


University of Alberta

AN APPLICATION LAYER BANDWIDTH ESTIMATION ALGORITHM

by

Yinzhe Yu 

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment  
of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta

Fall 2002



National Library  
of Canada

Acquisitions and  
Bibliographic Services

395 Wellington Street  
Ottawa ON K1A 0N4  
Canada

Bibliothèque nationale  
du Canada

Acquisitions et  
services bibliographiques

395, rue Wellington  
Ottawa ON K1A 0N4  
Canada

*Your file Votre référence*

*Our file Notre référence*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-81506-4

**Canada**

**University of Alberta**

**Library Release Form**

**Name of Author:** Yinzhe Yu

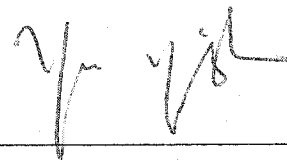
**Title of Thesis:** An Application Layer Bandwidth Estimation Algorithm

**Degree:** Master of Science

**Year this Degree Granted:** 2002

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.



---

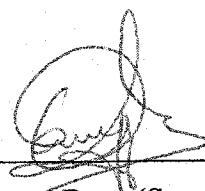
Yinzhe Yu  
Department of Computing Science  
133 Athabasca Hall  
University of Alberta  
Edmonton, AB  
Canada, T6G 2E8

**Date:** June 19, 2002

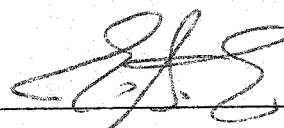
University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **An Application Layer Bandwidth Estimation Algorithm** submitted by Yinzhe Yu in partial fulfillment of the requirements for the degree of **Master of Science**.



Dr. Anup Basu (Supervisor)



Dr. Ehab Elmallah



Dr. Wayne Grover

Date: June 17, 2002

# Abstract

A network-aware application provides QoS to end-users without relying on service guarantees from the underlying networks. This kind of application actively monitors the performance variation of the network and adjusts resource demand accordingly. A critical issue for such an application is the accurate estimation of the available bandwidth. We propose a new bandwidth estimation algorithm for multimedia delivery network-aware applications. The main QoS parameter is the time limit for delivery of a media object. The algorithm uses the first fraction of the time limit to do bandwidth sampling and then uses this knowledge to estimate future bandwidth. The algorithm has two salient features. First, based on a t-distribution statistical model, it makes a controlled conservative estimation of the available bandwidth. This estimation preserves the time limit parameter with a confidence level. Second, the algorithm dynamically approximates the optimal amount of bandwidth sampling. It thereby maximizes the media object being transmitted within the time limit. Simulation results as well as experiments on real networks are presented to demonstrate the effectiveness of the algorithm.

*To my parents.*

# Acknowledgement

I would like to thank my supervisor, Dr. Anup Basu, for his support, advice and encouragement throughout this research. I would also like to thank my defense committee; Dr. Wayne Grover and Dr. Ehab Elmallah, for their truly valuable comments and critiques to my thesis, and Dr. Duane Szafron, for chairing my defense committee. I am also grateful to all the members of the Vision and Multimedia Communications Lab, Dr. Minghong Pi, Dr. Mark Fiala, Yixin Pan, Irene Cheng, and Matthew Bates, for their ideas and friendships. Thanks also go to Edith Drummond, Sunrose Ko, Sandi Sands and Carol Smith for their helpful administrative assistance during my stay in the Computing Science Department of University of Alberta. Finally, I am greatly indebted to my family. Without their inspirations, moral supports and all the sacrifices they have made, I could never have achieved this.

# Contents

<b>1. Introduction</b>	<b>1</b>
1.1 The Quest for Quality of Service .....	1
1.2 Resource Reservation .....	2
1.3 Network-aware Application .....	3
1.4 Bandwidth Estimation Problem .....	4
1.5 Thesis Objective .....	6
1.6 Thesis Organization .....	7
<b>2. Background and Related Works</b>	<b>8</b>
2.1 TCP Congestion Control .....	8
2.2 TCP Congestion Avoidance .....	12
2.3 TCP for Wireless .....	14
2.4 Related Researches .....	15
2.5 Contributions .....	18
<b>3. Single Server Bandwidth Estimation</b>	<b>20</b>
3.1 The Problem and Assumptions .....	20
3.2 Notations .....	21
3.3 The Algorithm .....	22
<b>4. Simulation of Single Server Algorithm</b>	<b>28</b>
4.1 Overview of Simulation Experiments .....	29
4.2 Moving Average Method .....	31
4.3 Experiment Results .....	33
<b>5. Multi-server Bandwidth Estimation</b>	<b>43</b>
5.1 Simple Extension of the Algorithm .....	44
5.2 Refinement of Multi-server Algorithm .....	47
5.3 Simulation Overview .....	50
5.4 Experiment Results .....	52



<b>6. Experiments on Real Networks</b>	<b>58</b>
6.1 Implementation of the Algorithm .....	59
6.2 Experiments on Campus Networks .....	61
6.3 Experiments on the Internet .....	65
6.4 Time Slice Choice .....	66
6.5 Experiments on Networks with a Wireless Link .....	69
<b>7. Conclusions and Directions for Future Research</b>	<b>72</b>
<b>Bibliography</b>	<b>75</b>
<b>Appendix 1 Experimental Data</b>	<b>81</b>
<b>Appendix 2 t-distribution Table</b>	<b>97</b>

# List of Tables

2.1	Features supported in major TCP implementations . . . . .	14
5.1	V values on K channels with K $\alpha$ values . . . . .	49
6.1	Experimental results for the campus network . . . . .	62
6.2	Average actual transmission time for the fixed sampling method . . . . .	63
6.3	Experimental results on the Internet . . . . .	66
6.4	Comparison of different time slice sizes . . . . .	67
6.5	Experimental results on the Internet with a wireless link . . . . .	70

# List of Figures

1.1	Network-aware Application Architecture .....	5
2.1	An example of congestion window dynamics in TCP connection .....	10
2.2	Fast retransmit based on duplicate ACKs .....	11
4.1	Screen Capture of Simulator at work .....	29
4.2	Screen Capture of Simulator using “Moving Average” method .....	32
4.3	Real Transmission Time for Fix Sampling, alpha 0.95, 500 total slices .....	34
4.4	Real Transmission Time comparison, alpha 0.95, 500 total slices .....	35
4.5	Real Transmission Time for Fix Sampling, alpha 0.95, 100 total slices .....	38
4.6	Real Transmission Time comparison, alpha 0.95, 100 total slices .....	39
4.7	Number of runs failing to finish within time limit, alpha 0.95, 100 total slices ..	41
4.8	Number of runs failing to finish within time limit, alpha 0.95, 500 total slices ..	42
5.1	Network environment of multi-server multimedia delivery application .....	43
5.2	Screen Capture of Multi-server Bandwidth Estimation Simulator .....	50
5.3	Comparison of transmitted object size for two algorithms, 100 total slices ....	53
5.4	Comparison of transmitted object size for two algorithms, 500 total slices ....	54
5.5	Number of runs failing to finish within time limit, multi-server .....	56
5.6	Number of runs using one channel for real transmission .....	57
6.1	Screen capture of the Client program .....	60
6.2	Experimental setup on campus network .....	62
6.3	Comparison of actual transmission times .....	64
6.4	Experimental setup on the Internet through broadband connection .....	65
6.5	Screen Capture of three experiments .....	68
6.6	Experimental setup on the Internet with a wireless link .....	69

# Chapter 1

## Introduction

### 1.1 The Quest for “Quality of Service”

Since its inception 40 years ago, the Internet has evolved from a few leased lines connecting half a dozen large computers to millions of inter-connected networks supporting all kinds of computing devices. Accordingly, the applications running on it have evolved from character-based programs to all kinds of multimedia applications. In the future, perhaps five years from now, the Internet is expected to connect various mobile devices like cell phones and personal data assistants (PDAs) with much more sophisticated features than exist today. These devices are expected to receive various information including that from emails, news, stock quotes, and online games. Much of this information will take the form of multimedia objects like images, sound, videos, and 3D graphic objects. These envisioned applications will create new challenges in the design of future Internet protocols, across all the levels of the protocol stack.

In terms of the network layer, the Internet was originally designed to offer only one level of service, the "best effort", to all its service users. In such an environment, all data packets put into the Internet have the same priority; the networks do not provide any guarantee on when a packet will be delivered or even whether a packet will be delivered at all. However, as new applications continue to emerge, requirements for the quality of data delivery become more diversified for different kinds of applications. Some applications may need guaranteed delivery for every bit of data in transmission, while others may focus much more on the average speed of the delivery; some applications are

sensitive mainly to the bandwidth the network is able to sustain, while others are equally sensitive to bandwidth, delay and jitter. The quest for solutions to these problems has been an active research area for the past decade, under terms like “Quality of Service” (QoS) and “Differential Services” [Miloucheva, 1995], [Vogel, 1995].

## 1.2 Resource Reservation

One solution to these problems that has currently attracted a lot of attention is the Resource Reservation Protocol (RSVP) and the “Integrated Services” it proposes [Zhang, 1993], [Braden, 1997]. The Internet envisioned by Integrated Services will provide a number of *service classes* other than the “best effort”. The examples of such classes include *guaranteed service* and *controlled load service*. For guaranteed service packets, the Integrated Services provider makes sure the delay of the packets will be within a specified time limit; for controlled load service packets, the Integrated Service provider emulates a lightly loaded network in delivering them, even though the actual network may be heavily loaded<sup>1</sup>. More service classes are under consideration and proposal.

To provide these services, substantial changes need to be made to hosts and routers. Apart from telling the network where the packet is intended to go, the hosts must also inform the network of the quality of service they expect for all the ensuing packets going to the same destination, which is called *flow specification*. Then, each router on the path from the sender to the receiver must determine whether it has enough resources (link capacity, buffer, etc.) to satisfy the flow. This process is called *admission control*. Based on admission control criteria, the router will either reject the request by sending an error message to the flow initiator, or commit the resources and maintain some internal states for the new flow. Finally, when packets of different flows arrive, routers need to meet the agreed-upon flow requirements by doing *packet scheduling* among all the

---

<sup>1</sup> The controlled-load service provides no firm quantitative guarantees as guaranteed service does. When a flow is accepted for controlled-load service, the router makes a commitment to provide a service equivalent to that seen by the “best effort” flow on a light-load network. Therefore, a controlled-load service flow does not noticeably deteriorate as the real network load increases. A good introduction to the implementation methods of both service modes is [White, 1997].

packets.

### 1.3 Network-aware Application

Although the resource reservation method provides a graceful solution to QoS, it has been suffering from slow acceptance and deployment for two reasons. Firstly, since RSVP requires the replacement (at least the upgrade of software) of a significant portion of the routers currently running the Internet, the cost is high. Secondly, since RSVP maintains states and schedules packets on the basis of each flow, it places a much heavier computation burden on routers. How to address these issues is still under research [Black, 1998] [Grossman, 2002].

One alternative approach that has generated a lot of interests in QoS-based multimedia delivery application is the *network-aware* application [Bolliger, 1998], [Bolliger, 1999] and [Wang, 1999]. These applications do not rely on the underlying network to provide guaranteed quality and are built on the old “best effort” service model of the Internet. As a result, they try to provide a trade-off between QoS parameters. For example, in interactive multimedia applications like tele-education [Maly, 1997], application designers may choose to trade image/video resolution for response time.

Instead of reserving a specific bandwidth before transmission, network-aware applications actively monitor the performance variation of the network and attempt to adjust their resource demands in response. For example, a server needing to deliver a certain multimedia object to a client adapts the volume of data (in turn, the quality of the multimedia object<sup>2</sup>) to be transmitted to the client, given different network bandwidth availability. If the available bandwidth is low, the application reduces the size of the object and thus reduces demands on the network; and if the bandwidth is high, the application increases its demand to fully utilize the additional resources. In such a

---

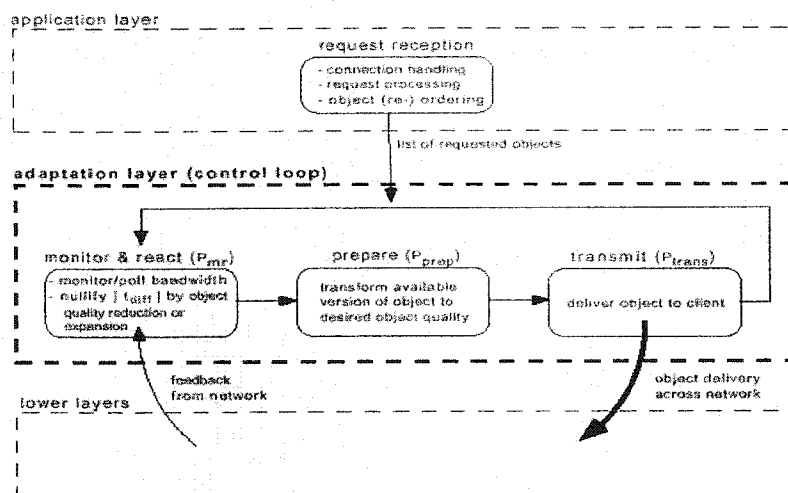
<sup>2</sup> In the real world, different applications may have different interpretations of what constitutes the “high quality” of a multimedia object. In this thesis, we will always assume quality correlates to the size of the object. Larger objects are assumed to have higher quality.

method, the application can guarantee the delivery of an object to its client with a meaningful QoS parameter like a user-specified time limit.

Several prototype systems have demonstrated this idea. In [Cheng, 2001], the author presented a Tele-Learning system that delivers multimedia course materials (mainly still images) to students over the Internet. The system allows the users to specify several QoS parameters. The first one is time limit, which allows the user to specify the time the transmission is expected to finish. The user can also set a target resolution and quality (the quality represents the quantization factor of JPEG image compression), as well as a set of criteria on how to trade off between resolution and quality in case the target cannot be met given the condition of the network. The server will adapt the resolution and quality of the requested image to provide the best image to the user while trying to restrict transmission time to within the time limit.

## **1.4 Bandwidth Estimation Problem**

Generally speaking, to deliver a user-requested multimedia object adaptively, the network-aware application needs to go through three phases. Firstly, the server needs to monitor the current network condition and estimate the available bandwidth in the brief future period during which requested object will be transmitted. Secondly, given the user-specified time limit and estimated bandwidth, the server calculates the volume of data the network is able to handle and then adapts the object to that size. For instance, if the object is a JPEG image, the application can adapt it in terms of the image resolution and the JPEG quantization factor to produce a tailored JPEG image with the target size. In the third phase, the tailored object is transmitted to a user. Figure 1.1 is an illustration of the architecture of such a network-aware application, taken from [Bollinger, 1998].



**Figure 1.1, Network-aware Application Architecture**

Obviously, the bandwidth monitoring and estimation are of critical importance to this strategy. Only an accurate estimation of the available bandwidth allows the user request to be met. In several previous studies, bandwidth-monitoring modules were implemented in different ways. In [Bolliger 1998], where the purpose of the application was to send a group of images (90 JPEG images in its experiment) from the server to the client, the author used the bandwidth achieved from one image to estimate the bandwidth for the next. The method is clever since in that way, all the data transmitted are real application data; there is no overhead for bandwidth monitoring. However, not many applications will be like the one described by the author, which had a large group of objects to be sent at one time. Therefore, more general methods are needed.

In [Cheng, 2001], the author used a different method to perform network monitoring and estimation. Since the Tele-Learning system described was intended to provide course materials interactively to students as they browsed the course content, it was expected that significant idle time would pass by between two consecutive media delivery requests. The bandwidth estimation method used in that system was to have the server maintain a session for each client currently active and send testing packets periodically (e.g., every 15 seconds) to the client. The client then calculates the available bandwidth based on the time needed to transmit the testing packet. In this way, each connection maintains a history of bandwidth samples. When a media delivery request is made to the server, the



server estimates the available bandwidth of the future period by using some sort of average of the history of the bandwidth samples. This method suffers from several problems. Since the method uses a relatively long history of the bandwidth to predict the future bandwidth, the estimation result is not very accurate. As the approach assumes the availability of the history of the bandwidth information, the applicability of the method is again restricted to those applications for which a long session is maintained, as in a Tele-Learn system. An even more serious problem is that periodically sending random bits along the network for bandwidth testing purposes when no real data transmission is actually required can be a significant waste of the total bandwidth of networks because it interferes with the traffic of other users running a congestion control protocol.

## **1.5 Thesis Objective**

The subject of this thesis is the design of a new method of bandwidth estimation for network-aware applications. Given a multimedia object to be transmitted and a user-specified time limit, an application will use the first fraction of the time limit to do bandwidth testing. Based on the average and the variance of the bandwidth samples measured during the bandwidth testing period, this method gives an estimation of the available bandwidth, guaranteeing that the transmission of an object will finish within the time limit by a specific confidence level (e.g., 95%). Compared to the methods discussed above, the new method obtains the most up-to-date bandwidth information; the entire traffic including both bandwidth testing and real object transmission is bounded by a user-specified time limit so that bandwidth testing will not occur when no actual transmission is required; and the method has wide applicability.

A challenge in designing this kind of method was how to balance the time for bandwidth testing and the time for data transmission. Since the bandwidth testing involves sending random bits along the path to the client in order to obtain knowledge of the available bandwidth, the time set aside for it is pure overhead. Using too much time for bandwidth testing obviously decreases the amount of time available for real transmission. This

prevents the sender from sending an object of the largest size within the time limit. On the other hand, using too little time for bandwidth testing yields less reliable knowledge of the available bandwidth, and thus poses a risk of exceeding the time limit. If we want to keep the same confidence level for finishing transmission within the time limit, a more aggressive under-estimation of the bandwidth is needed, but such an estimation also prevents us from achieving our goal of sending an object of largest size within the time limit. Thus, an optimal trade-off strategy is required.

To meet the challenge, a statistical model was used to quantify the advantages and disadvantages of bandwidth estimation spending. The new method used this model to determine the proportion of the time limit for bandwidth testing that would yield the largest size of the transmitted object. We will show that the bandwidth testing time chosen by our method is a good approximation of the optimal amount.

## **1.6 Thesis Organization**

In Chapter 2, we will first closely examine of the mechanisms of TCP congestion avoidance and controlling, which are the underlying source for bandwidth fluctuation in terms of the application layer. We also compare our method with those of related research and emphasize the contribution of this thesis. In Chapter 3, we present the mathematical model of bandwidth testing for the simple condition (where only one server exists to serve the client) and give an algorithm to determine the amount of bandwidth testing. Chapter 4 presents the simulation results for the algorithm. In Chapter 5, we extend the model to the condition of a multi-server environment and present the simulation results for the extended algorithm. In Chapter 6, we report the experimental results for real networks. Finally, we draw our conclusions in Chapter 7.

## Chapter 2

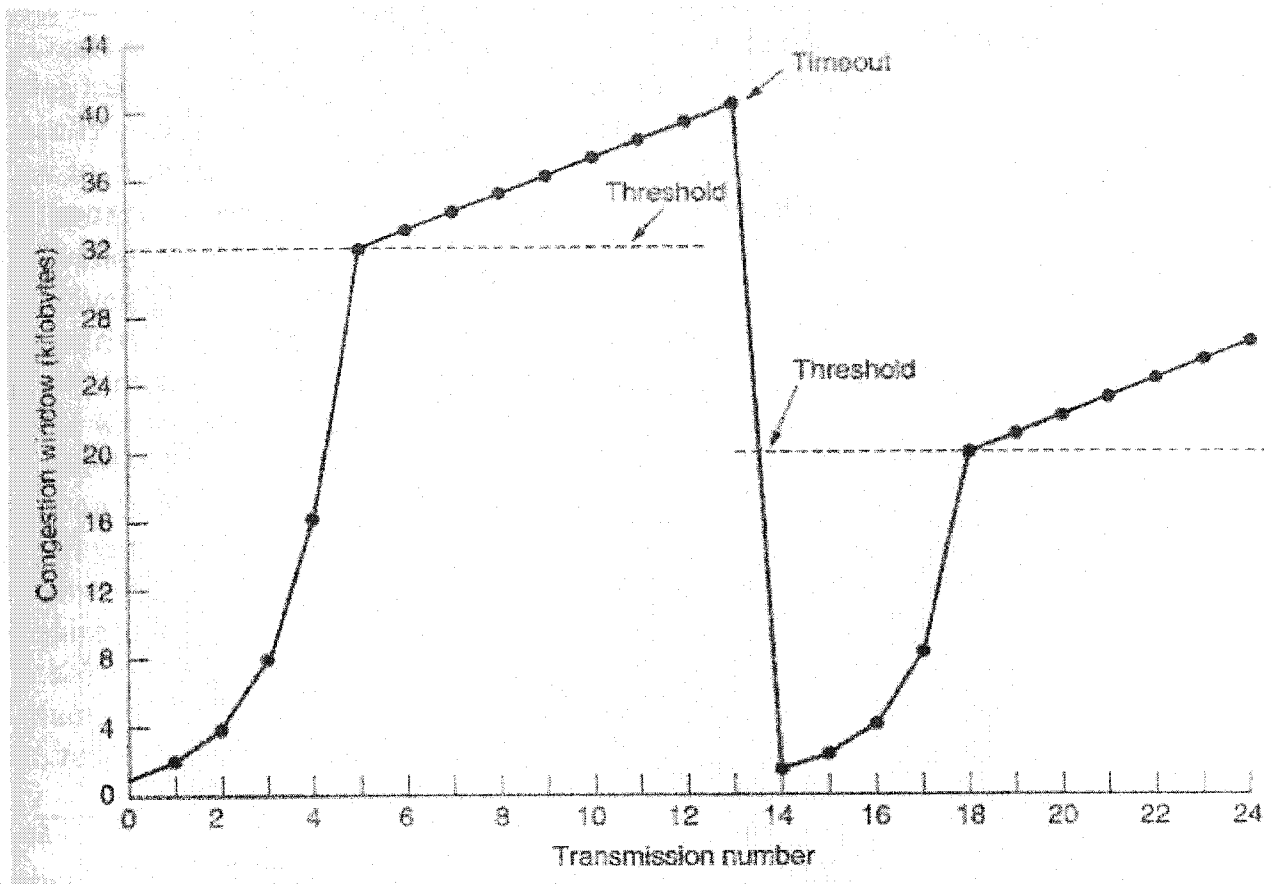
# Background and Related Works

In this chapter, we will take a close look at the bandwidth estimation problem. We will first explain some internal design issues of the TCP protocol, especially the congestion avoidance and control mechanisms, which are the main reason for the difficulty of bandwidth estimation. Then we will survey the related research in bandwidth modeling and estimation and emphasize the main contribution of this thesis.

### 2.1 TCP Congestion Control

Transmission Control Protocol (TCP) [Postel, 1981] [Braden, 1989] is one of the two most widely used protocols (together with IP) in today's Internet. It provides a guaranteed and in-order data stream delivery service on top of the unreliable service of IP. In order for each packet sent out by the sender to be received properly, several factors must be in place. Firstly, the receiver must have enough buffers to store the arrived data yet to pass to higher layer. The TCP protocol achieves this by having the receiver piggyback an *advertise window* on its ACK messages. The advertise window tells the sender how much buffer area is available on the receiver side; thus, the sender should never send out unacknowledged data larger than the advertised window size. Secondly, even if the receiver has enough buffers, the routers on the path may not be able to keep up with the speed at which the sender inserts data into the connection. In that case, we say *congestion* has occurred, and the router begins to drop packets. One design goal of the TCP protocol is to find out and sustain the largest bandwidth on a connection with minimum congestion.

In order to control the congestion, the TCP protocol introduces a parameter called the *congestion window*. The sender will never have outstanding data larger than the advertised window size *and* the congestion window. When a new connection is established, the congestion window is set to the maximum segment size (MSS, a TCP option negotiated between sender and receiver) of the connection, so that the sender can send out only one packet before the acknowledgement (ACK) is received. Each time an ACK is received on time, the sender increases the congestion window by one MSS. This algorithm is called *slow start* [Jacobson, 1988]. Ironically, it is not slow at all. Actually, when operating in the slow start mode, the congestion window grows exponentially to the number of the round trip time (RTT). Suppose at the beginning of the connection the speed at which the sender injects data into the network is much less than the path can handle, every packet will be acknowledged quickly and properly. Therefore, in the first RTT, one packet is sent; in the second, two packets are sent; in the third RTT it is four, and so on. After a few RTTs, the sender will exceed the maximum load the network is able to handle, so a timeout (due to the drop of packets) will happen. At that point, a new slow start process begins (from one packet per RTT), but this time, the sender will not continue the slow start process until another timeout happens. Actually, the TCP protocol also maintains a third parameter called a *Congestion Threshold*. It is initially set to be 64K. Once a timeout happens, the Congestion Threshold is set to half of the congestion window at that time, so when a new round of slow start occurs, the slow start will finish when the congestion window reaches the congestion threshold. After that threshold has been reached, the congestion window grows linearly instead of exponentially (to the RTT) by adding a fraction of the MSS for each ACK. Figure 2 (taken from [Tanenbaum, 1996]) shows the change of the congestion windows size in a typical TCP connection.



**Figure 2.1,** An example of congestion window dynamics in a TCP connection

The mechanisms of slow start and congestion window dynamics first appeared in [Jacobson, 1988]. The other two widely implemented congestion control schemes of the TCP protocol are *fast retransmit* and *fast recovery*. As the congestion control mechanism uses the timeout as the indicator of the loss of packets, and the timeout is usually set to be several times larger than the RTT (in order to avoid a false timeout), it usually takes a relatively long time before the sender realizes that a packet is lost and retransmits it. “Fast retransmit” was designed to address the problem. When using fast retransmit, every time a data packet arrives at the receiver side, the receiver responds with an ACK<sup>3</sup> (acknowledging the last in-order packet), even if that sequence number has already been acknowledged. (This happens when one packet is lost on the way, so

<sup>3</sup> The TCP protocol uses cumulative acknowledgements. The sequence number in an ACK is the next octet that the receiver expects. There are several reasons for this design choice. Firstly, with cumulative ACKs, lost acknowledgements do not necessarily force retransmission. Secondly, compared with a mechanism with NAK, cumulative ACK does not require the receiver to maintain timers. All the decisions about timeout and retransmission are made at the sender side. This is important. As the sender and receiver are separated by the Internet, different packets can take different paths, so the sender is in the better position to make the decision on when to timeout packets.

all the ensuing packets that arrive will ACK the packet before the lost one). In that case, the sender will receive several duplicate ACKs for the same sequence number. This event is a strong indicator that a packet (the one immediately after the last acknowledged) is lost. TCP will retransmit that packet after three duplicate ACKs have been observed at the sender. Figure 2.2 (taken from [Peterson, 2000]) illustrates fast retransmit based on duplicate ACKs.

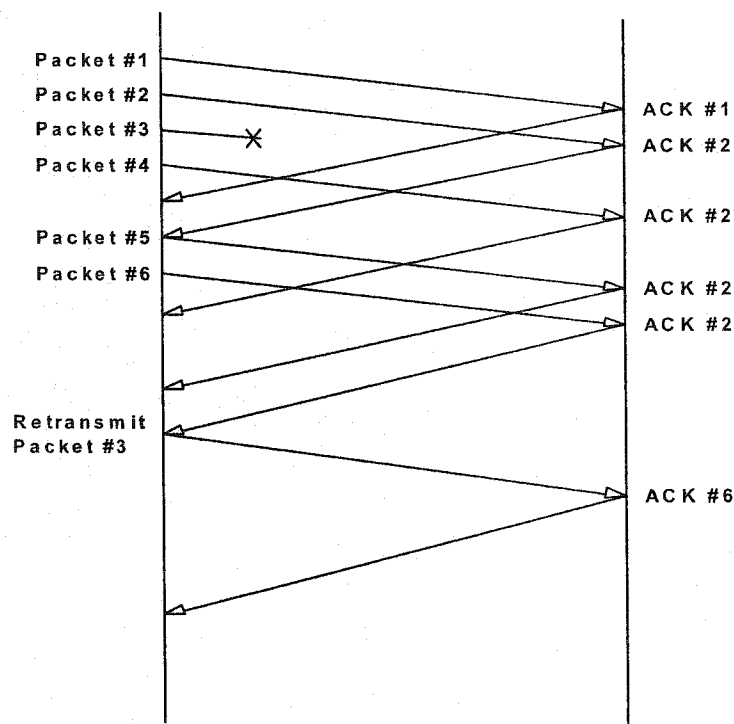


Figure 2.2, Fast retransmit based on duplicate ACKs

A technique related to fast retransmit is *fast recovery*. When the fast retransmit mechanism signals congestion, rather than shrink the congestion window size all the way back to that of one packet and run a slow start, the fast recovery mechanism uses the ensuing duplicate ACKs to clock the sending of new packets until the retransmitted packet is acknowledged. In other words, since the receiver can generate a duplicate ACK only when a packet has arrived (i.e., when the packet has left the network and is no longer consuming network resources), the sender can safely inject a new packet into the network without worrying that it will create more congestion. The fast recovery terminates when the sender receives the ACK for the retransmitted packet. The sender

then sets the congestion window to the congestion threshold and moves on. [Stevens, 1997] and [Allman, 1999] summarized the four schemes we have discussed so far, which are now part of the requirements for all the TCP implementations.

