiver uses the delayed ACK option, it sends only one ACK packet for every two data packets. In this case, the proposed algorithm does not work properly since it assumes the receiver host will send back a probe packet for each received packet. To solve this problem, Step 3 in Subsection 2.2.2 of the proposed algorithm should be changed so that intervals of three packets are used rather than intervals of two packets. That is, we calculate the transmission delay $(D_{i,2j'+2} - D_{i,2j'})$ $(1 \leq j' \leq \lfloor N/2 \rfloor)$ for the probing packets in stream i in order to check its increasing trend. This modification has almost the same effect as halving the number of packets in one stream, resulting in a degradation in measurement accuracy. Therefore, the number of packets in a stream should be increased appropriately.

Effect of packet fragmentation

In the case where TCP packets are transmitted through a queue or node for which the MTU (Maximum Transmission Unit) is smaller than the packet size, the packets will be fragmented into several pieces in the network. The problem here becomes a question of whether measurement result will still be accurate if the packets in measurement streams become fragmented somewhere on the way to the receiver. We argue that fragmentation has little effect on the measurement results. The measurement algorithm is based on the increasing trend of the packet stream in order to estimate available bandwidth. Even with fragmentation, the stream still shows an increasing trend when and only when the

transmission rate is larger than the available bandwidth. However, fragmentation does increase the packet processing overhead, which may in turn raise the increasing trend of packet streams if it occurs at a bottleneck link. This may lead to a slight underestimation in the measurement results.

Effect of packet retransmission

When 3 dupACKs arrive and TCP packet retransmission occurs, the measurement program transmits the retransmitted packet then immediately releases all packets stored in the ImTCP Buffer. While the dupACKs arrive, the measurement program can not determine the arrival intervals of the ACKs of the measurement streams, therefore, it can not deliver measurement results. The program stops sending measurement streams and waits until a new ACK, instead of dupACKs, arrives. This is done to that the network has recovered from the congestion, and then measurements are restarted. Thus, packet retransmission only interrupts the measurements for a while.

2.4 Simulation results

2.4.1 Effect of parameters

Number of packets in a measurement stream N

Figure 2.6 shows the network model used in the ns-2 [47] simulation. A sender host connects to a receiver host through a tight link. The capacity of the tight link is 100 Mbps and the one-way propagation delay is 90 msec. All of the links from the end hosts to the routers have a 100-Mbps bandwidth. There is cross traffic 1, 2 and 3 generated by end hosts connecting to the routers. The cross traffic is made up of UDP packet flows, in which various packet sizes are used according to the monitored results in the Internet reported in [48], as shown in Table 2.1. We make the available bandwidth on the tight link fluctuate by changing cross traffic 2's rate. Cross traffic 1 and 3 are for adding noise to the transmission delay of ACK packets.

To avoid counting on the effect from TCP behaviors, we investigate the results of the measurement algorithm when the sender uses the UDP streams for the measurement. In this case, the receiver simply echoes the UDP streams back to the sender. We show results in which we turn off cross traffic 1 and 3 and change the available bandwidth as follows:

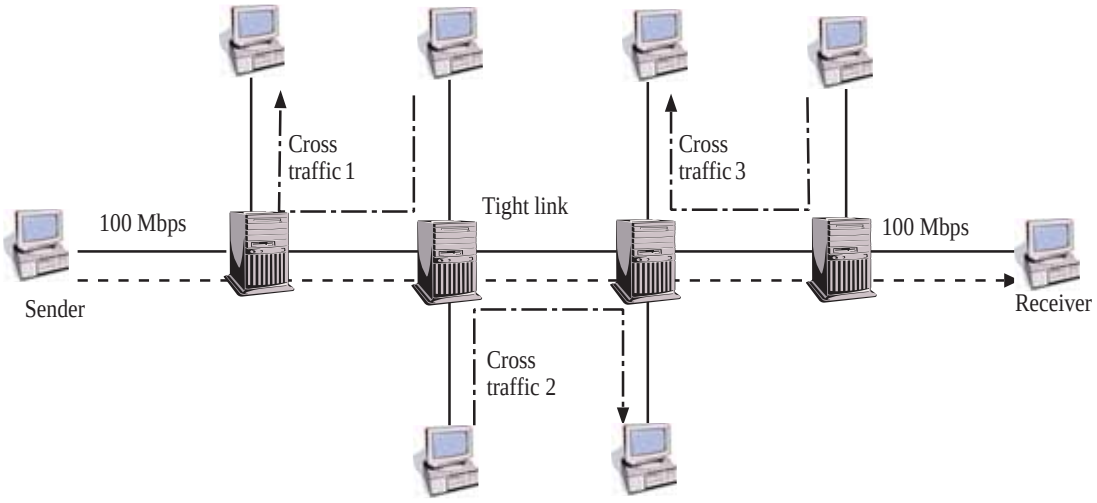
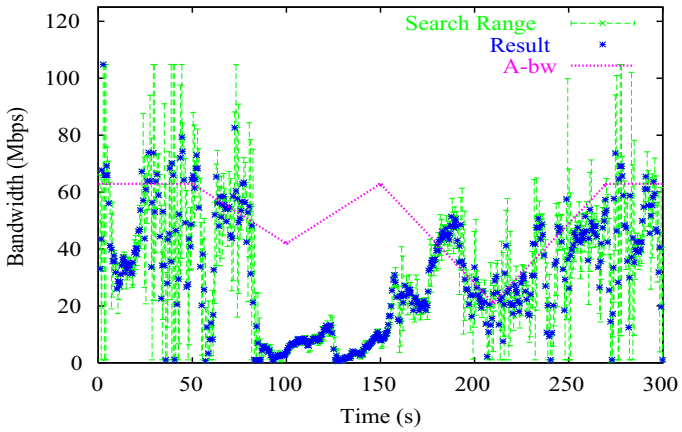


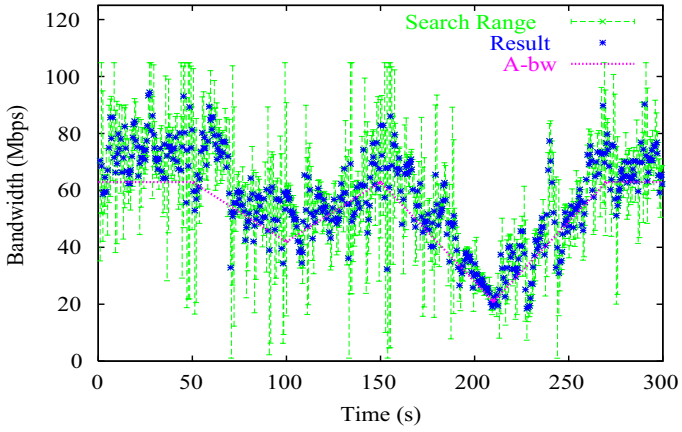
Figure 2.6: Network model for evaluation of ImTCP

Table 2.1: Distribution of packet size of the cross traffic

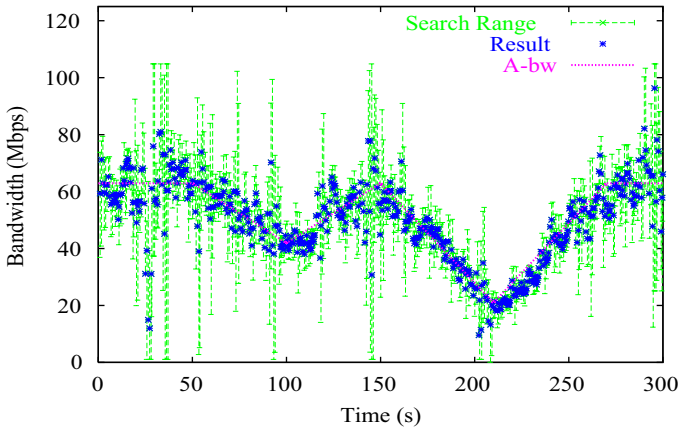
Packet size (Bytes)	Proportion of bandwidth (%)
28	0.08
40	0.51
44	0.22
48	0.24
52	0.45
552	1.10
576	16.40
628	1.50
1420	10.50
1500	37.10
40–80 (range)	4.60
80–576 (range)	9.60
576–1500 (range)	17.70



(a) N=3



(b) N=5



(c) N=8

Figure 2.7: Results of the proposed measurement algorithm

Table 2.2: Number of measurement results when m changes

m (packets)	2	3	4	5	6	7	Proposed
$N = 5$	387	441	424	389	-	-	424
$N = 6$	200	216	406	392	370	-	394
$N = 7$	77	173	277	279	348	349	359

from 0 sec to 50 sec, the available bandwidth is 60 Mbps; from 50 sec to 100 sec, decreases to 40 Mbps; from 100 sec to 150 sec, increases to 60 Mbps; from 150 sec to 210 sec, decreases to 20 Mbps; from 120 sec to 270 sec, increases to 60 Mbps; and from 270 sec to 300 sec the available bandwidth is 60 Mbps. The simulation results are shown in Figure 2.7. These figures indicate that when N is 3, the measurement results are far from the correct values. That is because, when N is very small, we cannot determine the increasing trend of the streams correctly in Step 3 in the proposed algorithm, which leads to the incorrect choice of sub-range in Step 4. When N becomes larger than 5, on the other hand, the estimation result accuracy increases.

With a large value of N, packet storing time for one measurement stream becomes longer. Therefore, we want to keep the value as small as possible to avoid degrading the TCP transmission rate. We use $N = 5$ as the default setting. In case the measurement accuracy is required, the N much be set to a larger value. In the following simulations, when there is no explicit mention, we use $N = 5$.

Effect of setting of m

We next examine number of measurement results yielded in 80 (s) of simulation by a ImTCP connection when the available bandwidth is set to 3 Mbps. Our simulations now uses the same topology as described in Figure 2.6 except that the UDP sender and receiver are replaced by an ImTCP sender and an ImTCP receiver, respectively. Table 2.2 shows the number of measurement results when N is set to different values. The results when using the proposed setting are shown in the column "Proposed" of the table. We vary the value of m to find at which value, the number of measurement results is almost the same as that in case $m = N$ (the underlined values). When $N = 5$, $m = 2$ is a good setting, because the number of results maintain highly while the average packet storing time is smallest. But when $N = 6$, the optimal value of m changes to 4 and $m = 2$ becomes a very bad setting because it decrees the number of results. Thus, the ideal value of m depends

Table 2.3: Number of measurement results when T changes

T (s)	0.04	Proposed	0.01	0.004
A-bw = 4 Mbps	379	371	324	105
A-bw = 7 Mbps	486	488	423	298

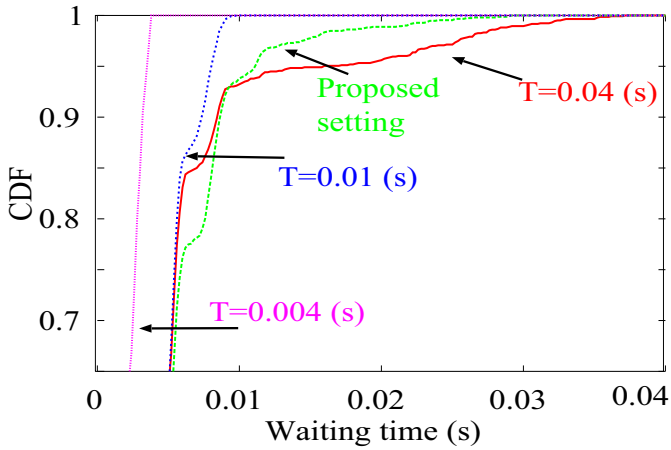


Figure 2.8: CDF of the waiting time of the first packet in measurement streams

on the value of N. On the other hand, the dynamic setting always delivers large number of measurement results while the average packet waiting time is kept low.

Packet waiting time T

We next examine the number of measurement results of ImTCP when we set T to 0.04 sec, 0.01 sec, 0.004sec. Table 2.3 shows the values when we set available bandwidth to 4 Mbps and 7 Mbps. The "Proposed" column shows the correspondent values when we use the proposed setting for T. When T takes small values such as 0.004 sec or 0.01 sec, ImTCP often fails to create measurement streams, therefore, the number of measurement results is small. On the other hand, as shown in Figure 2.8, when T takes a large values, such as 0.04 sec, the waiting time of the packets for stream creation is long. As a result, we found that the transmission rate of ImTCP when $T = 0.04$ sec is degraded. In contrast, the proposed setting for T can eliminate the cases when the packet waiting time is long, while maintaining the number of the measurement results.

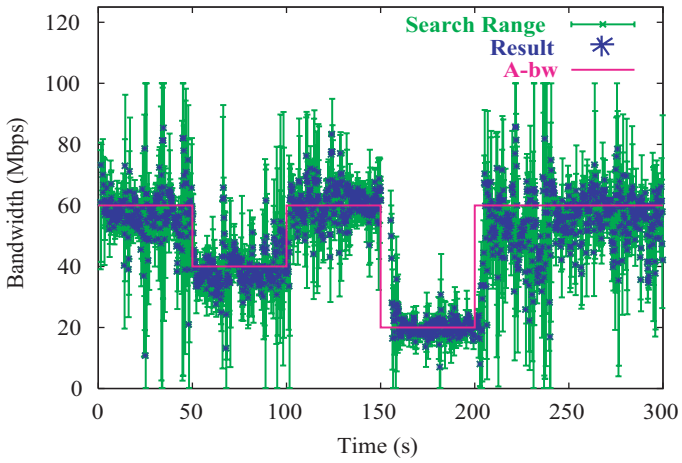
2.4.2 Measurement accuracy

We next examine the measurement accuracy of ImTCP when all the parameters are set to the values proposed above. We set cross traffic 1's transmission rate to 5 Mbps, cross traffic 3's rate to 15 Mbps and changes cross traffic 2' rate so that the available bandwidth is 60 Mbps from 0 sec to 50 sec, 40 Mbps from 50 sec to 100 sec, 60 Mbps from 100 sec to 150 sec, 20 Mbps from 150 sec to 200 sec and 60 Mbps from 200 sec to 300 sec. Figures 2.9(a) and 2.9(b) show the measurement results of ImTCP when the number of packets (N) in a measurement stream is five and eight, respectively. We also plot the correct values of the available bandwidth in all figures. Comparing Figure 2.9(a) with Figure 2.7(b) and Figure 2.9(b) with Figure 2.7(c), we observe that our measurement method can be successfully applied to TCP with no degradation in measurement accuracy. The maximum relative error of the measurement results is 20.8% and 20.1%, when N is five and eight, respectively. Experiments in high network environments in [44] have shown that the error of Iperf [49] is 15%, Pathload [28] is 16% and Pathchirp [38] is 15–20%. An other experiment in [27] shows that Spruce [27] has only 70% results that have the relative error smaller than 30% and, in this case, Pathload and IGI experience larger errors. The results mean that ImTCP with N=5 can perform measurement almost as well as existing active measurement tools.

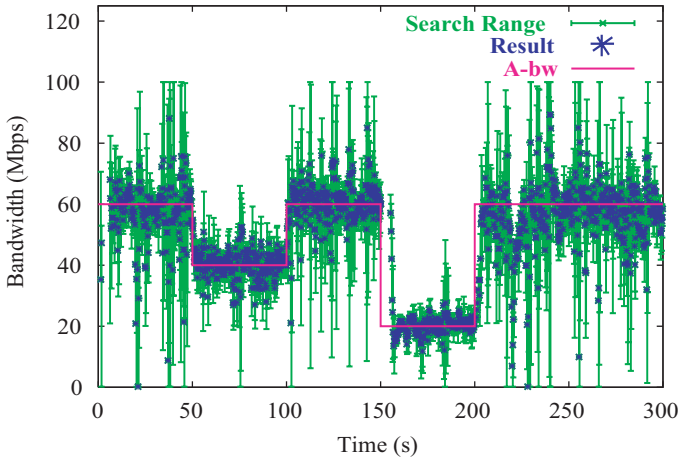
Figure 2.10 shows the changes in throughput of ImTCP in this simulation. For comparison, we also show the case of RenoTCP under the same network conditions. From the figure, we can see that ImTCP performs the measurement with a throughput almost the same as that of RenoTCP. An important point we can take from Figure 2.9 and 2.10 is that ImTCP yields accurate results even when the current throughput is lower than the available bandwidth. For example, from 0 sec to 50 sec in the simulation, although the throughput of ImTCP is less than 60 Mbps, the available bandwidth value is still realized, as shown in Figure 2.9.

2.4.3 Comparision with existing inline measurement methods

We compare the measurement accuracy of ImTCP with that of other inline measurement methods. We examine the average measurement results of every 0.5 sec of ImTCP, Westwood [16], the method proposed by Hoe [34] and TCP-Rab [35] in the network condition described in Subsection 2.4.2. In fact, the method by Hoe's performs only one measurement right after the connection starts. To compare with other methods, we repeat the



(a) N=5



(b) N=8

Figure 2.9: Measurement results of ImTCP

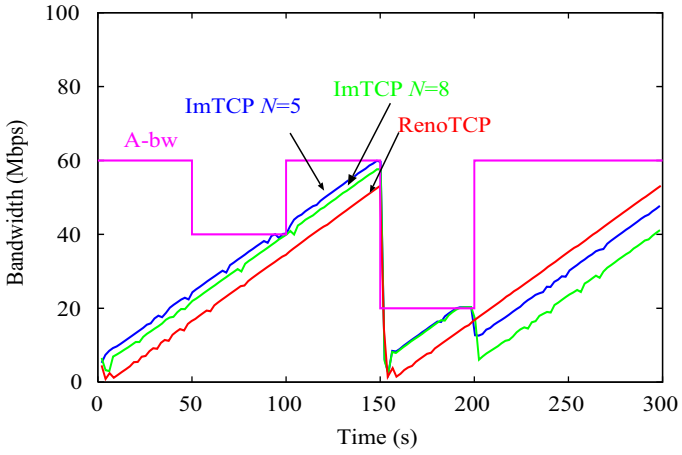


Figure 2.10: Throughput of ImTCP and RenoTCP

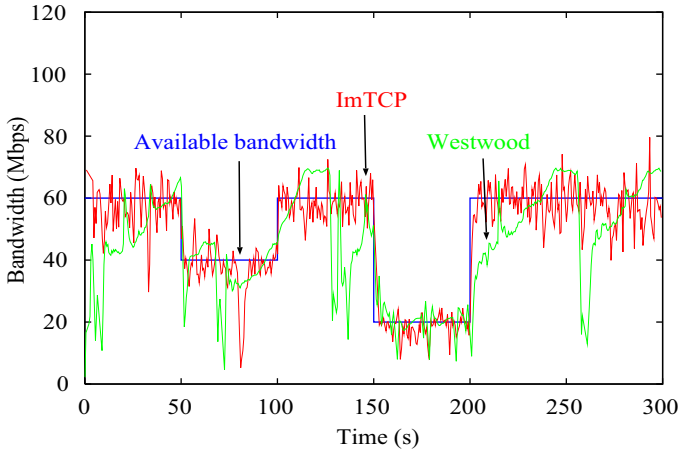
measurements in every RTT.

Figure 2.11(a) and 2.11(b) show that TCP-Rab can deliver accurate measurement results sometimes because the measurements do not interfered by the cross traffic 1 and 3. Hoe's method is based on only 3 closely transmitted ACK packets so the affect from cross traffic 1 and 3 is also small. Westwood performs worse in this condition because it counts on the arrival intervals of all ACK packets. However, the methods are all passive measurements so no one can detect the real value of available bandwidth if it changes fast from low to high. In contrast, ImTCP can detect the changes of available bandwidth fast because it actively adjusts the transmission rate of packets, even in the present of cross traffic 1 and 3.

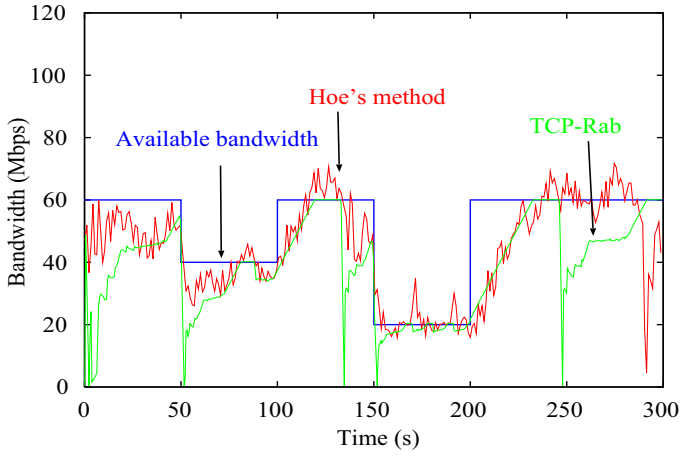
2.4.4 Effect of ImTCP on other traffic

We finally investigate the measuring accuracy in an environment where the cross traffic fluctuates greatly. We use the network model depicted in Figure 2.6 with the cross traffic 1 and 3 turn off. Cross traffic 2 is changed to Web traffic involving a large number of active Web document accesses. We use a Pareto distribution for the Web object size distribution. We use 1.2 as the Pareto shape parameter with 12 Kbytes as the average object size. The number of objects in a Web page is eight. The capacity of the tight link is set to 50 Mbps.

