Real-time Remote Visualization of Scientific Data

Mukta Nandwani

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

Dr. Mark T. Jones, Chair
Dr. Peter M. Athanas
Dr. Binoy Ravindran

May 20, 2002
Blacksburg, Virginia

Keywords: unstructured mesh, visualization, rendering, network protocol, compression, rate control

Real-time Remote Visualization of Scientific Data

Mukta Nandwani

Abstract

Visualization of large amounts of simulation data is important for the understanding of most physical phenomena. The limited capabilities of desktop machines make them unsuitable for handling excessive amounts of simulation data. The present day high speed networks have made it possible to remotely visualize the data being generated by a supercomputer in real time. In order for such a system to be reliable, a robust communication protocol and an efficient compression mechanism are needed. This work presents a remote visualization system that addresses these issues, and emphasizes the design and implementation of the application level network protocol. A control theory based adaptive rate control algorithm is presented for UDP streams that maximizes the effective throughout experienced by the stream while minimizing the packet loss. The algorithm is shown to make the system responsive to changing network conditions. This makes the system deployable over any network, including the Internet.

To my Grandparents

who instilled in me, the desire to learn.

Acknowledgments

Of all the chapters of my thesis, I find that this section is the hardest to write. As I proceed, I hope to do justice in acknowledging the support I have received from all the different quarters.

I met Dr. Mark Jones during my first semester at Virginia Tech. Not only did he help me define my research goals, he also guided me all through the way. His knowledge, insightfulness and patience have made my graduate study a most rewarding experience. Words elude me as I try to express my gratitude to him. I will always cherish my association with him.

I would like to thank Dr. Peter Athanas for his encouragement and support that I often needed during difficult times. I am especially grateful for the helpful hints he gave me during my struggle with circuit synthesis.

I have known Dr. Binoy Ravindran since my first course at Virginia Tech. He introduced me to socket programming that is the basis for the implementation of all my research work. I am highly grateful to him for his teachings.

During my two years of graduate study, I took a number of courses from Dr. Scott Midkiff. The knowledge I gained from these courses proved extremely useful in my networking studies that lie at the core of my research work. I am highly grateful to him for helping me gain a

better understanding of some of the most elusive concepts.

I cannot undermine the importance of the brainstorming sessions I had with Dr. Jae H. Park. I am thankful to him for helping me solve many difficult problems.

I am thankful to my project mates, Dr. Lori Freitag, Dr. Paul Plassmann, Dr. Raymond Loy, and Yanhua Yi without whom this work would not have been possible.

Most of my years as a graduate student were spent at the Configurable Computing Lab. I gratefully acknowledge the ideas and assistance I received from the lab members that helped me, among other things, in debugging my code. I wish them the very best for their future endeavors.

Last, though not the least, I would like to express my gratitude to my family and friends for standing by me through these years. Their encouragement, belief in my ability, and tolerance to my mood swings enabled me to go through the maze of graduate research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data analysis is an inherent and important component of any scientific study. One of the most basic and insightful forms of observing natural phenomenon is through visualization. However, accurate perception of visual data requires that the data be available in large amounts. Even though high processor speeds and cheap memory have made desktop computers quite powerful, their ability to handle huge amounts of simulation data is highly limited. It is therefore natural to harness the power of supercomputers and multi-processor clusters for generation of accurate simulation data. Ideally, viewing such data should be possible through the click of a button on a desktop machine. Because it is not possible to handle the data locally, a natural solution is to be able to connect to a remote system and obtain data from it.

This thesis presents a remote visualization system for viewing large scientific data sets in real time. It addresses the challenges involved in building such a system, from carefully designing the methods of data storage to compression, transfer, and visualization. In addition, the thesis proposes an adaptive rate control scheme for maximizing the effective throughput of

UDP streams over wide area networks.

Because the amount of data is very large, it is not possible to stream the entire data over a limited capacity network. It is therefore desirable that the data be compressed before it is sent. The proposed system uses polynomial compression based on spatial distribution of the data. The compressed 3-dimensional data is stored in an octree structure. Efficient storage of compressed data allows easy retrieval in real time.