Over the last decade, many new congestion control mechanisms have been proposed. [Hoe, 1996] proposed changes to the fast retransmit scheme to enable it to quickly recover from multiple packet losses during the slow start period. [Mathis, 1996] added “Selective Acknowledgement” as an option of the TCP protocol. This mechanism also helps addressing the problem of slow recovery from multiple packet losses. [Lin, 1998] proposed a modification to the sender behavior during the fast recovery period, improving both the throughput of individual TCP connection and the fairness among the TCP connections sharing network links.

## 2.2 TCP Congestion Avoidance

All the mechanisms we discussed in Section 2.1 are based on congestion control. They generally try to inject more and more traffic (quickly or slowly) into the connection until congestion is sensed (by timeout or duplicate ACKs), and then back off quickly to alleviate the situation. Other mechanisms have been proposed to *avoid* congestion from happening in the first place. [Ramakrishnan, 1990] and [Floyd, 1993] proposed putting additional functionality into the routers to assist the sender to anticipate congestions. [Brakmo, 1995] introduced TCP Vegas, a new congestion avoidance mechanism working purely in the end nodes. So far, TCP Vegas has received most supports of these new schemes.

The basic idea of TCP Vegas is as follows. As the sender expands the congestion window gradually, it also monitors the actually achieved throughput in the connection. If the congestion window grows constantly, while the throughput begins to flatten, then bandwidth limitation of the connection has been reached. Although congestion may not have happened yet (no timeout has occurred), excessive packets are queuing up in the buffer of the intermediate routers. If the sender continues to pump more packets into the

connection during each RTT, congestion will eventually occur. TCP Vegas tries to maintain the right amount of extra data in the network. The author argued that with too much extra data, the delay will increase, and congestion will occur; on the other hand, with too little extra data, the mechanism cannot respond rapidly to the transient increases in the available bandwidth.

TCP Vegas maintains two parameters in order to measure the “extra data”. First, it sets  $\text{ExpectedRate} = \text{CongestionWindow} / \text{BaseRTT}$ , where  $\text{BaseRTT}$  is the minimum of all the round trip time measured on the connection (usually the first RTT). Second, it maintains the parameter  $\text{ActualRate} = \text{DataSent} / \text{SampleRTT}$ , where  $\text{DataSent}$  and  $\text{SampleRTT}$  are derived from observing one “distinguished packet”. The  $\text{SampleRTT}$  is the time between when the distinguished packet is sent and when its acknowledgement arrives.  $\text{DataSent}$  is the number of bytes sent out at the sender during that period. TCP Vegas also defines two thresholds  $\alpha$  and  $\beta$  ( $\alpha < \beta$ ), roughly corresponding to having too little and too much extra data in the network. During each RTT, it compares  $\text{Diff} = \text{ExpectedRate} - \text{ActualRate}$  to  $\alpha$  and  $\beta$ . If  $\text{Diff} < \alpha$ , it increases the congestion window linearly during the next RTT; if  $\text{Diff} > \beta$ , it decreases the congestion window linearly during the next RTT; otherwise, it leaves the congestion window unchanged. The details on how to determine  $\alpha$  and  $\beta$  can be found in [Brakmo, 1995]. By avoiding congestion in first place, TCP Vegas was reported to have achieved between 40% to 70% better throughput than TCP Reno, which is the most widely used TCP implementation (distributed with BSD Unix) [Brakmo, 1995]. We will next summarize the features supported in three major implementations of the TCP protocol<sup>4</sup>.

---

<sup>4</sup> TCP Tahoe, also known as the BSD Network Release 1.0 (BNR1), was implemented by Jacobson, with the mechanisms described in [Jacobson, 1988]. It was released with the BSD UNIX version 4.3. TCP Reno, also known as the BSD Network Release 2.0 (BNR2), includes all the mechanisms described in [Stevens, 1997]. As the most widely used TCP implementation today, TCP Reno is the current de facto Internet standard.



	<b>TCP Tahoe</b>	<b>TCP Reno</b>	<b>TCP Vegas</b>
<b>Slow start</b>	Yes	Yes	Yes
<b>Additive Increase /Multiplicative Decrease</b>	Yes	Yes	Yes
<b>Fast Retransmit</b>	Yes	Yes	Yes
<b>Fast Recovery</b>	No	Yes	Yes
<b>Delayed ACK</b>	No	Yes	Yes
<b>Congestion Detection/Avoidance</b>	No	No	Yes

*Table 2.1, Features supported in major TCP implementations*

## 2.3 TCP for wireless

In recent years, as wireless network applications began to take off, the TCP protocol has been facing the challenge of a heterogeneous wired/wireless network environment. Researchers have pointed out several shortcomings of TCP's behavior in such an environment [Tsaoussidis, 2002]. Firstly, the TCP protocol usually assumes that all packet losses are caused by congestion, which is generally true in the wired network world. Therefore, when the TCP sender senses a packet loss, it always backs off and reduces the sending rate. However, in a wireless world, transient packet losses due to corrupted data are very common. Mislabeling these transient losses as "congestion" greatly reduces the throughput of the TCP protocol over wireless links. Secondly, the traditional TCP protocol does not control the tradeoff between performance measure (like throughput) and energy consumption. When a wireless link is experiencing burst errors (e.g., during handoffs and fading channels), the traditional TCP usually expends much energy in wasteful retransmission effort.

Many studies have been performed to improve the situation, including [Balakrishnan, 1995], [Balakrishnan, 1997], [Haas, 1997], [Ramakrishnan, 1999], [Parsa, 1999] and [Tsaoussidis, 2000]. [Balakrishnan, 1997] proposed that an Explicit Congestion Notification (ECN) be added to the IP protocol. The TCP protocol can be changed to

trigger congestion control only after getting an ECN from an intermediate router. [Tsaoussidis, 2000] proposed a mechanism called “TCP-probing” to address the energy-saving problem. In TCP-probing, when a data segment is detected as lost, the sender, instead of re-transmitting the lost packet again and again, initiates a “probe cycle”, during which data transmission is suspended and only probe segments (headers without payloads) are sent. A lost probe segment or its acknowledgement re-initiates the “probe cycle”, suspending data transmission for the duration of the error (due to fading channel or handoff). As a probe segment is short and only one probe segment is in transit at any time, energy is saved during the probing-cycle. Once one probe segment makes the round trip, the data transmission resumes.

## 2.4 Related Research

The discussion above has shown that the TCP protocol, particularly its congestion control/avoidance mechanism, is complex and fluid. The main purpose of these mechanisms is to achieve the maximum available bandwidth for individual TCP flows with minimal the congestion, and to maintain fairness among all TCP flows. The effective bandwidth realized on a TCP connection is affected not only by the network traffics, the processing power and buffer size of routers, but also by the specific mechanisms implemented on the host computers.

We will now describe some related research in modeling and estimating the throughput of the TCP protocol. Studies of TCP bandwidth modeling fall into two categories. In the first category, the authors try to model the TCP throughput in the context of the entire network. [Lakshman, 1997] first modeled the network as a system with one single bottleneck link (with a specific capacity and a FIFO buffer of a specific size). This model assumed a constant number of connections sharing the bottleneck link and modeled all delays other than the service time and queuing at the bottleneck into a single “propagation delay”. The author modeled the cyclical evolution of the TCP congestion window (as implemented in TCP Tahoe and Reno) and used Markovian analysis to develop a closed-form expression for the throughput of the TCP connections. [Kumar,

1998] used a similar system model and parameters, but modeled more versions of TCP mechanisms, like TCP-NewReno and TCP-Vegas. [Casetti, 2000] extended the previous two models by adding a new layer of model to mimic the application-level behavior, such as the ON-OFF activity of a web server responding to clients' request. During an OFF period, the application is assumed to be idle; during an ON period, the application is sending data over a TCP connection. This model represents the time spent in each state as a random variable with negative exponential distribution.

The common goal of these studies that adopts a system perspective is to understand the aggregate performance of the TCP connections and the impact of different schemes on the TCP throughput. For example, [Lakshman, 1997] concluded that for multiple connections sharing a bottleneck link, the TCP protocol is generally unfair to connections with higher round-trip delays. [Kumar, 1998] compared the throughput performance of different TCP implementations in a local network environment with a lossy wireless link. However, the throughput models described in these studies are not applicable to network-aware applications, because these models are built on parameters of the network systems like bottleneck capacity, router buffer size, and the number of concurrent TCP connections. This information is not available to a host running network-aware applications.

The second class of research tries to model the TCP throughput from the perspective of the TCP sender. [Jacobson, 1988] first proposed to model the available bandwidth with TCP's congestion window  $cwnd$  and round trip time  $rtt$ . The rationale behind this method is that when there are no packet losses, a TCP sender sends out a window of packets with size  $cwnd$  for each round trip time. Therefore, the product of the average  $cwnd$  and average  $rtt$  is an approximation of the TCP throughput. However, the method does not include the TCP behavior when there are packet losses. When packet losses happen, TCP's sending rate is no longer governed by the congestion window. Depending on whether packet losses are recovered by timeout or fast retransmit, this recovering period can be shorter or longer. However, by ignoring this factor, Jacobson's method is able to provide only a rough approximation. Another problem associated with

this method is that the product of  $cwnd$  and  $rtt$  measures the sender's throughput, instead of how much data is received by receiver. In the context of an application, obviously the later makes more sense.

[Mathis, 1997] attempted to use network level metrics to model the TCP throughput. The metrics used by this model include the packet loss rate  $p$ , the round-trip time  $rtt$ , and the packet size  $mss$ . Like the model of [Jacobson, 1988], it does not model timeout periods. It models the TCP congestion window with a cyclical dynamics, with each cycle ending at a packet loss. The probability of packet loss is modeled as a constant  $p$ . The main advantage of this method is that it uses network level metrics; therefore, its methodology could be extended to model other non-TCP protocols like reliable multicast. However, it shares the problem of [Jacobson, 1988] since it does not model the period in which the sending rate is not controlled by the congestion window.

[Padhye, 1998] extended [Mathis, 1997] by modeling timeout periods. In addition to the metrics used in [Mathis, 1997], Padhye added the number of timeouts and the average duration of timeouts into his model. By adding TCP level metrics like timeout, Padhye's model lost the generality of using only network level metrics. However, this model's accuracy was expected to be better than the previous two. [Bolliger, 1999] analyzed the accuracy of all these three models by using large-scale real network traces. For each trace (transfer of 1MB data among 35 hosts across Europe and North American), the author computed the predicted bandwidth according to the three models and compared it to the real bandwidth achieved. Padhye's model was reported as the most accurate, especially when timeout events occurred during the transfer. The other two models over-estimated the actual bandwidth.

However, gaps still need to be filled before these models can be applied to the bandwidth estimation in a real application. Firstly, all these models are based on a theoretical analysis. For example, both Mathis's and Padhye's models assume a TCP connection has existed for a time  $t$ . They give an estimation of the steady state throughput of that TCP connection based on the metrics observed in this TCP connection in time  $t$ .

However, these models do not answer the question of how long the  $t$  should be in order to get a relatively accurate estimation or of how long the estimation is valid. For this reason, no clearly defined application programming interfaces (APIs) based on these models exist to date. Secondly, as all these models use the knowledge of the TCP congestion control and avoidance mechanisms, the models depend on specific implementations of the TCP protocols. For example, both Mathis's and Padhye's models are built on the packet loss rate  $p$  because they all assume a Reno-style congestion window dynamics, in which the TCP sender generally approaches the bandwidth limit of the network and backs off when light congestion occurs (as signalled by packet losses). However, in a TCP Vegas implementation, since the sender tries to avoid congestion in the first place, it is very likely that no packet loss occurs during the entire life of a TCP Vegas connection. In such cases, neither of the two models is applicable. Therefore, bandwidth estimation models based on transport or network layers need continuous changes or at least re-evaluations as new mechanisms are proposed and added to TCP implementations. Finally, all the models we have described operate on the TCP sender. The throughput modeled represents how much data are sent out instead of how much data are received by the receiver, even though the amount of data received is more relevant to a real application.

## 2.5 Contributions

For this thesis, we did not try to create a general model of the TCP throughput like those in the previous models discussed above. Instead, we create a ready-to-use bandwidth estimation algorithm for network-aware applications. The algorithm works completely in the application layer and mainly on the receiver side. It assumes that one or more sender(s) and one receiver want to make a single transmission of a media object within a specified time limit. Our algorithm slices the entire time limit period and uses the first fraction of these slices for bandwidth testing. It makes an estimation for the entire time limit based on the statistical characteristics of the bandwidth samples. Although the algorithm does not have access to the internal metrics of the transport and network layers, it is able to confidently make a relatively accurate estimation of the available bandwidth

in the time limit period. Moreover, unlike the previous methods, which have trouble in dealing with specific TCP implementations (e.g., TCP Vegas), we expect the algorithm to work for all the implementations of the TCP protocol, since it is independent of the TCP's internal metrics. Throughout our design, the algorithm has been kept simple enough to be implemented in a wide range of devices with limited computation power. We believe the algorithm we will describe will directly benefit the development of network-aware applications.

## Chapter 3

# Single Server Bandwidth Estimation

### 3.1 The Problem and Assumptions

In this chapter, we will develop an application layer bandwidth estimation algorithm for network-aware applications. We assume that one server and one client need to make a one-time transmission of a multimedia object (from server to client). No previous bandwidth information is available for the connection between the server and client. The user on the client side has specified a time limit  $T$  for the transmission. Neither exceeding the time limit  $T$  nor under-utilizing the time  $T$  is desirable. The first fraction  $t$  of  $T$  will be used for bandwidth testing to obtain the bandwidth estimation  $B$  for the future period  $T - t$ . The server will then adapt the multimedia object to be exactly of the size  $(T - t) \cdot B$  and then transmit it to the client.

Over-estimation of available bandwidth will yield a target object size larger than the network can handle and therefore, will make the transmission exceed the time limit. On the other hand, under-estimation will under-utilize the time limit and therefore, deliver an object that could have had better quality (we assume an object with larger size always has better quality). Our problem is how to determine the appropriate  $t$  and use the bandwidth testing information obtained in  $t$  to make the bandwidth estimation so that to deliver as much data as possible to the client within the time limit by a probability (confidence level) of  $\alpha$ .

To simplify the model, we ignore the time needed for the adaptation of a media object, since objects can be stored with multiple resolution levels or be dynamically adapted in a short time, given a fast processor.

We will use time slices of equal length to do the bandwidth testing. Suppose each time slice has length  $t_s$ , we then have  $T/t_s$  time slices. Each time slice has a bandwidth value  $x_i = \frac{C_i}{t_s}$  ( $i=1,2\dots T/t_s$ ), where  $C_i$  is the count of bytes received during the  $i^{\text{th}}$  time slice. These bandwidth values can be viewed as random variables. We obtain the bandwidth value of the first  $t/t_s$  slices and then use the mean and variance of these samples to estimate the average of the  $T/t_s$  population.

## 3.2 Notations

We first define the following variables, which we will use in our discussion.

### DEFINITION 3.1: Basic Notation

- $T$  is the time limit for the multimedia object transmission specified by a user.
- $t$  is the time used for bandwidth testing.
- $t_s$  is the length of each time slice used for obtaining bandwidth samples.
- $N = \frac{T}{t_s}$ ,  $N$  is the number of time slices in period  $T$ , which generates our bandwidth *population*,  $X_1, X_2, \dots, X_N$ .
- $n = \frac{t}{t_s}$ ,  $n$  is the number of time slices in period  $t$ , which generates our bandwidth *samples*,  $x_1, x_2, \dots, x_n$ .
- $\mu = \frac{\sum X_i}{N}$ ,  $\mu$  is the average of the  $N$  bandwidth population, the average available bandwidth we are trying to estimate.



- $\sigma^2 = \frac{\sum (x_i - \mu)^2}{N}$ ,  $\sigma^2$  is the variance of the  $N$  bandwidth population.
- $\bar{x} = \frac{\sum x_i}{n}$ ,  $\bar{x}$  is the average of the  $n$  bandwidth samples  $x_1, x_2, \dots, x_n$ .
- $s^2 = \frac{\sum (x_i - \bar{x})^2}{n-1}$ ,  $s^2$  is the variance of the  $n$  bandwidth samples.  $\square$

In the context of a network-aware application,  $t_s$  is a system parameter (we will discuss the choice of  $t_s$  in Chapter 6). Since  $T$  is set by user,  $N = \frac{T}{t_s}$  can therefore be viewed as a fixed number. Our bandwidth estimation algorithm has two tasks: to determine  $n$ , the number of samples to take; and, given  $n, N, \bar{x}, s^2$ , to estimate  $\mu$ .

### 3.3 The Algorithm

In statistical practice,  $\bar{x}$  is usually used as an estimation of  $\mu$ . Actually, given  $n, N, \bar{x}$  and  $s^2$ , the random variable  $\mu$  has a probability distribution centred at  $\bar{x}$ . If we assume that the random variable  $x$  (the true bandwidth available in any time slice) is a normal variable (i.e., it follows the normal distribution), then according to statistical theory,

$$d = \frac{\mu - \bar{x}}{\frac{s}{\sqrt{n}} \cdot \sqrt{\frac{N-n}{N-1}}} \quad (3.1) \text{ is a continuous random variable following the "Student's t-}$$

Distribution" with degree of freedom  $n$  (number of samples) [Harnett, 1982].<sup>5</sup> We use the character  $d$  instead of  $t$  to specify the t-Distribution random variable to avoid confusion with the time  $t$ , which we have already defined.

We point out that the assumption of normality of  $x$  is a rough approximation of real

---

<sup>5</sup> Mathematically, the random variable  $d$  is defined as a standardized normal variable  $z$  divided by the square root of an independently distributed chi-square variable, which has been divided by its degree of freedom.  $d = z / \sqrt{\chi^2 / n}$ .

world bandwidth<sup>6</sup>. As well, we further assume that during the entire time limit period, the bandwidth random variable is stationary; therefore, the first fraction of the samples is a good representative of the entire population. This assumption is partially justified by recent findings of [Balakrishnan, 1997B] and [Bolliger, 1999]. For example, Bolliger compared the throughput of the first 50% of the time period of a TCP connection to that of the second half. He found that in 80% of connections, the difference was within a factor of 1.5, and in 90% of the connections, the difference was within a factor of 2. Moreover, when building our model, we ignored the correlation between successive samples, mainly to simplify our model and our algorithm so that they could be used in network-aware applications in a real-time style. We will check the validity of the algorithm developed for this model in our real network experiments (Chapter 6).

Thus, given  $n$ ,  $N$ ,  $\bar{x}$ ,  $s^2$ , and t-distribution, we have the probability distribution of  $\mu$ .

$$\mu = \bar{x} + d \cdot \frac{s}{\sqrt{n}} \cdot \sqrt{\frac{N-n}{N-1}} \quad (3.2)$$

This means that when we take  $n$  bandwidth samples, we can expect the average bandwidth within period  $T$  to be a random variable with a probability distribution following Equation (3.2).

With the distribution of  $\mu$ , we can infer the  $\alpha$  confidence interval for  $\mu$  as follows,

$$\bar{x} - d_{(\alpha,n)} \cdot \frac{s}{\sqrt{n}} \cdot \sqrt{\frac{N-n}{N-1}} \leq \mu \quad (3.3)$$

Equation (3.3) states that the value of  $\mu$  is greater or equal to  $\bar{x} - d_{(\alpha,n)} \cdot \frac{s}{\sqrt{n}} \cdot \sqrt{\frac{N-n}{N-1}}$  with a probability of  $\alpha$ . With this observation, we define the bandwidth estimation as follows:

---

<sup>6</sup> Fortunately, the t-distribution is known to be "robust", which means that the assumption of normality of  $x$  can be relaxed as long as  $n$  is relatively large.

**DEFINITION 3.2: Safe Bandwidth Estimation**

We define  $\mu_{est} = \bar{x} - d_{(\alpha,n)} \cdot \frac{s}{\sqrt{n}} \cdot \sqrt{\frac{N-n}{N-1}}$  to be a safe estimation of the future bandwidth.

According to our discussion above, we have a confidence of  $\alpha$  that the actual bandwidth will be higher.  $\square$

We have mentioned that  $\mu$  is a random variable centred at  $\bar{x}$ . From Definition 3.2, we can see that in order to provide a confidence level that will conform to the time limit, we actually have to under-estimate the bandwidth. This under-estimation has its cost; if we use  $\mu_{est}$  as the future bandwidth estimation to determine the target size of the media object, the actual transmission will most likely under-utilize the remaining time  $T-t$ . It can be shown that the under-utilized time is on the average  $(T-t) - \left(\frac{\mu_{est} \cdot (T-t)}{\mu}\right)$ ,

or  $(T-t) \left(1 - \frac{\mu_{est}}{\mu}\right)$ . Therefore, we define the under-utilized time  $\tilde{T}$  as follows.

**DEFINITION 3.3: Under-utilized Time**

$\tilde{T}(n) = \left(1 - \frac{\mu_{est}}{\bar{x}}\right) (T-t) = \frac{d_{(\alpha,n)} \cdot \frac{s}{\sqrt{n}} \sqrt{\frac{N-n}{N-1}}}{\bar{x}} \cdot (T-n \cdot t_s)$ . It determines the average

under-utilization time with the safe estimation.  $\square$

Given the definitions above, we are ready to determine how much bandwidth testing to perform in order to maximize the data delivered to a user while conforming to the time limit requirement by confidence level  $\alpha$ . We first prove an important property of  $\tilde{T}(n)$  in Theorem 3.1.

**Theorem 3.1:** For  $n < \frac{3}{4}N$ ,  $\tilde{T}(n)' < 0$ ,  $\tilde{T}(n)'' > 0$ . This means when  $n$  is relatively small compared to  $N$ ,  $\tilde{T}(n)$  decreases as  $n$  grows, and the rate of its decrease diminishes.

**Proof:** We write  $\tilde{T}(n)$  as follows:

$$\begin{aligned}\tilde{T}(n) &= \frac{d_{(\alpha,n)} \cdot \frac{s}{\sqrt{n}} \cdot \sqrt{\frac{N-n}{N-1}}}{\bar{x}} \cdot (T - n \cdot t_s) = \frac{s}{x \cdot \sqrt{N-1}} \cdot d_{(\alpha,n)} \cdot \sqrt{\frac{N}{n}-1} \cdot (T - n \cdot t_s) \\ &= c \cdot f(n) \cdot g(n) \cdot h(n)\end{aligned}$$

Since we view  $\tilde{T}(n)$  as a function of  $n$ , we will view  $s$  and  $\bar{x}$  part of the constant  $c$ . We

$$\text{denote } f(n) = d_{(\alpha,n)}, \quad g(n) = \sqrt{\frac{N}{n}-1}, \quad h(n) = (T - n \cdot t_s).$$

$$\text{Therefore } g(n)' = -\frac{N}{2 \cdot n^2 \cdot \sqrt{\frac{N}{n}-1}}$$

$$\text{and } g(n)'' = \frac{N(3N-4n)}{4 \cdot n^4 \cdot \left(\frac{N}{n}-1\right) \sqrt{\frac{N}{n}-1}}.$$

For  $n < \frac{3}{4}N$ ,  $g(n)' < 0$ , and  $g(n)'' > 0$ .

We also have  $h(n)' = -t_s < 0$ , and  $h(n)'' = 0$

Thus, if we denote  $l(n) = g(n) \cdot h(n)$ , then for  $n < \frac{3}{4}N$ ,

$$l(n)' = g(n)' \cdot h(n) + g(n) \cdot h(n)' < 0$$

$$l(n)'' = g(n)'' \cdot h(n) + 2 \cdot g(n)' \cdot h(n)' + g(n) \cdot h(n)'' > 0$$

Next, we need to show that the same properties hold for  $f(n) = d_{(\alpha,n)}$ . As  $f(n)$  is a function with a complex representation, a strict mathematical proof is beyond the scope of this thesis. Here we use a numerical hypothesis to serve the purpose. For every  $\alpha$  values we will use in this thesis, we calculate value  $d_{(\alpha,n)}$  for  $n=1 \dots 500$ , and verify that  $d_{(\alpha,n)} < d_{(\alpha,n-1)}$  and  $d_{(\alpha,n)} - d_{(\alpha,n-1)} > d_{(\alpha,n-1)} - d_{(\alpha,n-2)}$  (Part of the  $d_{(\alpha,n)}$  values are presented in Appendix 2). Therefore, we have similar results for  $f(n)$  on  $f(n)' < 0$ , and  $f(n)'' > 0$ .

Therefore, for  $n < \frac{3}{4}N$ ,  $\tilde{T}(n)' = f(n)' \cdot l(n) + f(n) \cdot l(n)' < 0$ ,

$$\tilde{T}(n)'' = f(n)'' \cdot l(n) + 2 \cdot f(n)' \cdot l(n)' + f(n) \cdot l(n)'' > 0. \quad \square$$

A direct result of Theorem 3.1 is a property of the expected object size  $V(n)$ . We first give the definition.

**Definition 3.4: Expected Object Size:**

$V(n) = \mu_{est} \cdot (N - n) \cdot t_s$ . It is simply our estimated available bandwidth times the remaining time within the time limit. This is the object size the client will request from the server if we take  $n$  samples for bandwidth testing.  $\square$

Theorem 3.1 means that  $\tilde{T}(n)$  decreases as  $n$  grows, and that the rate of the decrease diminishes when  $n$  is relatively small compared to  $N$ . The implication is that as we take more bandwidth samples, we can estimate the future bandwidth more accurately with the same confidence level  $\alpha$ , thus reducing the cost of time under-utilization. We denote the benefit of each new sample as  $\Delta\tilde{T}_n = |\tilde{T}_{n+1} - \tilde{T}_n|$ . As shown by Theorem 3.1,  $\Delta\tilde{T}_n$  decreases as  $n$  grows. On the other hand, the cost of each new sample is a constant  $t_s$ , the time slice. Therefore, our expected object size will first increase as we take more bandwidth samples (since at first  $\Delta\tilde{T}_n > t_s$ ), but the increasing rate of  $V(n)$  diminishes (because as  $\Delta\tilde{T}_n$  decreases,  $\Delta\tilde{T}_n - t_s$  diminishes). Then after passing a threshold,  $V(n)$  begins to decrease (since now  $\Delta\tilde{T}_n < t_s$ ), and the decreasing rate of  $V(n)$  grows with  $n$  (because as  $\Delta\tilde{T}_n$  decreases,  $t_s - \Delta\tilde{T}_n$  increases with  $n$ ). Therefore, to maximize the target object size, we can determine the optimal value  $n$  by comparing  $V(n)$  with  $V(n-1)$  each time a new bandwidth sample is obtained. As long as  $V(n) > V(n-1)$ , we should continue bandwidth testing. The pseudocode in Algorithm 3.1 illustrates the method. The algorithm uses our safe bandwidth estimation to calculate  $V(n)$  in each time slice. The process of bandwidth testing continues until the  $V(n)$  value begins to drop.

**Algorithm 3.1:** Single Server Bandwidth Estimation

Obtain samples  $x_1, x_2, x_3$ ;

Calculate  $V(1)$  and  $V(2)$ ;

$n \leftarrow 2$ ;

while ( $V(n) > V(n-1)$ ) {

$n \leftarrow n+1$ ;

    Obtain sample  $x_{n+1}$ ;

    Calculate  $V(n)$ ; }

return  $\mu_{est}$ ;  $\square$

## Chapter 4

# Simulation of Single Server Algorithm

In Chapter 3, we described an algorithm for bandwidth testing and estimation in a single server environment. This algorithm has two main features. First, given a number of bandwidth samples collected, the algorithm uses an under-estimation quantified by  $t$ -distribution to serve as the future bandwidth estimation. Second, the algorithm calculates the expected object size in each time slice. Since statistically this expected object size has a single maximum value, the algorithm uses a drop in the expected object size as the termination condition to stop bandwidth testing and start object transmission.

In this chapter, we use simulation to verify the effectiveness of the algorithm. We focus on two things: first, on how well the under-estimation satisfies the user specified time limit confidence level; and second, as the algorithm dynamically determines how many samples should be used for bandwidth testing, we want to see how the number of samples it chooses will approximate the optimal. Because too many testing samples waste a significant portion of the time limit and leave little time for real transmission, while too few testing samples will make an under-estimation too aggressive, the optimal amount of bandwidth testing should finally yield the largest portion of the time limit that can be used for real data transmission. To measure how well the algorithm approximates the optimality, we will compare it against a straightforward approach – using a fixed number of bandwidth testing samples. We will choose a series of different fixed numbers, use them to perform bandwidth testing, and compare the real transmission time yielded in this way to the real transmission time yielded by our algorithm.

In our simulation, we model the bandwidth in each time slice as a normal random variable with stationary mean and variance over the entire time limit period. Bandwidth samples are treated as completely independent random variables, with zero correlations between successive samples.

## 4.1 Overview of Simulation Experiments

We now describe our simulator. Figure 4.1 is a typical screen capture of the simulator at work<sup>7</sup>. It has a user interface to allow the experimenter to set the parameters of experiments. First, the experimenter can choose the statistical characteristics of the channel. He or she sets the mean and standard deviation of the bandwidth samples through two text-boxes in the bottom right area (labelled “Mu” and “Sigma”) of the simulator. Second, the experimenter can choose the number of time slices. In a real application, the user will set a time limit for media delivery. The total number of time slices is the time limit divided by the time of each time slice. For instance, if we simulate that the end user has chosen to transmit an object within 25 seconds, and the system has designated each sampling time slice to be 0.05 seconds, then the total number of time slices is 500. The total time slice is set in the “Parameter” menu. Finally, the experimenter can set the confidence level  $\alpha$  for the time limit in the “Parameter” menu as well.