We run the simulation for 500 sec and find that the average throughput of ImTCP is 25.2 Mbps while that of Reno TCP is 23.1 Mbps. The results therefore show that data transmission speed of ImTCP is almost the same as that of Reno TCP.



(a) ImTCP and TCP Westwood



(b) Method by Hoe and TCP-Rab

Figure 2.11: Average measurement results of inline measurement methods

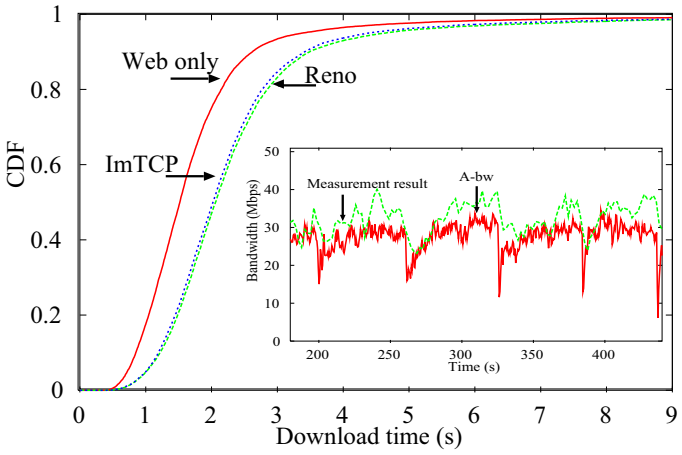


Figure 2.12: Comparison of Web page download times

We compare the effect of ImTCP and Reno TCP on Web page download time in Figure 2.12. This figure shows cumulative density functions (CDFs) of the Web page download time of Web clients. We can see that ImTCP and Reno TCP have almost the same effect on the download time of a Web page. This indicates that inline measurement does not affect other traffic sharing the link with ImTCP. Small graph in Figure 2.12 also confirms that the ImTCP measurement result reflects the change in available bandwidth well.

2.4.5 Bandwidth utilization and fair share

Two important characteristics of the Internet transport protocol are full utilization of link bandwidth and fair sharing of bandwidth among connections. We use the following simulations to show that ImTCP has these two characteristics. We use the network topology shown in Figure 2.13 with many ImTCP connections sharing a tight link. Using a small buffer (200 packets) in the router at the tight link to force conflict among connections, we vary the number of ImTCP connections while observing total throughput and fairness among the connections.

In Table 2.4 we show the Jain's fairness index [50] for the ImTCP connections as well as the total transmission rate of ImTCP connections in Mbps when the capacity of the tight link is set to 50, 60, and 70 Mbps. The number of connections is also varied. Also shown are the transmission rates when ImTCP is replaced by Reno TCP.

This Jain's fairness index takes a value from 0 to 1; a share is considered fair as its index is near 1. We can see that the ImTCP connections share the bandwidth link fairly

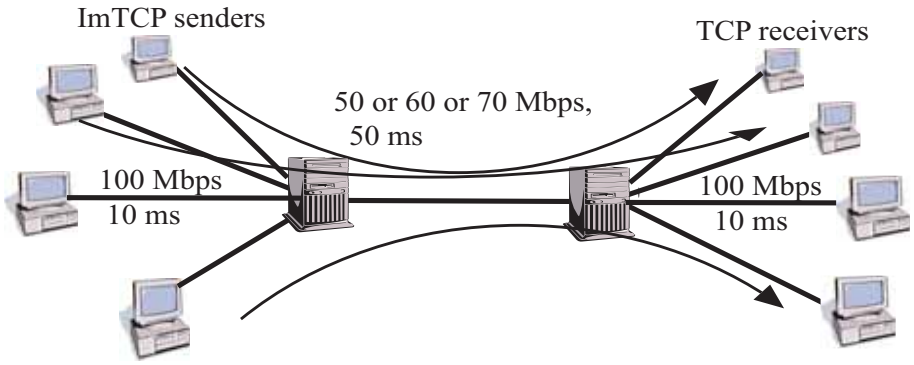


Figure 2.13: Network model for investigating bandwidth utilization and fair share

Table 2.4: Fairness and link utilization of ImTCP

Capacity	#flows	Jain's index	ImTCP	Reno
50	2	0.999	44.4	45.6
	10	0.997	46.7	46.0
	24	0.986	47.6	46.1
60	2	0.999	53.2	53.1
	10	0.992	55.3	54.0
	24	0.995	56.7	54.11
70	2	0.999	59.9	60.6
	10	0.992	63.7	61.9
	24	0.992	65.9	62.0

because the index is always near to 1. Due to the small buffer size of the tight link, when the number of connections are small the total throughput is not very high. When the number of connections is large, total throughput increases. We can see that ImTCP and Reno TCP have almost the same link utilization regardless of the number of connections.

2.4.6 TCP-friendliness and TCP-compatibility

ImTCP is *TCP-friendly*; it achieves the same throughput as Reno TCP under the same condition. Simulation results shown in Table 2.4 confirm this. Although ImTCP buffers packet stream at the sender host, the buffered packets is quickly transmitted after each transmission of a packet stream (in the EMPTY BUFFER state). Therefore, there is almost no degradation in transmission speed of data packets.

A network protocol is called *TCP-compatible* if the connections using this protocol

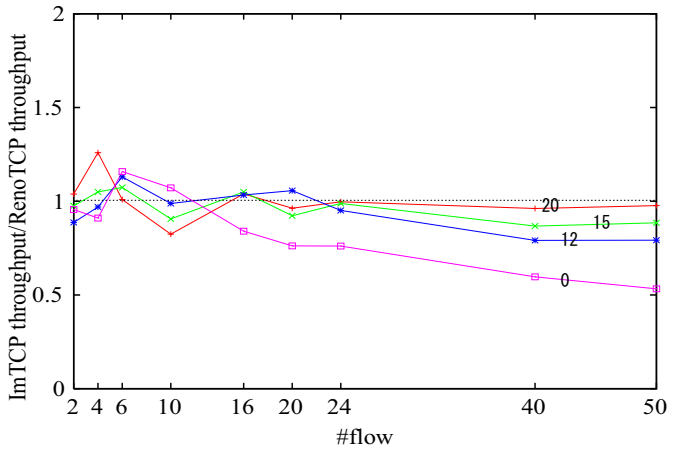


Figure 2.14: Comparision of ImTCP and Reno TCP throughput

fairly share the bandwidth in an tight link with Reno TCP [51]. We examine the TCP-compatibility of ImTCP by observing the throughput of ImTCP connections when they coexist with Reno TCP connections and non-TCP traffic. The non-TCP traffic is indicated by a 0.1 Mbps UDP flows with randomly varied packet size (300-600 bytes). All TCP and non-TCP traffic conflict at the 50 Mbps tight link. We use the same number of ImTCP and Reno TCP connections.

The ratio of the total throughput of ImTCP connections to that of Reno TCP connections is shown in Figure 2.14. When the ratio is around 1, ImTCP is TCP-compatible. The horizontal axis shows the total number of the TCP connections. In the current version of ImTCP, there is no time interval between 2 measurements. The result of this version is shown by the line numbered 0. We can see that ImTCP receives lower throughput than Reno TCP. The reason is as follows. Some of packets of ImTCP may not be transmitted in burst due to the affect of packets buffering at the sender. On the other hand, traditional TCP connections in competing environment have the trend to transmit packets in a bursty fashion. When the packets of ImTCP collide with the bursts of packets of Reno TCP, they have higher probability to be dropt. Therefore, ImTCP with high measurement frequency may lost more packets when conflicting with Reno TCP, leading to a lower throughput.

The simple and effective way to overcome this problem is increasing the measurement interval of ImTCP. We next consider the cases when the measurement intervals are 12, 15 and 20 RTTs, and show the results by the line numbered 12, 15 and 20, respectively, in Figure 2.14. Note that the RTT in this case is 0.14 seconds and each measurement takes at most 4 RTTs. Therefore, 12, 15 and 20 RTT interval means ImTCP releases measurement

results in 2.24(s), 2.66(s) and 3.36(s), respectively. When the measurement interval is relatively small, ImTCP achieves lower throughput than Reno TCP. On the other hand, when the measurement interval is equal to or larger than 20 RTTs, ImTCP is compatible to Reno TCP. In other words, when the measurement frequency is smaller than a certain value (in this simulation, that is 1/3.36 times per second) there is a trade-off relationship between the TCP compatibility and the measurement frequency.

In such a heavy congested network that there is no available bandwidth even when ImTCP does not exist, ImTCP must be TCP-compatible in order to gain the equal throughput to other connections. Moreover, in this environment, the measurement results themselves usually do not bring so much valuable information so they will be not required updated frequently. Therefore, in this case, ImTCP must take a low measurement frequency. When the network is vacant, ImTCP will not conflict with other connections so much. In this case, TCP-compatibility does not strictly required, because ImTCP is TCP-friendly so that ImTCP will perform exactly like traditional TCP. Besides, the information about the vacancy in the network will be of interest. In this case, ImTCP should increase its measurement frequency. Thus, there should be a dynamic adjustment for the measurement frequency according to the network status. We will consider the problem in our future works.

2.5 Transmission modes of ImTCP

2.5.1 Background transmission

The transmission for backup data or cached data (background traffic) should not degrade throughput of other traffic (foreground traffic), which may be more important. We introduce an example showing that ImTCP successfully uses the results of bandwidth availability measurements to prevent its own traffic from degrading the throughput of other traffic. We call this type of ImTCP data transmission *background mode*.

The main idea is to set an upper bound on the congestion window size according to estimated values so that the transmission rate does not exceed the available bandwidth. This reduces the effect ImTCP has on other traffic in the same network links. We use the following control mechanism. When

$$g \cdot RTT \cdot A > m \cdot N,$$

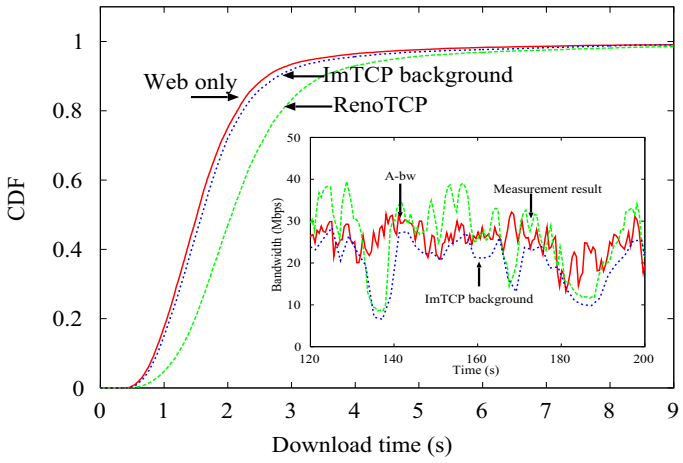


Figure 2.15: Average of Web page download time

we set

$$MaxCwnd = g \cdot RTT \cdot A,$$

where A is the estimated value of available bandwidth, $MaxCwnd$ is the upper bound of the congestion window size and N is the number of packets for a measurement stream. The parameter g can range from 0 to 1. When g is small, ImTCP uses less bandwidth and interferes only very slightly with foreground traffic. When g is near 1, ImTCP uses more bandwidth and its effect on foreground traffic grows. We set the upper bound of the congestion window size ($MaxCwnd$) to $g \cdot RTT \cdot A$ only when the value is large enough for ImTCP to continue performing measurements well.

We examine the behavior of ImTCP in background mode when foreground traffic is originated with Web document transfers. We replace the ImTCP connection in the simulation in Section 2.4 with a background mode ImTCP connection. Figures 2.15 compare the download time for Web pages under ImTCP and Reno TCP. We find that ImTCP has only a very small effect on the download time of the foreground Web traffic. The average throughput of ImTCP in this case is about 72% of that of Reno TCP. The small graph in Figure 2.15 shows the measurement value and throughput of ImTCP connection as a function of simulation time in this case. Note that the throughput of ImTCP does not approach the actual value of available bandwidth. This indicates that ImTCP background mode is successfully avoiding interference with Web traffic.

2.5.2 Full-speed transmission

We introduce another example of a modified congestion control mechanism to show that ImTCP can enhance link utilization using its measurement results.

To improve TCP throughput in wireless or high-speed networks, we introduce an available-bandwidth-aware window size adjustment. The idea is to use the measurement result to adjust the increasing speed of the congestion window size. When the available bandwidth is large, the window size increases quickly to make full use of available bandwidth, and when the available bandwidth is small due to the existence of other traffic, the window size increases slowly. We call this type of ImTCP data transmission *full-speed mode*.

In the congestion avoidance phase, we do not increase the congestion window size ($Cwnd$) by one in every RTT. Instead, we use the following adjustment:

$$Cwnd \leftarrow Cwnd + max\left(1, h \cdot (1 - \frac{Cwnd}{V})\right),$$

$$V = A \cdot RTT.$$

In the equation, h ($h \geq 1$) is a parameter that determines how fast the window size increases. If h is large, ImTCP can successfully utilize the bandwidth link. When h is small or equal to 1, ImTCP behaves the same as Reno TCP.

We perform the following simulation to investigate the performance of ImTCP in full-speed mode. The ImTCP sender and ImTCP receiver is connected by two routers with Gigabit links. The 500 Mbps link between the two routers becomes the bottleneck link in the path. We assume the buffer of the TCP receiver is large so the TCP throughput can achieve 500 Mbps.

Figure 2.16 shows the changes in the window size of ImTCP in full-speed mode, High-Speed TCP (HSTCP) [52] and Reno TCP in the network. Reno TCP requires a long time to reach a large window size. HSTCP increases the window size quickly to fully use the free bandwidth; however, the increasing speed is non-sensitive to the available bandwidth such that packet loss events occur frequently. Therefore, overall, the throughput of HSTCP is not as large as expected. ImTCP increases the window size quickly when the window size is small and decreases the speed when its transmission rate reaches the available bandwidth to avoid packet losses. Therefore, the throughput of ImTCP is better than the others.

Finally, we compare the throughput of ImTCP in full-speed mode with Reno TCP in

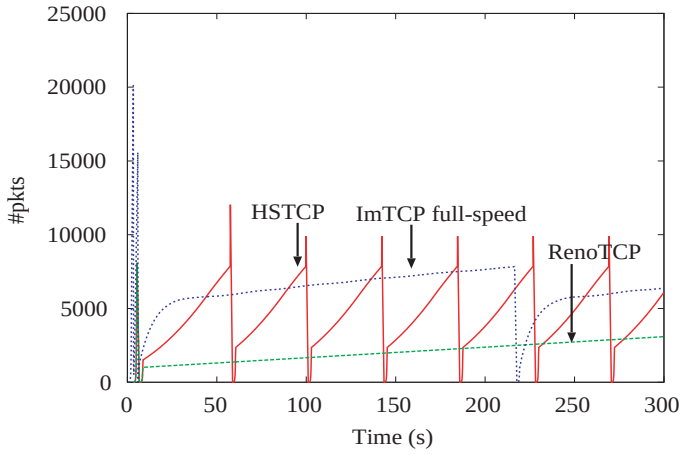


Figure 2.16: Comparision of TCP window sizes

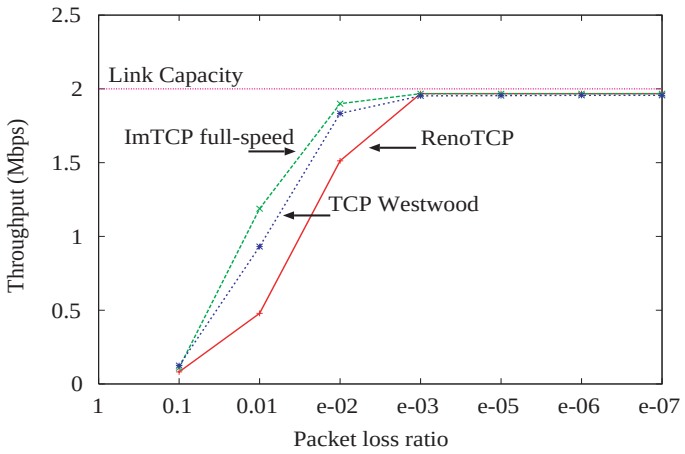


Figure 2.17: TCP throughputs in wireless network

a wireless network. We insert a 2 Mbps network link in the path between a TCP sender and TCP receiver to simulate a wireless link. We vary the packet loss rate of the network links and find that ImTCP can achieve a larger throughput than TCP Westwood and Reno TCP when the loss rate is high, as shown in Figure 2.17.

Parameter h is set to 100 in this case. When the packet loss rate is high, a higher value for parameter h can help ImTCP obtain higher available bandwidth. When the packet loss rate is low, the value of h should be low so that ImTCP will share bandwidth fairly with other traffic.

2.6 Conclusions

In this chapter, we introduced a method for measuring the available bandwidth in a network path between two end hosts using an active TCP connection. We first constructed a new measurement algorithm that uses a relatively small number of probe packets yet provides periodic measurement results quickly. We then applied the proposed algorithm to an active TCP connection and introduced ImTCP, a version of TCP that can measure the available bandwidth. We evaluated ImTCP through simulation experiments and found that the proposed measurement algorithm works well with no degradation of TCP data transmission speed. We also introduced examples of ImTCP special transmission modes.

The evaluation of ImTCP performance in real network environments as well as the Internet is performed in [15]. The source code of ImTCP implemented in BSD system is openly available at ImTCP homepage [53].

In future projects, we will consider a dynamic adjustment for the measurement frequency of ImTCP in order to make ImTCP compatible with traditional TCP. We will also develop new transmission modes for ImTCP as well as evaluate the performance of the modes introduced in this chapter.

Chapter 3

A Simultaneous Inline Measurement Mechanism for Capacity and Available Bandwidth

3.1 Introduction

The *capacity* of an end-to-end network path, which is considered to be the smallest capacity of network links along a path, is the maximum possible throughput that the network path can provide. Traffic may reach this maximum throughput when there is no other traffic along the path. The *available bandwidth* indicates the unused bandwidth of a network path, which is the maximum throughput that newly injected traffic may reach without affecting the existing traffic. The two bandwidth-related values are obviously important with respect to adaptive control of the network.

In many cases, both capacity and available bandwidth information are required at the same time. For example, network transport protocols should optimize link utilization according to available bandwidth. However, if a connection tends to fully using the available bandwidth, other connections that join the network later will find it difficult in obtaining bandwidth. Therefore, connections do not share the bandwidth fairly. In this case, if the connections are aware of the capacity, they can quickly change the used bandwidth so that the fairness with newly attended connections is maintained. One method of using capacity and available bandwidth information to optimize both bandwidth utilization and connection fairness for TCP is proposed in [14]. Another example is streaming multimedia data. Capacity information can be used for the decision of the size and the quality of the

multimedia data. Because available bandwidth is normally highly variable, it is better to use capacity information in this case. Available bandwidth information is then used to improve the performance of the transmission of the data. Besides, the billing policy of the Internet service provider may be based on both the capacity and the available bandwidth of the access link that they are providing to the customer.

Several passive and active measurement approaches exist for capacity or available bandwidth [26–29,33,37,42]. Although active approaches are preferred because of their accuracy and measurement speed, sending extra traffic onto the network is a disadvantage that is common to all active measurement tools. For example, Pathload [28] generates between 2.5 and 10 MB of probe traffic per measurement. The average per-measurement probe traffic generated by Spruce [27] is 300 KB. For routing in overlay networks, or adaptive control in transport protocols, these measurements may be repeated continuously and simultaneously from numerous end hosts. In such cases, the probes will create a large amount of traffic that may degrade the transmission of other data on the network, as well as the measurement accuracy itself.

We therefore propose an active measurement method that does not add probe traffic to the network. The proposed method, named *inline measurement*, uses the concept of *plugging* the new measurement mechanism into an active TCP connection. We have introduced ImTCP (Inline measurement TCP) in Chapter 2, a Reno-based TCP that deploys inline measurement for available bandwidth. The ImTCP sender not only observes the ACK packet arrival intervals in the same manner as TCP Westwood [16], but also actively adjusts the transmission interval of data packets, in the same way that active measurement tools use probe packets. When the corresponding ACK packets return, the sender utilizes the arrival intervals to calculate the measurement values.

The available bandwidth measurement algorithm for ImTCP is described in detail in Section 2.2. For each measurement, the ImTCP sender searches for the available bandwidth only within a given search range. The search range is a range of bandwidth that is expected to include the current available bandwidth and is calculated statistically from the previous measurement results. Without a search range, measurement tools (such as Pathload) must send packet in many transmission rates, from 0 Mbps to the upper limit of the physical bandwidth, to probe the network. The search range limits the range of the bandwidth that the measurement tool should probe, therefore, probe packets will not be sent in a high rate if not necessary. Thus, the measurements do not cause much effect on other traffic in the network. The search range also allows the number of packets for the measurement to be

kept small, so that measurement is still possible when the TCP window size is relatively small. The search range is divided into multiple sub-ranges of identical width of bandwidth. For each of the sub-ranges of the bandwidth, the sender transmits a group of TCP data packets (a packet stream), the transmission rate of which varies to cover the sub-range. The sender then determines whether an increasing trend exists in the transmission delay of packets in each stream when the echoed (ACK) packets arrive at the sender host. Delayed ACKs is supposed to be disabled at the receiver because the ImTCP sender will stop measurement and perform like a normal TCP sender if it finds out that many expected ACKs do not arrive. The increasing trend indicates that the transmission rate of the stream is larger than the current available bandwidth of the network path [28]. This fact allows the sender to infer the location of the available bandwidth in the search range. The simulation results show that the ImTCP sender can perform periodic measurements at short intervals, on the order of several RTTs and the measurements results reflect well the changes in the available bandwidth of the network.