The degree of compression can be varied as the simulation proceeds in order to take advantage of increased or reduced network capacity. Thus, the amount of data and accuracy of the image depends on the capability of the local machine and its connection to the remote host. Such a feature is highly desirable when viewing takes place over wide area networks with variable capacities.

The octree structure is replicated at the viewing end. The data in the leaves of the octree is decompressed to obtain a 3-dimensional image. As the simulation proceeds, octree updates are obtained from the remote host, and used for re-rendering the image.

Though this thesis addresses all the above-mentioned issues, the emphasis lies on an application level protocol design for data transfer. The proposed system uses Transmission Control Protocol (TCP) for exchange of control information and User Datagram Protocol (UDP) for sending the data in real time. In addition, a general and efficient rate control algorithm is presented that takes advantage of available bandwidth between the sender and receiver. The algorithm is then closely tied to the system in order to be able to change the compression degree based on available resources. The system not only gives the user a real-time moving picture of the simulation data, but also allows data exploration. The system may potentially be used to view any simulation data like that produced during seismic analysis, structural analysis, study of acoustics, computational fluid dynamics (CFD) or electro-magnetics.

This thesis proposes an adaptive control theory based rate control scheme for maximization of effective throughput experienced by UDP streams over wide area networks. As opposed to the conventional back-off algorithms for congestion control, the proposed theory assumes that the application is tolerant to some packet loss, and ensures a high effective bandwidth even in the face of congestion.

1.1 Thesis Organization

The following chapters specify details of the system structure, application level network protocol used, and display the advantage obtained by the proposed rate control technique.

Chapter 2 provides a background for the thesis and reviews current trends in the field. It also provides a review of the basic technologies used in the proposed system.

Chapter 3 presents a general system overview. It outlines the system structure and provides insights into the organization of each of its components.

Chapter 4 lists the issues encountered while developing a robust network protocol for the system. It also describes how the system addresses these issues. An efficient rate control algorithm used for maximizing the system utilization is described in this chapter.

Chapter 5 presents results from experimental system runs over actual networks. The system is found to be robust and adaptive to wide area networks. An analysis of the results brings forth the strengths of the system.

Chapter 6 summarizes the thesis and presents possible directions for future work.

The thesis concludes with Appendices A and B that list some finer details in reference to system results.

Chapter 2

Background and Review

Remote visualization of large scientific data sets is a challenging problem involving data compression, transfer, decompression, and rendering. The challenge escalates when data changes in time, and the user wishes to interact with it.

A great number of research efforts are targeting the various facets of this problem. This chapter highlights the work done on this problem and forms a background for the application presented in this thesis.

2.1 The Unstructured Meshes

In a structured mesh, "the local geometric and combinatorial structure at each node is identical, with the possible exception of the mesh's boundary nodes" [1]. On the other hand, in an unstructured mesh, "the local geometric and combinatorial structures may differ from one node to another" [1]. Thus, visual data with no regular format in terms of pattern of

connections among points may be viewed as an unstructured mesh. The geometric positions of the points may still be symmetric in an unstructured mesh. Figure 2.1 is an example of an unstructured mesh.