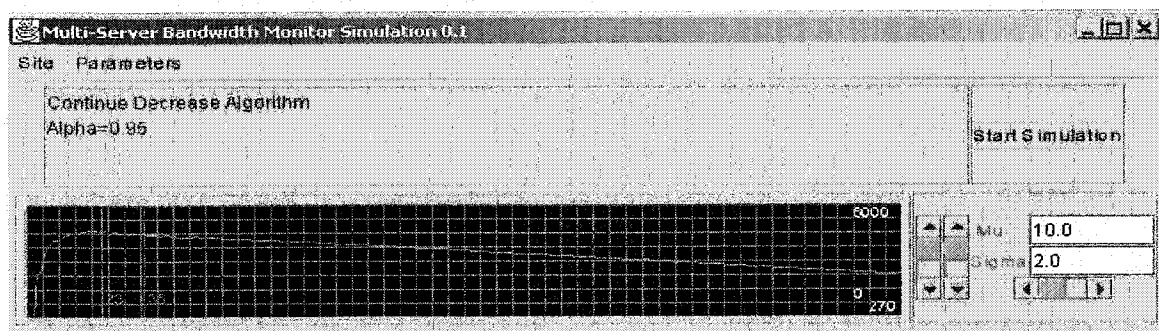


Figure 4.1, Screen capture of simulator in work

<sup>7</sup> We actually implemented one simulator that can be used for both single-server and multi-server experiments. Therefore the title bar of the window in Figure 4.1 shows “Multi-server Bandwidth Monitor Simulation”. In this chapter, we will set the number of servers to one.



After setting the parameters, the experimenter pushes the “Start Simulation” button on the top right corner of the window to begin a simulated run. The simulator simulates the operation of the 500 time slices in two phases, as in a real application. Phase One is the network-testing phase. In each slice, the simulator generates a random number (using a normal distribution generator with the mean and standard deviation specified by the experimenter). This random number simulates the bandwidth observed in the real world for this slice. Then it calculates the combined  $V$  value as defined in Chapter 3

$$(V(n) = \mu_{est} \cdot (N - n) \cdot t_s = \left( \bar{x} - d_{(\alpha, n)} \cdot \frac{s}{\sqrt{n}} \cdot \sqrt{\frac{N - n}{N - 1}} \right) \cdot (N - n) \cdot t_s ). \quad \text{It uses a slightly}$$

modified version of Algorithm 3.1 (see next section) to determine whether the bandwidth testing should continue or not.

When the bandwidth testing is finished, the simulator enters Phase Two, the real transmission, using the last  $V(n)$  as the target object size. We will assume the server will tailor the object and begin to send an object of the target size immediately. The simulator continues to generate random numbers for the channel to simulate the real transmission. In each time slice, the simulator decreases the size of the remaining object on the channel and checks whether it reaches zero. When the remaining object size does reach zero, the real transmission on that channel is finished. If the object size fails to drop to equal to or below zero at the end of the total time slices number, this run of simulation has failed to deliver the object within the user-specified time limit.

When a run of simulation finishes, the simulator reports several results. First, it reports whether this run has succeeded in delivering the object within the time limit. Second, the simulator stored the bandwidth numbers in all the slices. After the run is finished, the simulator recalculates the  $V(n)$  values for all the time slices ( $n=1\dots 500$ , in the example we discussed), and plots it in a graph, like the red curve over the black background in Figure 4.1. It also draws a red vertical line at the place where  $V(n)$  is largest over all slices, and a light red line at the place where our algorithm finished bandwidth testing and began real transmission. As the example in Figure 4.1 shows, the simulator models the bandwidth of slices as normal random number with a mean of 10.0 and a standard

deviation of 2.0. The  $V(n)$  value is maximum when  $n=23$ , and the algorithm finishes bandwidth testing at  $n=36$ .

## 4.2 Moving Average Method

Up to now, we have omitted an important detail in the implementation of the algorithm. As we discussed in Chapter 3, “statistically” (i.e. if we view  $\bar{x}$  and  $s$  as constant), the  $V(n)$  value has only one maximum. It continues to grow until the maximum is met and begins to drop all the way after that. Based on this factor, we can stop bandwidth testing and declare that we have found the largest  $V(n)$  once we observe a drop in its value. However, when taking the random effects of  $\bar{x}$  and  $s$  into consideration,  $V(n)$  no longer has one single local maximum. As shown in Figure 4.1, although the general trend of  $V(n)$  is to grow to a maximum rapidly and drop gradually, the curve is more or less serrated. (The larger the standard deviation of bandwidth random variable, the more serrated it is.) Therefore, if we simply stop bandwidth testing when we observe one decrease in  $V(n)$ , the method will yield a weaker estimation of target object size.

We tried a couple of methods to address the problem. The method used in the experiment shown in Figure 4.1 can be called the “continuous decrease method”. Instead of stopping bandwidth testing after observing one decrease in  $V(n)$ , this method requires a number of continuous decreases (actually four in the case of Figure 4.1) in  $V(n)$  before stopping bandwidth testing. The rationale behind this method is that if we observe several continuous decreases in  $V(n)$ , we are more sure that the random effects of  $\bar{x}$  and  $s$  have been dominated by the other parts of the formula of  $V(n)$ . Therefore, we can judge that  $V(n)$  has already entered the downward slope, and the bandwidth testing should stop.

The problem with the “continuous decrease method” is that it often detects the downward slope of  $V(n)$  too late. As shown in Figure 4.1, the curve of  $V(n)$  has the largest value at  $n=23$  and enters the downward slope at that point. However, only at  $n=36$  when four continuous decreases of  $V(n)$  were observed.

The method we finally adopted is slightly different from the “continuous decrease method”. We call it the “moving average method”. It keeps a moving average of the last four  $V(n)$  values, and compares the “moving average” of the current slice with the previous one. The average of the several past  $V(n)$  values helps to smooth out the serrated effects of  $V(n)$ . The bandwidth testing stops when a threshold number of continuous decreases (of the moving average) have been observed. The method uses two levels of thresholds<sup>8</sup>. When  $s/\bar{x} < 0.3$ , the threshold is set to 3; otherwise, the threshold is set to 5. This is because when the standard deviation of the bandwidth is large, false local maximums often occur before the actual maximum of  $V(n)$ . Making the threshold relatively large helps to eliminate them from being accepted as the real maximum. Figure 4.2 shows the result of using the “moving average method” on the same curve of  $V(n)$  values as in Figure 4.1 (We used the same random seeds to generate the bandwidth values). As we can see, the maximum  $V(n)$  value is reached at  $n=23$ , and the “moving average” method finishes bandwidth testing at  $n=26$ , a result significantly better than that achieved by the “continuous decrease” method, which finishes at  $n=36$ . This method is therefore a good heuristic for obtaining the approximation of the optimal bandwidth sampling number.

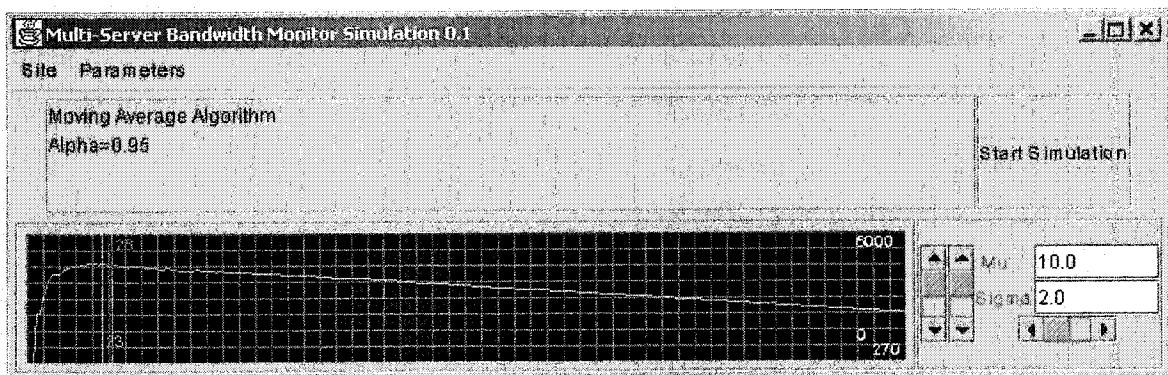


Figure 4.2, Screen Capture of Simulator using the “Moving Average” method.

Finally, before presenting the experimental results, we will discuss the issue of efficient implementation. Because the calculation of  $d_{(\alpha,n)}$  values is time-consuming and

<sup>8</sup> The levels and the values of thresholds were determined by trial and error in our experiments.

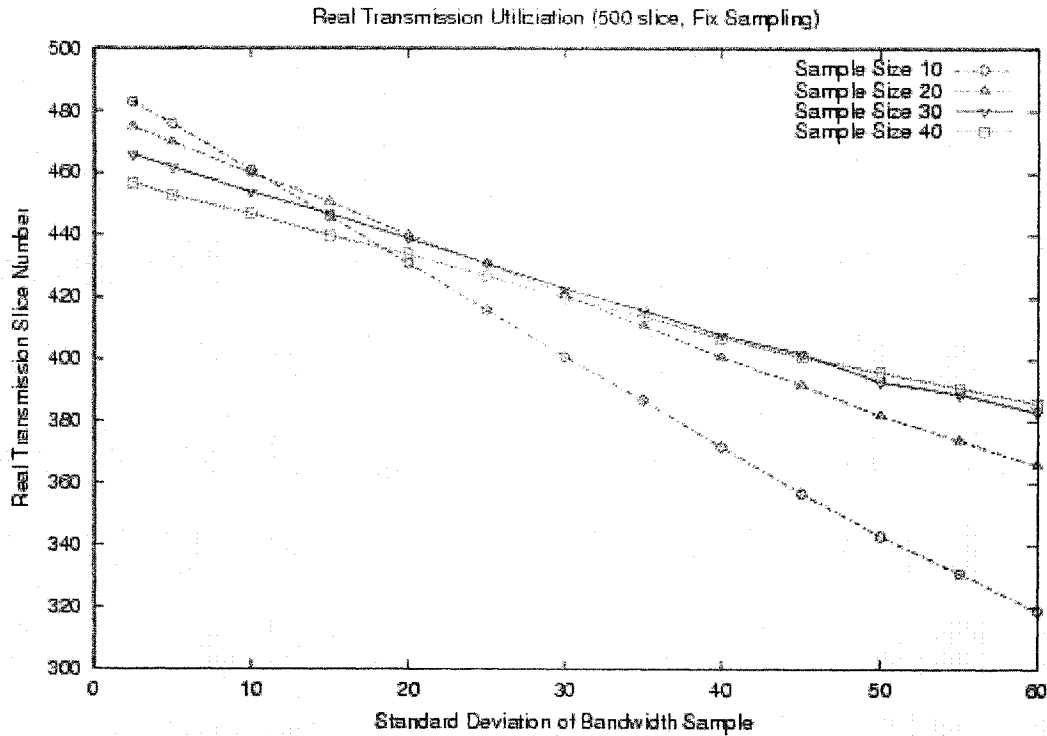
repetitive, we chose to pre-compute them and store them in a multi-dimension array for quick reference. Instead of letting the experimenter set an arbitrary value of  $\alpha$ , the simulator allows only three choices in  $\alpha$  value, 0.95, 0.90, and 0.75. The  $d_{(\alpha,n)}$  values were calculated by using the algorithm (which uses continued fraction representation of the t-function) described in [Press, 1993]. We then verified our results with the t-distribution critical value table in [Zwillinger, 1996] for those values available in it.

### 4.3 Experimental Results

In the experiments, we first show how well the algorithm approximates the optimal amount of bandwidth testing. As we discussed, a gauge to measure the optimality is the amount of time for real data transmission. We will compare the algorithm against a straightforward method we call “fixed sampling”. In Figure 4.3, we present the performance of a fixed sampling method. In that method, the simulator will first take a fixed number of bandwidth samples, calculate  $V(n)$  based on them and use the remaining time slices for real transmission. Figure 4.3 plots the number of time slices used for real data transmission (out of 500 total time slices, for  $\alpha = 0.95$ ) for four different numbers of fixed sample sizes (10, 20, 30, 40). The horizontal axis of the graph is the standard deviation of the bandwidth random variable; the vertical axis of the graph is the average real transmission time slice. The mean of the bandwidth random variables is always set at 100.0, while the standard deviations are from the set {2.5, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60}. Each data point is the average of 200 runs, excluding those runs that did not finish the transmission within the time limit<sup>9</sup>. The details of the data plotted in Figure 4.3 are presented in Appendix 1, Dataset 1.

---

<sup>9</sup> Those runs that did not finish within the time limit are excluded because they could distort the meaning of the actual transmission time. Usually, the larger the actual transmission time, the better is the algorithm. However, if the actual transmission time is so large that it exceeds the time limit, the time is not desirable.



*Figure 4.3, Real Transmission Time for the Fixed Sampling Method ( $\alpha=.95$ , 500 total slices)*

Figure 4.3 shows that different numbers of sample sizes perform well at different bandwidth variance regions. A small sample size like 10 outperforms others in low variances (when standard deviation equals to or is less than 10), but lags behind considerably when variances are large (e.g., when the standard deviation equals 60). This result coincides with our expectation. A small sample size performs well with a small variance because in this case, a small number of bandwidth samples are enough to serve the purpose of bandwidth estimation. Taking more bandwidth samples is just a waste of time slices. On the other hand, when the variance is large, a small number of samples do not contain enough knowledge of the channel to provide an accurate estimation. Therefore, to provide the same confidence level, the under-estimation margin must be very high. This hurts the real transmission time as well. The performance of a large sample size like 40 is exactly the reverse of that of a small sample size like 10.

Next, we compare the performance of our algorithm using moving average method with the performance shown in Figure 4.3 achieved by the fixed sampling method. We will call our new algorithm the “dynamic sampling” algorithm. The comparisons are presented in Figure 4.4 (a)(b)(c)(d), which compares our algorithm with the fixed sampling of 10/20/30/40 samples, respectively. The data points of the fixed sampling are the same as those in Figure 4.3, and the data points for our algorithm are the same across Figure 4.4 (a)(b)(c)(d). We run the dynamic sampling algorithm in three groups of experiments, 200 runs each group. For each group, we average the number of slices used for real transmission, excluding those runs that did not finish within the time limit. The results in Figure 4.4 are the average of the three groups. The details of these data are presented in Appendix 1, Dataset 2.

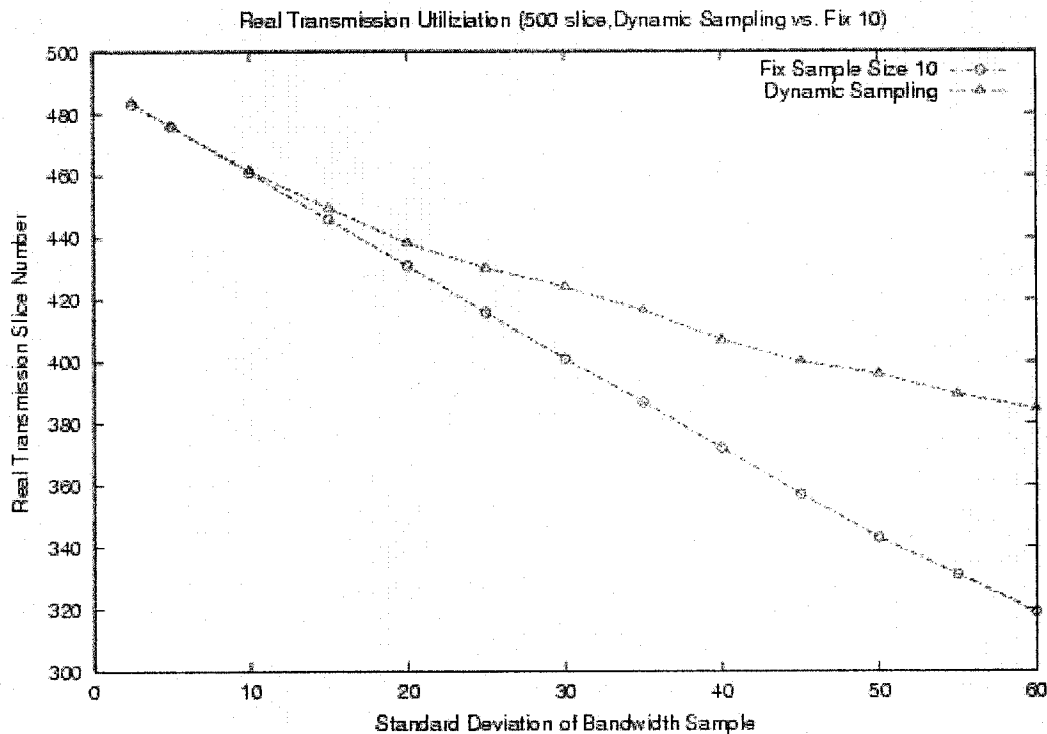


Figure 4.4(a), Real Transmission Time comparison( $\alpha=.95$ , 500 total slices, Dynamic v. Fix Sample 10)

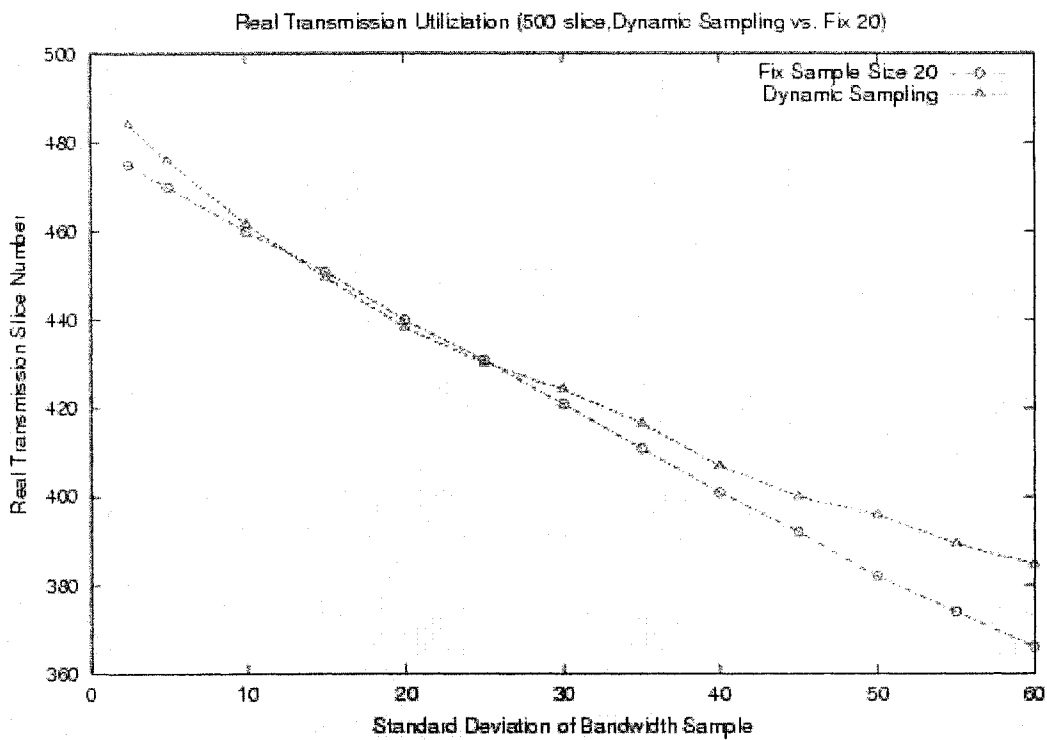


Figure 4.4(b), Real Transmission Time comparison ( $\alpha=.95$ , 500 total slices, Dynamic v. Fix Sample 20)

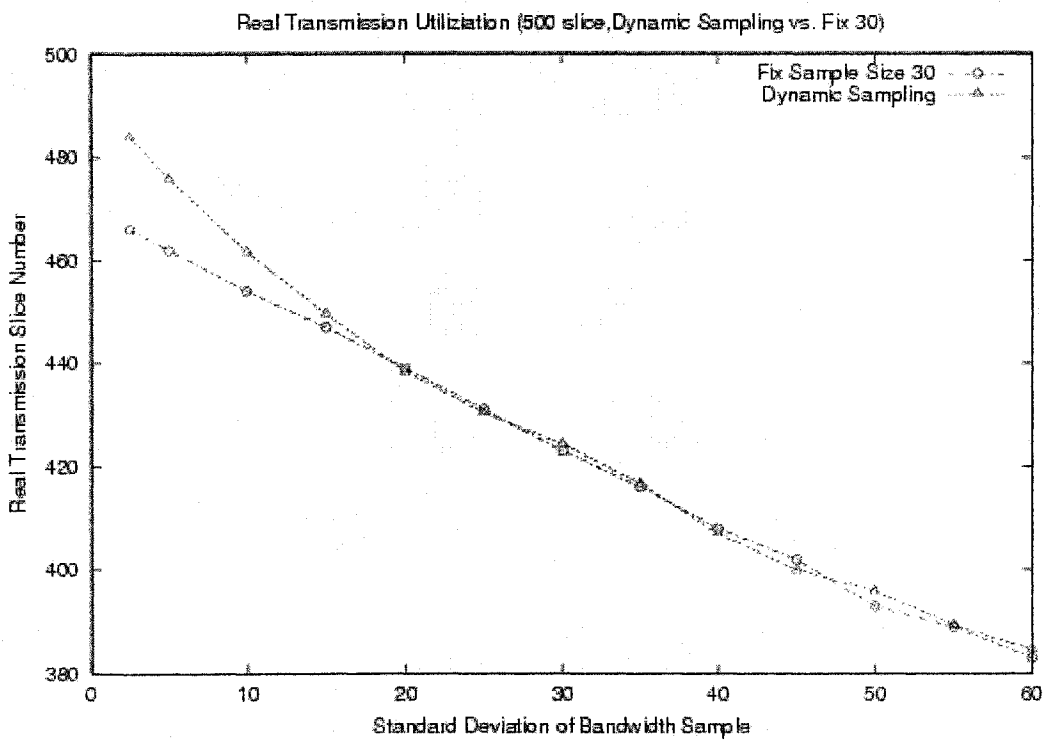
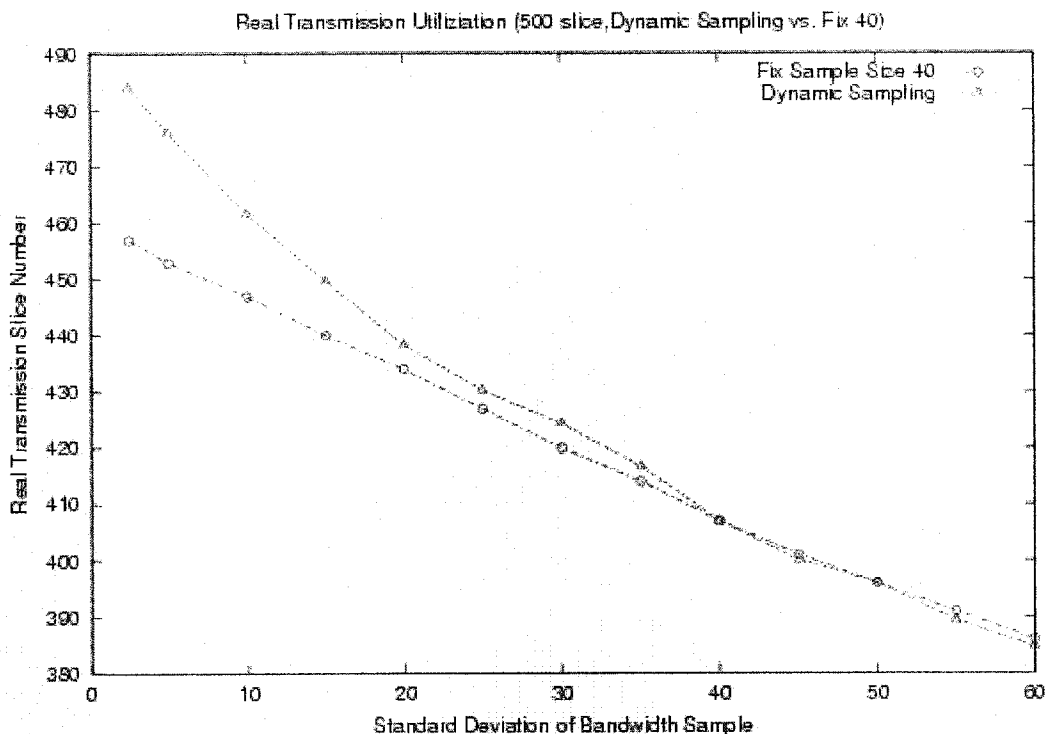


Figure 4.4(c), Real Transmission Time comparison ( $\alpha=.95$ , 500 total slices, Dynamic v. Fix Sample 30)



*Figure 4.4(d), Real Transmission Time comparison ( $\alpha=.95$ , 500 total slices, Dynamic v. Fix Sample 40)*

As shown in Figure 4.4, the dynamic sampling algorithm almost achieves results that resemble a contouring curve consisting of the best parts of the curves for each fixed sample size. For example, it achieves similar good results as fixed sample 10 when variance is low and similar good results as fixed sample 40 when variance is high. On the other hand, it beats fixed sample 10 up to 20% under large variance and beats fixed sample of 40 more than 5% when variance is small.

Under this set of experiments, the difference between the dynamic sampling algorithm and the fixed sampling algorithm with a sample size of 20/30 is not very significant. This finding indicates that the best sample size for fixed sampling should be somewhere between 20 and 30. However, this conclusion does not mean that a fixed sampling algorithm with a carefully chosen sample size can be used as a substitute for our dynamic algorithm, because the best sample size is determined by many parameters, like the total number of time slices and the confidence level  $\alpha$ .



To demonstrate this effect, we present the results from another set of experiments. The parameters of this experiment are almost the same as before:  $\alpha = 0.95$ ; the bandwidth averages are set at 100.0, and the standard deviations are from  $\{2.5, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60\}$ . However, this time, the total time slice is 100, and we change the sample size for the fixed sampling method to 5, 10, 15, and 20. The results are shown in Figure 4.5 and Figure 4.6 (a)(b)(c)(d). The details of the data are presented in Appendix 1, Dataset 3 and 4.