In the present chapter, we introduce an inline measurement algorithm for capacity for ImTCP. The proposed algorithm utilizes the arrival intervals of the ACK packets of packet pairs (PPs) that are sent back-to-back. Due to the characteristic of ImTCP that PPs are available after the transmission of each measurement stream, the capacity measurements do not require any further changes in ImTCP. With the proposed method, ImTCP measures the capacity at the early stage of the connection and continues to collect data to improve the measurement accuracy during the transmission. We do not intend to develop a new capacity measurement tool that is better than the existing ones [22, 37, 39]. Rather, with the effort of reducing the load over the network caused by probe traffic, our main focus is on how we can extract capacity information from a TCP connection with the smallest change in TCP.

The main concept of the proposed capacity measurement algorithm in ImTCP is that the available bandwidth information, which can be yielded periodically due to the deployed available bandwidth measurement mechanism, is exploited. In the existing PP-based capacity measurement algorithm [22, 37, 39], the PPs that are cut into by other packets from cross traffic at the bottleneck link causes incorrect capacity estimation and are therefore eliminated from the data used in the calculation. However, in the proposed method, the available bandwidth information is used for estimation of the quantity of the cross traffic that cuts in PPs at the bottleneck link. The interval of the PPs becomes usable for the capacity measurement, which enables ImTCP to collect more information from PPs so

that faster and more accurate measurements can be expected. The proposed algorithm also uses statistical analysis to calculate the confidence interval of the delivered results.

Through simulation validations, we show that ImTCP can deliver capacity measurement results quickly, independent of the characteristics of the network. In addition, we find that the capacity measurement algorithm works well in extremely high-load networks, in which current measurement algorithms do not work well; ImTCP can deliver results with small errors even when the network load is as high as 93% of the capacity.

The remainder of this chapter is organized as follows. In Section 3.2 we discuss PP-based measurement techniques used for inline measurement. In Section 3.3, we introduce the proposed measurement algorithm for network capacity. In Section 3.4, we evaluate its performance through simulation experiments. Finally, in Section 3.5, we present concluding remarks.

3.2 Packet-pair-based capacity measurement algorithms

Currently there are various approaches for measuring the capacity of an end-to-end network path [24, 25, 54–56]. Some of these approaches use packets of various size to probe the network and infer the network capacity from the difference in the transmission delays of packets of various sizes [54]. Other approaches use the probe packets in different TTLs (Time To Live) to measure all link bandwidth, rather than just the capacity of the bottleneck link [24,25,55,56]. However, the packet size in TCP is always set to path MTU, which is the maximum size a packet can have to avoid fragmentation. A change in packet size can therefore only be done by selecting smaller packets, which requires TCP to send more packets. Setting small TLL values to TCP packets in order to dropt them along the path causes packet retransmissions and reduction in TCP window size. Thus, changes in TCP data packet size or TLLs for the purpose of measurement may cause severe deterioration in the data transmission throughput of TCP so these approaches can not be used for inline measurement.

We found that only PP-based measurement can be used for inline measurement because no changes in packet size or TTL are required, whereas packets that are sent back-to-back can be created with the current ImTCP structure without requiring any changes.

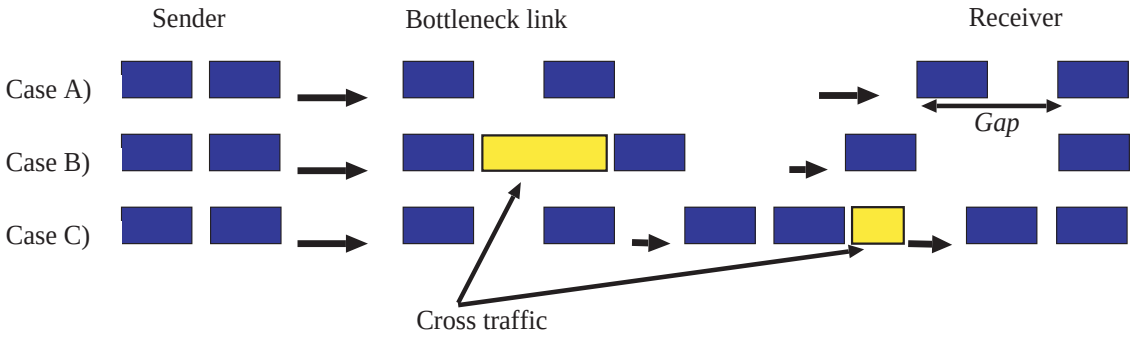


Figure 3.1: Three cases showing how the spacing between a pair of packets may change as the pair travels along a path.

3.2.1 Packet pair technique

The intuitive rationale of capacity measurement using PPs is that if two packets are sent close enough together in time to cause the packets to queue back-to-back at the bottleneck link, then the packets will arrive at the destination with the same spacing as when they left the bottleneck link [54]. The spacing is supposed to remain unchanged until the PPs reach the receiver, as shown in Case A of Figure 3.1, which is a variation of a figure taken from [22].

In this case, the capacity of the bottleneck link (C) can be calculated by the equation:

$$C = \frac{P}{Gap}, \tag{3.1}$$

where P is the size of the PPs, and Gap is the time spacing of the two packets when arriving at the receiver.

However, when a PP travels along the path, two more situations can occur. As shown by Case B in Figure 3.1, the two packets may be cut into by other packets from cross traffic at the bottleneck link. The result is that, the spacing between the two packets becomes larger than expected. In this case, Equation (3.1) leads to an under-estimation of the capacity. In another case, indicated by Case C in Figure 3.1, the PPs may pass back-to-back through the bottleneck link, but in a link downstream of the bottleneck link, the pairs again get in queue, and the spacing between the two packets is shortened. In this case, Equation (3.1) leads to over-estimation.

Current PP-based measurement techniques use only the PPs described in Case A to calculate capacity. These techniques have various mechanisms for determining the Case-A

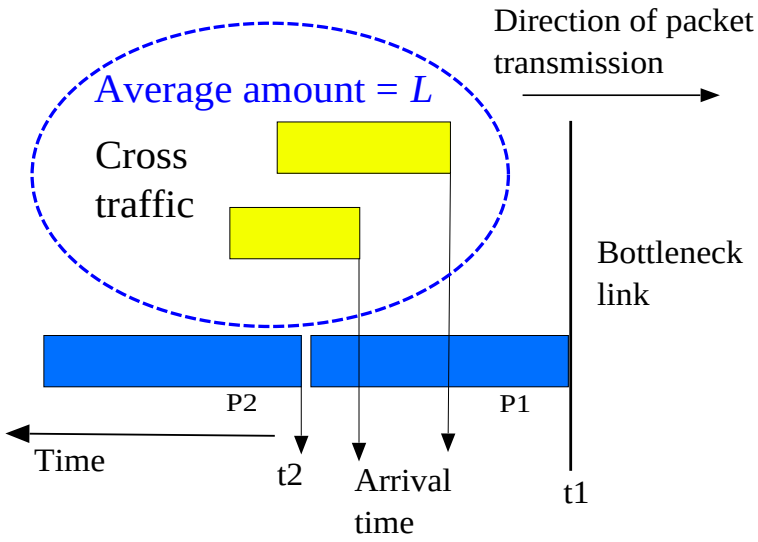


Figure 3.2: Arrival time at the bottleneck link of PPs and cross traffic

PPs from all of the received PPs. Some tools assume a high frequency of appearance of Case-A PPs and so search for these PPs from a frequency histogram (Pathrate [39]) or a weighting function (Nettimer [22]). CapProbe [37] repeatedly sends PPs until it discovers a Case-A PP, based on the transmission delay of the packets.

When the network path is almost empty, Case-A PPs may appear with the highest frequency. However, when other traffic appears in the network, there is a high probability that the cross traffic on the tight link (the link having smallest available bandwidth) stretches the PPs so that their intervals become large; the PPs then become Case-B. Case-C PPs do also exist, but some probing results from the Internet in [39] show that they are much fewer than Case-B ones. In this case, because Case-B PPs occur more often, CapProbe will spend an extremely long time for capacity searching, and Pathrate and Nettimer will deliver incorrect estimations.

Unlike those existing techniques, we propose a new technique by which to calculate capacity that can use both Case-A PPs and Case-B PPs. This is possible because of the available bandwidth information that is available in ImTCP.

3.2.2 Capacity calculation

Let us consider the timing of the arrival at the bottleneck link of a PP (Figure 3.2). We assume that the first packet arrives at t_1 and the second packet arrives at t_2. During

the interval from t_1 to t_2, packets from other traffic may arrive at the bottleneck link. The second packet (P2) must wait in the queue for the processing of the cross packets. Therefore, the time spacing (Gap) of the PP after leaving the bottleneck link is the total of the queuing time and the processing time of the second packet. That is:

$$Gap = \frac{P + L}{C}, \tag{3.2}$$

where L is the amount of the cross traffic that arrives at the bottleneck link during the interval (t_1, t_2). Supposing that the bottleneck link of a network path is the link having the smallest available bandwidth, we can then calculate the total transmission rate of the cross traffic at the bottleneck link as: $C - A$, where A is the current available bandwidth. Let δ be the time spacing of the PP upon arrival at the bottleneck link ($\delta = t_2 - t_1$). Then, the average value of L is:

$$L = \delta(C - A), \tag{3.3}$$

from Equations (3.2) and (3.3), we can write:

$$C = \frac{P + \delta(C - A)}{Gap},$$

or

$$C = \frac{P - \delta \cdot A}{Gap - \delta}. \tag{3.4}$$

Equation (3.4) enables the calculation of capacity from the PPs for both Case A and Case B. In the next section, we propose the new capacity calculation algorithm based on the equation.

3.3 Inline measurement algorithm for capacity

3.3.1 Overview

We introduce an inline PP-based measurement algorithm for capacity that utilizes available bandwidth to improve the measurement accuracy. The available bandwidth information can be used because an inline packet stream-based measurement mechanism for it already exists in ImTCP. Some existing available bandwidth measurement tools, such as IGI/PTR [29], take the reverse approach, that is obtaining capacity information first, then using it together with PP probing results to find available bandwidth. Moreover,

TOPP [42] measures both available bandwidth and capacity at the same time using PPs. However, as shown in recent experiments in real networks [44], measuring available bandwidth with packet streams is more valid than using packet pairs. Therefore, we think that the approach that we take in ImTCP is better.

The proposed capacity measurement mechanism has the following characteristics:

- The mechanism does not require any change in the current structure of ImTCP. Therefore, it does not affect the data transmission performance of ImTCP.

- The measurement starts and is able to provide results at the early stage of the connection. Unlike other methods such as the work by Hoe [34], the measurement also continues during the transmission. When the connection lasts for a long time, the measurement exploits the accumulated data to improve its accuracy.

3.3.2 Implementation of packet pairs in ImTCP

As introduced in Chapter 2, a measurement program is inserted into the sender program of TCP Reno to create an ImTCP sender. The measurement program is located at the bottom of the TCP layer, as shown in Figure 2.3. When a new data packet is generated at the TCP layer and is ready to be transmitted, the packet is stored in an intermediate FIFO buffer. The measurement program waits until the number of packets in the intermediate buffer becomes sufficient and then decides the time at which to send the packets in the buffer in order to create measurement streams. When no measurement stream is needed, the program immediately passes all of the data packets to the IP layer. In the previous version of ImTCP, we decided that the program forms and sends one measurement stream for the available bandwidth in each RTT in order to maintain fairness with respect to traditional TCP Reno.

During the transmission of a measurement stream, which includes five packets, there is a high probability that more than two packets arrive at and are stored in the intermediate FIFO buffer. Making use of the fact that after the transmission of a stream, ImTCP sends all stored packets in a bursty fashion, the capacity measurement program considers the first two packets in the burst as a PP to perform the measurement. Thus, there is no effect on the performance of ImTCP by introducing the capacity measurement mechanism. The creation of PPs is illustrated in Figure 3.3.

In ImTCP, 2–4 measurement streams are required in order to determine the available

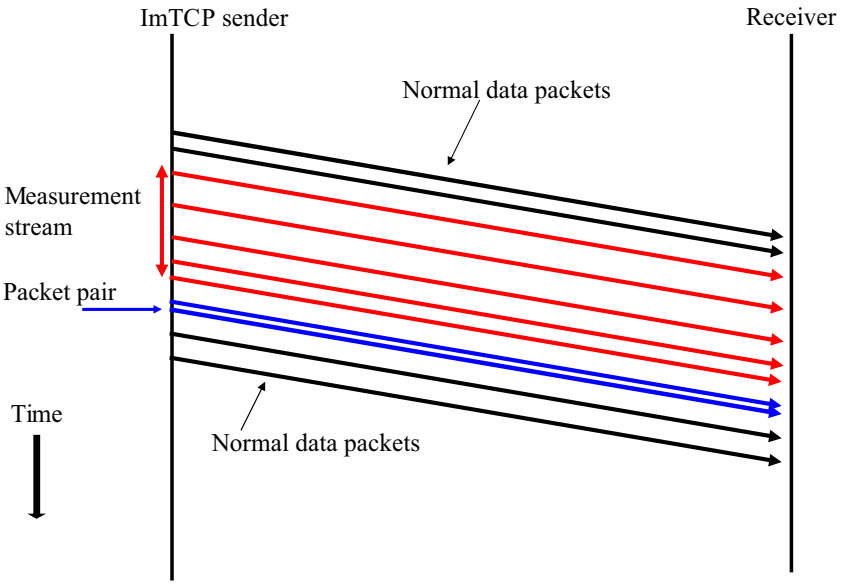


Figure 3.3: Creation of PPs in ImTCP

bandwidth. As mentioned above, each PP is formed and transmitted after each measurement stream. Therefore, 2–4 results for PPs can be obtained during the interval of two consecutive measurement results for available bandwidth.

3.3.3 Proposed measurement algorithm

We next explain the procedure for determining the capacity from the measurement results of PPs using Figure 3.4. The procedure involves the following steps:

- Grouping of PPs: PPs that sent when the measured available bandwidth remains unchanged are placed in the same `group`. The average value of arrival interval of PPs in a group, denoted by $\overline{Gap}$, is then calculated. To obtain a good average value, the number of PPs in each group should be enough large i.e. larger than or equal to 3, as determined herein. Therefore, after grouping, a group having only one or two PPs will be merged with the group that is collected right after that.

- Calculation: Based on the $\overline{Gap}$ value of a group, a `sample` of capacity is calculated using the following functions. If $\frac{A}{P/\delta} > \lambda$ we use

$$C = \frac{P}{\overline{Gap}}, \tag{3.5}$$

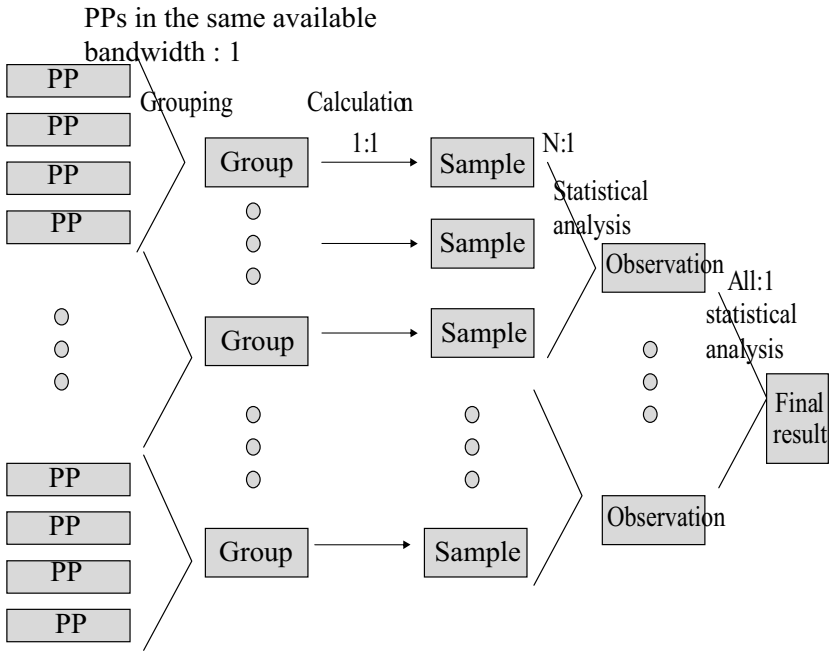


Figure 3.4: Proposed algorithm for inline capacity measurement

otherwise, we use

$$C = \frac{P - \delta \cdot A}{\overline{Gap} - \delta}, \tag{3.6}$$

where λ is the threshold showing the relation between the available bandwidth and the rate of the PPs upon arriving at the bottleneck link, that is defined as P/δ . We assume that the links before the bottleneck link do not have a noticeable effect on the time space, so that δ is approximated by the time interval in which the sender sends the packets. When the available bandwidth is approximately equivalent to the rate of the PPs upon arriving at the bottleneck link, which is considered as $\frac{A}{P/\delta} > \lambda$, the packets may pass through the link without being cut into by other packets (Case A). In this case, Equation (3.5) (based on Equation (1)) is used. On the other hand, since when the arrival rate of the PPs is much higher than the available bandwidth, which is considered as $\frac{A}{P/\delta} \leq \lambda$, the probability is high that the PP is a Case-B PP, Equation (3.6) (based on Equation (3.4)) is used. The changes in δ before the PP arriving at the bottleneck link make the calculation for sample of capacity using Equation (3.4) incorrect. However, we believe that the changes are small and do not occur so often. The task of grouping N `samples` in the next step of the algorithm is is an effort to reduce the effects of the changes.

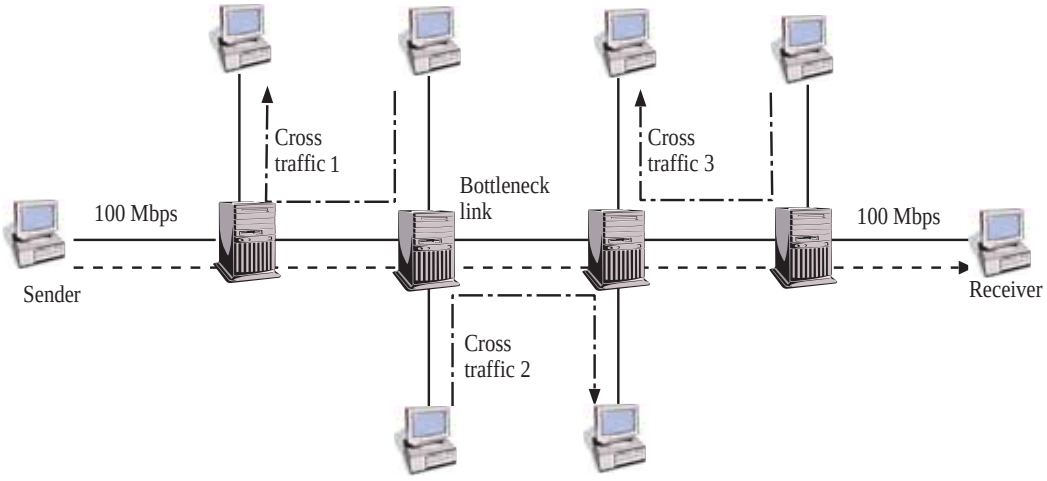


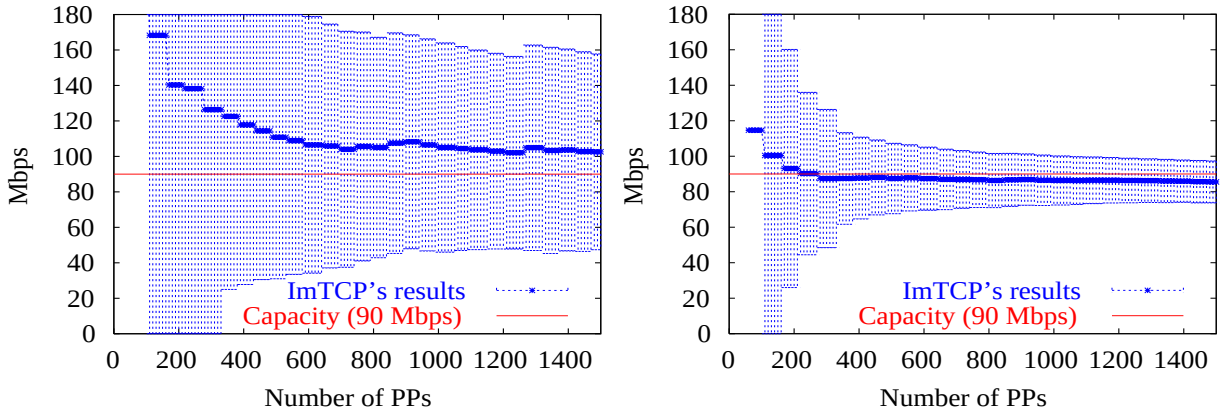
Figure 3.5: Simulation topology for evaluation of capacity measurement

- Statistical analysis:
  - We form `observations`, each of which is the average value of N `samples`. N should be large enough so that each `observation` has high accuracy. But when N is too large, the time required to finish an `observation` is long. This means that the proposed algorithm can not deliver the measurement results quickly. In the present paper, based on empirical experiments, we recommend $N = 10$.
  - The average value of the `observations` are calculated as the `final result`. The 90% confidence interval is also calculated to show the degree of fluctuation of the capacity.