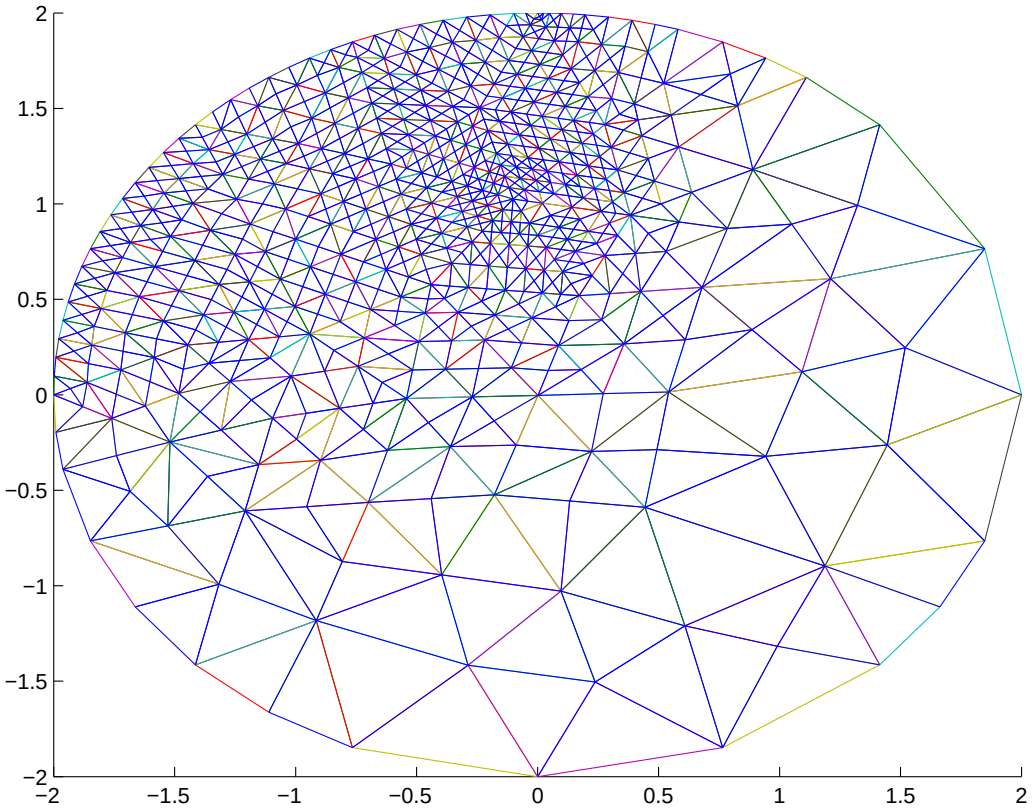

Figure 2.1: Unstructured Mesh

Most data corresponding to natural phenomena is found to be irregular. Therefore, unstructured meshes are found to be highly useful for numerical modeling of such phenomenon. In addition, the amount of data generated can be on the order of tera- and petabytes. Examples include data produced from simulation of thermonuclear flashes [2], arterial flow simulation [3] and weather prediction research [4].

The following sections give an overview of present day techniques used for handling such data.

2.2 The Need for Parallel Computations

The limited memory and processing capabilities of desktop workstations make them incapable of handling huge amounts of scientific data. Large machines with multiple processors and high computational power are better suited to the task. Computational efficiency can be increased by taking advantage of parallelism in the data. Today, a number of researchers employ this method of computation.

An estimate of the problem sizes may be obtained by considering the requirements of the identified 'Grand Challenge' problems [5]. The first global ocean simulation, in the early '90s, produced over 250 gigabytes of processed output using 3000 processor hours on a gigaflop supercomputer [5]. Biological problems also require high performance computing. In order to simulate the fluid dynamics of heart beats, calculations are performed on a 128 x 128 x 128 lattice used to represent blood. For calculations of this order, a CRAY C90 cpu-week with 50 Megawords of memory is required per beat [6]. Clearly, such applications cannot be deployed on single-processor machines.

Graffunder [7] lists the strengths of CRAY C916 used for nine large industrial problems. The applications deployed originate from a variety of scientific fields including computational fluid dynamics, computational chemistry, seismic analysis, structural analysis, acoustics, computational fluid dynamics (CFD) and electro-magnetics.

Wells et al [8] present the use of Intel iPSC/860 hypercube [9] for a computationally intensive numerical problem of solving the time-dependent Dirac Equation on a three-dimensional

lattice equation.

Elmroth et al [10] describe the advantages of using a parallel machine for simulating groundwater flow related problems. They present a parallel version of the TOUGH2 software package [11]. The time taken to solve a real problem is shown to reduce from 7984 to 126 seconds with the use of the parallel version of the software on a Cray T3E with 128 processors.

2.3 Visualization Software

The user interacts with the system through the visualization software. Therefore, this is one of the most important components of a remote visualization application. The demands on visualization software are diverse. Most importantly, it should be able to accept data in various formats, display it in a user-friendly manner, and allow easy interaction. This section lists some of the commonly used visualization tools.