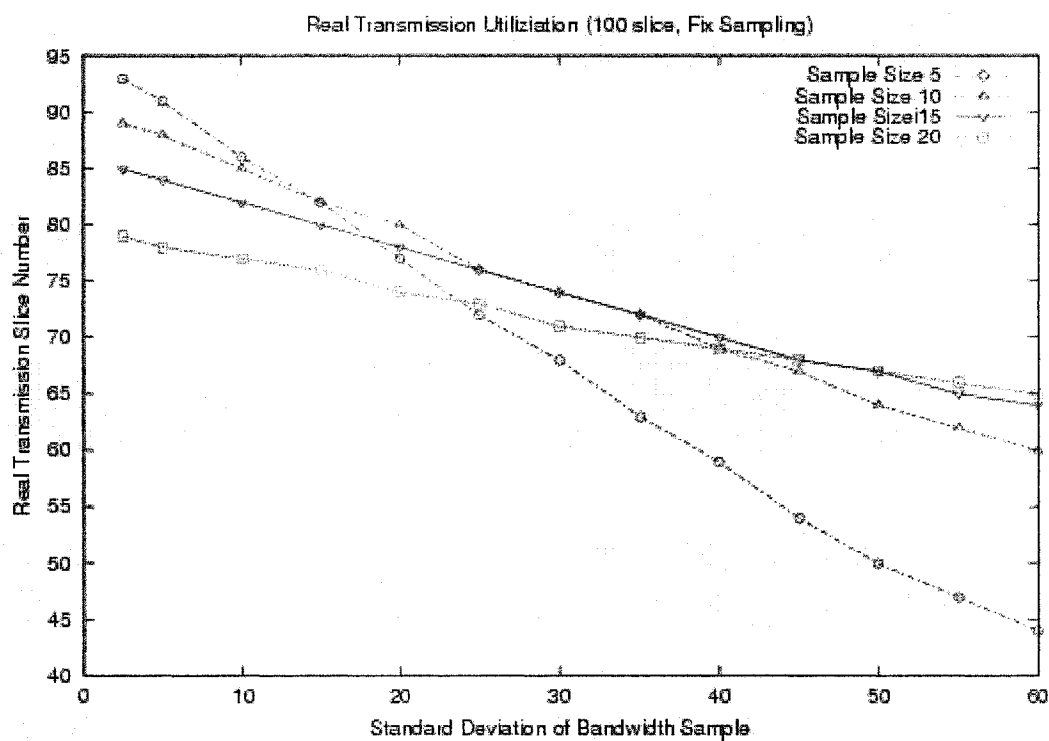


Figure 4.5, Real Transmission Time for Fix Sampling Method ( $\alpha=0.95$ , 100 total slices)

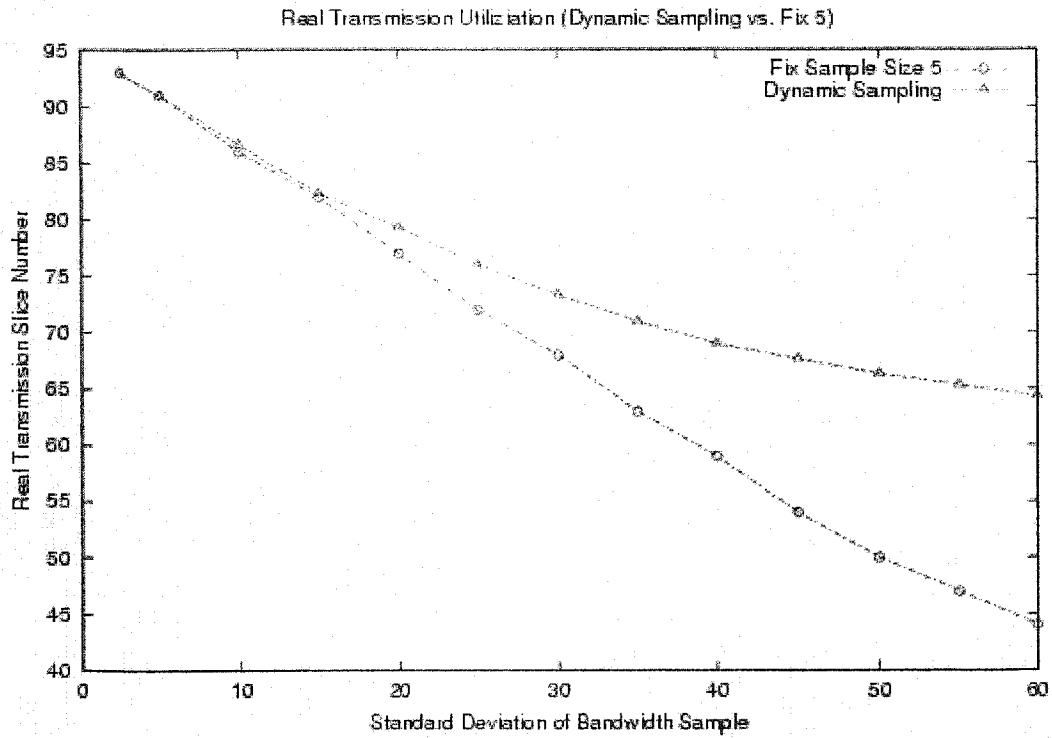


Figure 4.6(a), Real Transmission Time (Dynamic v. Fix Sample of 5. For  $\alpha=.95$ , 100 total slices,)

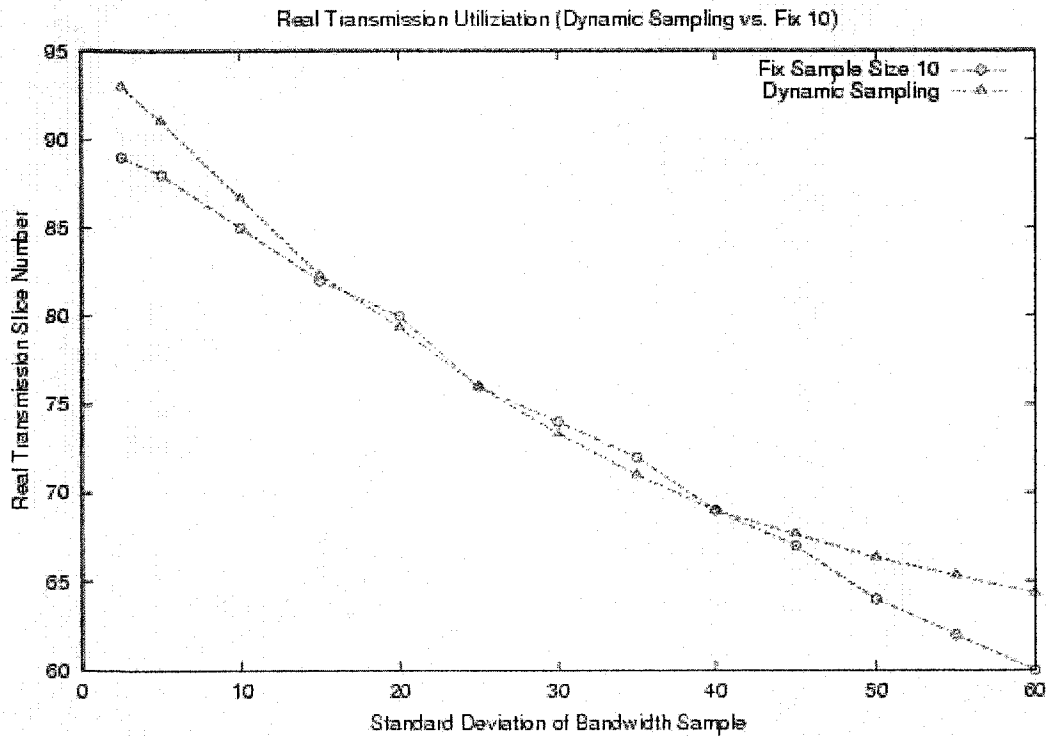


Figure 4.6(b), Real Transmission Time (Dynamic v. Fix Sample of 10. For  $\alpha=.95$ , 100 total slices)

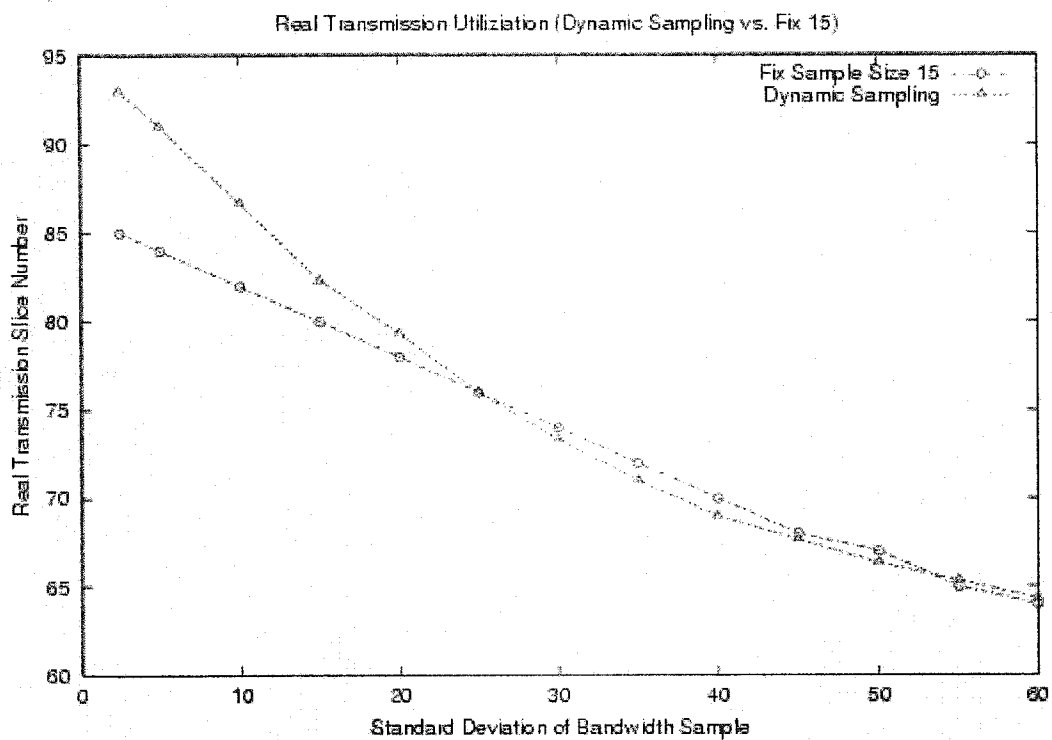


Figure 4.6(c), Real Transmission Time (Dynamic v. Fix Sample of 15. For  $\alpha=.95$ , 100 total slices.)

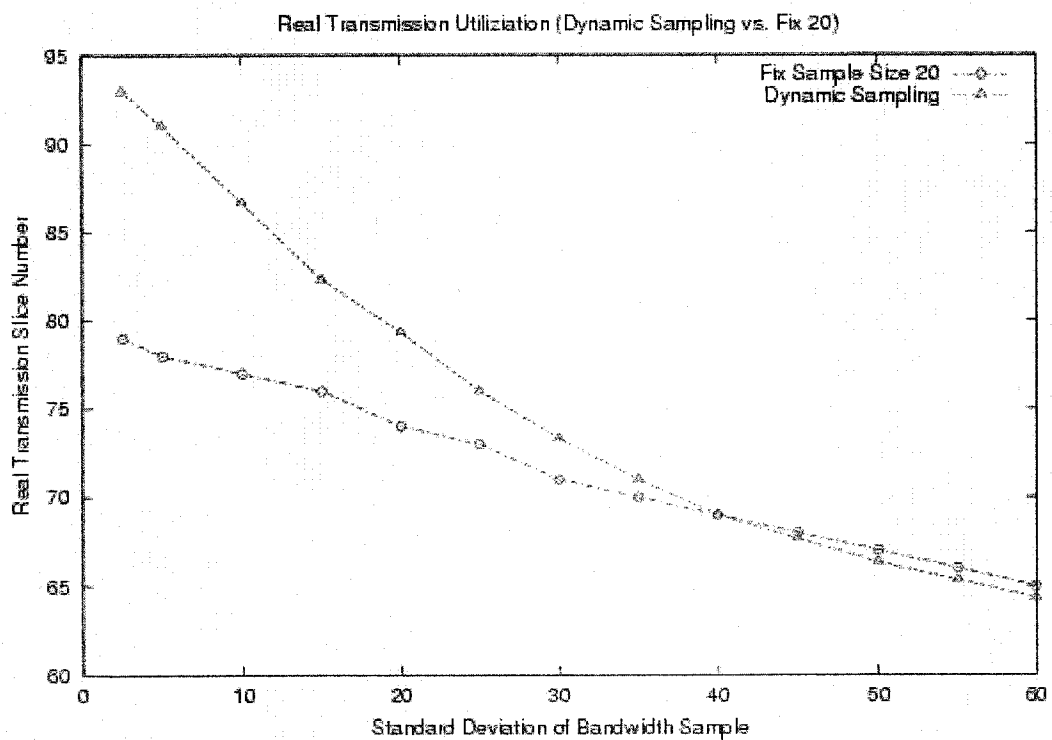
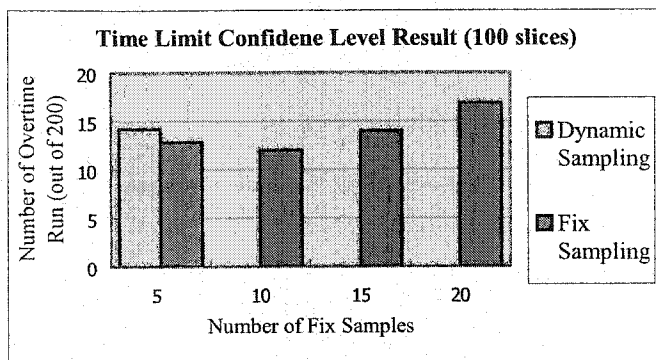


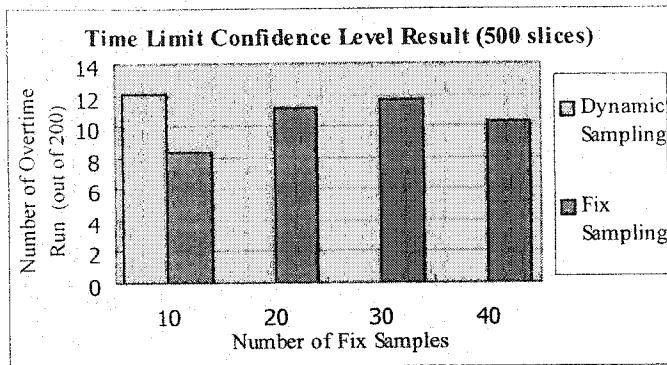
Figure 4.6(d), Real Transmission Time (Dynamic v. Fix Sample of 20. For  $\alpha=.95$ , 100 total slices.)

The results presented in Figures 4.5 and 4.6 follow the general trends of those in Figures 4.3 and 4.4, except for two important differences. First, as the total number of time slices decreases from 500 to 100, the relative importance of each slice increases. Therefore, the difference between a small fixed sample number and a large fixed sample number is much more significant in the low variances region. The dynamic sampling algorithm outperforms the fixed sampling of 20 samples by more than 15% when the standard deviation is 2.5. This result is significantly larger than that obtained when 500 total slices are used. Second, as the total number of time slice drops from 500 to 100, the best sample size is now between 10 and 15, much smaller than the best sample size of the 500 slices case (20-30), but not linearly.

Now that we have demonstrated the strength of the dynamic sampling algorithm in approximating the optimal amount of bandwidth sampling, we will show how well the algorithm provides the confidence level of the user-specified time limit. As we explained before, for the results presented in Figures 4.3-4.6, we discarded those runs that failed to finish within the time limit. Now, in Figures 4.7 and 4.8, we report how many of these runs there were. The details of the data are presented in Appendix 1, Datasets 5 and 6.



*Figure 4.7, Number of runs that failed to finish within the time limit (out of 200). Total Slice is 100, Alpha=0.95. The results are the average of all the bandwidth standard deviations in {2.5, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55,60}*



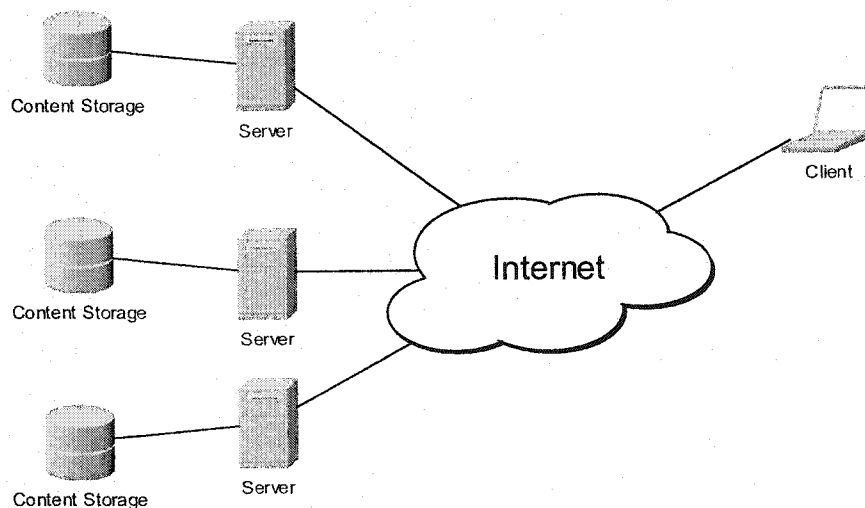
**Figure 4.8**, Number of runs that failed to finish within the time limit (out of 200). Total Slice is 500, Alpha=0.95. The results are the average of all the bandwidth standard deviations in {2.5, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60}.

As shown in Figure 4.7, when the total slice is 100, the percentage of runs that failed to finish on time ranges from 6% to 8%, and the dynamic algorithm generates about 7% of failure runs. When the total slice is 500, the percentage ranges from 4% to 6%, and the dynamic algorithm generates 6% of failure runs. These results mean the confidence level is preserved within a reasonable range for  $\alpha = 0.95$  nominally. Therefore, the estimation of the bandwidth based on t-distribution performs well in preserving the confidence level of the user-specified time limit.

## Chapter 5

# Multi-Server Bandwidth Estimation

In this chapter, we will extend the bandwidth estimation algorithm to a multi-server environment. In such an environment, several servers are hosting the same set of contents. A client is connected to all the servers via the Internet. We assume the client maintains a session with each of the servers, so that it can create TCP connections and receive data from the servers anytime it wishes. Figure 5.1 illustrates such an environment.



**Figure 5.1.** Network environment of a multi-server multimedia delivery application.

When the user selects a media object to be delivered and a time limit  $T$ , the client will first use a fraction of  $T$  (denoted  $t$ ) to perform the bandwidth testing on all the channels. The bandwidth testing follows the same sliced time method. In each time slice, the client will get a bandwidth sample on each channel. Based on the bandwidth samples collected, the client can make the available bandwidth estimation on all the channels. When bandwidth testing finishes, the client requests a strip of the object from each of the servers. The size of the strips is the product of the bandwidth estimation and the

remaining time ( $T-t$ ), which will be proportional to the relative bandwidth estimation on the channels. The transmission is successful only if all strips of the object finish transmission within the time limit  $T$ .

## 5.1 Simple Extension of the Algorithm

Suppose we have  $K$  channels<sup>10</sup> available. As defined in Chapter 3,

$V(n) = \mu_{est} \cdot (N-n) \cdot t_s = \left( \bar{x} - d_{(\alpha,n)} \cdot \frac{s}{\sqrt{n}} \cdot \sqrt{\frac{N-n}{N-1}} \right) \cdot (N-n) \cdot t_s$  is the expected object size

that will be delivered based on  $n$  bandwidth samples collected so far on any of these

channels. We use  $V_i(n) = \left( \bar{x}_i - d_{(\alpha_i,n)} \cdot \frac{s_i}{\sqrt{n}} \cdot \sqrt{\frac{N-n}{N-1}} \right) \cdot (N-n) \cdot t_s$  to represent the  $V(n)$

value on the  $i^{th}$  channel ( $i=1..K$ ), and represent the combined value of  $V_i(n)$  over all

channels  $V(n) = \sum_{i=1}^K V_i(n)$ . As we discussed in Chapter 3, statistically, each  $V_i(n)$  has a

single maximum value. Before it reaches its maximum, it grows as  $n$  grows with a decreasing growth rate; after reaching its maximum, it decreases when  $n$  grows, with an increasing decrease rate<sup>11</sup>. We first prove a new theorem about  $V(n)$ .

**Theorem 5.1**, If we view  $V(n)$  as a function of  $n$ , then i)  $V''(n) < 0$ ; ii) there exists  $\hat{n}$ ,  $V'(n) > 0$  for  $n < \hat{n}$ ,  $V'(n) < 0$  for  $n > \hat{n}$ .

**Proof:** We use induction on the number  $K$ . First we prove the theorem for  $K=2$ .

We have  $V_1(x) > 0$ ,  $V_1'(x) > 0$  for  $x < x_1$ ,  $V_1'(x) < 0$  for  $x > x_1$ , and  $V_1''(x) < 0$ .

$$V_2(x) > 0, V_2'(x) > 0 \text{ for } x < x_2, V_2'(x) < 0 \text{ for } x > x_2, \text{ and } V_2''(x) < 0.$$

<sup>10</sup> Throughout this chapter, we will assume the independence of all the channels, so that when we use several channels at the same time, each channel will have the same bandwidth as when that channel is used alone. This assumption is true in real world when the bottleneck of the network resides in the intermediate routers instead of the host. If the bottle-neck is in the host itself, then the whole idea of using multiple servers will not be attractive because it will not help in gaining more bandwidth.

<sup>11</sup> Or, mathematically speaking, i)  $V_i''(n) < 0$ , ii) there exists  $n_i$ ,  $V_i'(n) > 0$  for  $n < n_i$ ,  $V_i'(n) < 0$  for  $n > n_i$ .

$x_1$  and  $x_2$  are the  $x$  values that yield the maximum value for  $V_1(x)$  and  $V_2(x)$ . Without loss of generality, we assume  $x_1 < x_2$ .

$$V(x) = V_1(x) + V_2(x), \text{ so } V(x) > 0$$

$$V''(x) = V_1''(x) + V_2''(x), \text{ so } V''(x) < 0.$$

$V'(x) = V_1'(x) + V_2'(x)$ . We must show that there exist  $x_3$  so that  $V'(x) > 0$  for  $x < x_3$ ,  $V'(x) < 0$  for  $x > x_3$ .

First, for  $x < x_1$ , we have  $V_1'(x) > 0$  and  $V_2'(x) > 0$ , therefore  $V'(x) > 0$ ; for  $x > x_2$ , we have  $V_1'(x) < 0$  and  $V_2'(x) < 0$ , therefore  $V'(x) < 0$ .

In the range  $x_1 \leq x \leq x_2$ , we have  $V_1'(x_1) = 0$ ,  $V_1'(x) < 0$  for  $x_1 < x \leq x_2$ , and  $V_1''(x) < 0$ .

We also have  $V_2'(x) > 0$  for  $x_1 \leq x < x_2$ ,  $V_2'(x_2) = 0$ , and  $V_2''(x) < 0$ .

Therefore  $V'(x_1) = V_1'(x_1) + V_2'(x_1) > 0$ ,  $V'(x_2) = V_1'(x_2) + V_2'(x_2) < 0$ , and

$$V''(x) = V_1''(x) + V_2''(x) < 0.$$

Since  $V'(x)$  is a monotonic decreasing function in the given range, there must exist a single value  $x_3$  so that  $V'(x) > 0$  for  $x < x_3$  and  $V'(x) < 0$  for  $x > x_3$ .

We therefore have proved the theorem for  $K=2$ .

Assume we have proved that the theorem holds for  $K=m-1$ , then the combination of the  $m-1$  channels has the same property as specified. The process of proving the combination of  $m$  channels is exactly the same as  $K=2$ , viewing the first  $m-1$  channel as a single channel. This completes the proof.  $\square$

Theorem 5.1 basically means that we can design a similar algorithm as was used in the single-server environment to approximate the optimal amount of bandwidth testing by checking whether the combined  $V(n)$  value has reached its maximum. However, a new problem occurs in a multi-server environment, which is how to determine the value of  $\alpha_i$  for  $i=1..K$ .  $\alpha_i$  is the confidence level we will use on channel  $i$  when calculating  $\mu_{est,i}$  (which in turn determines  $V_i(n)$ ). As we assume the independence of all the channels,



we should have  $\alpha = \prod_{i=1}^K \alpha_i$ , where  $\alpha$  is the user-specified confidence level for the time limit. Our problem is that given the  $n$  samples taken on each of the  $K$  channels, we must determine  $\alpha_i$  for  $i=1..K$ , to maximize  $V(n) = \sum_{i=1}^K V_i(n) = \sum_{i=1}^K \left( \bar{x}_i - d_{(\alpha_i, n)} \cdot \frac{s_i}{\sqrt{n}} \cdot \sqrt{\frac{N-n}{N-1}} \right)$  (or to minimize  $\sum_{i=1}^K d_{(\alpha_i, n)} \cdot s_i$ ).

It is quite obvious that finding the optimal solution to the problem in real time is out of the question. Even the calculation of the  $d_{(\alpha_i, n)}$  value in real time, given arbitrary  $\alpha_i$  value, will be too time-consuming to be performed for each time slice, let alone for finding the optimal assignment of  $\alpha_i$ . In this chapter, we will use a simple heuristic, which is to set the same  $\alpha_i$  values on all the channels. Therefore,  $\alpha_i = \alpha^{1/K}$  for  $i=1..K$ . For example, when  $K=2$  and  $\alpha = 0.90$ , we will set  $\alpha_1$  and  $\alpha_2$  around 0.95 (more precisely,  $\alpha_1 = \alpha_2 = 0.9487$ ). The intuition behind this choice is that it minimize  $d_{(\alpha_1, n)} + d_{(\alpha_2, n)}$  for  $\alpha_1 \cdot \alpha_2 = \alpha$ . For example when  $\alpha = 0.90$ ,  $d_{(0.95, 1)} + d_{(0.95, 1)} \approx 6.314 + 6.314 < d_{(0.91, 1)} + d_{(0.99, 1)} \approx 3.397 + 31.821 < d_{(0.90, 1)} + d_{(0.9995, 1)} \approx 3.078 + 636.619$ . Therefore, when the variances on the channels do not vary too much, this heuristic is a good choice in making  $\sum_{i=1}^K d_{(\alpha_i, n)} \cdot s_i$  small. Another important merit of this heuristic is that all the  $d_{(\alpha, n)}$  values we need for the algorithm can be pre-calculated, eliminating the need to calculate  $d_{(\alpha, n)}$  in real time.

Based on this discussion, we have created the following simple extension of Algorithm 3.1 for determining the optimal amount of bandwidth testing.

**Algorithm 5.1: Multi-server Bandwidth Estimation (version 1)**

$\alpha_i \leftarrow \alpha^{1/K}$ , for  $i=1..K$ ;

Obtain samples  $x_1, x_2$  on each channel and calculate  $V(1) \leftarrow \sum_{i=1}^K V_i(1)$ ;

Obtain sample  $x_3$  on each channel and calculate  $V(2) \leftarrow \sum_{i=1}^K V_i(2)$ ;

$n \leftarrow 2$ ;

while ( $V(n) > V(n-1)$ ) {

$n \leftarrow n+1$ ;

    Obtain a new sample on each channel;

    Calculate  $V(n) \leftarrow \sum_{i=1}^K V_i(n)$ ;

}

return  $\mu_{est}$  on each channel;  $\square$

**5.2 Refinement of Multi-server Algorithm**

Unfortunately, Algorithm 5.1 does not perform well when the sample standard errors on channels vary greatly. Algorithm 5.1 assumes that the  $\alpha$  values on all the channels are the same. When it has  $K$  channels available, it always performs transmission on all the channels. However, in some cases (especially when the standard errors on channels vary a great deal), if we assume all channels use the same  $\alpha$  value, we may end up better off if we drop some of the channels in order to maximize the combined  $V(n)$  value<sup>12</sup>. Consider a simple example. Suppose we have only two channels, and the user wants to transmit an object within the time limit  $T$  at a confidence level of  $\alpha = 0.90$ . If we use two channels,  $\alpha$  will be about 0.95 on each channel; if we use either of the two channels alone,  $\alpha$  will be 0.90. Now suppose connection #1 has a much larger estimated

<sup>12</sup> This sounds like a paradox: in order to maximize the expected object size, we end up better off using only some of the channels instead of all of them. The reason is our simplification that all the channels are using the same  $\alpha_i$  values.

In fact, if we allow arbitrary  $\alpha_i$  values (as long as  $\alpha = \prod_{i=1}^K \alpha_i$ ), then there will always be a set of  $\alpha_i$  values that will maximize the expected object size by using all the channels.

bandwidth than connection #2 so that the difference between  $V_1$  by using  $\alpha = 0.90$  and  $\alpha = 0.95$  is larger than  $V_2$  using  $\alpha = 0.95$  (recall that

$$V = \mu_{est} \cdot (T - t) = \left( \bar{x} - t_{(\alpha, n)} \cdot \frac{s}{\sqrt{n}} \cdot \sqrt{\frac{N-n}{N-1}} \right) \cdot (N-n) \cdot t_s; \text{ in this case, we may be better}$$

off dropping connection #2 and using connection #1 for real transmission, with  $\alpha = 0.90$ .<sup>13</sup>

To accommodate the possibility of dropping channels during the bandwidth estimation, we propose the following method to refine the algorithm. Suppose we have  $K$  channels available. In each time slice when a new sample is obtained on each channel, the algorithm calculates  $K$   $V$  values by using  $K$  different  $\alpha$  ( $\hat{\alpha}_1, \hat{\alpha}_2 \dots \hat{\alpha}_K$ ) for each connection, where  $\hat{\alpha}_i$  is the  $\alpha$  value if  $i$  channels are finally used for real transmission (the other  $K-i$  channels are dropped). We have  $\hat{\alpha}_i = \alpha^{1/i}$ , where  $\alpha$  is the confidence level specified by end user. Therefore, we will have a total of  $K^2$   $V$  values in each time slice, as shown in Table 5.1 below. Each row of the table belongs to one channel, and the  $K$  values of the same row represent how much volume will be transmitted when the  $K$  different  $\alpha$  values are used. From this definition of the table values, it is obvious that the maximum value of column one of the table is the maximum  $V$  we can obtain if we use only one channel; the sum of the maximum two values of column two is the maximum  $V$  we can obtain if we use two channels; the sum of the maximum three values of column three is the maximum  $V$  we can obtain if we use three channels; finally, the sum of all the  $K$  values of column  $K$  is the  $V$  we can obtain if we use all the  $K$  channels. Therefore, by picking the largest of these  $K$  options, we can obtain the maximum  $V$  that can be obtained by using any subset of the  $K$  channels. As each new sample is obtained, we can go through this procedure to find out the maximum  $V(n)$  and the channels that we should use to achieve that  $V(n)$  (the rows that constitute the option we choose).

<sup>13</sup> Actually, in a real implementation, there are several strategies to make use of these “dropped” channels. One strategy, suggested by W. Grover, is to use them to transmit the last portions of the strips allocated to the “admitted” channels. In this way, the transmitted portions on “dropped” channels can serve as contingency backup in case the “admitted” channels do not finish within the time limit. Another strategy is to allocate strips of the expected object to all the channels including the “dropped” ones. However, when calculating the strip sizes, use a  $\alpha$  value close to 1.0 (E.g.  $\alpha = 0.999$ ) for the “dropped” channels.

$V_1(\hat{\alpha}_1)$	$V_1(\hat{\alpha}_2)$	$V_1(\hat{\alpha}_3)$	...	$V_1(\hat{\alpha}_K)$
$V_2(\hat{\alpha}_1)$	$V_2(\hat{\alpha}_2)$	$V_2(\hat{\alpha}_3)$	...	$V_2(\hat{\alpha}_K)$
$V_3(\hat{\alpha}_1)$	$V_3(\hat{\alpha}_2)$	$V_3(\hat{\alpha}_3)$	...	$V_3(\hat{\alpha}_K)$
...	...	...	...	...
$V_K(\hat{\alpha}_1)$	$V_K(\hat{\alpha}_2)$	$V_K(\hat{\alpha}_3)$	...	$V_K(\hat{\alpha}_K)$

**Table 5.1**,  $V$  values on  $K$  channels with  $K$   $\alpha$  values. Each row of the table represents  $K$   $V$  values for one channel. Please note that compared to the notation in Algorithm 5.1, we omit the time slice number as a parameter of the  $V$  value but introduce the new parameter  $\hat{\alpha}_i$ .

Based on this discussion, we present a refined version of our algorithm that accommodates dropping channel(s).