3.4 Simulation experiments

In this section, we examine the measurement results of the proposed capacity measurement algorithm through ns-2 [47] simulations. We also compare the proposed algorithm with two existing algorithms, CapProbe [37] and Pathrate [39]. We compare the algorithms in the scope of inline measurement, because we only focus on how to extract capacity information from a TCP connection without introducing any extra probe traffic in to the network.

We use the simulation topology shown in Figure 3.5. The transmission rate of cross traffic 1 is fixed to 5 Mbps and that of cross traffic 3 is fixed to 15 Mbps. The packet size distribution of cross traffic is set to the statistical results for the Internet traffic reported



(a) $\lambda = 0.9$ (b) $\lambda = 0.8$

Figure 3.6: Measurement results for ImTCP when cross traffic 2 is 5 Mbps

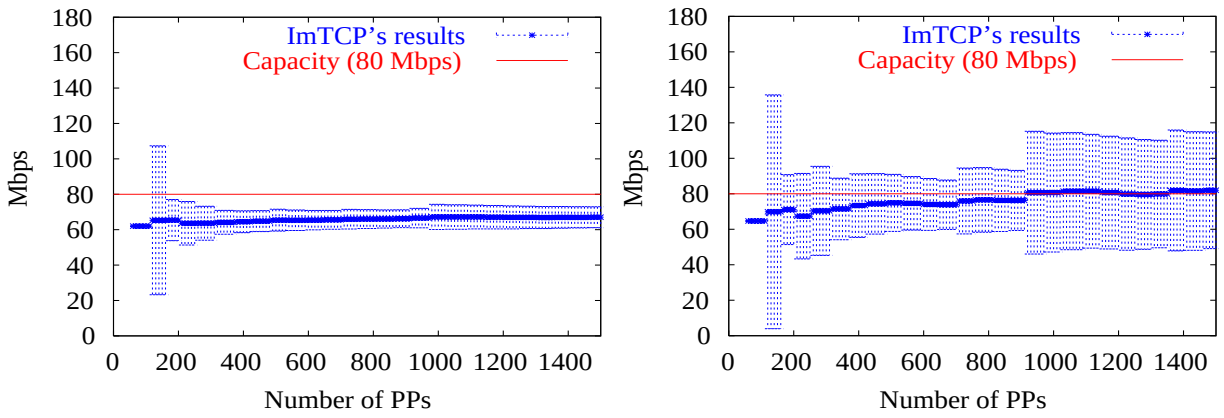
in [48], as shown in Table 2.1. This mixture has an average packet size of 404.5 bytes and has a correlation value of 0.999 when compared to realistic Internet traffic.

3.4.1 Effect of parameters

- Value of λ

  We set the bottleneck link capacity to 90 Mbps and the transmission rate of cross traffic 2 to 5 Mbps and examine the measurement results when $\lambda = 0.9$ (Figure 3.6(a)) and $\lambda = 0.8$ (Figure 3.6(b)). These figures show the changes of the capacity measurement results as the number of PPs sent for the measurement increases. The errors bars show the 90% confidence interval of the correspondent results. For the first some PPs, there is at most one `observation` is delivered therefore ImTCP can not calculate the confidence interval. In this case, the load on the bottleneck link is low, so the Equation (3.5) should normally be used. The setting $\lambda = 0.9$ does not allow the Equation (3.5) to be used so frequently and therefore leads to a bad result, that can be seen in large confidence intervals. We see that in this case $\lambda = 0.8$ (or lower than 0.8) is a better setting.

  We next show the case when the capacity is 80 Mbps and the rate of cross traffic 2 is set to 20 Mbps in Figure 3.7(a) ($\lambda = 0.5$) and 3.7(b) ($\lambda = 0.8$). In this case, the rate of the cross traffic is high, so Equation (3.6) should normally be used. Therefore,



(a) $\lambda = 0.5$ (b) $\lambda = 0.8$

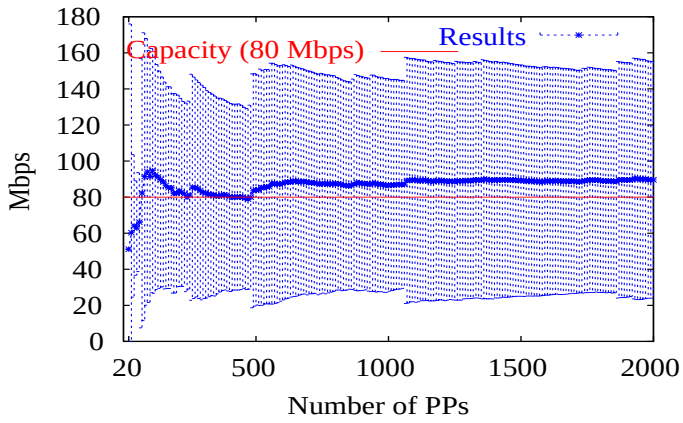
Figure 3.7: Measurement results for ImTCP when cross traffic 2 is 20 Mbps

a small value of λ, such as 0.5, gives incorrect results for the capacity, and, again, $\lambda = 0.8$ is a good setting in this case. Thus, $\lambda = 0.8$ is a suitable setting for the two cases above, and we found that it is a good setting in many other cases. Therefore, in the following simulations, we used $\lambda = 0.8$.

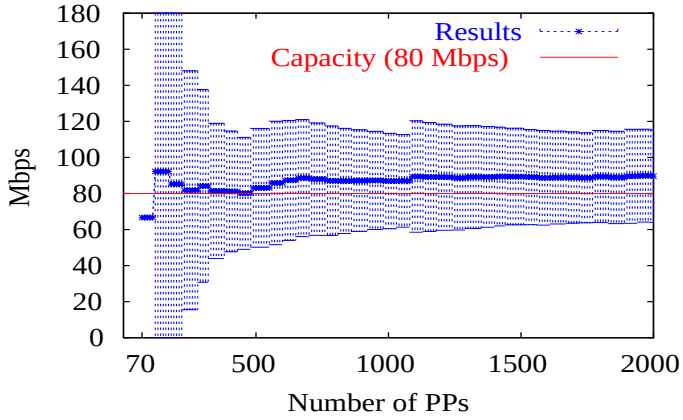
In general, for longer connections, because a larger number of PPs are sent, ImTCP's results approach nearer to the right value. However, we can see that the measurement results of ImTCP are sometimes not exactly the right value (for example results in Figure 3.6(b)) even when the connection lasts for along time. The reason for this is that, we suppose that the amount of the traffic that cut in every PPs is the average value of that ($L = \delta(C - A)$), but the amount of traffic that cut in a certain PP is sometimes too large or too small in comparison with L. In these cases, the Sample calculated from these outstanding values (using Equation (3.6)) is far from the right value of Capacity, this leads to a slight inaccuracy in the final result of ImTCP.

- Value of N

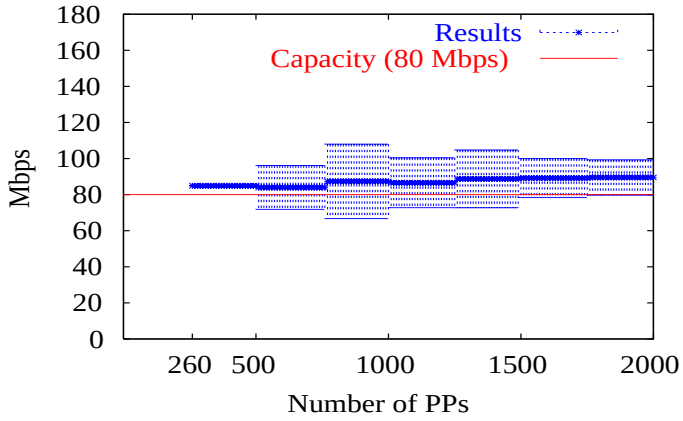
  N is the number of samples to form an `observation`. We set the bottleneck link capacity to 80 Mbps and the rate of cross traffic 2 to 40 Mbps. Figures 3.8(a), 3.8(b) and 3.8(c) show the measurement results when N is set to 1, 10 and 50, respectively. We can see that in Figure 3.8(a), the results are yielded after only 20 PPs are sent, while in Figure 3.8(b), the number is 70, and in Figure 3.8(c), it is 260. The large



(a) $N = 1$



(b) $N = 10$



(c) $N = 50$

Figure 3.8: Measurement results for the proposed algorithm when N changes

confidence interval in Figure 3.8(a) indicates that a small value of N ($N = 1$) is not suitable. On the other hand, Figure 3.8(c) indicates that a value of N ($N = 50$), that is too large, is unsuitable as well, because in this case the time required for the results to be yielded is long. Figure 3.8(b) shows the results with the proposed setting ($N = 10$), which can provide faster and better results. Unlike the results in Figure 3.6 and 3.7, the measurement results in Figure 3.8(a), 3.8(b) and 3.8(c) are all accurate from the first values. We can not expect the results to be better, as we have explained in Section 4.1 Therefore we could not see an improvement in the measurement accuracy in the figures, as the number of PPs increases.

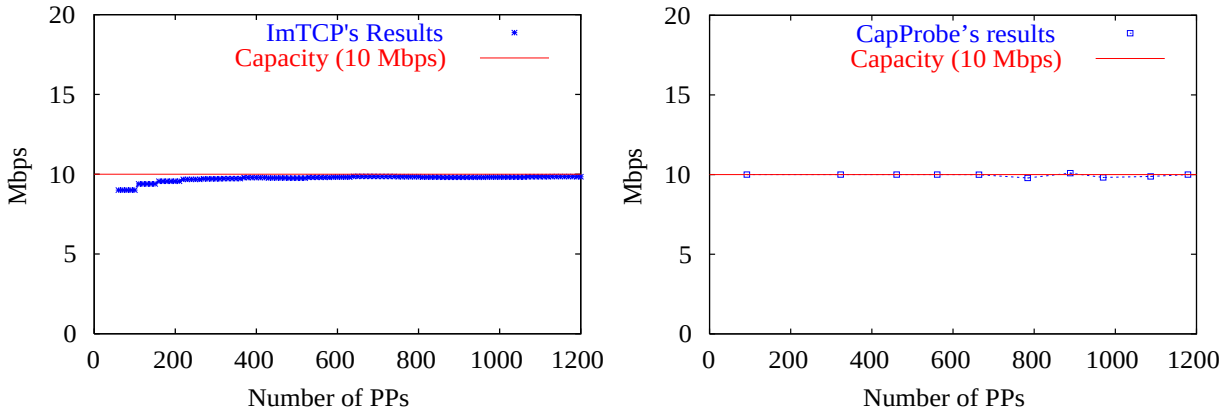
We use $N = 10$ for the following simulations. In fact, N must be set by applications or programs for that the measurement results are collected, depending on its uses. For example, if the application needs the measurement results to be updated in short intervals, it may choose a small value for N.

3.4.2 Comparison with CapProbe

We implement the CapProbe algorithm in TCP in order to compare the performance with the greatest possible impartiality. The difference from the original CapProbe algorithm proposed in [37] is that the packet size remains unchanged over the "runs" in the algorithm, because in TCP connections, changing the data packet size may have a bad effect on the TCP performance. The restriction on the packet size may be the reason for the poor performance of CapProbe in the following simulations. This means that CapProbe is not suitable for inline measurement.

Small capacity, low network load scenario

The capacity is set to 10 Mbps, and the rate of cross traffic 2 is set to 4 Mbps. Figures 3.9(a) and 3.9(b) show the measurement results for the proposed algorithm and CapProbe, respectively. Both of the measurement results are good. Moreover, we can see that the results obtained by CapProbe have high accuracy, because when CapProbe successfully finds the PP in Case A, the capacity can be calculated exactly. Another advantage of CapProbe is that, compared with the proposed algorithm, CapProbe is simple because it requires no complicated calculations. Howerver, CapProbe only delivers a measurement result after sending a large number of PPs. Table 3.1 shows the number of PPs sent until the proposed algorithm and CapProbe deliver the first measurement result. Here, the



(a) ImTCP

(b) CapProbe

Figure 3.9: Measurement results for ImTCP and CapProbe in small capacity, low network load scenario

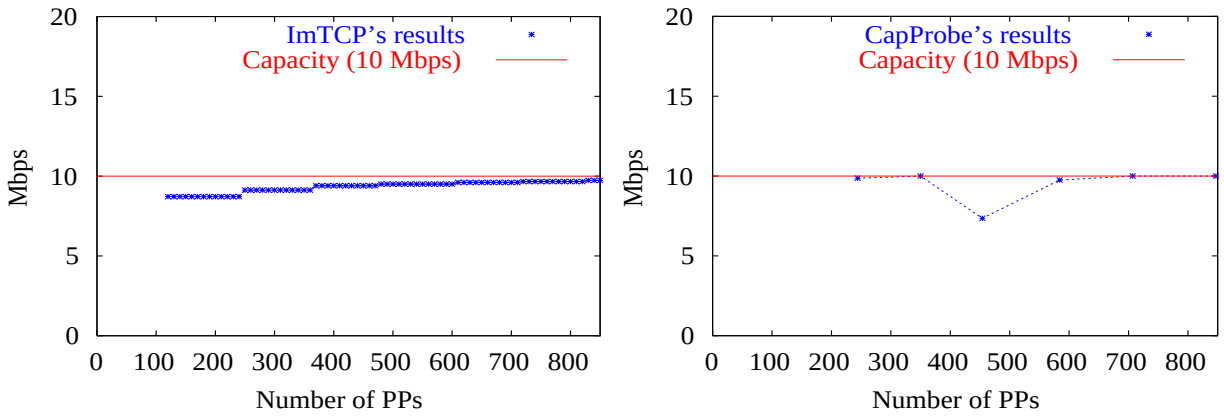
Table 3.1: Number of PPs required for the first measurement of ImTCP and CapProbe

Capacity (Mbps)	Cross traffic 2 (Mbps)	Proposed Alg. (PPs)	CapProbe (PPs)
10	1	60	87
10	2	60	85
10	4	60	92
10	5	60	159

capacity of the bottleneck is kept unchanged while the cross traffic 2 is varied from 1 Mbps to 5 Mbps. The table shows that, CapProbe only delivers a measurement result after 85 PPs or more are sent. The required number of PP is larger as the network load increases. In contrast, the proposed algorithm delivers good measurement results faster, after sending 60 PPs.

Small capacity, high network load scenario

We next change the cross traffic 2 to 9 Mbps to form a high network load environment. In this case, ImTCP still delivers good measurement results, as shown in Figure 3.10(a). On the other hand, CapProbe, as can be seen in Figure 3.10(b) introduces fewer results. It also delivers one incorrect measurement result. The reason for this is that, when the bottleneck link is crowded, many PPs are cut into by cross traffic so most of PPs are in



(a) ImTCP

(b) CapProbe

Figure 3.10: Measurement results for ImTCP and CapProbe in small capacity, high network load scenario

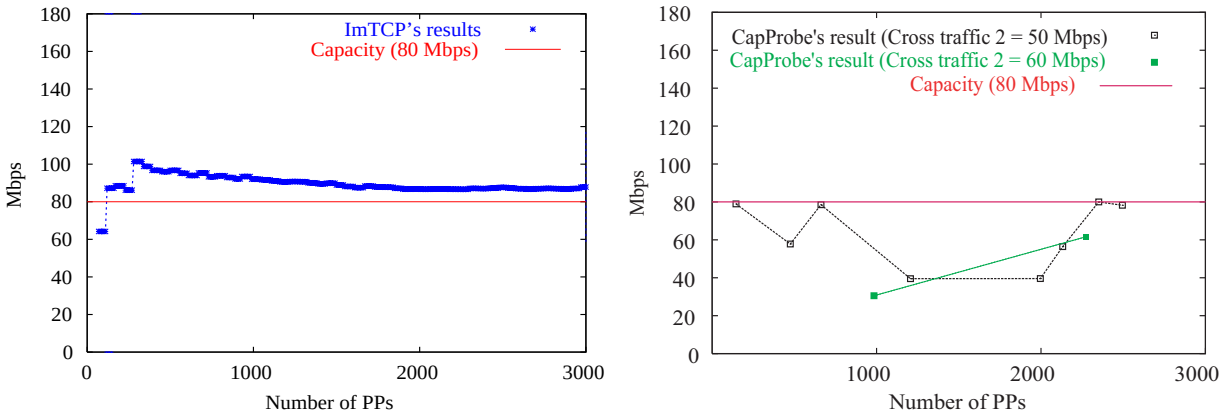
Case-B. It is easy for CapProbe to mistake a Case-A PP for a Case-B PP.

Large capacity, high network load scenario

The capacity is set to 80 Mbps, and the rate of cross traffic 2 is set to 60 Mbps. In a network with such a heavy load, the proposed algorithm can perform well (Figure 3.11(a)), whereas CapProbe can not deliver accurate results (Figure 3.11(b)), because, in this case, most of the PPs are cut into by other traffic so there are few Case-A PPs. In Figure 3.11(b) we also show the measurement results of CapProbe when the cross traffic 2 is decreased to 50 Mbps. These measurement results are still far from the correct value. We believe that CapPobe will perform better if the size of PPs is adapted appropriately, instead of being unchanged in the simulations, but changing packet size is not suitable with inline measurement.

3.4.3 Comparison with Pathrate

In order to accommodate the Pathrate algorithm into TCP, we use the interval of PPs delivered in ImTCP to form the histogram to be used in Pathrate. Pathrate also requires the measurement results of packet trains, referred to as the Average Dispersion Rate (ADR) in the Pathrate algorithm [39]. However, integrating the packet train into TCP is difficult because this has an adverse effect on the performance of TCP. Therefore, we perform the



(a) ImTCP

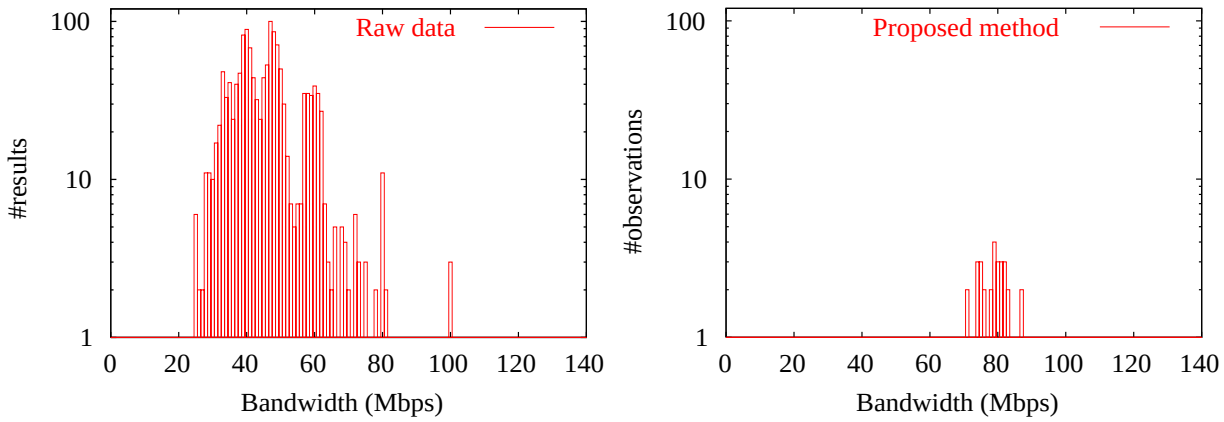
(b) CapProbe

Figure 3.11: Measurement results for ImTCP and CapProbe in high network load scenario

packet train measurement separately from TCP connection, in the same environment as that in the simulation with the ImTCP connection. The result of ADR is then used to find the measurement result for Pathrate.

We use the same network topology as that for the above-described simulations. The capacity is set to 80 Mbps and the transmission rate of cross traffic 2 is variable. We show the case when the cross traffic contains mainly packets of small size, by randomly varying the packet sizes of the cross traffic within the range of 400 to 600 B, because in this environment the difference between the proposed algorithm and Pathrate appears clearly. The performance of the proposed program in this environment is also examined, and the measurement results are listed in Table 3.2. In this case, since most PPs are cut into by cross traffic packets, Pathrate should not work very well. On the other hand, the proposed can yields good measurement independent on the value of cross traffic. ImTCP always delivers results with the relative error smaller 2% of the capacity even when cross traffic 2 is 75 Mbps (the network load is 93% of the capacity). However, when the cross traffic is small, (cross traffic 2 is 10 Mb/s), many PPs are not stretched at the bottleneck link so they do not become Case-B PPs. Instead, they become Case-C PPs due to the effect of cross traffic 3. Because the number of Case-B PPs decreases, the measurement algorithm introduces larger confidence intervals.