The Visualization Toolkit (VTK) [12] is a freely available API software coded in C++. It is a powerful tool that enables 3D visualization. It also provides functions to find isosurfaces and perform complex tasks such as Delaunay triangulations and polygon reduction. The capability for handling user interaction is also present. Interface layers for Java, Python, and Tcl/Tk are provided with the software; it can also be used on various platforms.

OpenGL [13] is another widely used API-based software package for rendering 2D and 3D images. It is also a freely available software package portable to various platforms; it provides a number of useful visualization functions. Images thus rendered can be made interactive. A huge advantage of using OpenGL as opposed to other imaging software is that it is well documented.

Application Visualization System (AVS/5) [14] facilitates the development of 2D/3D interactive applications through its API. It is useful for applications being developed for Unix/Linux. In addition to visualization modules, it can also transparently handle data communication. Thus, it is easily deployable in a distributed environment.

The Scientific Computing and Imaging Institute have developed SCIRun [15], a software for integrated computational steering and visualization. It accepts user inputs to interactively change computational parameters while the results are being visualized. It is a free software package deployable on a variety of platforms as well as in a distributed environment. It has been integrated with Genesis and Matlab for analysis of networks of neurons along with simultaneous simulation and visualization [16].

2.4 Real-time Applications

Common present day real-time applications include video conferencing, streaming audio and video. In streaming media applications, the audio and video data is typically compressed, and sent to the receiver as packets. The data is then reconstructed and reproduced at the receiver. For applications where audio and video both need to be transmitted, synchronization is done among the two streams. Extensive amount of work has been done towards developing useful real-time applications. RealPlayer [17] and Windows Media Player [18] are popular streaming media softwares.

In general, real-time applications have strict delay constraints and are moderately tolerant to losses. The delay constraints make User Datagram Protocol (UDP) [19] the transport protocol of choice as opposed to the reliable Transmission Control Protocol (TCP) [20] that introduces retransmissions, and higher overheads.

UDP is a transport layer protocol that uses Internet Protocol (IP) for transfer of packets between sender and receiver applications identified by port numbers. UDP neither guarantees delivery, nor uses any rate control mechanisms. Owing to its simple functionality, the header overhead and packet delivery delay is low. An optional checksum may, however, be used to identify and discard a packet corrupted on its way to the destination. Because it is an unreliable protocol, there may be packet loss, duplication or re-ordering. This makes UDP suitable for applications that need fast delivery, but can handle some packet loss.

TCP is another common transport layer protocol that sits on top of IP but provides an end-to-end reliable service. In addition to ensuring delivery through the use of acknowledgements and re-transmissions, it takes care of packet duplication and re-ordering. It also performs functions such as flow control and congestion control. The price of reliable delivery is the high overhead and introduction of delays.

Real-time applications typically use Real-time Transport Protocol (RTP) [21] over UDP. RTP is specially designed to support real-time applications. It enables synchronization between sender and receiver, and helps in rate control. It is a very useful protocol for sending video or audio over the network, because encoding can be changed depending on bandwidth availability. RTP also helps in handling jitter, and problems with delayed playback, packet loss, re-sequencing, and duplication common in a UDP application deployed over the Internet. RTP Control Protocol (RTCP) [21] works along with RTP to provide control functions like exchange of session data and statistics between source and receiver. It specifically helps in rate control.

Though the proposed system transfers data in real time, the requirements of such a system are significantly different from conventional applications such as streaming audio.

Most compression algorithms require that packets be delivered in order. If this is not ensured by the underlying protocol, the application has to sort the packets. Typically packets are

marked with sequence numbers to enable re-sequencing. The system that this thesis presents uses packets with no inter-dependencies. Therefore, strict ordering of packets is not required for data reconstruction.

RTP also provides for a timestamp field in the packet header. The system developed packs more than one data element in a packet, and each element has a 'life span' associated with it. Therefore, a common timestamp header is not sufficient for determining the playback time.

The system does not endeavor to exactly match receiver with sender side simulation rate. Thus, RTP's rate determination and control is not needed in this case. In addition, RTP facilitates multi-casting at the expense of high complexity. As of now, the system is not intended to have many-to-many communication making the increased complexity unwarranted.