**Algorithm 5.2:** Multi-server Bandwidth Estimation (refined version)

Calculate  $\hat{\alpha}_i = \alpha^{1/i}$  for  $i \in \{1, 2, \dots, K\}$ ;

Obtain sample  $x_1, x_2$  on each channel;

Calculate  $V_i(1, \hat{\alpha}_j)$  for  $i \in \{1, 2, \dots, K\}, \hat{\alpha}_j \in \{\hat{\alpha}_1, \hat{\alpha}_2, \dots, \hat{\alpha}_K\}$ ;

$V_1 = \text{GetMax}(1)$ ;

$n \leftarrow 1$ ;

while(TRUE) {

$n \leftarrow n + 1$ ;

Obtain sample  $x_{n+1}$  on each channel;

Calculate  $V_i(n, \hat{\alpha}_j)$  for  $i \in \{1, 2, \dots, K\}, \hat{\alpha}_j \in \{\hat{\alpha}_1, \hat{\alpha}_2, \dots, \hat{\alpha}_K\}$ ;

$V_n = \text{GetMax}(n)$ ;

if ( $V_n < V_{n-1}$ ) break;

}

return  $\hat{\mu}$  on each channel that constitutes part of  $V_n$ ;  $\square$

The function  $\text{GetMax}(n)$  takes the variable  $n$  as input and works on a two dimensional

array  $V_i(n, \alpha_j)$  (for  $i, j = 1 \dots K$ , as illustrated in Table 5.1). It first sorts each column of the table, then picks the largest value from column 1, the largest two values from column 2 and adds them together, and so on, and finally adds all the values in column  $K$ . It returns the largest of the  $K$  sums. Moreover, we can have  $GetMax(n)$  to set a vector (implemented as a global variable) of  $K$  indicators, indicating whether row  $i$  is part of the largest sum. This vector shows which channels are dropped and which channels are admitted.

### 5.3 Simulation Overview

Now we present the simulation result for our multi-server bandwidth estimation algorithm. As in Chapter 4, we try to find out how well the algorithm approximates the optimal amount of bandwidth testing and how well it preserves the confidence level of the user-specified time limit. Moreover, for the multi-server algorithm, we will study how effective the algorithm is in handling the drop of the connections to maximize the transmitted object size.

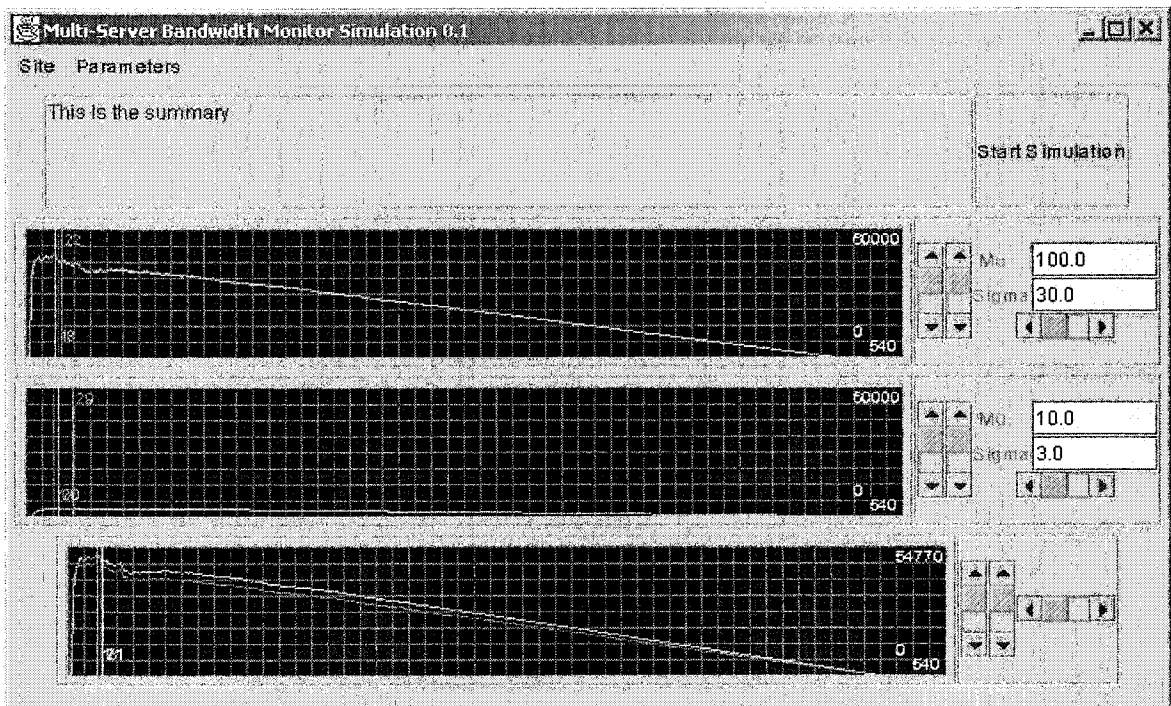


Figure 5.2, Screen Capture of Multi-server Bandwidth Estimation Simulator.

Figure 5.2 is a screen capture of our multi-server simulator. The experimenter can change the number of channels through the “Site” menu. The user-specified confidence level  $\alpha$  and total time slice number  $N$  are set through the “Parameter” menu. In Figure 5.2, we have chosen two channels and  $\alpha = 0.95$ . There are two panels in the middle of the window, each with text fields (labelled “Mu” and “Sigma”) on the right for the experimenter to enter the mean and standard deviation of the channel’s bandwidth population.

The simulator simulates a real application in two phases. Phase One is the bandwidth testing. In each slice, the simulator generates a random number by using a normal distribution generator with the mean and standard deviation specified by the experimenter for each channel. As we discussed, we assume the independence of the channels; therefore, the random numbers are independently generated for each channel. These random numbers simulate the bandwidth observed in the current time slice on each channel. Then the simulator calculates the combined  $V$  values and uses Algorithm 5.2 to determine whether the bandwidth testing should continue or not. When the bandwidth testing is finished, the simulator determines which channels will be used (admitted) for real transmission and what the strip sizes will be in each of them ( $V_i(n) = \mu_{est,i} \cdot (N - n) \cdot t_s$ ). In Phase Two, the real transmission is simulated. The simulator continues to generate random numbers in the admitted channels to simulate the real transmission. In each time slice, the simulator decreases the remaining object strip size in each admitted channel and checks whether it has reached zero. When the remaining object strip size does reach zero in a channel, the real transmission on that channel is finished. The entire object is transmitted to the client within the given time limit only if all admitted channels finish transmitting their strips on time.

As in Chapter 4, we make a small amendment to Algorithm 5.2 by using  $3/5$  (determined by variance) continuous decreases in the moving average of the past four  $V$  values as the termination condition of the bandwidth testing. We also limit the maximum number of channels to five. With that simplification, all  $\alpha$  values relevant are those for which  $\hat{\alpha}^i \in \{0.75, 0.90, 0.95\}$  for  $i=1..5$ . We then pre-compute these  $d_{(\alpha,n)}$  values and arrange

them in an array by using the same method discussed in Chapter 4.

After a run is finished, the simulator displays the  $V$  value curve on the graph of the channel panels. For each channel,  $K$  curves are drawn, one for each  $\hat{\alpha}_i$  value (i.e., if  $i$  channels are used for real transmission). In the case shown in Figure 5.2, the red curve represents  $V$  values if one channel is finally used; therefore, it uses  $\hat{\alpha}_1 = 0.95$  to calculate  $V$ ; the yellow curve represents  $V$  values if two channels will be used in real transmission; thus, it uses  $\hat{\alpha}_2 = 0.9747$  to calculate  $V$ . Finally, the lowest component of the window in Figure 5.2 shows the graph of the combined  $V$  values at each time slice. The red curve represents the maximum combined  $V$  values when one (the largest) channel is used, and the yellow curve represents the combined  $V$  values of both channels. In this case, we can see that using both channels outperforms using only one of them, as the yellow curve is almost always above the red one.

## 5.4 Simulation Results

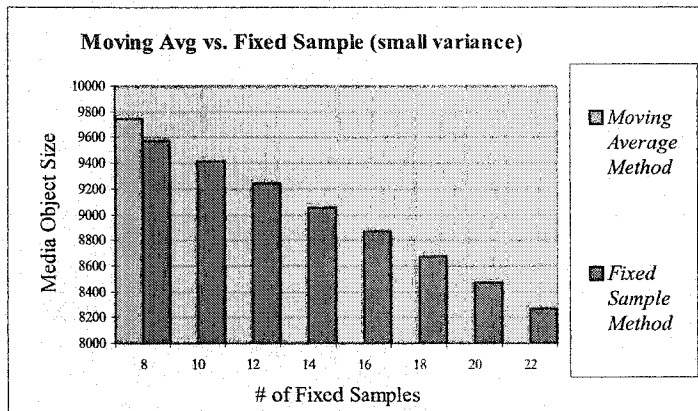
We first show how well the algorithm approximates the optimality in the amount of bandwidth testing. In this set of experiments, we use the size of the successfully transmitted object as a gauge in measuring the optimality of the sample size<sup>14</sup>. We compare our dynamic sampling algorithm with the algorithm that uses a fixed number of samples for the bandwidth testing. Simulation is performed in a setting of two channels. Channel #1 has a mean bandwidth of 100Kbps and channel #2 10Kbps. In order to measure the effectiveness of the algorithms under different network characteristics, we vary the variances on both channels. For channel #1, we vary the standard deviation from the set {2.5, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60}(Kbps), and choose the standard deviations of channel #2 from {0.25, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0}(Kbps). For each combination of channel characteristics, we run 1000 times.

---

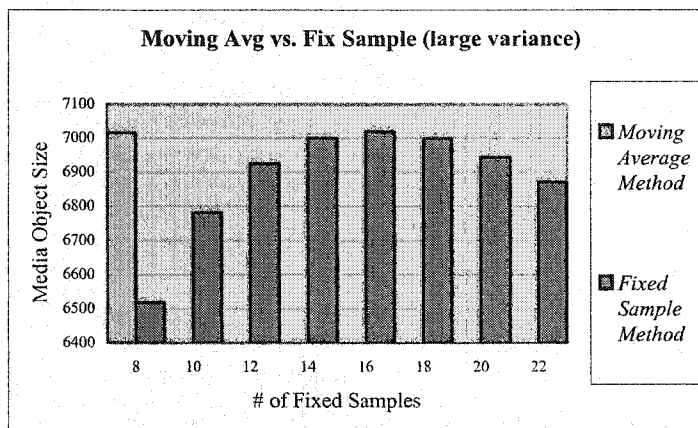
<sup>14</sup> Recall that in Chapter 4, we used the real transmission time as the gauge to measure the merit of sample size. In this chapter we change to the transmitted object size, because when several channels are each delivering a strip of the object, the real transmission time on each channel can be different. The transmitted object size is a more uniform gauge in this case.

We exclude those runs that exceed the time limit and average the remaining on the real transmitted object size. We test two total time slice numbers, 100 and 500.

The results of the tests are presented in Figure 5.3 (a), (b), (c) and Figure 5.4(a), (b), (c). Figure 5.3 shows the results for 100 time slices, and Figure 5.4 shows the results for 500 time slices. As both algorithms' performances differed significantly under different variances, we present each case in three graphs. The performance under small variances (where sigma is less than 15% of mu) is shown in (a); the performance under large variances (where sigma is larger than 35% of mu) is shown in (b); and the overall performance is shown in (c). The details of the data for Figure 5.3 and 5.4 are presented in Appendix 1, Datasets 9 and 10.

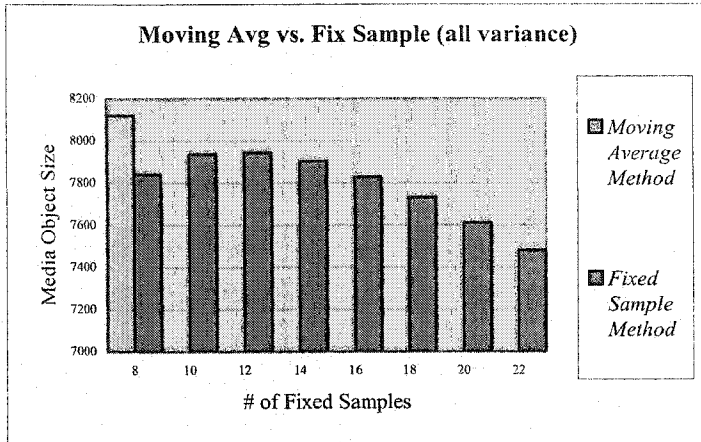


**Figure 5.3(a):** Comparison of the average size of the transmitted object for two algorithms (on 100 total slices). Averaged over all small variances ( $\sigma < 0.15 * \mu$ ) in 1000 runs.

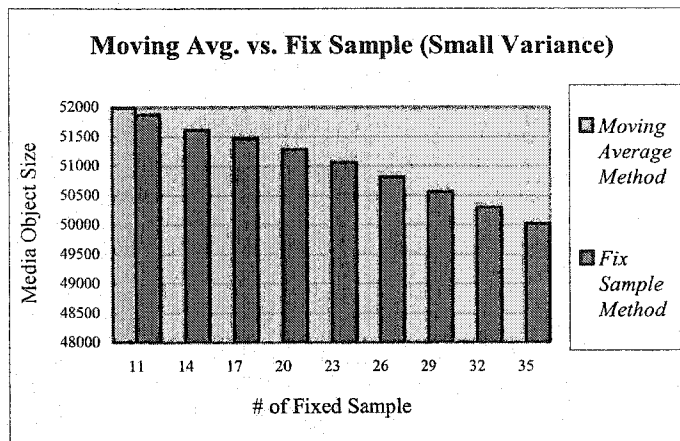


**Figure 5.3(b):** Comparison of the average size of the transmitted object size for two algorithms (on 100 total slices). Averaged over all large variances ( $\sigma > 0.35 * \mu$ ) in 1000 runs.

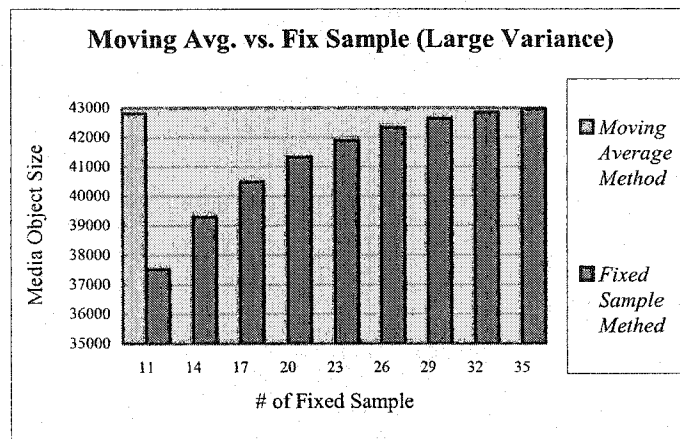




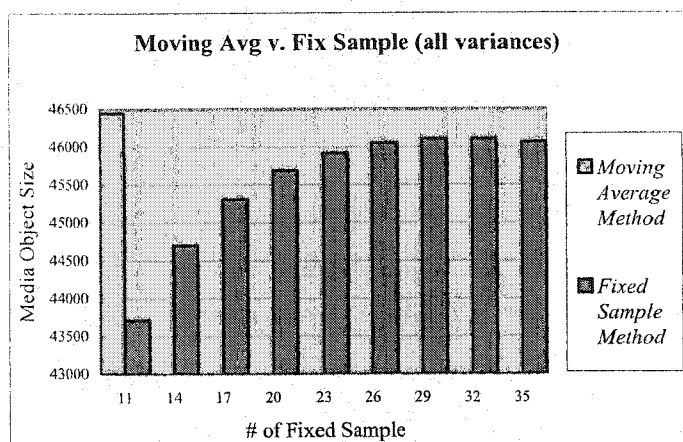
**Figure 5.3(c):** Comparison of the average size of the transmitted object size for two algorithms (on 100 total slices). Averaged over all variances in 1000 runs.



**Figure 5.4(a):** Comparison of the average size of the transmitted object size for two algorithms (on 500 total slices). Averaged over all small variances ( $\sigma < 0.15 * \mu$ ) in 1000 runs.



**Figure 5.4(b):** Comparison of the average size of the transmitted object size for two algorithms (on 500 total slices). Averaged over all large variances ( $\sigma > 0.35 * \mu$ ) in 1000 runs.

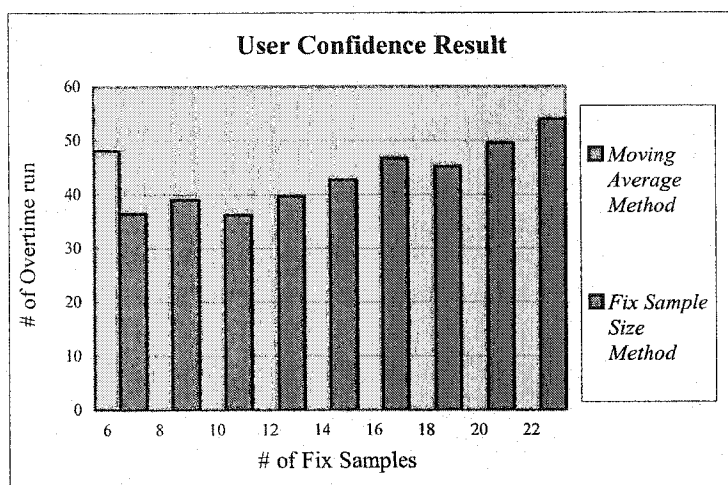


**Figure 5.4(c):** Comparison of the average size of the transmitted object size for two algorithms (on 500 total slices). Averaged over all variances in 1000 runs.

We have two observations concerning the above results. Firstly, the performance of the fixed sampling algorithm depends on the number of samples used, yet the optimal number of the sample varies under different total slice numbers and  $\alpha$  values. However, by using the dynamic sampling algorithm, we can always achieve results similar to those obtained by using the fixed sampling algorithm with the best sample size.

Secondly, the performance gain by using the dynamic sampling algorithm instead of the fixed sampling algorithm comes from different sources. For instance, the dynamic algorithm outperforms the fixed sampling algorithm by up to 17% in small variances on 100 time slices, but only by 4% in small variances for 500 time slices, because for small variances, the worst performing fixed sampling algorithms are those using a relatively large number of fixed samples. These additional samples constitute a much larger portion of the total time in 100 total time slices than in 500 total time slices. Conversely, in the large variance case, our algorithm outperforms fixed sampling by up to 8% over 100 time slices and up to 15% over 500 time slices. That is because this difference occurs when a small number of fixed samples are used, which cause aggressive underestimation of the bandwidth in the algorithms. This factor has more of an effect when there are more remaining time slices for real transmission, 500 total time slices in this case.

Another important factor we are interested in is how well the algorithms provide the confidence level that real transmission will be finished within time limit. In Figure 5.5, we present the results for the total time slice of 100. The details of the data plotted in Figure 5.5 are presented in Appendix 1, Dataset 11.

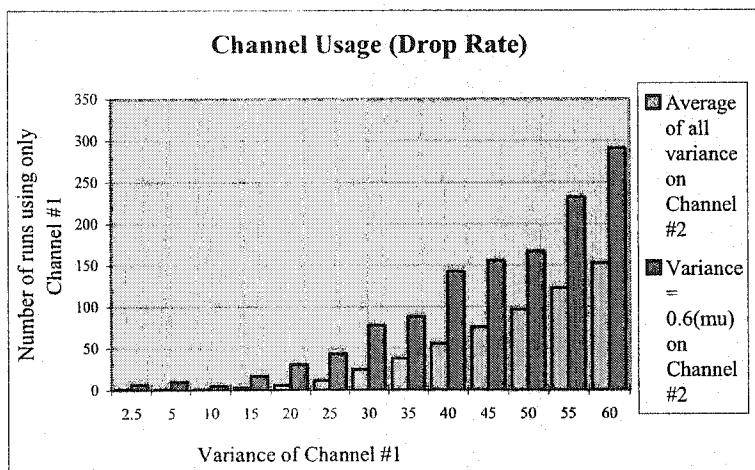


*Figure 5.5: The number of runs out of 1000 that failed to finish the transmission within the time limit (100 total slices).*

The graph shows the number of runs out of 1000 that failed to finish the real transmission within the time limit (averaged over all variances on both channels) for both the dynamic algorithm and the fixed sampling method. Recall that we have set  $\alpha = 0.95$ . We can see that the dynamic algorithm has around 95% of the runs finishing transmission within the time limit. The results for the fixed sampling algorithm with different numbers of samples vary slightly within the 94.5-96.5% range. Thus, both methods deliver the confidence level of the user-specified time limit very well.

Finally, we present data to show how often the dynamic algorithm really drops channels. In the previous discussions, we pointed out that when two channels with large variances have a relatively large difference in their average bandwidths, it may be better to drop the channel with the smaller bandwidth. We performed experiments on two channels with average bandwidths of 100kbps (Channel #1) and 10kbps (Channel #2), respectively. We counted how many times in 1000 runs our algorithm ended up using only one channel for

real transmission. The total time slices in use were 100. As before, the experiments were conducted over all the variances specified in the previous experiments, and the results are shown in Figure 5.6. The details of the data are presented in Appendix 1, Dataset 12.



**Figure 5.6:** This graph shows how many times in 1000 runs our algorithm uses only one channel for real transmissions. The x-axis is the variance of Channel #1. The dark-colored bar shows the number for the case of variance equals 6.0 on Channel #2, while the light-colored bar shows the average number for variances belonging to {0.25, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0}.

As the graph indicates, the percentage of runs using only one channel reaches as high as 30% when variance is high in both channels. This result justifies our efforts in developing the refined algorithm to determine which channels to admit and which ones to drop.

## Chapter 6

# Experiments on Real Networks

Up to now, the experiments we have discussed are all based on simulations. In this chapter, we describe an implementation of the algorithm and the experimental results in the real network environment. We have several goals in doing experiments on real networks.

Firstly, since the algorithm draws bandwidth samples from the first fraction of the time limit, it has been assumed that these samples will be the unbiased representatives of the entire population (the mean and variance of the bandwidth random variable are stationary). However, this assumption is only an approximation of the real scenario. The first group of samples are actually quite special, not only because they are from the period when the TCP protocol tries to figure out the available bandwidth on the network (as we discussed in Chapter 2), but also because the available bandwidth may be affected by other traffic on the network if the total time limit is relatively long. Therefore, we want to find out whether this assumption is valid in the real network. The positive results we will present relating to this issue only partially justify the assumption. The results do show that in several cases (the three cases we will describe), the assumption is acceptable. However, they cannot be readily generalized to all applications.

Secondly, practical issues still must be addressed once our algorithm can be applied to real applications. One important question is how to determine the proper length of each time slice. In the simulation experiments we described, the unit of time was one time slice, and we did not mention how long each time slice actually was. When determining the time slice in a real network environment, several factors must be taken into consideration. Although the TCP protocol provides the upper layer protocols a service as

if the data were transmitted in a stream, they are actually delivered in packets. Therefore, if the time slice is too small, the observed bandwidth samples will fluctuate a great deal and exaggerate the variance of the bandwidth. Also, as each time slice introduces some computation overhead in calculating bandwidth sample statistics, an excessively small time slice will enlarge such overhead. On the other hand, if the time slice is set too large, the cost of each sample grows, which also hurts the accurate estimation of the bandwidth. We aim to understand these effects better by performing experiments in real networks.

Finally, we want to check the computation cost of the algorithm to determine whether the algorithm is simple enough to be applied in real-time bandwidth estimation for network-aware applications. To keep the project's scope within a reasonable range, we have restricted the experiments in real networks to a single-server environment.

## 6.1 Implementation of the Algorithm

Our implementation of the algorithm consists of two programs, a server and a client. The server is a background process on the server host. It listens to a certain TCP port number on the server machine and waits for connection requests from clients. The client is a Java Applet embedded in an HTML page. It includes a user interface that allows the user to specify parameters for an object transmission. There are three parameters: the "Time Limit" is the user specified time for object transmission; the "Confidence Goal" is the confidence level associated with the time limit; the "Time Slice" is the time used for each bandwidth sample. Figure 6.1 is a screen capture of the client program, showing the GUI for user parameter setting.

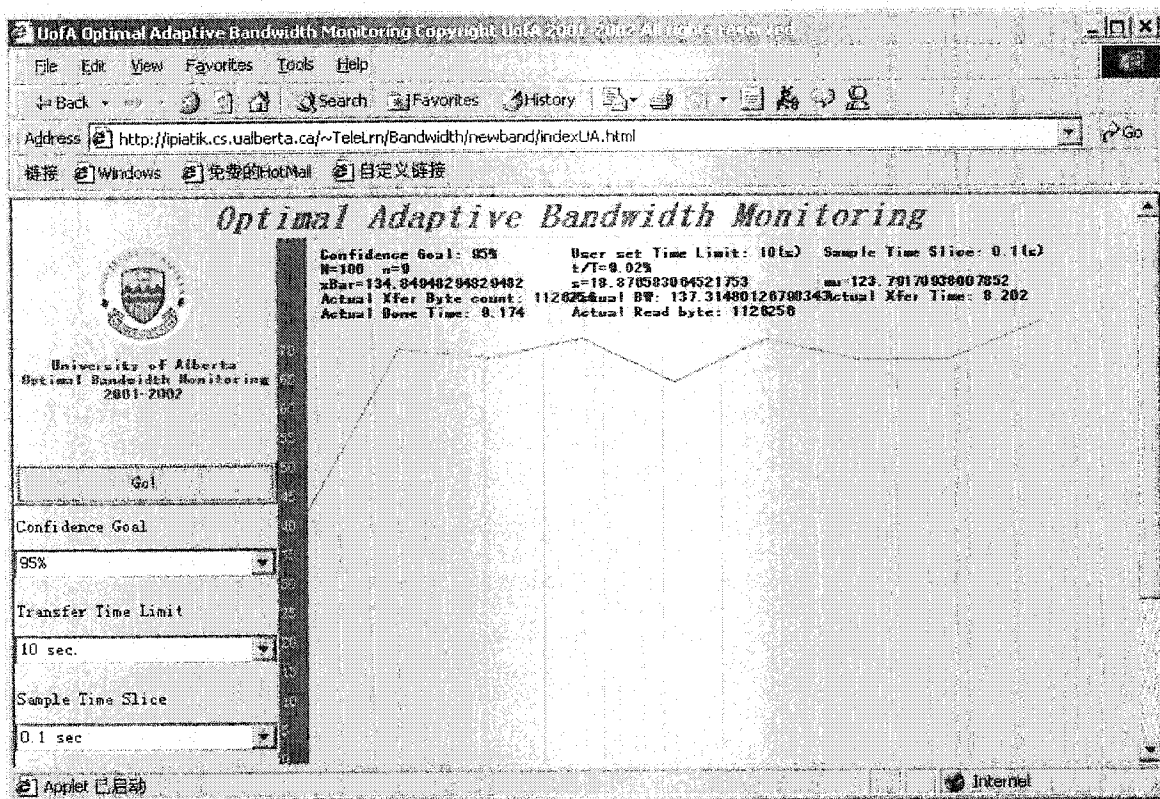


Figure 6.1, Screen capture of the Client program

When the experimenter sets the parameters and presses the “Go” button, the following steps are executed:

1. The client creates a TCP socket connection to the server program by connecting to the port on the server host to which the server program is listening. Then it sends a TEST command to the server. The server responds by sending a stream of random bytes to the client via the connection.
2. The client receives the testing stream on its side and at the same time counts the bytes received. For every time slice, the client divides the number of bytes received in this time slice to determine the bandwidth. This is a new bandwidth sample  $x_n$ .
3. The client calculates  $\bar{x}$ ,  $s^2$ ,  $\mu_{est}$ , and  $V(n)$  as defined in Chapter 3. We use the “moving average” method described in Chapter 4 as the termination condition. When

there are two/three (depending on  $s/\bar{x}$ ) consecutive drops in the average of the past four  $V$  values, proceed to Step 4; otherwise, go to Step 2.

4. The client tears down the testing connection, makes a new connection<sup>15</sup> to the server and sends a TRANSMIT command to the server, along with the target object size, which equals  $(T - t) \cdot \mu_{est}$ .

5. The server sends a byte stream of length  $(T - t) \cdot \mu_{est}$  to the client.

6. The client receives the byte stream (which represents the object), records the finish time, calculates the statistics about this transmission and updates them on the right-hand region of the applet, as shown in Figure 6.1.

To minimize the computation of the statistics when each new sample is obtained, part of them can be pre-computed and stored in a static array. We organize the component

$\frac{d_{(\alpha,n)}}{\sqrt{n}} \cdot \sqrt{\frac{N-n}{N-1}}$  into a multi-dimensional array of constants, with  $(\alpha,n)$  representing two

dimensions. Moreover, the computation  $s^2$  can be performed using the fast computation

form  $s^2 = \frac{1}{n-1} \left( \sum_n x_i^2 - n\bar{x}^2 \right)$ . Therefore, if we maintain the intermediate values

$\sum_n x_i$  and  $\sum_n x_i^2$ , the computation costs of  $\bar{x}$  and  $s$  are minimized.