We explain in detail the respective behaviors of these two algorithms in Figures 3.12(a) and 3.12(b). In Figure 3.12(a), the "Raw data" histogram indicates the measurement



(a) Data collected for Pathrate algorithm.

(b) `Observation` value calculated by the proposed algorithm

Figure 3.12: Histograms collected by Pathrate and ImTCP in heavy load network

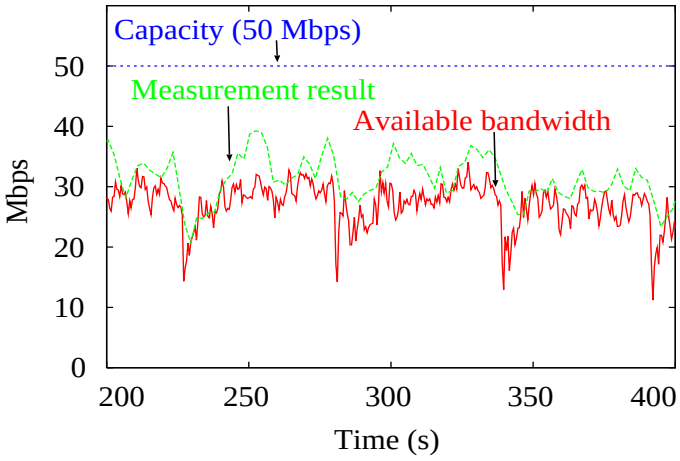
Table 3.2: Measurement results of ImTCP and Pathrate when cross traffic 2 changes

Cross traffic 2 (Mbps)	ImTCP's results (90% confidence interval)	Pathrate
75	79.35 (18.26)	49.00
60	80.24 (23.03)	48.00
40	78.32 (26.04)	80.00
10	81.57 (46.98)	80.00

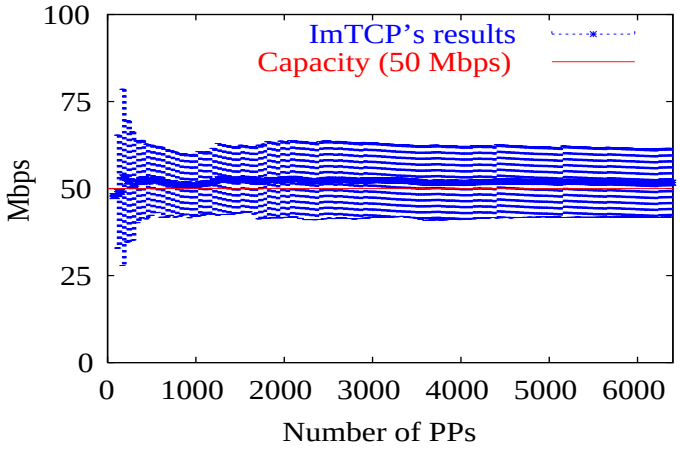
results calculated using Equation (1) that are used in Pathrate, and in Figure 3.12(b), the "Proposed method" histogram shows the `observation` results obtained using proposed algorithm, when the cross traffic 2's rate is 75 Mbps. In this case, Pathrate fails to deliver good measurement results because in this case number of Case-A PPs are fewer than Case-B PPs. This can be seen in some high peaks near 50 Mbps (while the correct value of capacity is 80 Mbps) in Figure 3.12(a). In contrast, the proposed algorithm can deliver good results, because the `observation` values always concentrate at the correct value of capacity, regardless of the network load.

3.4.4 Measurement in Web traffic environment

We finally investigate the measurement results for ImTCP in the network model depicted in Figure 3.5 with the cross traffic 1 and 3 turn off. Cross traffic 2 is now changed to



(a) Available bandwidth measurement results for ImTCP



(b) Capacity measurement results for ImTCP

Figure 3.13: Measurement in Web traffic environment

Web traffic involving a large number of active Web document accesses. We use a Pareto distribution for the Web object size distribution. We use 1.2 as the Pareto shape parameter with 12 Kbytes as the average object size. The number of objects in a Web page is eight. The capacity of the bottleneck link is set to 50 Mbps.

Figure 3.13(a) shows the measurement results for available bandwidth given by ImTCP. Also shown are the correct values of available bandwidth. We can see that the available bandwidth fluctuates frequently, because of the changes in the number of TCP connections in the crossing Web traffic. Figure 3.13(b) shows the measurement results for capacity also introduced by ImTCP in this case. The results are always approximately 50 Mbps, the correct value. The figure confirms that, even when the available bandwidth fluctuates frequently, ImTCP can deliver good measurement results for capacity.

3.5 Conclusions

In this chapter, we have proposed a new capacity measurement technique that is suitable for use in TCP connections. In contrast to existing techniques, the proposed mechanism uses available bandwidth information that is available in ImTCP, which enables the utilization of packet pairs that can not be used in existing techniques to calculate the capacity. The simulation results have shown that, the proposed technique can deliver measurement results quickly, even for a heavily loaded network, in which other techniques do not work well.

As our future works, we are currently implementing ImTCP using the proposed technique on a FreeBSD system. We will also consider a bandwidth measurement algorithm that can be deployed at the TCP receiver.

Chapter 4

Inline Bandwidth Measurement Techniques for Gigabit Networks

4.1 Introduction

Current efforts to improve TCP performance over high-speed networks can be divided into two main approaches. The first one changes the AIMD (Additive-Increase-Multiplicative-Decrease) congestion control of Reno TCP toward a faster increase of window size when there is no congestion and a smaller decrease when packet loss occurs. The representatives of this approach are HighSpeed TCP [52], Scalable TCP [57] and BIC (Binary Increase Congestion) TCP [58]. They can perform better than Reno/NewReno TCP in high-speed networks. However, these TCP versions do not have optimal speed for increasing the window size for all network environments because they still rely on network feedback, that is, a packet loss event, for adjustment of the speed. The packet loss event requires TCP to retransmit a certain amount of data, which can be large in high-speed networks. Therefore, these TCP variants cannot fully utilize the bandwidth of a network path. The second approach tries to avoid packet loss by deploying a delay-based congestion control mechanism. This approach uses queuing delay as a multi-bit congestion signal. The representative of this approach is FAST TCP [59]. The difficulty with this approach is matching the delay with the congestion signal because the relation is affected by the capacity of the network path. A mismatch may cause oscillation of the congestion window size, leading to serious performance degradation.

The effective way for transport protocols to achieve the desired throughput on a high-bandwidth and long-delay network path is to observe the available bandwidth/capacity

of an end-to-end network and adapt accordingly. Active measurement of the available bandwidth/capacity of an end-to-end network path has been vigorously investigated [26–29,33,37,42]. Compared with passive measurement, active measurement can deliver faster and more accurate results because the network can be investigated in detail using probe traffic. However, the sending of probe traffic is a drawback of active measurement. In many cases, probe traffic may interfere with data transmission in the network, as well as degrading the measurement itself.

We have proposed active measurement methods that overcomes the problem mentioned above in Chapters 2 and 3. We proposed the concept of *inline measurement*, that is, the idea of *plugging* the active measurement mechanism into an active TCP connection. This method has the advantage of requiring no extra traffic to be sent on the network, and provides fast and accurate measurement. When the sender transmits data packets, TCP adjusts the transmission rate of some packets, and considering arrival intervals of the corresponding ACK packets, the TCP sender estimates the available bandwidth/capacity of the network path between the sender an the receiver of the TCP connection.

Inline measurement results enable TCP to optimize its bandwidth utilization. Study in [14] has previously proposed a congestion control mechanism based on a logistic equation and a Lotka-Volterra competition model, by which TCP can fully utilize the bandwidth when the available bandwidth and capacity of the network path are provided. There also is a new TCP version that sets the upper limit of the congestion window based on the results of the inline network measurement [15]. In this case, TCP can provide background data transfer without affecting the foreground traffic, whereas previous methods cannot avoid network congestion essentially. Another study [60] shows the performance of TCP by using the capacity information for congestion control in the Internet. The performance of the TCP proposed by [60] is much better than that of Reno TCP, while the fairness with Reno TCP is still maintained. Besides its use as a part of congestion control mechanism, inline measurement can be used in many other cases, such as for routing or server selection in grid or overlay networks,

In this chapter, we focus on the problem: Can TCP still perform inline measurement in Gbps-level bandwidth? The problem arises because even stand-alone active measurement tools such as Pathload and CapProbe still cannot work well in a Gbps network for two reasons. First, measurement in fast networks requires short transmission intervals of the probe packets (for example, 12 μs for a 1-Gbps link and 1500-byte packet). However, regulating such short intervals causes a heavy CPU load. Second, network cards for high-speed

networks usually employ Interrupt Coalescence (IC) [40, 41], which rearranges the arrival intervals of packets and causes bursty transmission, and, therefore, algorithms utilizing packet arrival intervals do not work properly.

We introduce a new inline measurement mechanism that works well in high-speed networks. We call this ICIM (Interrupt Coalescence-aware Inline Measurement). Unlike other active measurement tools which observe the inter-intervals of the packets, ICIM adjusts the number of packets that are transmitted in a burst caused by IC and estimates the capacity and available bandwidth by checking whether the inter-intervals of the bursts of corresponding ACK packets are increased or not as they pass through the network. ICIM does not set the sending interval of the packets, so the overhead for packet spacing at the sender is eliminated.

The contributions of this chapter are as follows.

- We propose ICIM-abw, an inline measurement algorithm for available bandwidth that works in a gigabit network. The simulation results show that ICIM-abw performs accurate measurements in 1-Gbps and faster network. More than 94% of the results delivered by ICIM-abw has the relative errors smaller than 20% when measuring a 5-Gbps network path. The measurement frequency is also about 60 times higher than that of an existing algorithm. Meanwhile, TCP with ICIM can transmit data with the same performance as Reno TCP.

- We introduce ICIM-cap, an algorithm for inline measurement of the capacity in a gigabit network. Unlike current measurement algorithms, the ICIM-cap algorithm works well in extremely high-load networks. However, the algorithm can work well only when the tight link of the path, which has the smallest available bandwidth, is identical to the bottleneck link, which has the smallest capacity. We then discuss the range of errors if this supposition is not true, and show that the range of errors is small.

- We implement ICIM-abw in the FreeBSD system to test its performance in a laboratory environment. The measurement results show that ICIM-abw can work well in a real system.

The remainder of this chapter is organized as follows. In Section 4.2, we discuss the problems of bandwidth measurement in high-speed networks and look at a number of related studies. In Section 4.3, we introduce ICIM-abw and explain how to realize it in

Reno TCP. We then evaluate the performance of Reno TCP that is utilizing ICIM-abw. In Section 4.4, we introduce ICIM-cap and discuss the range of errors that may occur with this algorithm. In Section 4.5, we validate the performance of ICIM-abw in a real environment. Finally, in Section 4.6, we present concluding remarks.

4.2 Bandwidth measurement in high-speed networks

In this section, we discuss some of the difficulties encountered by existing active available bandwidth/capacity measurement tools in high-speed networks (1 Gbps or higher). We assume that the machines that run the measurement tools are general purpose machines, for example, a x86-based CPU machine with a normal operating system (OS), such as 4.4 BSD or Gnu/Linux (or similar). The problems mentioned here may not occur in high-performance machines that are designed especially for network bandwidth measurement.

4.2.1 Limitation of packet pacing in general-purpose machines

In current active measurement tools, probe packets must be sent at a rate higher than the bandwidth of the network path, otherwise the packet space will not be expanded and the tools will not be able to determine the bandwidth. When the bandwidth reaches 1 Gbps or higher, the transmission intervals of the probe packets must be 12 μs (for measuring 1-Gbps bandwidth) or smaller. As we discuss below, for a general-purpose machine, sending packets in such small intervals causes high CPU overhead.

For pacing packets, there are two approaches. The first is to continuously check the hardware clock (for example, using `gettimeofday()` in UNIX systems) and send the packets when the clock reaches a determined timing. In a Linux system with an x86-based CPU, one access of the hardware clock requires approximately 1.9 μs (in the FreeBSD system, one access requires 9 μs) [61]. The `write()` system call requires an average of 2 μs (in the case of a Pentium III CPU). Therefore, a Linux system can only send packets in intervals greater than $2 + 1.9 = 3.9$ μs. This means that, the system can measure the bandwidth up to 3 Gbps (for the case in which the probe packet size is 1,500 Bytes). However, in order to send packets at 3 Gbps, the CPU has to spend most of the time checking the hardware clock overhead. If the measurement is repeated continuously, then the CPU will not be able to process tasks from other applications. The system performance then will be

deteriorated. Thus, checking the hardware clock to send packets in a high-speed network is not a good approach.

The second approach is to register the packet sending program to an Interrupt Service Routine (ISR) of the hardware clock interrupt. In a general-purpose UNIX OS, the ISR `hardclock()` is provided for this purpose. In 4.4BSD OS and 2.4 LINUX kernels , the `hardclock()` system call is called by the interrupt of hardware clock every 0.01 s. In the 2.6 LINUX kernels, it is called every 0.001 s. However, with this low interrupt frequency, the program called by `hardclock()` can only send packets at the rate of up to 12 Mbps (assuming that the packet size is 1,500 Bytes). To obtain a higher interrupt frequency, a new interrupt schedule of the hardware clock can be implemented. However, one hardware interrupt (in 4.4 BSD OS) normally requires more than 1 μs [62]. If the packet transmission rate is 1 Gbps, then the sending interval is 12 μs. This means that, in this case, the overhead of the hardware interrupt is as high as 1/12 of the total working time of the CPU. In addition, a new interrupt schedule for the hardware clock requires many changes in the OS.

4.2.2 Effects of Interrupt Coalescence

Another reason for the difficulty in the task of measurement in high-speed networks is IC, which is deployed in most high-bandwidth Network Interface Cards (NICs). IC is a technique in which NICs group multiple packets that arrive in a short time interval and pass them to the OS in a single interrupt. IC reduces the CPU overhead when the arrival intervals of packets become small. Because the inter-arrival intervals of the packets observed by the kernel are changed, IC has an enormous impact on bandwidth measurement tools, in which the arrival intervals of packets are utilized for bandwidth estimation.

There are a number of types of timer setting in IC. For example, Intel Gigabit Ethernet Controllers [40] contains the following mechanisms for IC:

- Absolute timer: The absolute timer delays the assertion of an interrupt to allow the controller to collect additional interrupt events before delivering them to software.
- Packet timer: The packet timers are inactivity timers, triggering interrupts when the link has been idle for an appropriately long interval.
- Master timer for throttling all interrupt sources: An interrupt throttling mechanism is used to set an upper bound for the interrupt rate.

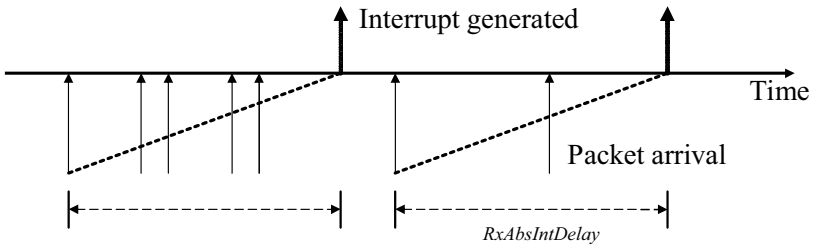


Figure 4.1: Receive absolute timer

Under sustained loads, the absolute timers will be the primary source of device interrupts [40]. We investigate the absolute timers in greater detail. There are two absolute timers. One is for transmit interrupts, and the other is for receive interrupts. Because transmit interrupts only inform the kernel as to the completion of packet sending, delays in transmit interrupts do not affect the real transmission intervals of the packets. In contrast, delays in receive interrupts change the intervals of all receiving packets observed by the kernel. As shown in Figure 4.1, the receive absolute timer starts to count down upon receipt of the first packet. Subsequent packets do not alter the countdown. Once the timer reaches zero, the controllers generate an interrupt to pass all of the packets to the OS in a bursty manner. The length of the timer is decided by the parameter *RxAbsIntDelay*, which is defaulted to 0.1312 ms in Intel Gigabit Ethernet Controllers. Thus, all packets that have time intervals smaller than *RxAbsIntDelay* will belong either to the same burst, in which case the time interval between the packets becomes zero, or to two successive bursts, in which case the time interval becomes *RxAbsIntDelay* or larger. Therefore, the software cannot detect packet intervals smaller than *RxAbsIntDelay*. With the default value of 0.1312 ms for *RxAbsIntDelay*, the software cannot perceive transmission rates larger than 100 Mbps (if the packet size is 1,500 Bytes).

Without IC, an OS interrupt occurs whenever a single packet arrives; this leads to a high CPU overhead when the system performs high speed data transmission. Therefore, we should not disable IC feature for the purpose of measurement. There are some studies that have discussed measuring bandwidth using the existing IC. For example, one study [61] suggests that in order to obtain the real arrival intervals of packets, the onboard timestamp of some network cards (for example SysKonnect GigE NIC [41]) should be used. However, the same study also concludes that this solution is not useful for general-purpose network measurement tools, because very few NICs have an onboard timer. Furthermore, using an onboard NIC timer requires modification of the device driver. This prevents the tool from

being easy to run on numerous systems.

Another study [63] reports that since the last packet in a burst formed by IC has the smallest delay in the NIC buffer, the intervals of the last packets in the bursts can be used for estimation of the available bandwidth, according to the Pathload [28] algorithm. However, because only a small part of stream is used for the measurement, the stream must be very long. This is not suitable in inline measurement, because making long measurement streams in TCP badly affects the TCP transmission performance.

4.2.3 Bursty packet transmission in TCP

The behavior of TCP when the network cards enable IC has been investigated in previous studies [62, 63], and IC has been shown to be detrimental to TCP self-clocking. IC causes the ACK packets to arrive at the sender in bursts, and this bursty arrival in turn causes bursty transmission of data packets and, subsequently, bursty transmission of ACK packets from the TCP receiver. According to one study [63], with IC, 65% of ACKs arrive with intervals of less than 1 μs, because they are delivered to the kernel with a single interrupt. Meanwhile, without IC, almost no ACK packets arrive with small intervals.

In the present chapter, we propose an algorithm that can exploit the burst of data packets in TCP under the effects of IC to measure the available bandwidth/capacity of the network path between TCP sender and receiver. The TCP sender adjusts the number of packets involved in a burst and checks whether the inter-intervals of the bursts of corresponding ACK packets are increased or not to investigate the bandwidth. ICIM can be employed into any version of TCP. Using previously reported results [63], ICIM first checks to see if the network card has IC enabled. If the IC is enabled, ICIM continues measurement based on the bursty transmission of TCP.

4.3 ICIM-abw: Interrupt Coalescence-aware inline measurement for available bandwidth

4.3.1 Packet burst-based measurement algorithm

The basic idea of measuring bandwidth larger than 1 Gbps is that, we consider a burst of packets as a big packet. Two consecutive big packets are then treated as a packet pair, of which the time interval in large enought so that general purpose operating systems can

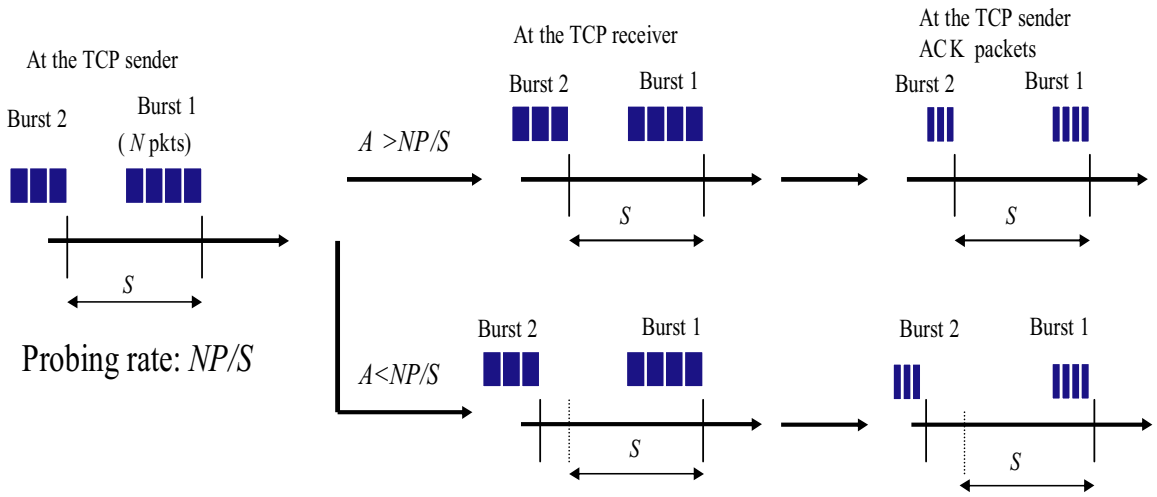


Figure 4.2: Packet burst-based available-bandwidth measurement principle

read exactly. The measurement algorithm is described below. Though the algorithm can work somehow when Delayed ACK in the TCP receiver is on, we recommend the Delayed ACK is off so that the algorithm can works properly. Because the absolute timer (described in Section 4.2) is the primary source of device interrupts in the high speed transmission, we assume that the NIC uses the absolute timer when receiving packets.