RTP also provides for stream mixing. This is a very important requirement for systems that transfer more than one stream of data and need to synchronize the streams at the receiver. Because only a single stream of data needs to be transmitted for the purpose of data visualization, this feature is not needed.

Because most features of RTP do not conform to the above-mentioned requirements for transferring scientific data in real-time, RTP was not chosen for the purpose. Chapter 4 elaborates on network and protocol issues, and gives insight into how the system handles real-time requirements.

The developed system is meant to work not only for LANs, but also over the Internet. The problem of packet loss induced by the use of UDP becomes challenging when the application is deployed over a WAN. The various causes of packet loss are:

- Packet discard due to checksum failure that occurs if there is a transmission error. Such errors are rare in present day networks.
- Packet drop due to congestion at intermediate routers.
- Buffer overflow at the receiver. Here two buffers, the receiver buffer and the playback buffer, are dealt with. The receiver buffer overflows if the sender sends so fast that the receiver runs out of buffer space. The playback buffer overflows if the receiver's playback rate is slower than the sender's sending rate.

An overflow of the playback buffer can easily be determined, and control signals may be sent to the source asking it to slow down. Congestion avoidance techniques such as Random Early Detection (RED) [22] have been deployed to sense congestion and trigger TCP congestion control mechanisms.

Solutions have been proposed to handle the more difficult problem of receiver buffer overflow. One of the suggestions was made by Kapadia et al [23]. They present a packet-spaced protocol as a means of making UDP adaptive to the network. They found that if inter-packing spacing is increased, resulting in a reduction in the sender's transmission rate, packet loss decreases and receiver's throughput is increased. This technique works for a slight increase in packet spacing. The paper also presents an adaptive algorithm to determine when to increase the packet spacing. The sender and receiver keep an account of the number of packets sent and received in one round-trip time (RTT). If packet loss is detected, the sender doubles the inter-packet spacing. If there is no loss in the network, the sender reduces inter-packet spacing, thus increasing the sending rate. An upper limit is imposed on the maximum inter-packet spacing.

Balakrishnan et al [24] propose the use of a congestion manager (CM) to make applications adaptive to congestion. They present CM algorithms, a protocol, and an API for use at the

sender and receiver ends. The CM is a good technique when multiple flows co-exist and it is important to grant them resources in a fair manner in the face of congestion. The CM uses probes and sends feedback to monitor the state of the network. The CM lies above IP and introduces its own set of headers.

This thesis presents a rate control algorithm that makes the application responsive to network congestion. An overflow of the playback buffer is also interpreted by the system as packet loss. Chapter 4 elaborates this adaptive rate control scheme.

2.5 Current Work

A number of techniques are being developed for the purpose of interactive real-time visualization of large scientific data sets. These schemes are based on one of three basic approaches [25]:

Image-based rendering uses reference images from various view-points to produce the actual image. As the user interacts with the image, new images are constructed using the available reference images.

Chang et al [26] use image-based rendering for 3D image warping [27]. They present a Layered Depth Image (LDI) [28] tree which is used to adaptively select the sampling rate of the image determined by the level of the tree.

Image-based rendering schemes induce errors when trying to re-construct a viewpoint for which no reference image is available. Depending on how many reference images are available and how closely the desired viewpoint relates to them, the amount of error may vary. An important advantage of these schemes lies in the fact that the amount of data transmitted depends on the complexity of the reference images as against the complexity of the scene.

Parallel visualization server techniques attempt to visualize the entire data set. They rely on remote computational resources, a sophisticated graphics work station, and a high bandwidth network. The computation involves extraction of relevant geometric data, such as iso-surfaces, from the image which is sent over the network to the workstation.

Haimes [29] presents a system that uses this technique for visualization of large data sets. The proposed system sends data as it performs the computations. This alleviates the burden on the system components that can now work with small amounts of data.

Crockett [30] evaluates the feasibility of distributed software architecture for parallel visualization. He points out the problems of scalability and image handling in such a scheme.