## 6.2 Experiments on Campus Network

We first performed experiments on the University of Alberta campus network. The hosts

---

<sup>15</sup> In order to determine the maximum available bandwidth between client and server, the server will keep pumping random bytes to the first TCP connection (TEST) as quickly as possible. As a result, the first connection is still filled with random bytes in its buffer when bandwidth sampling is finished. Therefore, we choose to tear it down (the server socket will catch an exception and discard the random bytes) and make a new connection for real object transmission. We assume that the bandwidth of a TCP connection is determined by the path between client and server; therefore, the estimation made on the first connection can be applied to the new one.



and network setting are shown in Figure 6.2. Both the server and the client run Red Hat Linux 6 and the client uses Netscape Communicator 4.0 as browser. The experiment results are presented in Table 6.1<sup>16</sup>.

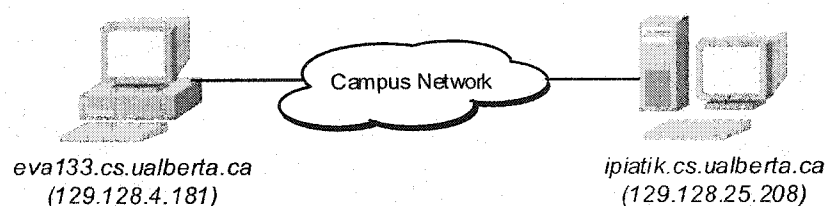


Figure 6.2, Experiment set up on campus network

<b>Confidence Goal</b>	95%	75%
<b>Average No. Of Sample</b>	14.6	8.7
<b>Sample Average (kbps)</b>	141.2	140.6
<b>Sample Standard Error (kbps)</b>	7.71	8.65
<b>Estimation (kbps)</b>	137.8	138.5
<b>Actual Bandwidth Average (kbps)</b>	141.0	140.8
<b>Actual Object size (Byte)</b>	1337328	1360624
<b>Actual Transmission Time (s)</b>	9.49	9.67
<b>Actual Total Time (s)</b>	9.80	9.85
<b>Total No. of Exception in 100 runs</b>	4	20

Table 6.1, Experiment results on the campus network, with a Time Limit of 10s, Time Slice of 0.02s. Results are for two different confidence goals  $\alpha = 0.95/0.75$ .

The results are based on experiments with the parameters of a “Time Limit” = 10 seconds, a “Time Slice” = 0.02 second, and a “Confidence Goal” = 95% and 75%. For each parameter setting, we perform the transmission 100 times and report the averages.

Table 6.1 reveals that when  $\alpha = 0.95$ , it takes 14.6 samples (on average) out of 500 for

<sup>16</sup> Experiments reported in this section (Table 6.1 and Table 6.2) were performed in normal office hours on weekdays in November 2001. Transmission requests were issued from the client every 30 seconds to avoid excessive network burden.

bandwidth testing. 4% of the transmissions exceed the time limit; 94.8% of the entire time limit is used for actual object transmission. The results for  $\alpha = 0.75$  are similar, except that it takes fewer testing samples and gives a less aggressive under-estimation on the available bandwidth. This feature, in turn results in more transmissions exceeding the time limit (20%) but a higher utilization of the time limit for actual object transmission (96.6%).

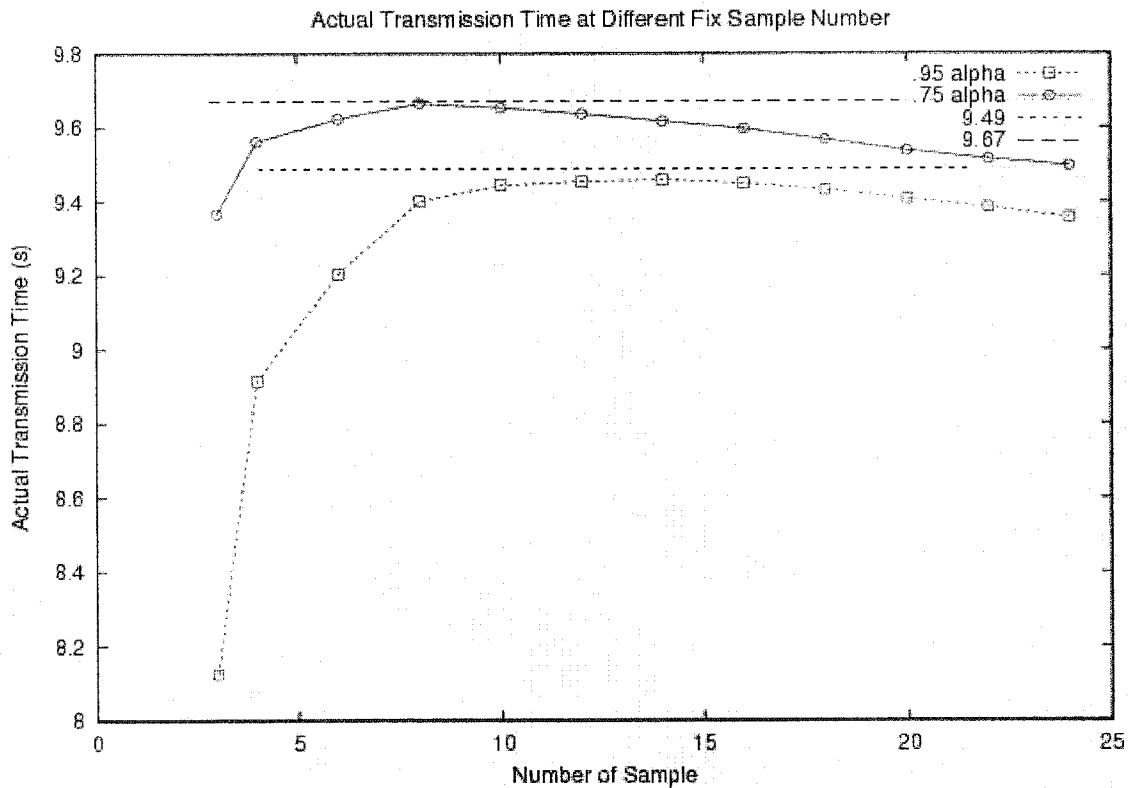
The campus network is a network environment where server and client are very close in network topology. Therefore, the path between the server and the client is extremely stable. The round trip time (RTT, obtained by the “ping” utility of Linux) between two computers is in the order of several milliseconds; the sample standard error is only around 6% of the sample average, as shown in Table 6.1. In this network setup, the results of the experiments are extremely good. The actual bandwidth average of the entire time limit is very close to the sample average, verifying our assumption that the bandwidth samples can serve as good representatives of the bandwidth of the entire time limit. The confidence level of the user-specified time limit is also preserved very well.

To demonstrate the strength of the algorithm to approximate the optimal amount of bandwidth samples, we will next present the comparison of our algorithm and the fixed sampling method. As mentioned in our simulation experiments, the fixed sampling method uses the same formula to estimate available bandwidth (based on t-distribution), except that it always uses a fixed number of bandwidth samples. Table 6.2 shows the actual data transmission time achieved with the fixed sampling method. The parameters used in the experiments of Table 6.2 are the same as those in the experiments of Table 6.1.

Sample No.	3	4	6	8	10	12	14	16	18	20	22	24
0.75	9.37	9.56	9.62	9.66	9.65	9.64	9.62	9.60	9.57	9.54	9.52	9.50
0.95	8.13	8.91	9.20	9.40	9.44	9.45	9.46	9.45	9.43	9.41	9.38	9.36

**Table 6.2.** Average actual transmission time for the fixed sampling method (each figure is the average of 100 runs)

To compare it with our algorithm, which determines the sample number dynamically, we plot the results of the fixed sampling method along with the results of our dynamic algorithm in Figure 6.3.

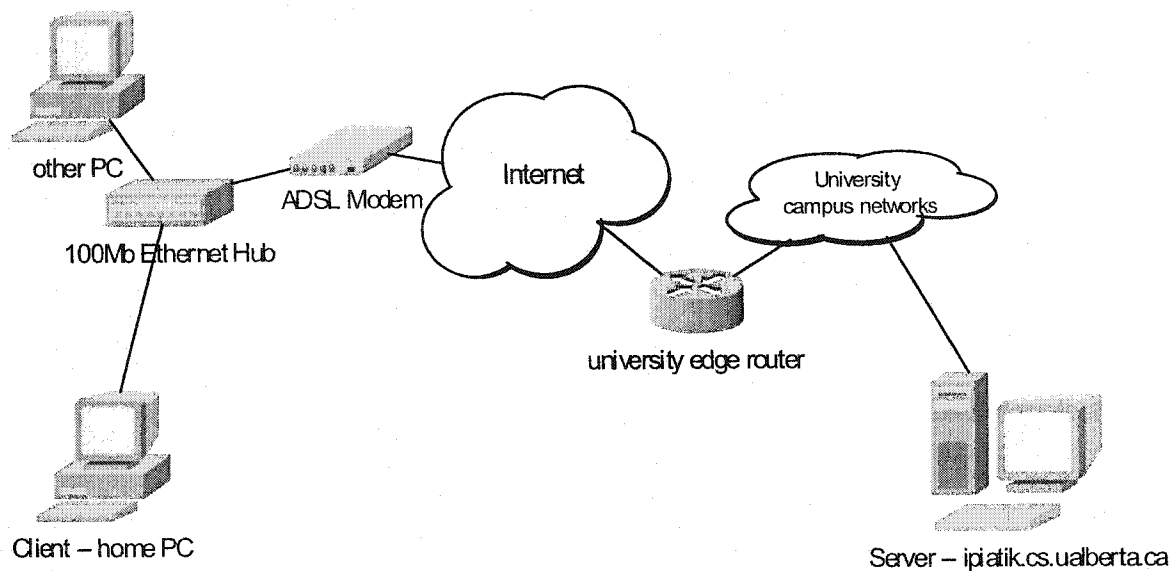


*Figure 6.3, Comparison of actual transmission time.*

The horizontal axis of Figure 6.3 is the number of fixed samples used in the “fixed sampling method”. The vertical axis is the (average) actual object transmission time. Apart from the two curves representing the actual transmission time of the fixed sampling method with  $\alpha = 0.75$  and  $\alpha = 0.95$ , we also draw two horizontal dotted lines at  $y=9.67$  and  $y=9.49$ , which are the actual transmission time achieved by our “dynamic sampling method” at  $\alpha = 0.75$  and  $\alpha = 0.95$ , respectively (see Table 6.1). We can clearly see that our algorithm approximates the optimal amount of samples much better and therefore achieves the best actual object transmission time.

### 6.3 Experiments on the Internet

In the second experimental environment setting, the server is the same computer as in the campus network experiments. It is located in the Vision and Multimedia Communication Lab of the Computing Science department at the University of Alberta. The client is a home PC connected to the Internet via a local broadband ISP with ADSL modem. Figure 6.4 is the detailed network diagram. A trace route result shows that there are 11 hops between the client and the server. The client runs Microsoft Windows 2000 Professional and Netscape Navigator 4.0.



**Figure 6.4.** Experiment set-up on Internet through broadband connection

We present the experiment results in Table 6.3.<sup>17</sup> We set the “Time Limit” parameter to 10 seconds, the “Time Slice” parameter to 0.05 seconds, and the “Confidence Goal” parameter to 95% and 75%. We performed transmissions 100 times for both 95% and 75% confidence levels and report the averages.

<sup>17</sup> All the experiments described in sections 6.3, 6.4 and 6.5 were performed in normal office hours on weekdays in March 2002. Transmission requests were issued from the client every 30 seconds to avoid excessive network burden.

<i>Confidence Goal</i>	95%	75%
<i>Average No. Of Sample</i>	14.0	8.4
<i>Sample Average (kbps)</i>	156.0	152.9
<i>Sample Standard Error (kbps)</i>	24.34	28.93
<i>Estimation (kbps)</i>	144.8	145.7
<i>Actual Bandwidth Average (kbps)</i>	156.4	156.4
<i>Actual Object size (Byte)</i>	1344991	1394117
<i>Actual Transmission Time (s)</i>	8.60	8.92
<i>Actual Total Time (s)</i>	9.61	9.69
<i>Total No. of Exception in 100 runs</i>	3	16

*Table 6.3, Experiment results on the Internet, with a Time Limit of 10s and a Time Slice of 0.05. Results are for two different confidence goals.*

The results in Table 6.3 follow a similar pattern to those in Table 6.1. One obvious difference is that this time, the sample standard error is significantly larger (up to about 20% of the sample average). Therefore, the algorithm makes a more conservative estimation of the future bandwidth, and, in turn, yields a lower actual transmission time. In spite of this difference, 86-89% of the time limit is used for real data transmission, and the time limit confidence level is well preserved. The 14.0/8.4 sample numbers used for bandwidth testing are almost same as those in Table 6.1. However, this time it is 14.0/8.4 out of 200 total slices, as each time slice is 0.05 second in this experiment. Because the bandwidth on the Internet shows more variance than on the campus network, a larger portion of the time limit is devoted to bandwidth testing.

## 6.4 Time Slice Choice

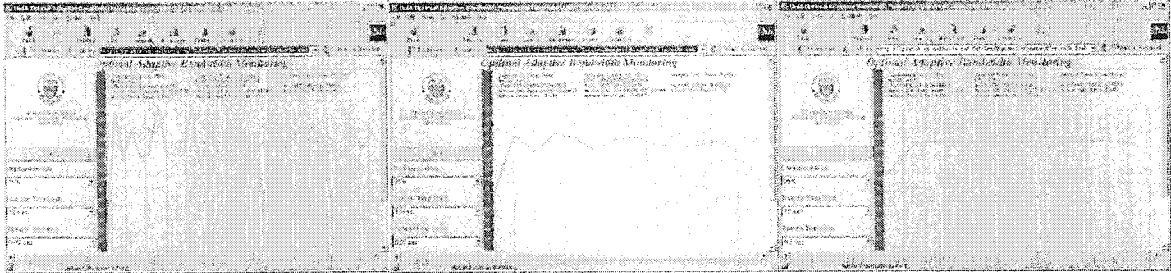
In the experiments discussed in the previous two sections, we have chosen two different time slices, 0.02 second for the campus network and 0.05 second for the home Internet connection. How to choose a suitable time slice size is an interesting question. When the time slice is small, the benefit is that the cost of each sample is small, and the

problem is that the sample standard error will be large, which force the algorithm to make a more conservative estimation. A large time slice has exactly the reverse problems and benefits. Our experience is that a reasonable slice size is within the range of 0.01-0.5 seconds. Different network characteristics have different optimal slice sizes within that range. As we will demonstrate in the experiment described in Table 6.4, the differences in the results obtained by using different slice sizes within the reasonable range are relatively small.

In this experiment, we ran the transmissions in the home Internet connection with three different time slices: 0.02 second, 0.05 second, and 0.1 second. The time limit was 10 seconds and confidence goal was set at 95%. We ran the transmission 100 times and report the average in Table 6.4. Figure 6.5 shows examples of the bandwidth sample curves drawn by our client program.

<i>Time Slice Size (s)</i>	0.02	0.05	0.10
<i>Total Time Slices Number</i>	500	200	100
<i>Average No. Of Sample</i>	25.5	14.0	9.0
<i>Sample Average (kbps)</i>	153.8	156.0	154.7
<i>Sample Standard Error (kbps)</i>	47.43	24.34	20.12
<i>Estimation (kbps)</i>	136.4	144.8	143.3
<i>Actual Bandwidth Average (kbps)</i>	156.3	156.4	155.4
<i>Actual Object size (Byte)</i>	1290585	1344991	1299929
<i>Actual Transmission Time (s)</i>	8.26	8.60	8.34
<i>Actual Total Time (s)</i>	9.06	9.61	9.51

*Table 6.4, Comparison of different time slice sizes.*



*Figure 6.5, Screen Capture of three experiments (0.02s time slice on the left, 0.05s at the center and 0.10s on the right)*

We can make several interesting observations from these experiment results. First, as the time slice size becomes larger, the sample standard error becomes smaller, and so does the number of samples the algorithm takes for bandwidth testing. However, even when the number of samples goes down, the time actually spent on bandwidth testing goes up (from around 5% to 7% to 9%) since the time slice is larger for each sample. Second, the under-estimation margin is determined not only by the sample standard error, but also by the number of samples. We will compare the under-estimation margin of a 0.05s time slice and a 0.10s time slice. Although the sample standard error of the 0.10s slice is smaller than that of 0.05s slice (20.12 vs. 24.34), its under-estimation margin is larger: for 0.05, it is  $156.0 - 144.8 = 11.2$ ; for 0.10, it is  $154.7 - 143.3 = 11.4$ , because the under-

estimation margin  $(d_{(\alpha,n)} \cdot \frac{s}{\sqrt{n}} \cdot \sqrt{\frac{N-n}{N-1}})$  is determined not only by  $s$ , but also by  $n$ . The

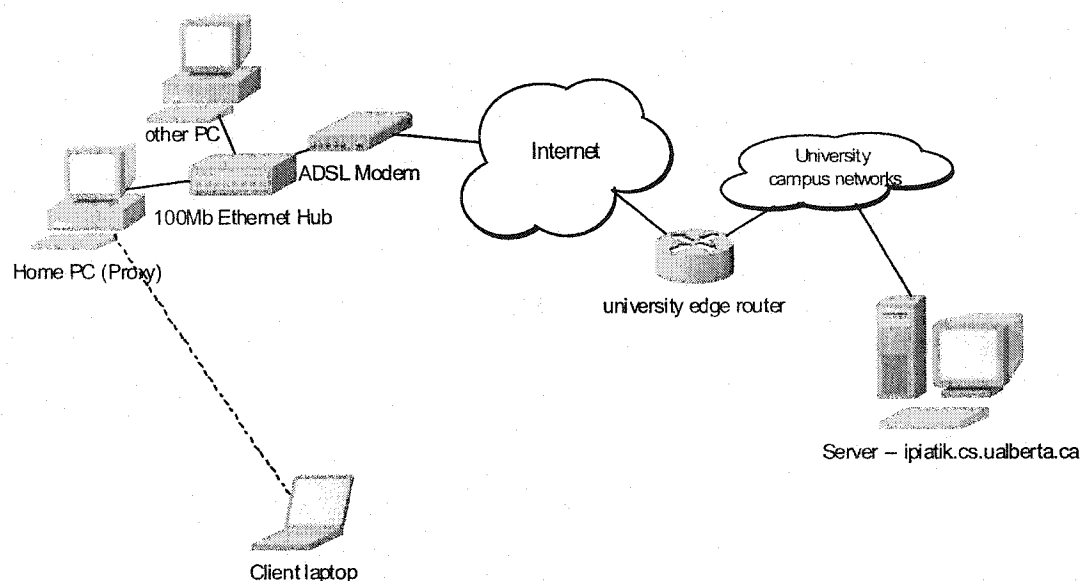
0.05s time slice has an advantage because it takes 14 samples, while the 0.10s time slices takes only 9. Therefore, the 0.10s time slice loses to 0.05s time slice because it uses a larger portion of time limit for bandwidth testing and makes larger under-estimation margin. On the other side, when we comparing 0.02s and 0.05s, the 0.02s time slice uses a smaller portion for bandwidth testing, but suffers a huge under-estimation margin because of its huge sample standard error.

With these observations, we can see that there are several dynamics going on when the time slice changes. Often one dynamic offsets another. Our experiments results show that when the time slices are within a reasonable range, the difference in performance is

relatively small. For example, in Table 6.4, the average transmitted object sizes yielded by 0.02s and 0.10s time slice are only about 3-4% under that of the 0.05s time slice.

## 6.5 Experiments on Network with Wireless Link

Our last experiment setting is similar to that described in Section 6.3, except that we add a wireless link on the last hop of the path. The network topology is like that in Figure 6.6.



*Figure 6.6. Experiment set-up on Internet with a wireless link*

As shown in Figure 6.6, the settings are almost identical to those in Figure 6.4 except that on the client side, a new laptop PC now serves as the client. The dashed line between the “Home PC” and “Client laptop” indicates that a wireless link is used to connect the two. The laptop is equipped with Intel Pentium MMX 233Mhz CPU, 64MB RAM and is running the Window 98 operating system. Both the home PC and laptop are installed with a US Robotics Wireless Access card conforming to the 802.11b standard, with a maximum specified throughput of 11Mbps. The two 802.11b cards operate in the “Ad-Hoc” mode. No dedicated Access Point (a.k.a. base station) is present. The IP addresses assigned to the two wireless cards start with 192.168., so that they are internal addresses



invisible from the outside Internet. The Home PC acts as the http proxy of the client laptop.

Although the current 802.11b wireless technology has become mature and reliable, there are special challenges to our algorithm related to the wireless link. The experiments setting we created will be the typical network architecture in future wireless application: wireless devices connected to the Internet with an unreliable wireless link as the last hop. Therefore, how our algorithm performs in such environment is especially interesting.

As the performance of the wireless link is extremely sensitive to the distance between the hosts and the existing radio interference, we performed the experiments at two distances of the link. In first experiment, we set the laptop close to the home PC (around 10 feet). Then in second experiment, we tried to find out the largest distance in which the two computers can detect each other (around 100 feet in a building with walls and other obstacles) and set the distance to that. We set the time limit to 10 seconds, the time slice to 0.10 second, and the confidence level to 95% and ran the transmission 100 times in each experiment. The results are presented in Table 6.5.

<i>Wireless link distance (feet)</i>	10	100
<i>Total Time Slices Number</i>	100	100
<i>Average No. Of Sample</i>	10.0	11.1
<i>Sample Average (kbps)</i>	155.9	124.5
<i>Sample Standard Error (kbps)</i>	26.3	39.7
<i>Estimation (kbps)</i>	141.6	101.4
<i>Actual Bandwidth Average (kbps)</i>	162.3	124.0
<i>Actual Object size (Byte)</i>	1259704	896529
<i>Actual Transmission Time (s)</i>	7.77	7.29
<i>Actual Total Time (s)</i>	9.01	8.64
<i>Total No. of Exception in 100 runs</i>	2	11

*Table 6.5, Experiment results on the Internet with a wireless link*

Table 6.5 reveals that the results for a distance of 10 feet (column 2 of Table 6.5) are similar to those in which no wireless link exists (column 3 of Table 6.4). The average bandwidth is similar; the confidence level is well preserved. The sample standard error is relatively larger with the presence of a wireless link, but the algorithm adjusts itself well to accommodate it.

On the other hand, the results for a distance of 100 feet show a new pattern. The average bandwidth achieved is significantly smaller than the bandwidth achieved before; the sample standard error is significantly larger; and most importantly, the confidence level is not preserved very well. As we investigated the details of the 100 runs, we developed a better understanding of the problem. Among the 11 runs that failed to finish in time, 6 of them finished after 13 seconds, which was very rare in previous experiments. We believe that the cause was the transient loss of the connection (blackout) of the wireless link. Therefore, with the presence of a wireless link, especially when the distance is on the verge of the specification distance, the unpredictable nature of the wireless link invalidates, to a degree, our assumption that “bandwidth samples are unbiased representatives of the whole population”. However, we still consider 11% of exception runs an acceptable result under extreme conditions. In most application cases, the client is most likely to be somewhere between 10 feet and 100 feet – i.e., between “very close” and the “maximum distance in the specification”.

Finally, we point out that our implementation of the algorithm ran smoothly on a relatively old model of laptop computer (with Intel Pentium MMX 233MHz CPU and 64MB RAM). Moreover, we have implemented the client program as a Java applet running within the web browser. The system overhead associated with the Java virtual machine is expected to be higher than most real network-aware applications using native languages in development. We are therefore very confident that our algorithm is simple enough to be integrated into network-aware applications running on a variety of hardware, including handheld devices with limited computing power.

## Chapter 7

# Conclusions and Directions for Future Research

In this thesis, we have presented a bandwidth estimation algorithm for network-aware multimedia delivery application. The algorithm is suitable for the interactive transmission of relatively large multimedia objects in both single server and multiple server network environments. The main target QoS parameter is a user-specified time limit for transmission of the object. The algorithm uses the first fraction of the time limit to do bandwidth sampling and then uses this knowledge to make an estimation of the future available bandwidth. We designed of the algorithm with two goals. First, we wanted to provide a guarantee on the time limit QoS parameter to a certain confidence level. This confidence level was achieved by making an estimation of the future bandwidth on the basis of a t-distribution statistical model. It enables the algorithm to make an accurate conservative estimation to preserve the user-specified time limit. Second, we aimed to approximate the optimal amount of bandwidth sampling in order to maximize the transmitted object size. This result was achieved by monitoring the  $V(n)$  values we have defined and fine tuning the termination conditions of bandwidth testing. Our simulation results and experiments in real networks both confirm that both of our goals were well achieved.

Unlike related works we have discussed, which mainly focus on building a general model of the TCP throughput using transport and network layer metrics, our algorithm

aims to solve the bandwidth estimation problem in real applications. The algorithm works completely in the application layer and mainly on the receiver side. It measures the throughput of the TCP connection in terms of the actually received object size. Since it does not make use of specific transport and network layer metrics, it is expected to work with all versions of TCP implementations.

During our building of the bandwidth model, we made several assumptions about the TCP bandwidth samples. We assumed that bandwidth samples follow a normal distribution; the bandwidth random variable is stationary during the entire period of time; and we ignored the potential auto-correlations of the bandwidth samples. Our experiment results in real networks show that despite these simplifications, our algorithm works accurately enough to meet our goal of providing a meaningful confidence level in preserving the time limit. Our implementation of the algorithm in the real network environments also demonstrates the simplicity of our algorithm. We believe the computation cost of the algorithm is low enough to be implemented in real-time on a wide range of devices with relatively limited computation power.

Several future studies are feasible. Firstly, as the main purpose of this research is to facilitate the development of network-aware applications, we would like to create a clearly defined API for the bandwidth estimation module in these applications and build an implementation ready to use for application programmers. We plan to use the Java language to define a set of interfaces, including the methods needed for the bandwidth estimation module in both the client and the server, and to implement them in Java classes.

Secondly, one component of the algorithm that we are particularly not very satisfied with is the multi-server extension. As we have discussed in Chapter 5, when we are using multiple channels, an important question is how to determine the confidence level for each of them. Our current method is to set all the confidence levels to be the same and then use an algorithm to “drop” some of them. This method is a very rough approximation of the optimal strategy, which should allocate confidence levels according

to the relative variance on all the channels. We plan to look into the issue and find better strategies for approximating the optimal result.

Finally, given a reliable bandwidth estimation scheme, another important issue in network-aware application development is fast and accurate transcoding algorithms of media objects. After determining the target size of the object, the server(s) need the transcoding algorithm to tailor the requested object to the target size accurately and quickly. The media object can be still images, audio/video clips, or 3D graphic objects. For still images, the popular JPEG image compression algorithm is notorious for its poor control of the compressed image size. The new image standard JPEG2000 [Christopoulos, 2000] has more support for the *scalability* of image coding. We plan to look into the issue of how to apply these new techniques in network-aware application design.

## Bibliography

- [Allman, 1999] M. Allman, V. Paxson, W. Stevens. TCP Congestion Control. Request for comment 2581, April 1999.
- [Balakrishnan, 1995] H. Balakrishnan, S. Seshan et al. Improving TCP/IP performance over wireless networks. In *Proceedings of the 1<sup>st</sup> ACM International Conference on Mobile Computing and Networking (Mobicom)*, November 1995.
- [Balakrishnan, 1997] H. Balakrishnan, V. Padmanabhan et al. A Comparison of mechanisms for improving TCP performance over wireless links. *ACM/IEEE Transactions on Networking*, Volume 5, No. 6, December 1997.
- [Balakrishnan, 1997B] H. Balakrishnan, M. Stemm et al. Analyzing stability in wide-area network performance. In *Proceeding of ACM SIGMETRICS '97*, pages 2-13, June 1997.
- [Blake, 1998] S. Blake, D. Black et al. An Architecture for Differentiated Services, Request for comment 2475, December 1998.
- [Bolliger, 1998] J. Bolliger and T. Gross. A Framework-based Approach to the Development of Network-aware Applications. *IEEE Transactions on Software Engineering*, May 1998.

- [Bolliger, 1999] J. Bolliger, T. Gross, U. Hengartner. Bandwidth Modelling for Network-Aware Applications. *In Proceeding of IEEE INFOCOM 1999*, page 1300-1309, March, 1999.
- [Braden, 1989] R. Braden. Requirements for Internet hosts – Communication Layers. Request for comment 1122, October 1989.
- [Braden, 1997] R. Braden, L. Zhang et al. Resource ReSerVation Protocol - Version 1 Functional Specification, Request for comment 2205, September 1997.
- [Brakmo, 1995] L. Brakmo, and L. Peterson. TCP Vegas: End-to-end congestion avoidance on a global internet. *IEEE Journal of Selected Areas in Communication*. Volumn 13, No. 8, page 1465-1480, October 1995.
- [Casetti, 2000] C. Casetti, M. Meo. A New Approach to Model the Stationary Behavior of TCP Connections. *In Proceedings of INFOCOM 2000*, Vol. 1, page 367-375, 2000.
- [Cheng, 2001] I. Cheng, A. Basu. QoS Specification and Adaptive Bandwidth Monitoring for Multimedia Delivery. *In Proceeding of EUROCON 2001*, Bratislava, Slovakia, July 2001.
- [Christopoulos, 2000] C. Christopoulos, A. Skodras et al. The JPEG2000 Still Image Coding System: An Overview. *IEEE Transactions on Consumer Electronics*, Vol. 46, No. 4, pages 1103-1127, November 2000.
- [Floyd, 1993] S. Floyd, V. Jacobson. Ramdom Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, page 397-413, August 1993.