As shown in Figure 4.2, we consider the situation in which two bursts of packets are sent at the interval S. The number of packets in the first burst (Burst 1) is N. Assume that C is the capacity of the tight link. C_{Cross} is the average transmission rate of cross traffic over the tight link when the two bursts pass the link, and P is the packet size. We suppose that C_{Cross} is not changed much by the TCP connection performing inline measurement. Otherwise, the available bandwidth is not determined; the measurement becomes meaningless. Then, the amount of traffic that enters the tight link during the period from the point at which the first packet of Burst 1 reaches the link until the point at which the first packet of Burst 2 reaches the link will be the sum of the packets in Burst 1 and the cross traffic packets arriving in S, i.e., $C_{Cross} \cdot S + N \cdot P$. If the amount is larger than the transfer ability of the link during this period, considered to be $C \cdot S$, then Burst 2 will go to the buffer of the link. This results in a tendency for the interval between the two bursts to increase after leaving the tight link.

We can write that the burst interval will be increased if

$$C_{Cross} \cdot S + N \cdot P > C \cdot S, \tag{4.1}$$

or,

$$\frac{N \cdot P}{S} > C - C_{Cross}. \tag{4.2}$$

Note that $C - C_{Cross}$ is the available bandwidth (A) of the tight link. Therefore, Equation (4.1) becomes

$$\frac{N \cdot P}{S} > A. \tag{4.3}$$

Since we assume that the absolute timer is used, S is always larger than *RxAbsIntDelay*. Therefore, at the NIC of the TCP receiver, since the arrival interval of the two bursts are larger or equal to S, the two bursts are passed to the kernel in two different interrupts. The TCP receiver then sends the ACK of the two bursts in the same intervals to the sender TCP. Thus, by checking the arrival intervals of the corresponding ACK packets of the two bursts, the TCP sender can determine if $A > NP/S$. By sending numerous bursts with various values of NP/S (by changing N), we can search for the value of the available bandwidth A. This is the measurement principle of the proposed inline measurement mechanism. Note that the bursty transmission in TCP appears when IC is enables, as mentioned in Section 4.2.3. ICIM-abw performing in reasonable measurement intervals do increase the burstiness but the increase is not effect much the performance of TCP, as shown in the simulation results in Subsection 4.3.3.

4.3.2 ICIM-abw

ICIM-abw inherits the concept of the *search range* from the available bandwidth measurement algorithm in ImTCP described in Chapter 2. This is the idea of limiting the bandwidth measurement range using statistical information from previous measurement results rather than searching from 0 bps to the upper limit of the physical bandwidth for every measurement. By limiting the measurement range, we can keep the number of probe packets small.

At first, we explain how to search for the available bandwidth in a determined search range and then we present an overview of the measurement algorithm.

Assume that the search range for a measurement is (B_l, B_u). The algorithm then check k values in the range to determine which is nearest to the real available bandwidth. We use $k = 4$ in the following simulations. The k points are:

$$B_i = B_l + \frac{B_u - B_l}{k - 1}(i - 1),\ i = 1, ..., k.$$

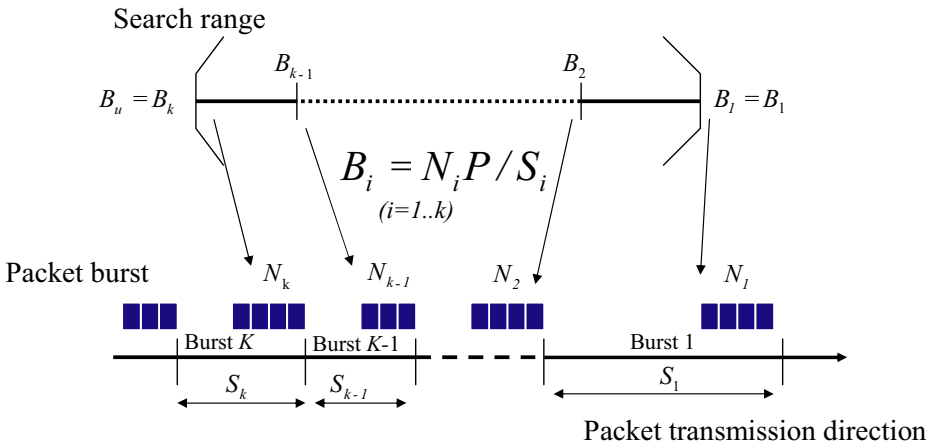


Figure 4.3: Probing a search range in ICIM-abw

The TCP sender then sends k consequence bursts and the number of packets are adjusted so that the probe rate of Burst i is B_i;

$$\frac{N_i \cdot P}{S_i} = B_i. \tag{4.4}$$

We illustrate the setting in Figure 4.3.

Realization of Equation (4.4) requires the following:

- The value of S_i is required at the timing of the transmission of Burst i.

  Infact, S_i is unknown until Burst $i+1$ is transmitted. But we need the value at the timing of the transmission of Burst i in order to guarantee Equation (4.4). We therefore estimate the value of S_i by assuming that the amount of data in Burst i is proportional to the length of the interval as follow:

$$S_i = \frac{N_i \cdot P}{T}, \tag{4.5}$$

  where T is the average throughput of TCP.

- In case the number of packets in Burst i is smaller than N_i, additional packets must be added to the burst so that the packet number becomes N_i.

  ICIM-abw utilizes a buffer located at the bottom of the TCP layer in order to store the packets temporarily before sending them to the IP layer, in the manner of ImTCP. ICIM-abw stores all of the packets of the burst that preceded Burst 1 in the buffer.

Packets are added to Burst i ($i = 1..k$) when necessary in order to maintain the desired number of packets (N_i) in these bursts.

ICIM-abw sends k bursts and checks the corresponding ACK of the bursts. If from burst number j, $j = 1..k$, the arrival interval of the bursts becomes larger, then B_j is considered to be the value of the available bandwidth in that measurement. Here, the burst interval is consider to become larger if the arrival interval is larger then λ times of the sending interval. We set λ to 1.01 in the following simulations.

ICIM-abw first checks whether IC is enabled for the network card. For the reasons explained in Section 4.2.3, ICIM-abw checks the arrival intervals of the ACK packets. If more than 50% of the intervals are less than 1 μs, then ICIM-abw decides that IC is enabled. If the IC is enabled, then ICIM-abw continues the following measurement steps. Otherwise, the measurement algorithm introduced in ImTCP is used.

The measurement algorithm of ICIM-abw is as follows:

1. Set the initial search range

   We set the initial search range as $(T, 2 \cdot T)$ where T is the throughput of TCP.

2. Search for the available bandwidth in the decided search range.

   ICIM-abw waits until the window size ($cwnd$) is larger than C_{min} (large enough to create bursts for measurement). We use $C_{min} = 50$ in the following simulations. Data packets are then sent in order to search the available bandwidth in the decided search range, as described above.

3. Add the new measurement result to the database and calculate the new search range.

   The measurement result in the last step is added to a dabatase of measurement results. We then calculate the new search range (B'_l, B'_u) from the database. We use the 95% confidential interval of the data stored in the database as the width of the next search range, and the current available bandwidth is used as the center of the search range. The search range is calculated as follows:

$$B'_l = R - \max\left(1.96\frac{V}{\sqrt{q}}, \frac{R}{10}\right), \tag{4.6}$$

$$B'_u = R + \max\left(1.96\frac{V}{\sqrt{q}}, \frac{R}{10}\right), \tag{4.7}$$

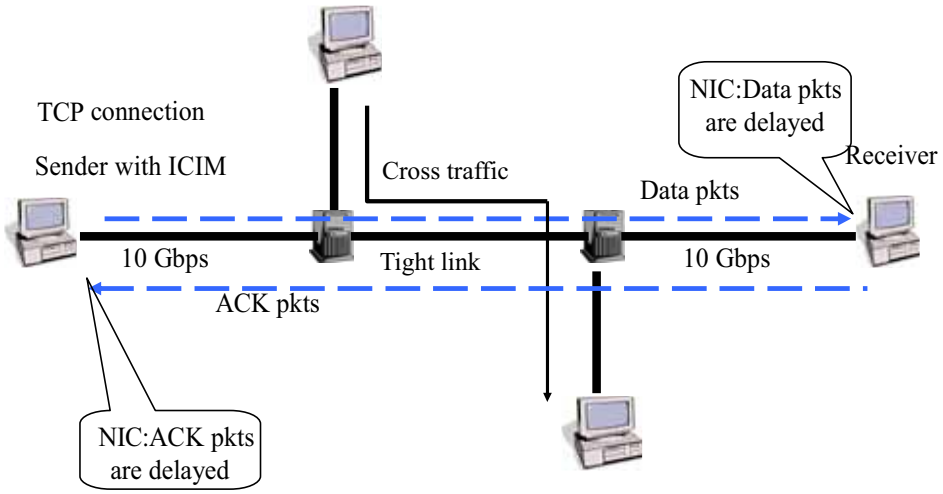


Figure 4.4: Simulation topology for evaluation of ICIM-abw

where R is the latest measurement result. V is the variance of stored values of the available bandwidth and q is the number of stored values. $R/10$ is a value that ensures that the search range does not become too small. Moreover, when measurement result in Step 3 falls to B_l (B_u), it is possible to consider that the network has changed greatly so that the real value of the available bandwidth is lower (higher) than the search range. In this case, we discard the accumulated measurement results because they become unreliable as statistic data and enlarge the search range (B_l, B_u) twice towards the lower (higher) direction to create (B'_l, B'_u).

4. Wait for Q seconds then return to Step 2 and start the next measurement. During the waiting time Q, TCP transmits packets in the normal manner. The waiting time is needed for the TCP transmission to return to the normal state after the packets store-and-forward process at Step 2.

4.3.3 Simulation experiments

Measurement results

We show the measurement results for ICIM-abw through ns-2 [47] simulations. We implement ICIM-abw via Reno TCP, the most popular version of TCP, and use the topology shown in Figure 4.4 for the simulation. The sender and receiver of TCP are connected through 10-Gbps access links and a tight link. The NICs of both the sender and receiver host employ IC with an absolute timer. The cross traffic on the tight link is made up of

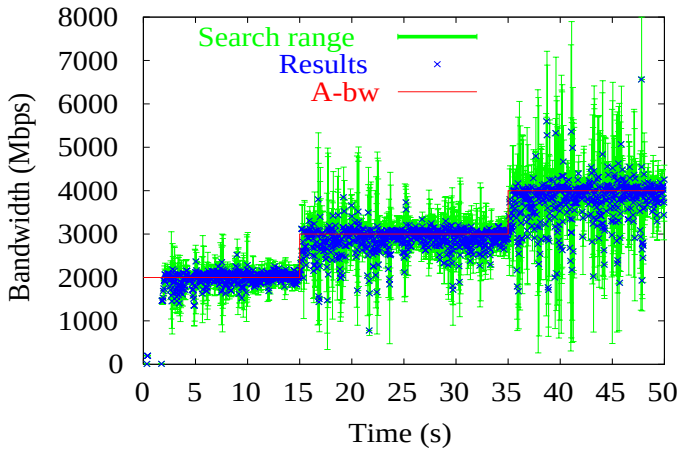
UDP flows in which various packet sizes are used, according to results monitored on the Internet [48], as shown in Table 2.1. The capacity of the tight link is 5 Gbps, and the available bandwidth (A-bw) is 2 Gbps (from 0 to 15 sec), 3 Gbps (from 15 to 35 sec) and 4 Gbps (from 35 to 50 sec).

Figures 4.5(a) and 4.5(b) show the measurement results for ICIM-abw when the interval between two measurements is set to one RTT or two RTTs, respectively. Also shown are the search ranges for each measurement. The search ranges, in most cases, successfully cover the correct value of the A-bw. Therefore, ICIM-abw can quickly detect the A-bw, even in such a high-speed network. When $Q = 1$, the throughput of TCP oscillates slightly, the estimation of the burst interval in Equation (4.5) becomes incorrect. Therefore, the probing rate of each Burst i may not be exactly equal to B_i (in Step 2 of Subsection 4.3.2). This leads to a large dispersion of the measurement results in Figure 4.5(a); in this case, for 9% of the measurements, the relative error is larger than 20%. When $Q = 2$, the TCP sender creates fewer packet bursts so that the measurement results are nearer to the correct value of the A-bw, as shown in Figure 4.5(b); only 6% of the measurements have relative errors higher than 20%. However, the measurement frequency (16.7 results/second) becomes half of that when $Q = 1$ (34.2 results/second).

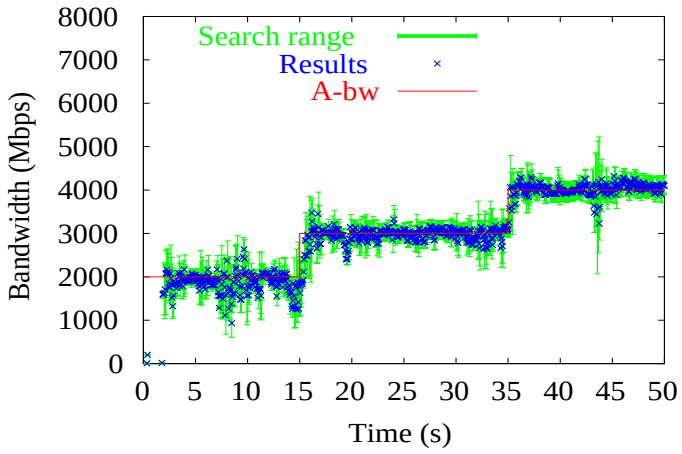
Comparison with IC-aware Pathload

We compare ICIM-abw with the only measurement tool we have found that can work in Gbps network. That is a version of Pathload that can detect and filter the effects of IC [63]. We call this version IC-aware Pathload. For the comparison between ICIM-abw and Pathload, the TCP sender and receivers are next replaced by the sender and receiver of Pathload. To make the measurement of Pathload faster, we set the starting probing rate to 200 Mbps (instead of the default setting of 1 Mbps). In addition, ω and χ are set to 200 Mbps and 150 Mbps, respectively, and the size of probing packets is set to 1,500 bytes.

The measurement results of IC-aware Pathload when the number of packets in a stream K is set to 160 are shown in Figure 4.6(a). Because the default value of *RxAbsIntDelay* used in NIC is 0.000132 (s) and the packet size is 1,500 bytes, the average number of packets in a burst is 22 when the A-bw is 2 Gbps, 33 when the A-bw is 3 Gbps and 44 when the A-bw is 4 Gbps. Therefore, when $K = 160$, there are approximately nine bursts in each stream when the A-bw is 2 Gbps. This means that Pathload has approximately nine packets (the last packet in the bursts) for measurement. The increasing trend in the stream in this case can be well determined so Pathload can deliver good measurement results. However,

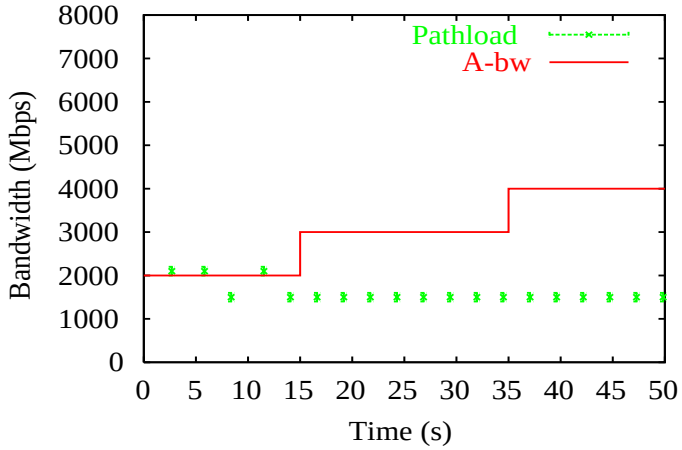


(a) Q=1 RTT

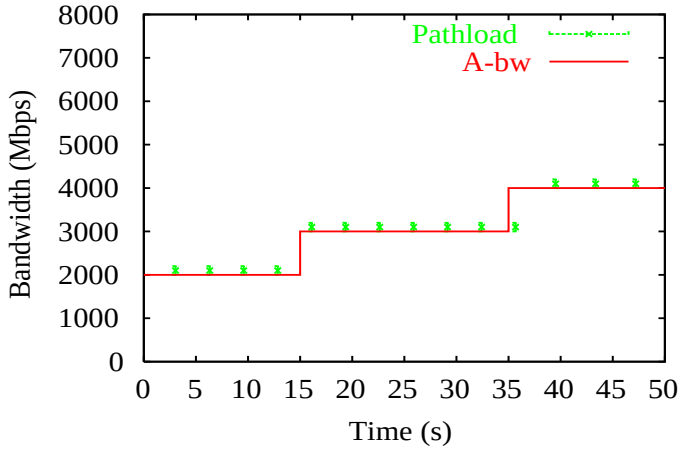


(b) Q=2 RTT

Figure 4.5: Measurement results for ICIM



(a) $K = 160$ packets



(b) $K = 200$ packets

Figure 4.6: Measurement results for IC-aware Pathload

when the A-bw becomes 3 Gbps or greater, the number of bursts becomes approximately six or fewer. Then, Pathload does not have enough packets to detect well the increasing trend in the stream. Therefore, as shown in Figure 4.6(a), Pathload fails to deliver good measurement results when the bandwidth is equal to or greater than 3 Gbps.

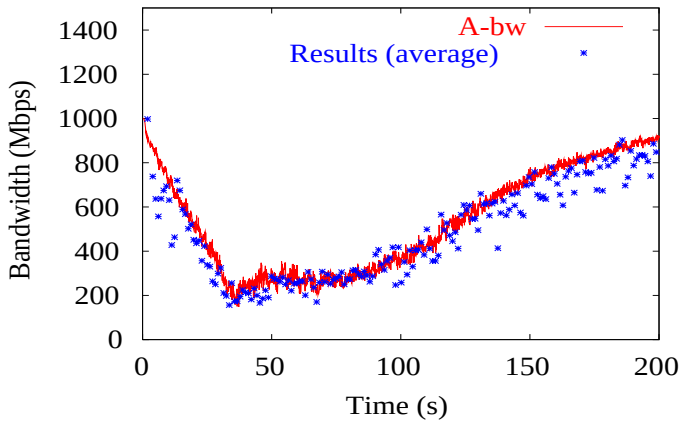
Figure 4.6(b) shows the measurement results of Pathload when K is set to 200. In this case, Pathload has a sufficient number of packets for detecting the increasing trend of streams. Therefore, the measurement results are correct. However, since Pathload searches for the A-bw from a low value, a long time is required to yield one result. The measurement frequency is only 0.28 results/second, which is 60 times smaller than that of ICIM-abw (with $Q = 2$ RTTs). Figure 4.6(b) shows that, if the A-bw changes during a measurement, Pathload may not detect the change well. At 15 seconds, the A-bw changes from 2 Gbps to 3 Gbps while Pathload is probing a rate smaller than 2 Gbps. When the probing rate reaches 2 Gbps, the A-bw is already changed, therefore Pathload can successfully detect the value of 3 Gbps. However, at 35 seconds, the probing rate of the ongoing measurement reaches 3 Gbps before the change in the A-bw from 3 to 4 Gbps, so Pathload assumes that the A-bw is smaller than or equal to 3 Gbps. Therefore, Pathload delivers a value of approximately 3 Gbps at the end of that measurement, which is far from the value of the A-bw at this timing.

Table 4.1 compares the number of packets used in the measurement of ICIM, and Pathload. ICIM-abw sends four bursts of packets for each measurement. The average number of total packets in four bursts is shown in the second column of the table. On the other hand, Pathload probes 8, 9 and 10 times for one measurement result when the A-bw is 2, 3 and 4 Gbps, respectively. Each probe requires 12 streams, the number of packets of which is 200. We can see that the number of packets used by ICIM-abw is less than one percent of that of Pathload.

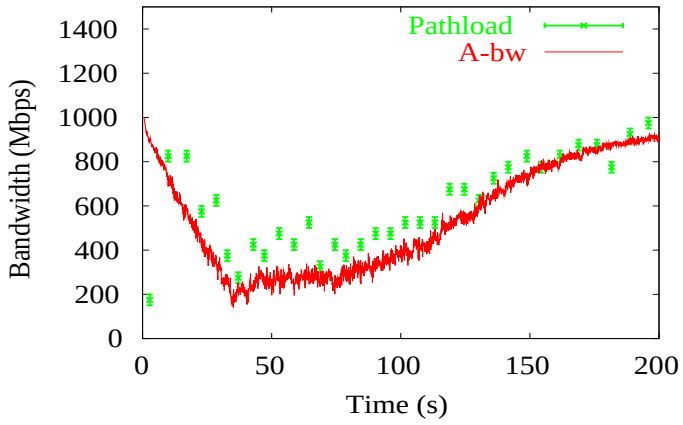
Figures 4.5 and 4.6 show that the measurement results of ICIM-abw have a larger dispersion compared to Pathload because, based on the nature of the algorithm, ICIM-abw cannot increase the length of each measurement burst to obtain high accuracy, as Pathload does. Instead, the accuracy can be improved by taking the exponential moving average in suitable intervals.