These techniques do not encounter errors due to change in viewpoints or other approximations, and enable easy manipulation of data. The cost in terms of data size, however, depends largely on the image complexity and size. Large data sets require higher bandwidths and computational power.

Sub-sampling and clustering techniques are data reduction methods that sample the data set or cluster points to form smaller representative data sets. The data may be sampled uniformly or randomly. How well the reduced data set represents the original data depends on the sampling technique, complexity of the image and sampling rate. Some researchers have also employed hierarchical methods for sampling and representation of data. Progressive Meshes [31], clustering methods [32] and wavelets [33] are a few such techniques.

Lindstrom et al [34] present a multi-resolution method for real-time continuous rendering of polygonal surface data typical in digital terrain models. The algorithm uses different levels of detail to control image quality. The data is assumed to be a uniformly distributed set of points with varying heights. Mesh rendering uses IRIS GL and OpenGL [13]. Their technique is shown to introduce less than 5% of error with polygonal data reduction by a

factor of 300.

Another multi-resolution technique is presented by LaMar et al [35] for visualization of texture based images. They represent texture data as an octree and use selection filters to display a higher resolution in the region of interest, with uniform degradation in image resolution and quality as the distance from the region of interest increases. The leaf nodes of the tree store the original image resolution which is coarsened as one moves up the tree. Since there is no data reduction, the memory requirement is very high, though visualization is made easy.

The advantage of these techniques is that they allow a trade-off between quality and resources. In addition, they allow easy and fast data exploration. On the down side, depending on the reduction factor, the reduced data set may not closely reflect the original data. This may prove to be an important drawback when data needs to be interpolated for rendering a specific view-point. Also, the available resources may determine the number of sample points leading to reduction in the ratio of sample points to original data as the data set size increases.

The efficiency of these methods may be defined in terms of resource requirements with respect to problem size. Each of the above techniques is found to be efficient under different circumstances [25], but none of them consider the importance of network protocol for real-time transfer of data. Though the assumption of a fast underlying network is important, the networking issues play an important role in making the system useful and efficient.

Supercomputing centers around the world have recognized the need to deploy fast networks to facilitate data transfer during high performance computing. This has led to the deployment of fast gigabit testbeds that alleviate the data transfer bottleneck.

Very High Speed Backbone Network (vBNS), developed by MCI and NSF in 1995, connects

five supercomputing centers across the continent and uses ATM and SONET [36]. It is used for a variety of high speed, large-scale supercomputing applications such as simulated merger of the Milky Way and Andromeda galaxies, interaction between biological cells, virtual reality and weather prediction applications.

NSF, DARPA and CNRI initiated a gigabit testbed program in early 90's which led to the development of five testbeds used for a variety of applications. CASA is used for distributed supercomputing applications; MAGIC for high-speed access for terrain visualization; BLANCA for remote visualization and mutimedia digital libraries; AURORA for distributed virtual memory, NECTAR for coupling of supercomputers for studying chemical reaction dynamics; and VISTANET for radiation treatment planning applications [37].

Cheng et al [38] present a remote visualization system that carries out electromagnetic scattering simulation on a supercomputer and uses a high speed network for sending data for visualization on a graphical workstation. No special network protocol was developed for the purpose of communication. Instead, AVS was used for both visualization and communication over a 10 Mbps Ethernet based local network. AVS performs message passing over TCP/IP protocol. One of the factors limiting performance is identified to be the high latency of data transfer. Interestingly, this problem would be more prominent over wide area networks, bringing out the need to carefully design a network protocol.

All the above-mentioned techniques rely on a high capacity network to provide for bandwidth requirements. The performance of these systems becomes limited if they are deployed over wide area networks where there may be no bandwidth guarantee. This thesis addresses the problems involved with deploying visualization systems over such networks and identifies the need for a robust application level network protocol.

Chapter 3

System Overview

This chapter provides a general overview of the visualization system presented in this thesis, and its basic components. The system supports the following features:

- a variable rate of compression,
- maximum network utilization, and
- stable system response.