- [Grossman, 2002] D. Grossman. New terminology and clarifications of Diffserv, Request for comment 3260, April 2002.
- [Haas, 1997] Z. Haas, P. Agrawal. Mobile-TCP: an asymmetric transport protocol design for mobile systems In *Proceedings of the IEEE International Conference on Communications (ICC '97)*, 1997.
- [Harnett, 1982] D. Harnett. *Statistical Methods*, Third Edition. Addison-Wesley Publishing Company, Inc. 1982
- [Hoe, 1996] J. Hoe. Improving the start-up behavior of a congestion control scheme for TCP. In *Proceeding of ACM SIGCOMM '96*, page 270-280, August 1996.
- [Jacobson, 1988] V. Jacobson. Congestion Avoidance and Control. In *Proceeding of the SIGCOMM '88 Symposium*, page 314-332, August 1988.
- [Kumar, 1998] A. Kumar. Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link. *IEEE/ACM Transactions on Networking*, Volume 6, No. 4, page 485-498, August 1998.
- [Lakshman, 1997] TI Lakshman, U. Madhow. The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss. *IEEE/ACM Transactions on Networking*, Volume 3, No. 3, page 336-350, June 1997.
- [Lin, 1998] D. Lin, H. Kung. TCP fast recovery strategies: Analysis and improvements. In *Proceeding of INFOCOM '98*, page 263-272, March 1998.



- [Maly, 1997] K. Maly et al., Interactive Distance Learning over Intranets. *IEEE Internet Computing*, Volume 1, No. 1, Jan/Feb 1997.
- [Mathis, 1996] M. Mathis, J. Mahdavi, S. Floyd. TCP Selective Acknowledgement Options. Request for comment 2018, October 1996.
- [Mathis, 1997] M. Mathis, J. Semke, J. Mahdavi. The Macroscopic Behaviour of the TCP Congestion Avoidance Algorithm. *Computer Communication Review*, Volume 27, Number 3, July 1997.
- [Miloucheva, 1995] I. Miloucheva, Quality of Service Research for Distributed Multimedia Applications. *ACM Pacific Workshop on Distributed Multimedia Systems*, Honolulu, Hawaii, 1995.
- [Padhye, 1998] J. Padhye, V. Firoiu, D. Towsley, J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. *In Proceedings of the ACM SIGCOMM*, 1998.
- [Parsa, 1999] C. Parsa, G. Aceves. Improving TCP congestion control over Internets with heterogeneous transmission media. *In Proceedings of IEEE International Conference on Network Protocols (ICNP '99)*, 1999.
- [Peterson, 2000] L. Peterson, B. Davie. *Computer Networks, A Systems Approach*. Morgan Kaufman Publishers, 2000.
- [Postel, 1981] J. B. Postel. Transmission Control Protocol. Request for comment 793, September, 1981.
- [Press, 1993] W. Press, B. Flannery, S. Teukolsky, W. Vetterling. *Numerical*

Recipes in C, second Edition. Cambridge University Press, 1993.

- [Ramakrishnan, 1990] K. Ramakrishnan, R. Jain. A Binary Feedback scheme for congestion avoidance in computer networks with a connectionless network layer. *ACM Transactions on Computer Systems*, Volume 8, No. 2, page 158-181, May 1990.
- [Ramakrishnan, 1999] K. Ramakrishnan, S. Floyd. A proposal to add explicit congestion notification (ECN) to IP. Request for comment 2481, January 1999.
- [Stevens, 1997] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. Request for comment 2001, January 1997.
- [Tanenbaum, 1996] A. Tanenbaum. Computer Networks, third Edition. Prentice-Hall, Inc., 1996.
- [Tsaoussidis, 2000] V. Tsaoussidis, H. Badr. TCP-probing: towards an error control schema with energy and throughput performance gains. In *Proceedings of the 8<sup>th</sup> IEEE International Conference on Network Protocols*, 2000.
- [Tsaoussidis, 2002] V. Tsaoussidis, I. Matta. Open issues on TCP for mobile computing. *Wireless Communications and Mobile Computing*, No.2, 2002.
- [Vogel, 1995] A. Vogel, B. Kerherv'e, G. von Bochmann, J. Gecsei. Distributed Multimedia and QoS: a Survey, *IEEE Multimedia*, Vol.2 No.2, Summer 1995.

- [Wang, 1999] X. Wang and H. Schulzrinne. Comparison of adaptive Internet multimedia applications, *IEICE Transactions on Communications*, Volume E82-B, Number 6, pages 806-818, June 1999.
- [White, 1997] P. White. RSVP and Integrated Services in the Internet: A Tutorial, *IEEE Communications*, May 1997.
- [Zhang, 1993] L. Zhang. RSVP: A new resource reservation protocol. *IEEE Network Magazine*, 7(5):8-18, September 1993.
- [Zwillinger, 1996] D. Zwillinger. CRC Standard Mathematical Tables, 30th Edition. CRC Press, January 1996.

# Appendix 1

## Experimental Data

1. Data plotted in Figure 4.3. The data below are number of time slices used for real data transmission (out of 500 total time slices, for  $\alpha = 0.95$ ) for four different number of fixed sample size (10, 20, 30, 40). The mean of the bandwidth random variables are always set at 100.0, while the standard deviation is from the set {2.5, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60}. Each number is the average of 200 runs, excluding those runs that fail to finish the transmission within time limit.

	10	20	30	40
2.5	483	475	466	457
5	476	470	462	453
10	461	460	454	447
15	446	451	447	440
20	431	440	439	434
25	416	431	431	427
30	401	421	423	420
35	387	411	416	414
40	372	401	408	407
45	357	392	402	401
50	343	382	393	396
55	331	374	389	391
60	319	366	383	386

2. Data plotted in Figure 4.4(a)(b)(c)(d). The data below are number of time slices used for real data transmission (out of 500 total time slices, for  $\alpha = 0.95$ ) for our

algorithm that determine the number of samples dynamically. The mean of the bandwidth random variables are always set at 100.0, while the standard deviation is from the set {2.5, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60}. Each number is the average of one group (200 runs), excluding those runs that fail to finish the transmission within time limit. We have run 3 groups of experiments.

	run 1	run 2	run 3
2.5	484	484	484
5	477	475	476
10	460	463	462
15	449	450	450
20	440	439	436
25	431	428	432
30	421	426	426
35	414	418	418
40	407	406	408
45	400	400	400
50	395	394	399
55	390	389	389
60	384	384	386

3. Data plotted in Figure 4.5. The data below are number of time slices used for real data transmission (out of 100 total time slices, for  $\alpha = 0.95$ ) for four different number of fix sample size (5, 10, 15, 20). The mean of the bandwidth random variables are always set at 100.0, while the standard deviation is from the set {2.5, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60}. Each number is the average of 200 runs, excluding those runs that fail to finish the transmission within time limit.

	5	10	15	20
2.5	93	89	85	79
5	91	88	84	78
10	86	85	82	77
15	82	82	80	76
20	77	80	78	74
25	72	76	76	73
30	68	74	74	71
35	63	72	72	70

40	59	69	70	69
45	54	67	68	68
50	50	64	67	67
55	47	62	65	66
60	44	60	64	65

4. Data plotted in Figure 4.6(a)(b)(c)(d). The data below are number of time slices used for real data transmission (out of 100 total time slices, for  $\alpha = 0.95$ ) for our algorithm that determine the number of samples dynamically. The mean of the bandwidth random variables are always set at 100.0, while the standard deviation is from the set {2.5, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60}. Each number is the average of one group (200 runs), excluding those runs that fail to finish the transmission within time limit. We have run 3 groups of experiments.

	run 1	run 2	run3
2.5	93	93	93
5	91	91	91
10	87	87	86
15	83	82	82
20	80	79	79
25	76	76	76
30	73	74	73
35	71	72	70
40	69	69	69
45	68	68	67
50	67	66	66
55	65	66	65
60	65	64	64

5. This dataset is used to create Figure 4.7. It reports how many runs (out of 200) fail to finish within time limit in fixed sampling algorithm in 100 total slices.

	5	10	15	20
2.5	13	12	14	17
5	13	12	14	16
10	13	12	14	17

15	13	12	14	17
20	13	12	14	17
25	13	12	14	17
30	13	12	14	17
35	13	12	14	17
40	13	12	14	17
45	13	12	14	17
50	13	12	14	17
55	12	12	14	17
60	12	12	14	16

6. This dataset is used to create Figure 4.7. It reports how many runs (out of 200) fail to finish within time limit in dynamic sampling algorithm (100 total slices). We have run three groups (each 200 runs). Results in Figure 4.7 are the average of three groups.

	run 1	run 2	run 3
2.5	14	11	10
5	15	11	14
10	8	12	14
15	17	14	13
20	20	7	18
25	11	13	18
30	11	17	10
35	18	11	16
40	15	21	14
45	17	17	13
50	11	13	17
55	16	18	15
60	15	15	14

7. This dataset is used to create Figure 4.8. It reports how many runs (out of 200) fail to finish within time limit in fixed sampling algorithm in 500 total slices.

	10	20	30	40
2.5	8	11	13	11
5	8	11	13	11
10	8	11	13	11
15	8	11	13	11
20	8	11	13	11
25	8	11	13	11

30	8	11	13	11
35	8	11	13	11
40	8	11	12	11
45	9	11	10	11
50	9	12	9	9
55	9	12	9	7
60	9	12	8	7

8. This dataset is used to create Figure 4.8. It reports how many runs (out of 200) fail to finish within time limit in dynamic sampling algorithm (500 total slices). We have run three groups (each 200 runs). Results in Figure 4.8 are the average of three groups.

	run 1	run 2	run 3
2.5	7	13	8
5	13	10	8
10	7	14	13
15	14	9	13
20	6	16	8
25	15	9	13
30	13	15	11
35	11	14	13
40	9	17	14
45	11	10	16
50	16	10	13
55	11	14	15
60	17	11	16

9. This dataset is the experiment results of multi-server bandwidth estimation algorithm for total slice of 100 (Figure 5.3). We present the real transmitted object size for both dynamic algorithm and fixed sampling algorithm (for fix sample size of 8, 10, 12, 14, 16, 18, and 20). The result of each setting is in one table; each table contains results for 169 parameter settings of variances (13 variances for channel #1 and 13 variances for channel #2). Each value in the table is an average of 1000 runs.

#### Real Transmitted Object Size for Dynamic Algorithm

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	10043	9836	9452	9079	8720	8471	8161	7861	7592	7361	7172	7037	6781
0.5	10013	9809	9440	9048	8725	8371	8070	7812	7691	7356	7188	6997	6790
1.0	9960	9766	9385	9021	8699	8456	8080	7830	7532	7338	7135	7055	6851
1.5	9915	9724	9359	9017	8683	8403	8044	7786	7597	7308	7237	6996	6786



2.0	9858	9670	9323	8973	8644	8335	8001	7792	7541	7323	7183	6990	6789
2.5	9818	9652	9291	8910	8630	8296	7983	7727	7483	7299	7096	6913	6821
3.0	9766	9592	9250	8892	8569	8293	7941	7707	7460	7291	7086	6970	6782
3.5	9713	9550	9213	8869	8543	8269	7867	7702	7503	7249	6995	6959	6748
4.0	9672	9507	9172	8859	8540	8243	7963	7663	7444	7172	7068	6848	6674
4.5	9639	9473	9155	8805	8470	8127	7835	7617	7399	7223	7014	6799	6691
5.0	9598	9431	9136	8767	8442	8123	7868	7621	7429	7200	7009	6816	6748
5.5	9568	9400	9113	8780	8436	8135	7855	7557	7335	7160	6878	6873	6684
6.0	9534	9365	9079	8745	8390	8107	7742	7538	7402	7126	6961	6790	6654

#### Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 8

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	9822	9658	9315	8963	8613	8304	7979	7642	7355	6999	6723	6435	6126
0.5	9807	9637	9311	8958	8643	8242	7938	7633	7366	6933	6728	6438	6146
1.0	9770	9606	9259	8933	8585	8268	7924	7605	7221	6948	6610	6456	6214
1.5	9743	9577	9242	8898	8572	8296	7895	7552	7264	6897	6708	6442	6246
2.0	9702	9532	9202	8873	8519	8219	7800	7537	7239	6916	6675	6348	6145
2.5	9672	9513	9180	8810	8510	8193	7810	7475	7119	6861	6679	6268	6126
3.0	9641	9477	9130	8807	8433	8140	7737	7472	7113	6888	6606	6355	6159
3.5	9604	9431	9101	8752	8430	8117	7743	7494	7078	6854	6468	6316	6090
4.0	9568	9402	9057	8753	8415	8091	7785	7416	7100	6722	6562	6268	6020
4.5	9540	9379	9040	8697	8323	8020	7648	7318	7050	6794	6495	6171	6079
5.0	9517	9332	9014	8665	8315	7978	7658	7341	7100	6796	6456	6306	6119
5.5	9483	9314	9006	8662	8296	7960	7681	7298	7020	6720	6377	6293	6064
6.0	9461	9286	8963	8622	8254	7956	7619	7241	7016	6679	6449	6268	6052

#### Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 10

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	9631	9485	9193	8908	8597	8331	8055	7788	7520	7210	6996	6760	6552
0.5	9616	9470	9194	8904	8614	8278	8017	7741	7550	7171	7003	6741	6498
1.0	9585	9446	9151	8862	8553	8311	8022	7756	7420	7180	6877	6803	6515
1.5	9562	9420	9131	8853	8559	8312	7990	7674	7439	7124	7004	6754	6570
2.0	9528	9385	9094	8814	8515	8265	7936	7702	7440	7173	6941	6652	6490
2.5	9501	9369	9074	8773	8517	8245	7938	7662	7288	7088	6921	6627	6463
3.0	9474	9334	9047	8762	8449	8203	7846	7601	7328	7072	6841	6690	6469
3.5	9446	9295	9030	8730	8459	8147	7824	7615	7297	7061	6763	6634	6437
4.0	9420	9276	8969	8718	8425	8156	7888	7594	7302	6979	6848	6564	6393
4.5	9387	9247	8966	8674	8369	8082	7762	7527	7271	7024	6745	6506	6323
5.0	9369	9218	8950	8641	8344	8044	7792	7510	7257	6991	6759	6567	6410
5.5	9343	9200	8927	8643	8334	8041	7791	7463	7223	6971	6668	6587	6406
6.0	9321	9173	8904	8614	8296	8040	7737	7391	7226	6952	6697	6534	6342

#### Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 12

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	9431	9304	9046	8799	8510	8300	8044	7811	7525	7321	7114	6949	6700
0.5	9417	9288	9043	8792	8536	8233	8041	7773	7614	7305	7119	6910	6713
1.0	9386	9268	9013	8755	8485	8287	8002	7787	7478	7280	7014	6947	6703
1.5	9368	9245	8995	8750	8491	8259	7987	7728	7534	7220	7130	6899	6762
2.0	9340	9219	8964	8717	8466	8221	7934	7731	7485	7291	7089	6829	6669

2.5	9319	9201	8936	8696	8450	8205	7930	7682	7395	7235	7028	6846	6678
3.0	9297	9169	8916	8667	8403	8185	7871	7673	7404	7191	6995	6820	6677
3.5	9266	9139	8903	8633	8404	8141	7839	7659	7374	7200	6943	6782	6618
4.0	9244	9118	8854	8631	8370	8119	7900	7616	7393	7107	6989	6727	6608
4.5	9218	9096	8843	8599	8335	8090	7789	7583	7362	7154	6898	6671	6592
5.0	9205	9074	8832	8565	8312	8034	7816	7562	7328	7148	6893	6743	6586
5.5	9180	9051	8814	8566	8298	8037	7814	7523	7297	7065	6806	6733	6594
6.0	9165	9033	8792	8536	8269	8023	7788	7480	7308	7061	6856	6710	6510

#### Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 14

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	9227	9114	8882	8668	8413	8211	8011	7777	7525	7324	7171	7027	6815
0.5	9214	9097	8877	8658	8439	8171	7986	7746	7627	7318	7175	6974	6855
1.0	9185	9079	8859	8627	8384	8205	7957	7756	7460	7301	7095	7043	6806
1.5	9171	9061	8844	8611	8395	8174	7939	7710	7536	7274	7171	6988	6844
2.0	9144	9038	8813	8592	8385	8160	7899	7704	7485	7318	7137	6905	6782
2.5	9126	9024	8791	8573	8348	8117	7887	7687	7413	7282	7085	6945	6778
3.0	9106	8991	8765	8542	8311	8126	7835	7657	7429	7251	7062	6916	6823
3.5	9079	8967	8753	8514	8301	8081	7813	7658	7391	7216	7021	6877	6744
4.0	9061	8952	8716	8508	8281	8058	7862	7622	7410	7166	7033	6837	6662
4.5	9037	8928	8699	8477	8250	8024	7764	7603	7392	7197	6970	6795	6738
5.0	9026	8912	8696	8460	8233	7969	7796	7562	7366	7140	6985	6841	6698
5.5	9002	8889	8675	8453	8214	7993	7800	7520	7335	7136	6917	6819	6708
6.0	8991	8879	8662	8433	8200	7976	7740	7492	7316	7082	6969	6817	6663

#### Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 16

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	9019	8917	8709	8519	8279	8107	7927	7736	7479	7317	7155	7038	6844
0.5	9007	8903	8707	8507	8304	8070	7898	7677	7570	7313	7174	7003	6865
1.0	8982	8888	8690	8475	8259	8093	7872	7687	7448	7304	7130	7072	6846
1.5	8970	8871	8670	8474	8270	8073	7862	7643	7483	7252	7168	7005	6882
2.0	8943	8848	8646	8442	8254	8057	7812	7648	7437	7287	7132	6957	6805
2.5	8926	8838	8627	8423	8234	8022	7806	7645	7397	7260	7108	6954	6815
3.0	8911	8806	8605	8402	8193	8017	7766	7610	7394	7239	7082	6944	6827
3.5	8885	8785	8591	8378	8195	7987	7753	7593	7381	7224	7038	6910	6780
4.0	8869	8770	8558	8373	8159	7968	7785	7561	7371	7164	7022	6854	6769
4.5	8853	8755	8547	8344	8141	7933	7689	7537	7376	7204	6978	6841	6801
5.0	8840	8735	8542	8330	8126	7894	7748	7527	7335	7163	7002	6891	6769
5.5	8817	8715	8521	8328	8110	7909	7742	7488	7308	7141	6952	6864	6760
6.0	8809	8700	8511	8302	8097	7892	7674	7459	7302	7090	6968	6841	6702

#### Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 18

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	8808	8716	8521	8360	8132	7982	7831	7636	7447	7268	7126	7014	6816
0.5	8798	8703	8526	8344	8165	7942	7796	7592	7489	7277	7129	6950	6849
1.0	8777	8691	8510	8317	8119	7968	7757	7617	7381	7238	7117	7039	6876
1.5	8765	8670	8487	8318	8127	7946	7753	7552	7414	7181	7145	7002	6871
2.0	8742	8657	8471	8284	8113	7932	7712	7588	7374	7251	7098	6940	6811
2.5	8726	8644	8454	8263	8108	7906	7703	7556	7342	7223	7076	6934	6834

3.0	8709	8616	8436	8254	8055	7885	7668	7523	7328	7194	7070	6946	6843
3.5	8688	8597	8420	8224	8054	7869	7651	7490	7333	7161	7014	6902	6778
4.0	8674	8583	8393	8219	8031	7837	7688	7491	7307	7117	7002	6877	6760
4.5	8658	8570	8387	8198	8014	7832	7599	7450	7322	7166	6971	6821	6798
5.0	8644	8547	8381	8176	8003	7792	7659	7450	7289	7118	6995	6887	6767
5.5	8627	8529	8358	8170	7982	7809	7649	7423	7244	7116	6929	6864	6774
6.0	8619	8524	8352	8161	7974	7788	7579	7401	7262	7077	6976	6827	6707

**Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 20**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	8597	8512	8337	8183	7967	7838	7702	7521	7348	7190	7053	6981	6793
0.5	8588	8501	8339	8167	8005	7815	7673	7480	7391	7198	7057	6880	6793
1.0	8568	8491	8324	8157	7965	7828	7641	7499	7281	7170	7048	6992	6828
1.5	8556	8472	8307	8150	7973	7815	7636	7445	7314	7125	7079	6957	6833
2.0	8535	8457	8293	8120	7956	7785	7599	7465	7285	7166	7045	6900	6774
2.5	8522	8446	8275	8099	7943	7778	7584	7458	7249	7141	7005	6885	6797
3.0	8506	8423	8258	8087	7916	7753	7550	7414	7239	7120	6997	6896	6807
3.5	8486	8405	8244	8060	7906	7740	7534	7391	7257	7084	6963	6876	6773
4.0	8474	8394	8217	8058	7886	7714	7574	7390	7220	7066	6930	6824	6721
4.5	8458	8377	8212	8036	7872	7712	7498	7361	7253	7096	6899	6787	6766
5.0	8447	8357	8208	8018	7860	7675	7539	7366	7211	7042	6940	6866	6735
5.5	8430	8345	8181	8009	7838	7683	7522	7333	7166	7052	6875	6831	6738
6.0	8424	8338	8182	8009	7833	7668	7480	7320	7169	7033	6912	6780	6696

**Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 22**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	8385	8307	8148	8005	7810	7690	7551	7392	7227	7096	6975	6892	6731
0.5	8377	8298	8146	7989	7848	7653	7530	7352	7277	7108	6967	6819	6725
1.0	8358	8286	8140	7974	7806	7679	7514	7366	7180	7091	6961	6890	6775
1.5	8347	8271	8117	7967	7807	7659	7503	7333	7218	7041	6986	6881	6770
2.0	8328	8256	8106	7942	7800	7648	7456	7339	7169	7075	6951	6833	6714
2.5	8316	8245	8088	7929	7780	7625	7452	7338	7159	7037	6926	6811	6730
3.0	8300	8224	8075	7915	7766	7610	7427	7295	7142	7009	6909	6818	6726
3.5	8282	8208	8062	7887	7746	7593	7423	7273	7167	6987	6906	6787	6716
4.0	8273	8199	8035	7895	7736	7581	7431	7278	7115	6987	6867	6773	6663
4.5	8255	8181	8030	7870	7721	7577	7380	7262	7138	6991	6836	6725	6681
5.0	8248	8168	8027	7849	7708	7537	7411	7253	7108	6964	6874	6805	6700
5.5	8231	8153	8004	7845	7686	7543	7391	7220	7075	6967	6806	6764	6670
6.0	8223	8147	8005	7846	7680	7521	7353	7210	7074	6957	6859	6740	6645

10. This dataset is the experiment results of multi-server bandwidth estimation algorithm for total slice of 500 (Figure 5.4). We present the real transmitted object size for both dynamic algorithm and fixed sampling algorithm (for fix sample size of 11, 14, 17, 20, 23, 26, 29, 32 and 35). The result of each setting is in one table; each table

contains results for 169 parameter settings of variances (13 variance for channel #1 and 13 variance for channel #2). Each value in the table is an average of 1000 runs.

#### Real Transmitted Object Size for Dynamic Algorithm

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	53219	52295	50724	49532	48384	47316	46372	45631	44585	43943	43422	42712	42573
0.5	53159	52267	50705	49498	48388	47492	46614	45399	44596	44123	43392	42857	42235
1.0	52933	52062	50535	49326	48212	47220	46156	45439	44600	43867	43184	42863	42335
1.5	52759	51933	50523	49221	48169	47115	46175	45161	44537	43803	42922	42984	42006
2.0	52639	51749	50345	49062	48028	47090	46050	45195	44240	43621	43153	42678	42194
2.5	52409	51623	50317	49077	47961	46818	46167	45214	44284	43410	42944	42728	42068
3.0	52266	51455	50168	48982	47782	46860	46005	44933	44243	43510	42940	42252	41840
3.5	52080	51360	50007	48767	47738	46820	45839	45013	44127	43333	42881	42318	41720
4.0	51973	51160	49839	48695	47545	46747	45800	44924	44041	43308	43008	42144	41914
4.5	51793	51032	49722	48538	47531	46639	45673	44869	44147	43464	42845	42262	41806
5.0	51653	50885	49621	48408	47539	46489	45545	44896	43864	43471	42533	42482	41952
5.5	51544	50726	49515	48301	47463	46556	45681	44608	43861	43325	42693	42155	41473
6.0	51408	50663	49569	48112	47279	46385	45251	44761	43810	43447	42735	41989	41661

#### Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 11

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	52824	52042	50460	48953	47359	45738	44367	42755	41322	40017	38453	37402	36487
0.5	52773	51992	50419	48963	47483	45828	44514	42692	41207	40160	38532	37238	36261
1.0	52593	51793	50191	48786	47247	45551	43901	42860	41228	39639	38462	37325	35729
1.5	52446	51662	50100	48612	47168	45478	43779	42535	41154	39471	38126	37299	36243
2.0	52334	51522	50015	48418	46894	45388	43627	42447	40780	39102	38489	37397	36106
2.5	52118	51342	49919	48350	46880	45039	43773	41983	40698	38995	37997	37282	35637
3.0	52002	51199	49699	48231	46667	45242	43506	42024	40391	38927	38037	36169	35641
3.5	51805	51083	49463	47877	46436	44742	43493	41897	40096	38865	37561	36149	35526
4.0	51712	50897	49373	47866	46156	44757	43273	41655	40309	38888	37299	36323	35592
4.5	51553	50780	49149	47720	46075	44614	43191	41914	40144	38595	37238	36536	35277
5.0	51404	50615	49005	47599	46061	44501	42924	41779	39716	38856	37068	36177	35186
5.5	51310	50454	48908	47459	45839	44366	42925	41295	39735	39055	37584	36279	34657
6.0	51178	50420	48947	47206	45793	44455	42742	41207	39876	39049	37388	36025	35000

#### Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 14

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	52608	51916	50541	49299	47848	46423	45200	43944	42588	41531	40010	39317	38509
0.5	52558	51868	50538	49230	47985	46627	45413	43803	42596	41598	40174	39206	38325
1.0	52417	51697	50343	49098	47736	46341	44960	43884	42384	41037	40246	39321	37979
1.5	52276	51577	50284	48942	47589	46169	44838	43700	42616	41050	39771	39124	37990
2.0	52186	51474	50128	48769	47427	46238	44783	43498	42085	40736	40257	39108	37892
2.5	52005	51324	50047	48720	47456	45891	44797	43139	42161	40685	39655	39034	37940
3.0	51888	51182	49929	48585	47266	45876	44477	43169	41919	40806	39926	38154	37764
3.5	51715	51092	49719	48350	47089	45658	44511	43085	41728	40423	39364	38508	37663
4.0	51640	50915	49625	48331	46846	45646	44314	42949	41664	40288	39168	38148	37573
4.5	51509	50833	49426	48188	46731	45523	44240	43118	41597	40408	39074	38350	37510

5.0	51368	50688	49349	48072	46726	45387	44001	42982	41227	40432	38772	38120	37283
5.5	51282	50550	49199	47919	46635	45354	43937	42472	41272	40732	39356	38066	36955
6.0	51173	50524	49223	47787	46378	45318	43873	42452	41310	40600	39306	37858	37224

**Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 17**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	52353	51735	50513	49400	48120	46862	45725	44661	43414	42499	41261	40464	39920
0.5	52306	51693	50497	49364	48177	47004	46001	44415	43392	42454	41351	40554	39859
1.0	52186	51542	50342	49211	47964	46735	45514	44569	43225	42201	41231	40441	39590
1.5	52058	51456	50280	49062	47881	46611	45421	44448	43232	42081	40947	40296	39325
2.0	51967	51325	50127	48923	47733	46715	45370	44135	42854	41755	41329	40271	39370
2.5	51823	51198	50079	48906	47766	46402	45361	44011	43053	41710	40743	40188	39187
3.0	51709	51088	50010	48781	47485	46335	45126	43970	42704	41812	40988	39634	39235
3.5	51565	51021	49825	48578	47457	46180	45064	43853	42661	41531	40632	39663	39205
4.0	51492	50867	49653	48497	47177	46175	44957	43815	42683	41455	40250	39517	39000
4.5	51370	50771	49579	48428	47148	46142	44923	43818	42551	41514	40400	39824	38792
5.0	51259	50627	49421	48266	47166	45890	44670	43734	42210	41423	39965	39402	38796
5.5	51186	50524	49344	48171	46986	45829	44766	43295	42135	41668	40521	39512	38380
6.0	51091	50498	49331	47977	46798	45853	44461	43299	42299	41529	40454	39142	38689

**Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 20**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	52082	51526	50421	49387	48231	47103	45996	45206	43894	43189	42194	41234	40706
0.5	52040	51476	50409	49376	48269	47227	46236	44889	43964	43068	42262	41471	40679
1.0	51931	51368	50235	49258	48117	47045	45852	45028	43848	42739	42074	41138	40437
1.5	51818	51264	50209	49079	48017	46856	45763	44754	43815	42647	41681	41243	40079
2.0	51732	51168	50043	48969	47933	46971	45700	44678	43432	42455	42002	41052	40333
2.5	51604	51039	49982	48914	47850	46672	45806	44532	43605	42350	41734	41106	40247
3.0	51501	50951	49955	48829	47662	46607	45523	44493	43282	42523	41764	40694	40142
3.5	51350	50854	49765	48630	47595	46470	45409	44323	43210	42162	41497	40478	40040
4.0	51283	50721	49636	48554	47393	46474	45446	44331	43195	42147	41222	40416	40035
4.5	51171	50662	49537	48524	47379	46378	45313	44413	43177	42286	41141	40688	39888
5.0	51087	50553	49448	48407	47401	46168	45132	44349	42942	42204	40854	40411	39733
5.5	51022	50451	49351	48259	47194	46217	45171	43920	42684	42340	41216	40573	39454
6.0	50961	50394	49368	48071	47085	46164	44942	43895	42934	42242	41339	40264	39641

**Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 23**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	51793	51284	50253	49331	48297	47199	46187	45485	44281	43554	42742	41975	41310
0.5	51762	51248	50230	49327	48288	47340	46409	45148	44281	43554	42805	41987	41324
1.0	51650	51145	50110	49155	48116	47129	46050	45279	44127	43209	42753	41783	41104
1.5	51555	51053	50076	49036	48031	47000	45940	45055	44285	43121	42222	42014	40789
2.0	51479	50943	49906	48933	47974	47094	45925	44949	43861	42983	42520	41714	41000
2.5	51351	50842	49869	48908	47915	46770	45969	44836	43958	42786	42287	41677	40962
3.0	51261	50743	49798	48774	47753	46768	45748	44750	43778	42997	42165	41205	40819
3.5	51133	50661	49676	48632	47691	46691	45681	44732	43640	42827	42021	41136	40664
4.0	51068	50548	49563	48570	47488	46529	45727	44634	43624	42586	41962	41023	40796
4.5	50959	50486	49454	48501	47464	46544	45571	44812	43600	42748	41817	41208	40443
5.0	50897	50412	49378	48391	47462	46354	45378	44671	43363	42783	41532	41219	40510

5.5	50830	50311	49277	48313	47308	46345	45537	44279	43178	42733	41775	41157	40160
6.0	50769	50240	49301	48152	47194	46292	45243	44293	43351	42740	41873	40753	40200

#### Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 26

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	51501	51025	50083	49262	48277	47233	46324	45649	44514	43801	43149	42366	41738
0.5	51474	50989	50031	49220	48236	47351	46496	45394	44529	43873	43161	42538	41784
1.0	51374	50896	49912	49076	48062	47174	46170	45386	44403	43566	42945	42261	41559
1.5	51285	50812	49916	48962	48038	47088	46131	45257	44540	43498	42638	42471	41252
2.0	51207	50713	49752	48825	47924	47132	46047	45201	44143	43312	42839	42128	41622
2.5	51090	50619	49723	48798	47843	46813	46068	45030	44199	43175	42638	42244	41388
3.0	50999	50534	49668	48703	47722	46825	45866	44939	44075	43245	42663	41850	41323
3.5	50885	50451	49525	48539	47733	46716	45823	44925	43878	43139	42377	41640	41139
4.0	50824	50344	49444	48524	47504	46651	45857	44840	43882	42982	42385	41415	41319
4.5	50728	50277	49318	48457	47468	46632	45678	44956	43904	43146	42357	41622	40983
5.0	50671	50212	49238	48375	47457	46464	45566	44843	43735	43080	41995	41719	41070
5.5	50618	50132	49176	48264	47339	46496	45679	44556	43491	43125	42208	41641	40843
6.0	50571	50075	49188	48119	47235	46379	45472	44479	43772	43079	42341	41365	40854

#### Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 29

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	51211	50751	49869	49129	48138	47165	46398	45682	44715	43980	43384	42727	42054
0.5	51180	50717	49844	49059	48132	47331	46541	45549	44614	44056	43386	42813	42208
1.0	51087	50642	49723	48901	47980	47158	46236	45481	44556	43774	43250	42685	42020
1.5	51000	50557	49729	48818	47972	47024	46163	45363	44702	43611	42889	42653	41629
2.0	50930	50463	49545	48724	47836	47105	46125	45259	44288	43625	43077	42476	41977
2.5	50822	50368	49537	48688	47773	46797	46146	45191	44415	43405	42935	42535	41727
3.0	50735	50290	49477	48574	47672	46832	45891	45048	44202	43540	42904	42075	41592
3.5	50630	50208	49337	48453	47691	46708	45872	45064	44038	43312	42644	42044	41449
4.0	50565	50121	49293	48447	47476	46666	45922	44933	44055	43238	42679	41840	41727
4.5	50479	50061	49171	48378	47438	46594	45777	45104	44073	43331	42693	41915	41380
5.0	50421	49994	49096	48283	47412	46463	45671	45005	43917	43494	42287	42028	41504
5.5	50380	49939	49014	48170	47317	46539	45722	44678	43779	43375	42540	41984	41331
6.0	50319	49875	49032	48052	47229	46373	45461	44606	43901	43180	42709	41740	41242

#### Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 32

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	50907	50475	49640	48934	48011	47073	46380	45688	44783	44062	43441	42875	42308
0.5	50880	50444	49632	48869	48036	47277	46442	45658	44714	44188	43501	43008	42347
1.0	50796	50383	49505	48766	47885	47110	46207	45525	44694	43936	43421	42828	42343
1.5	50708	50299	49490	48656	47863	46949	46167	45460	44759	43826	42997	42931	42021
2.0	50643	50207	49343	48559	47720	47037	46096	45274	44392	43732	43247	42591	42320
2.5	50545	50124	49343	48522	47659	46788	46132	45261	44483	43618	43100	42738	42054
3.0	50459	50044	49277	48405	47551	46789	45955	45057	44323	43639	43133	42309	41914
3.5	50363	49967	49151	48327	47597	46660	45881	45092	44160	43430	42797	42309	41777
4.0	50310	49874	49118	48316	47396	46662	45880	45008	44198	43370	42919	42109	41921
4.5	50229	49820	49003	48222	47344	46607	45833	45151	44163	43463	42891	42322	41700
5.0	50167	49751	48922	48147	47320	46467	45740	44982	44075	43542	42512	42333	41823
5.5	50118	49710	48820	48030	47249	46493	45752	44745	43927	43437	42754	42138	41617

6.0	50080	49654	48900	47957	47143	46359	45521	44769	44058	43442	42856	42071	41519
-----	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

**Real Transmitted Object Size for Fix Sampling Algorithm, Sample size 35**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	50601	50198	49392	48735	47855	46961	46266	45679	44864	44150	43657	43066	42550
0.5	50575	50157	49391	48662	47889	47181	46394	45615	44782	44294	43508	43103	42533
1.0	50497	50102	49268	48570	47743	46991	46197	45475	44738	44028	43452	42942	42498
1.5	50420	50026	49273	48425	47728	46865	46119	45454	44792	43869	43107	43041	42194
2.0	50348	49935	49108	48367	47612	46940	46074	45303	44408	43845	43375	42798	42471
2.5	50259	49857	49107	48355	47540	46729	46138	45254	44513	43696	43225	42875	42203
3.0	50184	49786	49072	48245	47434	46701	45864	45046	44373	43671	43189	42497	42198
3.5	50089	49721	48945	48160	47461	46589	45790	45108	44164	43549	43056	42499	42003
4.0	50041	49625	48913	48164	47280	46610	45834	45000	44266	43383	43050	42299	42073
4.5	49961	49581	48799	48042	47210	46526	45753	45161	44273	43624	43065	42564	41889
5.0	49905	49508	48721	47980	47202	46425	45730	45042	44146	43611	42655	42462	42037
5.5	49861	49473	48651	47893	47152	46429	45679	44806	44009	43539	42875	42313	41839
6.0	49825	49421	48685	47795	47035	46343	45466	44835	44069	43474	42913	42200	41689

11. This dataset is the experiment results of the number of runs that fail to finish in time limit in multi-server environment (Figure 5.5). Total time slice of 100. We present the number of overtime run for both dynamic algorithm and fixed sampling algorithm (for fix sample size of 6, 8, 10, 12, 14, 16, 18, 20 and 22). The result of each setting is in one table; each table contains results for 169 parameter settings of variances (13 variance values for channel #1 and 13 variance value for channel #2). The value plotted in Figure 5.5 is the average of the entire tables.

**Number of Overtime Runs for Dynamic Sampling Algorithm (out of 1000 runs)**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	34	31	36	38	44	50	53	52	59	56	69	65	60
0.5	29	27	38	38	39	40	41	52	67	65	56	61	57
1.0	29	31	41	46	39	56	53	53	56	55	56	58	66
1.5	36	34	33	37	36	46	43	63	56	55	62	65	52
2.0	34	33	40	40	39	52	42	57	54	60	66	58	74
2.5	40	33	36	36	42	49	48	54	56	64	64	58	61
3.0	30	27	37	35	42	47	47	52	63	59	65	47	51
3.5	28	39	38	42	43	46	45	49	60	52	58	64	62
4.0	32	30	39	39	46	46	51	50	62	63	73	58	61
4.5	39	33	36	35	43	47	46	50	55	59	70	56	61
5.0	26	40	41	36	47	42	51	54	57	60	65	58	67
5.5	41	31	36	31	41	37	50	47	52	59	52	70	70
6.0	30	29	32	31	37	51	49	50	60	61	62	56	60

**Number of Overtime Runs for Fix Sampling Algorithm, Sample size 6 (out of 1000 runs)**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	38	34	38	34	32	39	42	29	33	36	35	28	26
0.5	32	31	35	34	36	35	27	38	37	37	40	35	29
1.0	36	31	47	38	38	44	35	25	35	34	32	33	32
1.5	37	32	35	39	31	33	34	31	35	34	39	34	34
2.0	36	37	39	40	32	42	32	41	39	38	42	37	37
2.5	44	38	40	34	37	37	31	32	34	35	39	37	38
3.0	31	34	40	42	35	43	39	34	47	43	46	43	40
3.5	34	42	35	34	30	37	38	40	39	35	36	40	38
4.0	32	35	35	38	39	40	37	39	38	42	47	46	33
4.5	41	34	36	31	36	40	38	36	33	38	37	31	30
5.0	33	35	39	36	36	38	38	37	40	50	42	37	40
5.5	44	32	33	33	37	36	44	31	40	44	36	41	40
6.0	37	33	38	29	39	38	39	35	47	41	37	40	41

**Number of Overtime Runs for Fix Sampling Algorithm, Sample size 8 (out of 1000 runs)**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	41	36	40	40	34	43	41	37	35	41	42	33	31
0.5	36	32	36	32	37	39	34	37	43	40	40	42	32
1.0	38	37	39	42	39	48	44	35	39	40	36	37	36
1.5	38	30	39	42	37	34	37	39	44	39	49	36	31
2.0	35	39	44	40	35	40	42	51	40	41	46	40	42
2.5	48	39	34	35	37	37	40	43	37	36	45	36	41
3.0	39	32	39	35	38	42	37	38	46	44	43	47	39
3.5	34	46	39	41	33	40	36	39	38	38	40	42	43
4.0	39	37	42	44	49	43	38	39	41	43	49	51	39
4.5	43	36	42	35	38	40	43	39	32	39	35	39	41
5.0	28	45	47	37	43	36	46	41	43	40	40	36	40
5.5	44	31	31	35	37	37	43	34	40	43	39	46	41
6.0	34	27	35	42	36	39	40	40	41	42	43	40	48

**Number of Overtime Runs for Fix Sampling Algorithm, Sample size 10 (out of 1000 runs)**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	40	35	38	37	32	40	45	35	34	41	39	34	30
0.5	37	30	35	34	31	40	38	39	43	39	38	36	31
1.0	32	33	40	41	31	45	37	36	36	35	37	36	36
1.5	37	31	36	35	39	33	29	40	38	35	35	32	32
2.0	33	35	40	35	30	35	36	44	43	43	43	43	37
2.5	51	42	42	35	36	34	33	40	41	39	35	31	37
3.0	43	33	33	33	32	40	34	37	38	48	43	31	36
3.5	30	39	35	37	37	40	34	37	35	36	39	39	33
4.0	33	30	37	42	44	42	38	34	39	43	34	40	27
4.5	41	36	35	31	30	36	39	32	30	38	37	32	40
5.0	30	36	40	34	39	35	45	31	39	38	38	31	33
5.5	39	34	32	36	33	39	39	37	36	32	35	42	38
6.0	34	30	29	39	31	34	37	32	39	33	40	37	46



**Number of Overtime Runs for Fix Sampling Algorithm, Sample size 12 (out of 1000 runs)**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	43	41	36	40	44	48	53	36	40	43	43	38	36
0.5	37	32	39	40	36	42	41	39	45	40	41	37	34
1.0	41	36	45	47	37	46	44	35	44	44	42	45	37
1.5	43	37	42	32	33	41	32	45	34	41	46	35	38
2.0	37	35	46	43	36	45	37	48	46	39	46	41	46
2.5	46	47	48	35	43	47	40	43	45	40	41	39	36
3.0	36	43	37	37	34	42	37	38	42	47	44	35	38
3.5	40	37	35	40	41	44	43	35	39	34	35	50	36
4.0	38	39	42	39	48	41	43	44	41	40	39	46	30
4.5	41	41	42	32	39	36	42	42	36	41	45	35	46
5.0	39	39	41	40	42	36	45	39	39	36	41	42	34
5.5	37	38	39	34	38	42	46	43	39	37	42	43	42
6.0	30	39	32	35	32	40	35	33	36	34	44	37	45

**Number of Overtime Runs for Fix Sampling Algorithm, Sample size 14 (out of 1000 runs)**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	41	43	39	46	40	48	53	42	52	50	49	41	37
0.5	41	42	46	45	33	46	44	40	46	44	46	38	39
1.0	44	45	41	45	40	53	46	42	43	42	46	41	38
1.5	48	37	42	46	34	50	35	53	42	40	47	43	42
2.0	43	39	50	44	40	46	42	53	43	41	49	45	46
2.5	53	49	43	40	45	44	42	41	45	44	50	41	42
3.0	45	48	41	47	35	39	48	46	40	47	43	36	39
3.5	41	48	46	42	44	44	43	35	46	40	38	43	36
4.0	41	41	43	51	49	46	46	44	42	46	51	44	38
4.5	48	42	49	41	44	46	39	41	41	42	47	37	43
5.0	44	38	44	41	45	42	50	41	41	43	42	39	38
5.5	43	48	40	36	37	43	44	47	39	36	43	42	39
6.0	31	44	39	39	36	44	47	33	44	39	38	40	45

**Number of Overtime Runs for Fix Sampling Algorithm, Sample size 16 (out of 1000 runs)**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	47	46	41	52	42	56	54	46	49	52	54	47	42
0.5	47	38	49	54	35	47	48	48	53	50	50	41	41
1.0	44	48	49	50	47	59	52	45	54	51	47	48	49
1.5	44	38	53	47	42	57	42	54	47	50	53	50	45
2.0	46	44	53	49	40	42	48	56	44	50	53	48	56
2.5	55	47	45	50	45	48	43	49	42	47	46	46	45
3.0	47	51	48	48	38	48	49	48	50	51	55	41	46
3.5	48	49	50	47	44	51	44	41	48	36	44	46	41
4.0	46	48	48	51	56	47	49	49	46	49	56	46	34
4.5	46	45	53	43	45	51	45	48	42	49	53	44	43
5.0	47	44	52	42	51	42	48	42	50	42	49	45	47
5.5	48	47	42	39	39	43	42	48	45	45	43	44	47
6.0	43	49	44	45	39	50	45	41	48	41	48	44	51

**Number of Overtime Runs for Fix Sampling Algorithm, Sample size 18 (out of 1000 runs)**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	46	45	42	46	44	59	49	44	46	48	52	48	39
0.5	53	37	45	46	37	47	49	46	54	47	45	45	40
1.0	37	46	51	47	45	56	48	43	53	43	48	41	44
1.5	39	45	50	48	47	56	46	53	45	45	47	48	46
2.0	42	41	44	54	38	50	43	45	43	48	55	50	53
2.5	51	47	41	50	48	50	51	45	40	39	44	40	43
3.0	57	49	46	39	43	49	44	45	45	43	48	39	39
3.5	41	44	54	47	49	51	41	44	45	35	42	44	43
4.0	43	49	41	48	54	50	46	47	45	43	60	43	33
4.5	48	43	52	38	47	47	44	57	42	40	43	45	41
5.0	50	44	46	39	47	39	47	45	44	46	45	51	46
5.5	44	54	36	43	44	40	41	48	44	41	41	43	45
6.0	38	41	42	41	36	48	51	35	47	40	41	47	44

**Number of Overtime Runs for Fix Sampling Algorithm, Sample size 20 (out of 1000 runs)**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	50	51	41	51	57	62	52	50	53	51	58	49	44
0.5	52	42	49	54	46	49	48	47	60	49	48	53	42
1.0	40	42	57	42	42	55	51	53	60	43	49	41	55
1.5	49	48	50	59	49	56	44	52	51	50	57	45	49
2.0	53	54	47	56	43	54	44	57	43	51	52	45	57
2.5	54	54	47	53	54	54	46	47	49	44	46	49	44
3.0	57	52	52	48	48	54	48	45	50	48	55	42	48
3.5	48	46	55	46	53	50	52	42	57	40	49	45	45
4.0	50	49	45	57	61	54	52	52	52	48	65	53	45
4.5	52	49	58	50	49	48	46	54	44	46	49	49	48
5.0	53	50	53	50	58	38	54	47	47	56	51	51	54
5.5	46	55	46	48	54	39	50	53	49	46	47	47	53
6.0	42	47	50	44	46	53	51	41	49	43	51	55	56

**Number of Overtime Runs for Fix Sampling Algorithm, Sample size 22 (out of 1000 runs)**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	55	54	42	54	55	64	59	55	61	54	60	56	51
0.5	52	42	51	57	44	58	55	51	62	57	50	58	53
1.0	47	49	58	52	46	61	54	65	56	50	56	47	59
1.5	54	58	60	58	51	67	53	54	55	47	58	58	51
2.0	52	56	55	55	48	58	50	65	49	55	62	52	67
2.5	60	56	52	50	64	64	49	57	56	58	46	60	51
3.0	58	56	57	54	47	56	54	53	56	55	62	54	52
3.5	52	53	53	46	56	54	55	49	57	47	51	54	50
4.0	54	49	50	60	61	55	63	56	60	55	59	51	54
4.5	67	48	64	50	56	51	50	50	50	48	56	49	57
5.0	57	50	56	55	60	42	58	51	51	58	61	48	57
5.5	51	56	47	55	62	43	55	54	44	53	48	55	50
6.0	49	50	53	46	58	61	49	48	55	52	46	52	62

12. This dataset is the experiment results of the number of runs that use only one channel for real transmission (Figure 5.6). The number of total time slices is 100. We present the number for dynamic algorithm only. The table contains results for 169 parameter settings of variances (13 variance values for channel #1 and 13 variance values for channel #2).

**Number of Runs using only one channel for dynamic sampling algorithm (out of 1000 runs)**

	2.5	5	10	15	20	25	30	35	40	45	50	55	60
0.25	0	0	0	0	0	0	1	7	18	22	42	59	68
0.5	0	0	0	0	0	0	8	10	21	27	45	61	83
1.0	0	0	0	0	0	0	3	7	26	32	46	54	98
1.5	0	0	0	0	0	2	6	16	19	39	53	73	109
2.0	0	0	0	0	0	1	8	24	32	51	44	73	112
2.5	0	0	0	0	0	2	12	17	35	47	81	104	100
3.0	0	0	0	0	1	1	16	34	38	80	94	102	149
3.5	0	0	0	0	3	9	21	35	37	72	99	121	171
4.0	1	0	0	1	7	17	32	51	64	82	119	159	176
4.5	1	0	1	6	6	22	25	58	85	112	143	165	191
5.0	3	0	2	6	8	26	56	64	98	116	156	199	214
5.5	5	2	9	10	16	34	54	82	114	146	166	194	225
6.0	7	10	5	17	31	44	78	88	143	156	167	232	291

## Appendix 2

### t-distribution Table

The t-distribution values used in this thesis are those  $d_{(\alpha,n)}$  for  $\alpha^i \in \{0.75, 0.90, 0.95\}$ ,  $i=1 \dots 5$ , and  $n=1 \dots 500$ . We list here part of these values in the table below. Each row of the table represents one n value, while each column represents one  $\alpha$  value.

	0.75	0.866025404	0.9	0.948683298	0.95	0.974679434
1	1.000000001	2.233914076	3.077683551	6.149019487	6.313751452	12.544672930
2	0.816496584	1.519671374	1.885618083	2.875841223	2.919985587	4.273146763
3	0.764892332	1.356668770	1.637744345	2.324800286	2.353363438	3.165878914
4	0.740697084	1.285645447	1.533206273	2.108763997	2.131846780	2.764005866
5	0.726686843	1.246039302	1.475884044	1.994688367	2.015048359	2.560088784
6	0.717558200	1.220818394	1.439755748	1.924432651	1.943180291	2.437533764
7	0.711141777	1.203362504	1.414923929	1.876892522	1.894578617	2.355963644
8	0.706386613	1.190568844	1.396815316	1.842612098	1.859548036	2.297842083
9	0.702722149	1.180791791	1.383028738	1.816734532	1.833112934	2.254360914
10	0.699812063	1.173077978	1.372183638	1.796513028	1.812461112	2.220622124
11	0.697445328	1.166837197	1.363430324	1.780278771	1.795884827	2.193688741
12	0.695482868	1.161684596	1.356217335	1.766959787	1.782287564	2.171694487
13	0.693829305	1.157358557	1.350171294	1.755836443	1.770933392	2.153397165
14	0.692417071	1.153675083	1.345030381	1.746407652	1.761310128	2.137938305
15	0.691196951	1.150500966	1.340605610	1.738313958	1.753050345	2.124705864
16	0.690132254	1.147737441	1.336757164	1.731290738	1.745883678	2.113251761
17	0.689195073	1.145309710	1.333379396	1.725138957	1.739606727	2.103240415
18	0.688363807	1.143160090	1.330390945	1.719706005	1.734063611	2.094415645
19	0.687621461	1.141243405	1.327728212	1.714872895	1.729132817	2.086578496
20	0.686954497	1.139523752	1.325340709	1.710545551	1.724718243	2.079572110
21	0.686351990	1.137972227	1.323187875	1.706648571	1.720742905	2.073271060
22	0.685805030	1.136565348	1.321236743	1.703120803	1.717144367	2.067574114
23	0.685306278	1.135283784	1.319460245	1.699912176	1.713871537	2.062398335
24	0.684849627	1.134111511	1.317835937	1.696981280	1.710882084	2.057675412
25	0.684429967	1.133035123	1.316345076	1.694293576	1.708140760	2.053348417
26	0.684042973	1.132043311	1.314971864	1.691819984	1.705617923	2.049369598
27	0.683684979	1.131126500	1.313702913	1.689535915	1.703288453	2.045698604
28	0.683352843	1.130276488	1.312526781	1.687420416	1.701130928	2.042300999
29	0.683043862	1.129486226	1.311433653	1.685455465	1.699127024	2.039147378
30	0.682755694	1.128749637	1.310415030	1.683625556	1.697260886	2.036212385

31	0.682486309	1.128061430	1.309463551	1.681917254	1.695518778	2.033474064
32	0.682233920	1.127416990	1.308572793	1.680318802	1.693888754	2.030913276
33	0.681996980	1.126812276	1.307737129	1.678819954	1.692360314	2.028513304
34	0.681774103	1.126243715	1.306951593	1.677411667	1.690924261	2.026259480
35	0.681564078	1.125708170	1.306211809	1.676085987	1.689572458	2.024138835
36	0.681365823	1.125202840	1.305513887	1.674835861	1.688297722	2.022139914
37	0.681178377	1.124725235	1.304854383	1.673655002	1.687093615	2.020252543
38	0.681000878	1.124273147	1.304230210	1.672537799	1.685954467	2.018467616
39	0.680832559	1.123844570	1.303638588	1.671479258	1.684875123	2.016777010
40	0.680672718	1.123437728	1.303077058	1.670474862	1.683851017	2.015173482
41	0.680520735	1.123050996	1.302543364	1.669520582	1.682878006	2.013650443
42	0.680376045	1.122682937	1.302035491	1.668612729	1.681952365	2.012202004
43	0.680238137	1.122332213	1.301551610	1.667748019	1.681070705	2.010822762
44	0.680106538	1.121997641	1.301090058	1.666923449	1.680229977	2.009507967
45	0.679980830	1.121678122	1.300649333	1.666136272	1.679427386	2.008253103
46	0.679860628	1.121372660	1.300228050	1.665383996	1.678660419	2.007054258
47	0.679745572	1.121080358	1.299824950	1.664664387	1.677926723	2.005907707
48	0.679635344	1.120800379	1.299438882	1.663975324	1.677224203	2.004810097
49	0.679529645	1.120531955	1.299068786	1.663314924	1.676550903	2.003758401
50	0.679428200	1.120274392	1.298713696	1.662681438	1.675905031	2.002749778
51	0.679330761	1.120027038	1.298372715	1.662073226	1.675284957	2.001781622
52	0.679237087	1.119789301	1.298045018	1.661488833	1.674689155	2.000851557
53	0.679146973	1.119560632	1.297729847	1.660926873	1.674116240	1.999957394
54	0.679060214	1.119340520	1.297426492	1.660386100	1.673564909	1.999097061
55	0.678976631	1.119128492	1.297134301	1.659865305	1.673033973	1.998268696
56	0.678896048	1.118924114	1.296852669	1.659363415	1.672522305	1.997470530
57	0.678818306	1.118726982	1.296581049	1.658879430	1.672028890	1.996700978
58	0.678743265	1.118536707	1.296318893	1.658412395	1.671552774	1.995958504
59	0.678670779	1.118352945	1.296065724	1.657961449	1.671093027	1.995241689
60	0.678600720	1.118175371	1.295821095	1.657525753	1.670648868	1.994549250
61	0.678532972	1.118003670	1.295584568	1.657104574	1.670219493	1.993879979
62	0.678467422	1.117837557	1.295355766	1.656697178	1.669804170	1.993232686
63	0.678403961	1.117676764	1.295134297	1.656302903	1.669402223	1.992606348
64	0.678342493	1.117521035	1.294919825	1.655921130	1.669013035	1.991999941
65	0.678282930	1.117370150	1.294712012	1.655551267	1.668635977	1.991412533
66	0.678225176	1.117223868	1.294510567	1.655192766	1.668270514	1.990843238
67	0.678169157	1.117081993	1.294315199	1.654845127	1.667916123	1.990291243
68	0.678114791	1.116944326	1.294125628	1.654507848	1.667572279	1.989755780
69	0.678062008	1.116810681	1.293941610	1.654180465	1.667238551	1.989236102
70	0.678010742	1.116680884	1.293762901	1.653862570	1.666914485	1.988731511
71	0.677960925	1.116554771	1.293589268	1.653553743	1.666599652	1.988241379
72	0.677912499	1.116432191	1.293420512	1.653253601	1.666293701	1.987765078
73	0.677865406	1.116312994	1.293256413	1.652961783	1.665996225	1.987302070
74	0.677819591	1.116197044	1.293096796	1.652677963	1.665706886	1.986851729
75	0.677775004	1.116084214	1.292941475	1.652401802	1.665425370	1.986413635
76	0.677731592	1.115974365	1.292790269	1.652132999	1.665151352	1.985987229
77	0.677689313	1.115867397	1.292643033	1.651871251	1.664884540	1.985572069
78	0.677648123	1.115763187	1.292499597	1.651616301	1.664624643	1.985167712
79	0.677607983	1.115661638	1.292359829	1.651367883	1.664371416	1.984773763
80	0.677568847	1.115562650	1.292223588	1.651125751	1.664124581	1.984389778
81	0.677530683	1.115466113	1.292090735	1.650889660	1.663883915	1.984015433
82	0.677493453	1.115371950	1.291961141	1.650659391	1.663649187	1.983650355
83	0.677457120	1.115280075	1.291834703	1.650434732	1.663420186	1.983294217
84	0.677421660	1.115190401	1.291711302	1.650215476	1.663196681	1.982946647
85	0.677387038	1.115102851	1.291590824	1.650001435	1.662978507	1.982607367
86	0.677353219	1.115017349	1.291473175	1.649792434	1.662765443	1.982276095
87	0.677320187	1.114933821	1.291358250	1.649588288	1.662557351	1.981952554
88	0.677287905	1.114852214	1.291245944	1.649388834	1.662354032	1.981636463

89	0.677256350	1.114772446	1.291136199	1.649193896	1.662155334	1.981327566
90	0.677225500	1.114694460	1.291028899	1.649003336	1.661961083	1.981025632
91	0.677195331	1.114618202	1.290923980	1.648817025	1.661771163	1.980730402
92	0.677165820	1.114543609	1.290821354	1.648634789	1.661585399	1.980441692
93	0.677136943	1.114470634	1.290720957	1.648456510	1.661403675	1.980159245
94	0.677108686	1.114399219	1.290622714	1.648282073	1.661225851	1.979882922
95	0.677081023	1.114329317	1.290526544	1.648111327	1.661051821	1.979612466
96	0.677053942	1.114260877	1.290432399	1.647944190	1.660881448	1.979347714
97	0.677027419	1.114193862	1.290340204	1.647780510	1.660714613	1.979088481
98	0.677001441	1.114128216	1.290249906	1.647620229	1.660551224	1.978834625
99	0.676975987	1.114063908	1.290161442	1.647463196	1.660391153	1.978585939
100	0.676951045	1.114000892	1.290074759	1.647309342	1.660234330	1.978342282
200	0.675718411	1.110890554	1.285798795	1.639729226	1.652508113	1.966354577
300	0.675308416	1.109857659	1.284379873	1.637218054	1.649948675	1.962390216
400	0.675103586	1.109341945	1.283671602	1.635965356	1.648671948	1.960413903
500	0.674980739	1.109032752	1.283247024	1.635214651	1.647906855	1.959229982