Measurement results in Web traffic environment

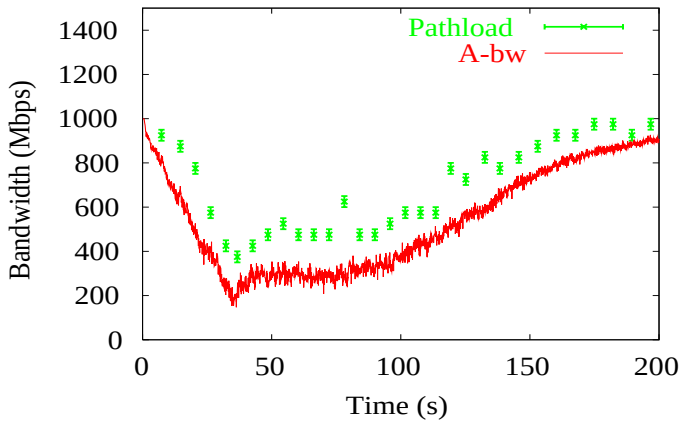
We next investigate the measurement results for ICIM-abw in the network model depicted in Figure 4.4. Cross traffic is now changed to Web traffic involving a large number of active



(a) Average measurement results of ICIM-abw



(b) Measurement results for IC-aware Pathload. $K = 160$



(c) Measurement results for normal Pathload. $K = 200$

Figure 4.7: Measurement results in Web traffic environment

Table 4.1: Number of packets required for a measurement

$A - bw$ (Gbps)	ICIM-abw	IC-aware Pathload
2	110	$200 \cdot 12 \cdot 8 = 19\ 200$
3	130	$200 \cdot 12 \cdot 9 = 21\ 600$
4	154	$200 \cdot 12 \cdot 10 = 24\ 000$

Web document accesses. We use a Pareto distribution for the Web object size distribution with 1.2 as the Pareto shape parameter and 12 Kbytes as the average object size. The number of objects in a Web page is 20. The capacity of the tight link is set to 1 Gbps. The access links are also set to 1 Gbps.

The available bandwidth is calculated as the capacity of the tight link minuses the total amount of Web traffic passing the link. Figure 4.7(a) shows the changes of available bandwidth and the average measurement results for each second. ICIM-abw under-estimates the available bandwidth a little because the cross traffic, composed of so many connections, arrives at the tight link in a bursty fashion. The burst of cross traffic may enlarge the intervals of the measurement bursts of ICIM-abw even when the probing rate is still lower than the average available bandwidth. However, the measurement results deviate only a little from the correct values and in general they can follow the changes of available bandwidth.

Figure 4.7(b) shows the measurement results for IC-aware Pathload in the same environment. We set K to 160 and the starting probing rate to 100 Mbps and ω and χ are both set to 50 Mbps. Overall, the results have a trend of over-estimation. We think that the problem can be solved if we adjust the PCT/PDT thresholds of Pathload appropriately, instead of using the default values. Figure 4.7(c) shows the measurement of normal Pathload. Because the probe packets are grouped at the NIC, the increasing trend in the measurement streams becomes difficult to discover. Therefore, Pathload over-estimates in most of the time. This frequent overestimation of bandwidth may lead to more aggressive systems. A conservative system caused by frequent underestimation of ICIM-abw will give less effect to the others sharing the same network environment.

TCP compatibility

We finally examine the data transmission performance of Reno TCP when it employs ICIM-abw. We perform a simulation where a number of Reno TCP connections that have

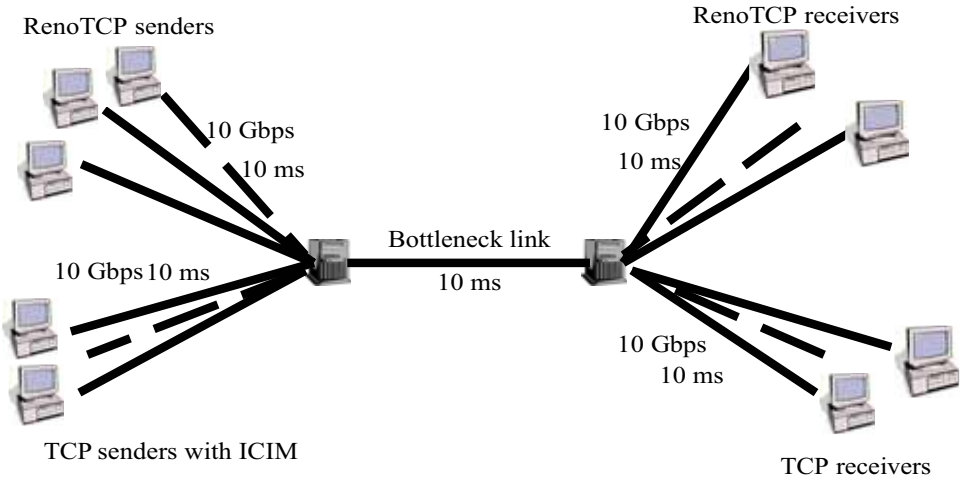


Figure 4.8: Simulation topology for examining TCP compatibility

ICIM-abw conflict with the same number of Reno TCP connections that do not have ICIM-abw through a 1 Gbps tight link, as shown in Figure 4.8. All the connections have the same RTT (0.018 s) and the same access link's bandwidth (10 Gbps). The number of connections is set to 4, 8 and 12. For each value of connection numbers, simulation is repeated 10 times, and the throughputs of the TCP connections that have and do not have ICIM-abw (and the ratio of thereof) are calculated and compared.

Table 4.2 shows the results when Q of ICIM-abw is set to 1 RTT and 2 RTTs and when ICIM-abw does not perform. In case ICIM-abw performs measurement in every RTT, the TCP achieves lower throughput than TCP that does not perform ICIM-abw when conflicts occur because ICIM-abw has to delay several data packets for measurement in this case. As shown in Table 4.2, the ratio of throughput between TCP with ICIM-abw compared to RenoTCP is always less than 1. When the number of connections increases, the ratio is lower because conflicts between TCP connections are more intense. If ICIM-abw takes a lower measurement frequency, for example, when $Q = 2$ RTT, then the TCP connections performing ICIM-abw can obtain the same throughput as normal Reno TCP, as shown in the third column of the table. We also disable ICIM-bw in all the TCP connections and show the throughput in this case in the fourth column of the table. We can see that that total throughput of TCP connection without ICIM-bw is almost the same of that when ICIM-abw with $Q = 2$ RTT is enabled. This means that ICIM-abw with reasonable measurement intervals does not effect the TCP connection performance.

Table 4.2: Comparison on throughputs of Reno TCP with and without ICIM

#con.	Q=1 RTT	Q=2 RTTs	No measurement
4	466.4 : 490.6 (0.95:1)	483.7 : 475.6 (1.01:1)	489.5 : 478.9 (1.02:1)
8	451.1 : 544.4 (0.82:1)	505.1 : 490.5 (1.02:1)	510.7 : 485.9 (1.05:1)
12	418.7 : 577.7 (0.72:1)	503.5 : 493.2 (1.02:1)	503.1 : 493.3 (1.02:1)

4.4 ICIM-cap: Interrupt Coalescence-aware inline measurement for capacity

Together with available bandwidth, the end-to-end capacity of a network path is important for adaptive control in a transport protocol. We have mentioned an inline measurement algorithm for capacity in Chapter 3. In this section, we focus on measuring this metric in high-speed networks.

4.4.1 Existing capacity measurement techniques and their problems

Many measurement techniques have been proposed for capacity measurement, such as Bprobe [26], Pathrate [39], CapProbe [37]. All of these techniques utilize packet pairs for measurement. However, because packets transmitting back-to-back are always grouped at the NIC under the effect of IC, the receiver cannot read the correct inter-arrival intervals of the packets. Thus, the packet pair-based techniques fail to perform the measurement when IC is enabled.

The first algorithm that can work in an IC environment is an enhanced version of Pathrate, suggested by [63]. The work of the algorithm is as follows.

The sender sends a measurement train (a group of packets) that is long enough that at least two bursts are observed in the received train. Then the number of the packets in the first burst (N) is used for the calculation of capacity:

$$C = \frac{N \cdot P}{L}, \tag{4.8}$$

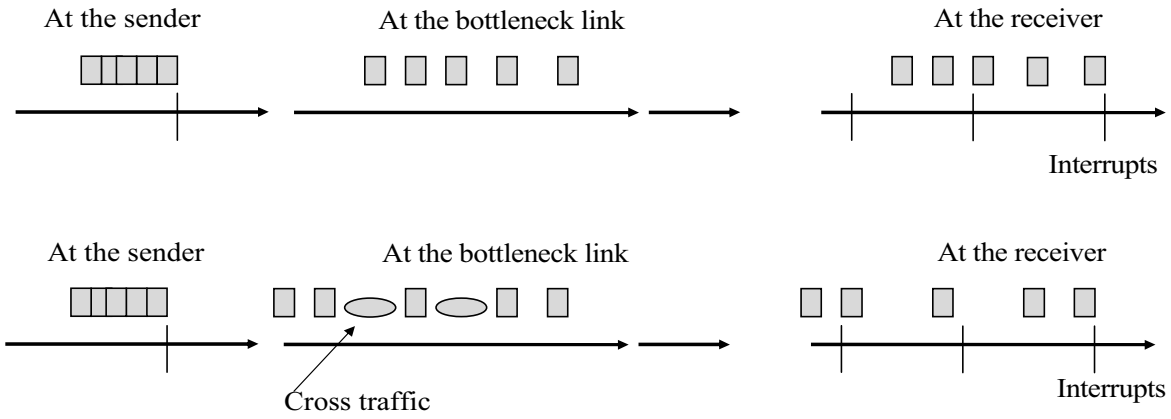


Figure 4.9: Enhanced Pathrate algorithm

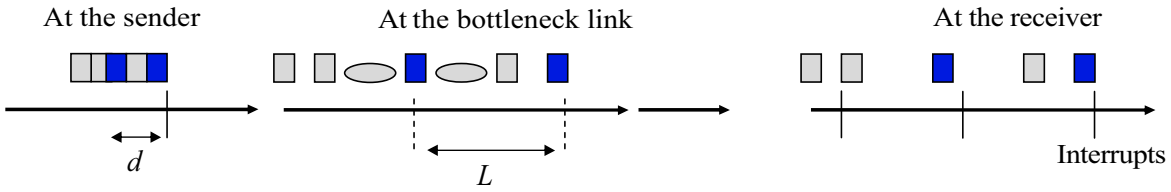


Figure 4.10: ICIM-cap algorithm

where L is the inter-arrival interval of the first and second burst (See the upper part of Figure 4.9).

However, the approach can work only when there is no cross traffic in the network path. If cross traffic exists, the cross traffic may cut into the measurement burst so that the number of packets received in the first interrupt may not reflect the value of the capacity correctly (See the lower part of Figure 4.9). This means the approach will not work well when the traffic load on the network is high.

4.4.2 ICIM-cap

We propose a burst-based capacity measurement algorithm that can overcome the problems mentioned above. The main concept of the proposed algorithm is that the available bandwidth information, which can be yielded periodically due to ICIM-abw (introduced in the last section), is exploited. The available bandwidth information is used to estimate the quantity of cross traffic that cuts into the first burst. In Figure 4.10, the top packets of the bursts are drawn in blue. The sending interval of the two packets is d. The amount

of traffic that arrives at the bottleneck link between the arrivals of the two packets is as follow:

$$C \cdot L = N \cdot P + d \cdot C_{Cross}, \tag{4.9}$$

where C_{Cross} is the average arrival rate of the cross traffic at the bottleneck link. Existing capacity measurement algorithms do not know the amount of C_{Cross}, so they cannot find the exact value of C in the case shown in the lower part of Figure 4.9. To the contrary, ICIM-cap can know the C_{Cross}, so it can perform well in such a case. That is the biggest feature of this algorithm.

If we suppose that the bottleneck link, which has the smallest capacity, is identical to the tight link, which has the smallest available bandwidth, we can write:

$$C_{Ccross} + A = C.$$

We can then calculate the capacity as follow:

$$C = \frac{N \cdot P - d \cdot A}{L - d}. \tag{4.10}$$

The TCP sender sends the bursts for ICIM-abw alternately with the bursts for ICIM-cap. The newest result of ICIM-abw is used for the next measurement of ICIM-cap. The length of the burst of ICIM-cap must be decided properly. It must be long enough that it can arrive in two interrupts. However, if it is too long, the TCP transmission will be adversely affected. We, therefore, propose a dynamic setting for the length, as follows:

- Quickly determine the initiative value.

  During the starting phase of the TCP connection, the TCP sender observes the length of the burst of packets and records the length of the longest one (G). The longest value is near the maximum number of packets that can be grouped in the same interrupt. So, we set the initial length of the measurement burst L to $1.5G$ in order to be sure that the burst can be divided into two small bursts at the receiver.

- Adapt the length dynamically to the changes of the environment.

  If L is too short, which can be noticed when the measurement burst is not divided into multiple bursts at the receiver, then the sender doubles the length. If L is too long, which can be noticed when the measurement burst is divided into more than

two bursts, the sender sets the length to $1.5B$, where B is the number of packets passed to the receiver in the first burst.

4.4.3 Simulation experiments

Through simulation validations, we show that ICIM-cap can deliver capacity measurement results quickly and correctly. Especially, it can deliver good results in extremely high-load networks, where current measurement algorithms such as (enhanced) Pathrate do not work well.

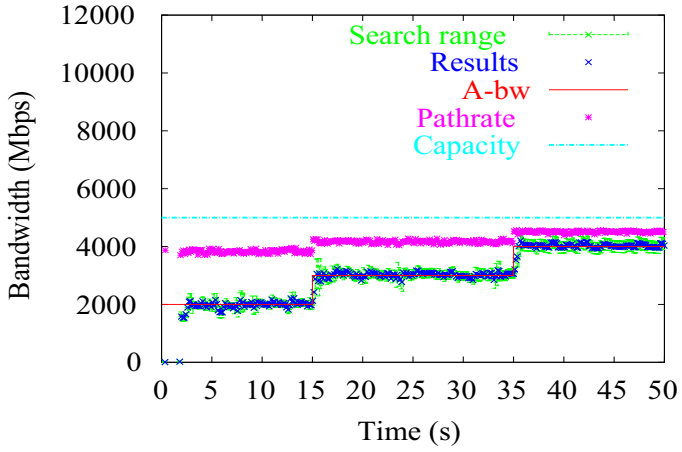
We repeat the simulation using the topology shown in Figure 4.4. This time, the TCP sender performs both ICIM-abw and ICIM-cap. For comparison, we also show the measurement results when we replace the ICIM-cap measurement algorithm by enhanced Pathrate. The measurement results are shown in Figures 4.11(a) and 4.11(b). In these figures, the measurement results of ICIM-abw are almost the same as those in Figure 4.5. Figure 4.11(a) shows the measurement results of enhanced Pathrate. As we can see, when the traffic load on the bottleneck link is heavy (when the available bandwidth is 2 Gbps while the capacity is 5 Gbps), Pathrate underestimates the capacity. On the other hand, as shown in Figure 4.11(b), ICIM-cap can deliver good measurement results regardless of the load on the network.

4.4.4 Discussions

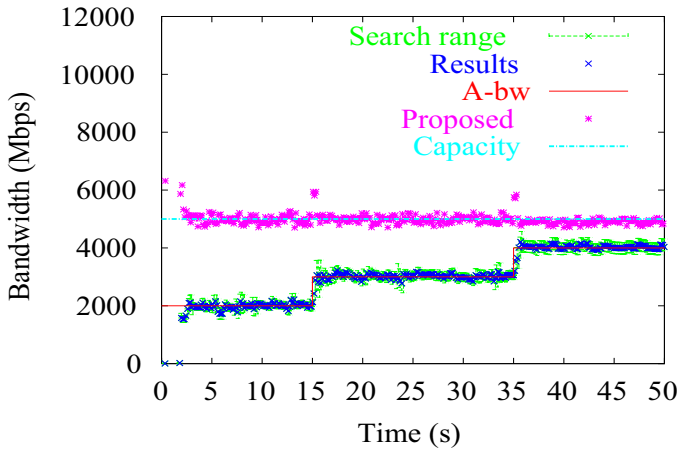
ICIM-cap relies on the supposition that the bottleneck link and tight link are identical. In this session, we discuss the errors in the measurement results of ICIM-cap when the above supposition is incorrect.

The case when the tight link is the upper link of the bottleneck link

Figure 4.12 shows the case when the tight link is the upper link of the bottleneck link. In this case, we suppose that traffic on another link does not affect much of the probe traffic. Moreover, the effect from the cross traffic on the bottleneck link is small. When the supposed condition is not true, the curve showing the relation between R_{in}, R_{out} will be more complex, but the tendency in general is unchanged. R_{in}, R_{out} are the sending rate



(a) Measurement results for enhanced Pathload



(b) Measurement results for ICIM-cap

Figure 4.11: Measurement results in gigabit network

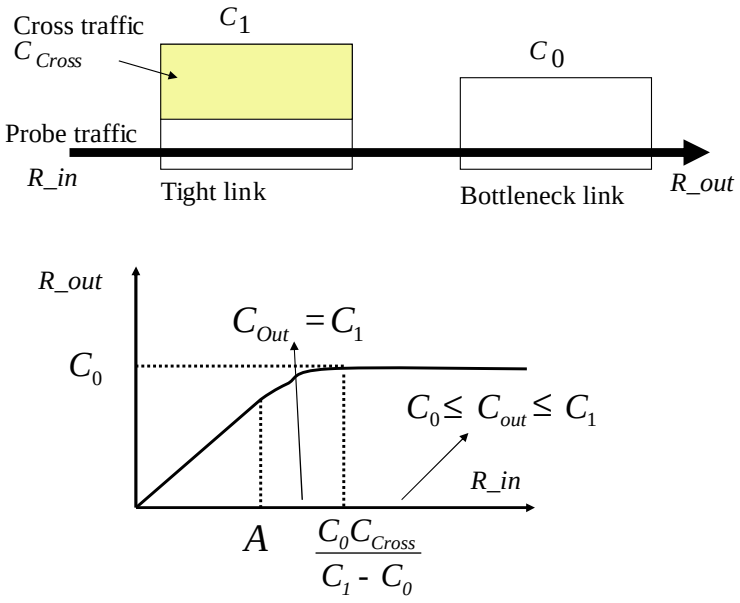


Figure 4.12: The case when the tight link is the upper link of the bottleneck link

and arrival rate, respectively, of the probe traffic. From Equation (4.10), we can write:

$$C = \frac{\frac{NP}{L}}{\frac{NP}{d}}\Big(C_{Cross} + \frac{NP}{d}\Big). \tag{4.11}$$

Because $\frac{NP}{L} = R_{in}$, $\frac{NP}{d} = R_{out}$, we can rewrite this using R_{in} and R_{out} as follow:

$$C = \frac{R_{in}}{R_{out}}(C_{Cross} + R_{out}). \tag{4.12}$$

We call C_{out} the result of the above calculation. C_{out} is the measurement result given by ICIM-cap. In this case, we examine the relation between C_{out} and the real capacity value (C_0) as well as the capacity of the tight link (C_1).

Figure 4.12 shows the changes of R_{out} when R_{in} increases. When R_{in} is smaller than the available bandwidth (A), R_{out} increases in proportion to R_{in}. When R_{in} reaches A (but is still smaller than ($\frac{C_0 \cdot C_{Cross}}{C_1 - C_0}$), the probe traffic starts to conflict with the cross traffic, and the increasing trend of R_{out} becomes slower. When R_{in} becomes larger than $\frac{C_0 \cdot C_{Cross}}{C_1 - C_0}$, R_{out} does not change regardless of the value of R_{in}. That happens because R_{out} is limited by the bottleneck link C_0 in this case. We summarize the results as follows:

- When $A < R_{in} < \frac{C_0 \cdot C_{Cross}}{C_1 - C_0}$, the measurement result of ICIM-cap C_{out} will be $C_0 \leq$

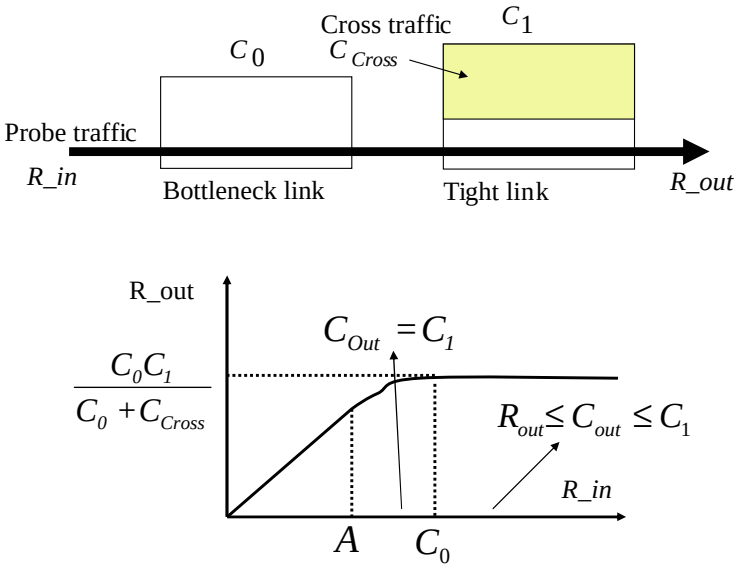


Figure 4.13: The case when the bottleneck link is the upper link of the tight link

$C_{out} \leq C_1$,

- When $R_{in} \leq \frac{C_0 \cdot C_{Cross}}{C_1 - C_0}$, we also have $C_0 \leq C_{out} \leq C_1$.

The case when the bottleneck link is the upper link of the tight link

If the tight link is the upper link of the bottleneck link, with the same observations as the above case, we can go to the following results (Figure 4.13):

- When $A < R_{in} < C_0$, the result of ICIM-cap C_{out} is equal to C_1 ($C_{out} = C_1$),
- When $C_0 \leq R_{in}$, the result of ICIM-cap C_{out} is included in the range: $R_{out} \leq C_{out} \leq C_1$.

4.4.5 Interpretation of the results

When a tight link and bottleneck link are not identical, ICIM-cap's measurement result can overestimate, but the measurement never gets higher than the capacity of the tight link. In fact, a link with a large capacity does not often become a tight link, so the overestimation will not be very large. The measurement results can be an underestimation, however, the result is never smaller than R_{out}, which is the measurement result of enhanced Pathrate.

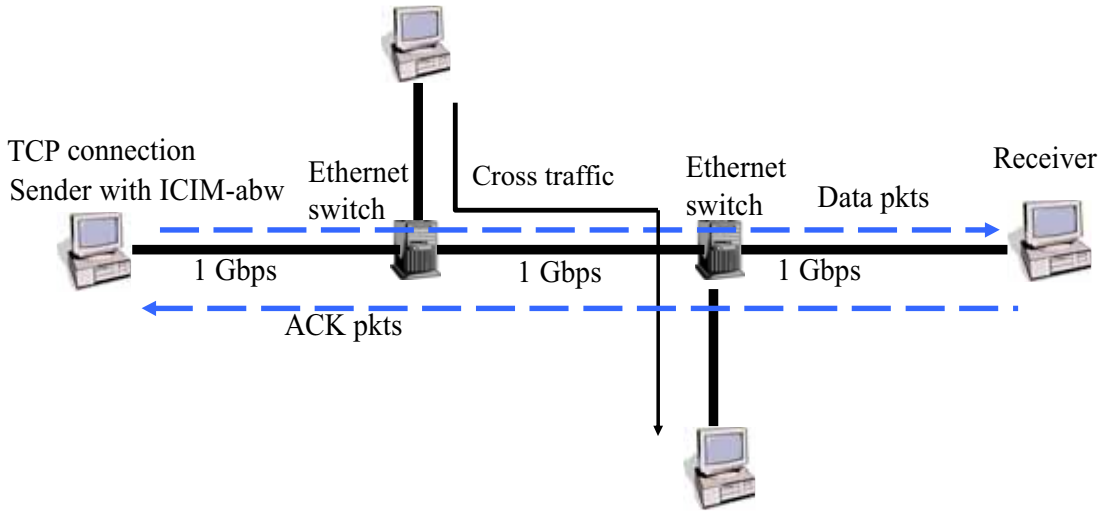


Figure 4.14: Network topology of the experiment

Thus, the measurement results for ICIM-cap may not be correct when the supposition of the tight link and bottleneck link is not true, but the error is not large.

4.5 Experiments in a real environment

In this section, we present the experiment result in a real environment to validate the burst-based measurement algorithm. We implement the basic algorithm, ICIM-abw, in a FreeBSD system and use the simple network shown in Figure 4.14 to examine whether the algorithm works well. This network consists of two switches equipped with 1-Gbps Ethernet ports; all links are 1 Gbps. Table 4.3 shows the specifications of the PCs. The cross traffic is made up of UDP traffic sent by Iperf [49]. One TCP connection is established between the sender and the receiver. In the TCP sender program, the ICIM-abw program is implemented. In this case, the link connecting the two switches becomes the bottleneck link. We control the Iperf flows so that the available bandwidth on the bottleneck link is 600 Mbps from 0 to 50 sec, 300 Mbps from 50 to 100 sec and 500 Mbps from 100 to 150 sec. Both NICs of the sender and the receiver enable IC; the *RxAbsIntDelay* parameter of IC is set to 0.1312 ms.

In Figure 4.15, we plot the correct values of the available bandwidth. The measurement results for ICIM-abw and the search ranges are shown in the same figure. We can

Table 4.3: Specifications of the PCs in the experiment

	Sender	Receiver
CPU	Intel P4 3.0 GHz	Intel P4 3.4 GHz
Mem.	1,024 MB	1,024 MB
OS	Free BSD 4.10	FedoraCore 4
NIC	Int. PRO/1000 Adapter	Int. PRO/1000 Adapter

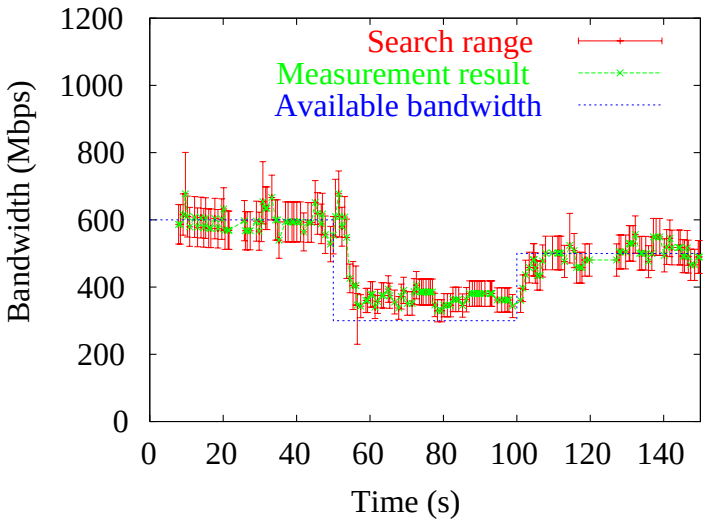


Figure 4.15: Changes of the available bandwidth and the measurement results

see that ICIM-abw can suitably measure the available bandwidth in this experimental network. Moreover, the measurement accuracy is as high as the evaluation of the simulation experiments in Section 4.3.3. As future studies, we will perform an experiment on ICIM algorithms in a large-scale network as well as on the Internet.

4.6 Conclusions

In this chapter, we introduced ICIM-abw and ICIM-cap, the methods that can measure the available bandwidth and capacity on a 1-Gbps or higher network path. The proposed measurement algorithms do not require regulation of packet transmission intervals and work well with Interrupt Coalescence. The simulation experiments showed that the proposed measurement algorithm works well in networks as high or higher than 1 Gbps.

As our future works, we will evaluate the performance of ICIM in a real Internet environment. We are also testing the performance of the congestion control mechanism proposed by [14] when ICIM is used for bandwidth estimation, in the real network environments.

Chapter 5

Conclusions

In summary, in this thesis, we introduced measurement algorithms that can be applied well in an active TCP connection. We refer to the proposed approach as inline network measurement. Inline measurement algorithms for two important bandwidth-related metrics of the end-to-end network path are proposed. In addition, special algorithms are also proposed for gigabit networks. The proposed algorithms make ingenious use of the TCP mechanism so that they can provides measurement results continuously and quickly while interfering with a small number of TCP data packets and having no effect on other traffic on the network. The algorithms are summarized below.

We first proposed a new measurement algorithm suitable for inline available bandwidth measurement that generates periodic measurement results in short intervals, on the order of several RTTs. The key concept in rapid measurement is to limit the bandwidth measurement range using statistical information from previous measurement results. We then introduced Inline measurement (TCP ImTCP), a Reno-based TCP that includes the proposed algorithm. When the ImTCP sender transmits data packets, it first stores a group of up to several packets in a queue and subsequently forwards them at a transmission rate determined by the measurement algorithm. Then, considering the arrival intervals of the ACK packets, the ImTCP sender perform bandwidth estimation. The simulation experiments showed that the measurement function works well with the window-based congestion control algorithm and has no effect on the traffic sharing the network path.

We next developed a new capacity measurement function and combined it with ImTCP in order to enable simultaneous measurement of both capacity and available bandwidth in ImTCP. The capacity measurement algorithm is a new packet-pair-based measurement technique that utilizes the estimated available bandwidth values for capacity calculation.

This new algorithm promises faster measurement than current packet-pair-based measurement algorithms for various situations and works well for high-load networks, in which current algorithms do not work properly.

Finally, we introduced inline measurement algorithms that overcome problems faced by current measurement tools when measuring high-speed networks: ICIM-abw for available bandwidth measurement and ICIM-cap for capacity measurement. The proposed algorithms utilize the data packets of an active TCP connection for measurement, as do existing inline measurement methods. However, rather than adjusting the packet transmission intervals, the proposed algorithms adjust the number of packets involved in a packet burst and utilize the inter-intervals of the bursts of the corresponding ACK packets for bandwidth measurement. Simulations and results showed that the proposed algorithms can measure the bandwidth in the network paths of 1 Gbps and higher.

For more than 40 years, TCP has successfully played the major role in data transmission on the Internet. With the proposed built-in measurement techniques, TCP becomes much more useful than just a transmission tool; TCP can act as an active measurement tool. The wide deployment of new versions of TCP will provide a large infrastructure that monitors the end-to-end bandwidth in real time for any network, and, most importantly, having no adverse effect on the network. End-to-end bandwidth inferred in TCP can be then passed to a higher network layer and utilized for adaptive control, optimal route selection, network topology design, or isolating fault locations. In addition, TCP can use such bandwidth information to optimize link utilization or improve the transmission performance of TCP itself.

Implementation codes for some of the measurement algorithms proposed in this thesis can be found on the Internet at the ImTCP homepage [53]. Currently, ImTCP with the available bandwidth measurement function (released in March, 2005) and ICIM-abw (released in December, 2005) are available. All of the implementations are for the BSD 4.10 [64] kernel system

There are several challenging tasks for future research. The first is the development of special TCP types that make use of inline measurement. TCP can definitely react more intelligently with changes in the bandwidth of an IP network when inline measurement can report such changes in a timely manner. As a first step, TCP for background transfer (ImTCP_bg) [15], TCP for high-speed transfer (TCP symbiosis) [14], and a congestion control mechanism of TCP for achieving predictable throughput [65], have been proposed. We believe that the "intelligent" TCP versions are a new trend in the improvement of

the network transport protocols. Second, in an effort to reduce the load over the network caused by probe traffic, inline measurement techniques for streaming protocols, such as the Real Time Streaming Protocol [66], are also of interest.

Bibliography

[1] G. Pierre and M. van Steen, "Design and implementation of a user-centered content delivery network," in *Proceedings of the 3rd IEEE Workshop on Internet Applications*, June 2003.

[2] S. Chidlow, "Storage area networks," *JISC Technology and Standards Watch Report: Storage Area Networks*, Nov. 2003.

[3] P. Rodriguez, C. Gkantsidis, and J. Miller, "Comprehensive view of a live network coding P2P system," in *Proceedings of Internet Measurement Conference 2006*, Oct. 2006.

[4] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in structured P2P systems," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS 2003)*, Feb. 2003.

[5] M. Bishop, S. Rao, and K. Sripanidkulchai, "Considering priority in overlay multicast protocols under heterogeneous environments," in *Proceedings of INFOCOM 2006*, Apr. 2006.

[6] Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in *Proceedings of the 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, Aug. 2001.

[7] Y. Zhao and Y. Hu, "GRESS - a Grid replica selection service," in *Proceedings of the 16th International Conference on Parallel and Distributed Computing Systems (PDCS-2003)*, Aug. 2003.

[8] R. Gao, C. Dovrolis, and E. Zegura, "Avoiding oscillations due to intelligent route control systems," in *Proceedings of IEEE INFOCOM 2006*, Apr. 2006.

[9] Akamai homepage, available at http://www.akamai.com/.

[10] Exodus homepage, available at http://www.exodus.com/.

[11] R. Kokku, P. Yalagandula, A. Venkataramani, and M. Dahlin, "NPS: A non-interfering deployable Web prefetching system," in *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)*, Mar. 2003.

[12] M. Natu and A. Sethi, "Active probing approach for fault localization in computer networks," in *Proceedings of the E2EMON Workshop 2006*, Apr. 2006.

[13] TPTEST, available at http://tptest.sourceforge.net/about.php.

[14] G. Hasegawa and M. Murata, "TCP Symbiosis: Congestion control mechanisms of TCP based on Lotka-Volterra competition model," in *Proceedings of Workshop on Interdisciplinary Systems Approach in Performance Evaluation and Design of Computer and Communications Systems (Inter-Perf 2006)*, Oct. 2006.

[15] T. Isugawa, G. Hasegawa, and M. Murata, "Background TCP data transfer with inline network measurement," *IEICE Transactions on Communications*, vol. E89-B, no. 8, pp. 2152–2160, Aug. 2006.

[16] M. Gerla, B. Ng, M. Sanadidi, M. Valla, and R. Wang, "TCP Westwood with adaptive bandwidth estimation to improve efficiency/friendliness tradeoffs," *Computer Communication Journal*, vol. 27, no. 1, pp. 41–48, Jan. 2004.

[17] R. Wang, G. Pau, K. Yamada, M. Sanadidi, and M. Gerla, "TCP start up performance in large bandwidth delay networks," in *Proceedings of INFOCOM 2004*, Mar. 2004.

[18] S. Asif, T. Nguyen, G. Xu, and B. Song, "Streaming video with bandwidth adaptation and error concealment for lowbit rate live wireless applications," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '05)*, Mar. 2005.

[19] Cisco Netflow, available at http://www.cisco.com/warp/public/732/Tech/netflow/.

[20] MRTG Web site, available at http://people.ee.ethz.ch/ oetiker/webtools/mrtg/.

[21] R. Anjali, C. Scoglio, L. Chen, I. Akyildiz, and G. Uhl, "ABEst: An available bandwidth estimator within an autonomous system," in *Proceedings of IEEE GLOBECOM 2002*, Nov. 2002.

[22] K. Lai and M. Baker, "Nettimer: A tool for measuring bottleneck link bandwidth," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Mar. 2001.

[23] T. En-Najjary and G. Urvoy-Keller, "PPrat: A passive capacity estimation tool," in *Proceedings of the E2EMON Workshop 2006*, Apr. 2006.

[24] Pathchar, available at http://www.caida.org/tools/utilities/others/pathchar/.

[25] Pchar, available at http://www.ca.sandia.gv/ bmah/Software/pchar.

[26] R. Carter and M. Crovella, "Measuring bottleneck link speed in packet-switched networks," *Boston University Computer Science Department*, Tech. Rep. TR-96-006, Mar. 1996.

[27] J. Strauss, D. Katabi, and F. Kaashoek, "A measurement study of available bandwidth estimation tools," in *Proceedings of Internet Measurement Conference 2003*, Oct. 2003.

[28] M. Jain and C. Dovrolis, "End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput," in *Proceedings of ACM SIGCOMM 2002*, Aug. 2002.

[29] N. Hu and P. Steenkiste, "Evaluation and characterization of available bandwidth probing techniques," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 6, Aug. 2003.

[30] M. Gerla, Y. Sanadidi, R. Wang, A. Zanella, C. Casetti, and S. Mascolo, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proceedings of the SIGCOMM 1994*, pp. 24–35, Aug. 1994.

[31] S. Savage, "Sting: A TCP-based network measurement tool," in *Proceedings of the 3rd Usenix Symposium on Internet Technologies and Systems (USITS 1999)*, Oct. 1999.

[32] Sprobe, available at http://sprobe.cs.washington.edu.

[33] D. Antoniades, M. Athanatos, A. Papadogiannakis, E. Markatos, and C. Dovrolis, "Available bandwidth measurement as simple as running wget," in *Proceedings of the 7th Passive and Active Measurement Workshop (PAM 2006)*, Mar. 2006.

[34] J. C. Hoe, "Improving the start-up behavior of a congestion control sheme for TCP," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, vol. 26, no. 4, pp. 270–280, Aug. 1996.

[35] T. Xu-hong, L. Zheng-lan, and Z. Miao-liang, "TCP-Rab: A receiver advertisement based TCP protocol," *Journal of Zhejiang University SCIENCE*, vol. 5, no. 11, pp. 1352–1360, 2004.

[36] A. Persson, C. Marcondes, L. Chen, Y. Sanadidi, and M. Gerla, "TCP Probe: A TCP with built-in path capacity estimation," in *Proceedings of the 8th IEEE Global Internet Symposium*, Mar. 2005.

[37] R. Kapoor, L. Chen, L. Lao, M. Gerla, and M. Sanadidi, "CapProbe: A simple and accurate capacity estimation technique," in *Proceedings of ACM SIGCOMM 2004*, Aug. 2004.

[38] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell, "PathChirp: Efficient available bandwidth estimation for network paths," in *Proceedings of the 4th Passive and Active Measurement Workshop (PAM 2003)*, Apr. 2003.

[39] C. Dovrolis, P. Ramanathan, and D. Moore, "Packet dispersion techniques and a capacity-estimation methodology," *IEEE/ACM Transactions on Networking*, vol. 12, no. 6, pp. 963–977, Dec. 2004.

[40] Intel, "Interrupt moderation using Intel Gigabit Ethernet Controllers," available at http://www.intel.com/design/network/applnots/ap450.pdf (2003).

[41] Syskonnect, "SK-NET GE Gigabit Ethernet Server Adapter," available at http://www.syskonnect.com/syskonnect/technology/SK-NET_GE.PDF (2003).

[42] B. Melander, M. Bjorkman, and P. Gunningberg, "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks," in *Proceedings of IEEE GLOBECOM 2000*, Nov. 2000.

[43] J. Navratil and R. Cottrell, "ABwE: A practical approach to available bandwidth estimation," in *Proceedings of the 4th Passive and Active Measurement Workshop (PAM 2003)*, Apr. 2003.

[44] A. Shriram, M. Murray, Y. Hyun, N. Brownlee, A. Broido, M. Fomenkov, and k claffy, "Comparison of public end-to-end bandwidth estimation tools on high-speed links," in *Proceedings of the 6th Passive and Active Measurement Workshop (PAM 2005)*, Mar. 2005.

[45] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of ACM Symposium on Operating Systems Principles (SOSP 2001)*, Oct. 2001.

[46] R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[47] NS homepage, available at http://www.isi.edu/nsnam/ns/.

[48] NLANR Web site, available at http://moat.nlanr.net/Datacube/.

[49] Iperf, available at http://dast.nlanr.net/Projects/Iperf/.

[50] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley-Interscience, 1991.

[51] S. Jin, L. Guo, I. Matta, and A. Bestavros, "TCP-friendly SIMD congestion control and its convergence behavior," in *Proceedings of the 9th IEEE International Conference on Network Protocols (ICNP 2001)*, Nov. 2001.

[52] S. Floyd, "Highspeed TCP for large congestion windows," *RFC 3649*, Dec. 2003.

[53] ImTCP homepage, available at http://www.anarg.jp/imtcp/.

[54] K. Lai and M. Baker, "Measurering link bandwidths using a deterministic model of packet delay," in *Proceedings of ACM SIGCOMM 2000*, Aug. 2000.

[55] A. B. Downey, "Using Pathchar to estimate internet link characteristics," in *Proceedings of ACM SIGCOMM 1999*, Aug. 1999.

[56] M. Goutelle and P. Vicat-Blanc, "Study of a non-intrusive method for measuring the end-to-end capacity and useful bandwidth of a path," in *Proceedings of the 2004 IEEE International Conference on Communications*, June 2004.

[57] T. Kelly and T. Ott, "Performance sensitivity and fairness of ECN-aware modified TCP," in *Proceedings of Second International IFIP-TC6 Networking Conference*, May 2002.

[58] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast long-distance networks," in *Proceedings of INFOCOM 2004*, Mar. 2004.

[59] C. Jin, D. Wei, and S. Low, "FAST TCP: Motivation, architecture, algorithms, performance," in *Proceedings of INFOCOM 2004*, Mar. 2004.

[60] C. Marcondes, A. Persson, M. Sanadidi, M. Gerla, H. Shimonishi, T. Hama, and T. Murase, "Inline path characteristic estimation to improve TCP performance in high bandwidth-delay networks," in *Proceedings of the International Workshop on Protocols for Fast Long-Distance Networks*, Feb. 2006.

[61] G. Jin and B. Tierney, "System capability effect on algorithms for network bandwidth measurement," in *Proceedings of Internet Measurement Conference 2003*, Oct. 2003.

[62] M. Zec, M. Mikuc, and M. Zagar, "Estimating the impact of interrupt coalescing delays on steady state TCP," in *Proceedings of the 10th SoftCOM Conference*, Oct. 2002.

[63] R. Prasad, M. Jain, and C. Dovrolis, "Effects of interrupt coalescence on network measurements," in *Proceedings of the 5th Passive and Active Measurement Workshop (PAM 2004)*, Apr. 2004.

[64] FreeBSD homepage, available at http://www.freebsd.org/.

[65] K. Yamanegi, G. Hasegawa, and M. Murata, "Congestion control mechanism of TCP for achieving predictable throughput,," in *Proceedings of Australian Telecommunication Networks and Applications Conference*, Dec. 2006.

[66] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)," *RFC 2326*, Apr. 1998.