A variable rate of compression allows the system to adapt to available resources. The system uses polynomial approximation as the compression technique. The compression error bound defines the amount of acceptable error and is used to tune the compression rate. The system tries to use whatever network capacity is available over the path connecting the server and the client machines. This is highly desirable as it allows the system to adapt its rate of compression so as to send higher or lower resolution images depending on the network capacity. Despite the changes in the compression and data rates, the system response remains

stable in terms of the image quality and the display rate. Such a feature is necessary for a visualization application since it directly affects user perception.

The simplest view of the system is that of a server and a client connected through a communication channel. Figure 3.1 shows client-server components as well as the communication channels between them. The communication among components is indicated by the direction of the connecting links. The two main modules - client and server, communicate over a network through their respective communication entities.

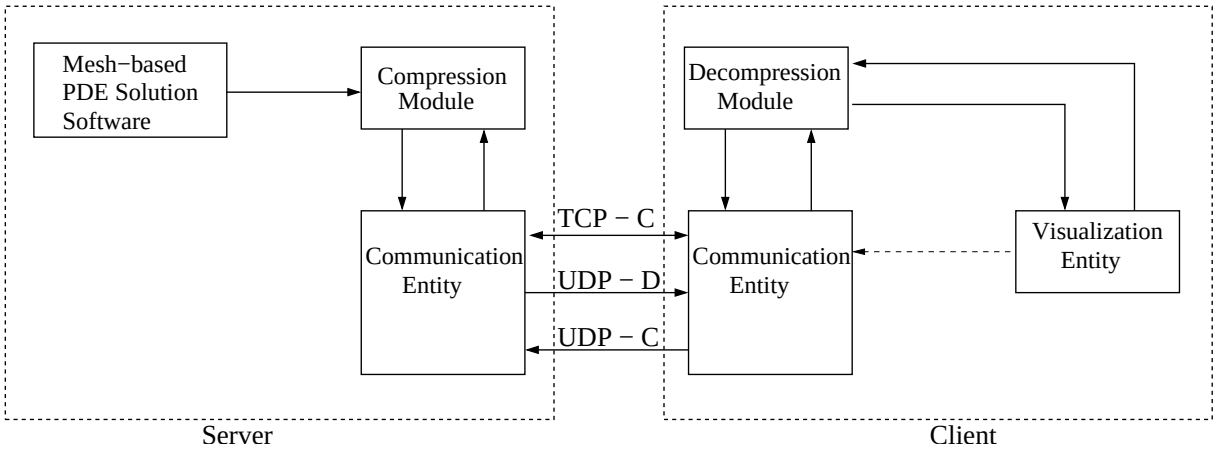

Figure 3.1: System structure including client and server components with associated communication channels

The following sections describe the functions of each component and outline how it is implemented in the system.

3.1 Server

The server is required to produce huge amounts of raw data, compress it, and send it to the client machine. Therefore, it needs multiple processors with high processing power

and memory. It is composed of a mesh-based Partial Differential Equation (PDE) solution software , a compression module and a communication block. The PDE solution software solves partial differential equations related to various physical phenomena. It uses mesh-based solutions such as adaptive refinement to produce unstructured mesh data that is sent to the compression module. Once the data is compressed, the compression module signals the communication block. The communication block asks for more data from the compression module as and when needed. Figure 3.2 shows the data flow between the server components and shared data structures. The communication entity is shown to communicate with the client through TCP-Control (TCP-C), UDP-Control (UDP-C) and UDP-Data (UDP-D) channels. Each of the server components is discussed in the following sub-sections.

3.1.1 Mesh-based PDE Solution Software

Most solution packages for solving PDEs rely on mesh-based discretizations of the solution domain. Such meshes may have billions of nodes and solution results may be generated on each node at thousands to millions of time steps. Because of the large amount of data and processing involved, these packages typically run on large parallel computing systems.

3.1.2 Compression Module

The compression module accepts mesh data from the PDE solution software and performs sampling and compression on the data, finally storing it for use by the communication block.

The compression module may be implemented as a separate process or thread from the PDE solver. The present system may be interfaced to accept mesh input from a separate process. The mesh points are distributed into an octree based on their spatial organization such that

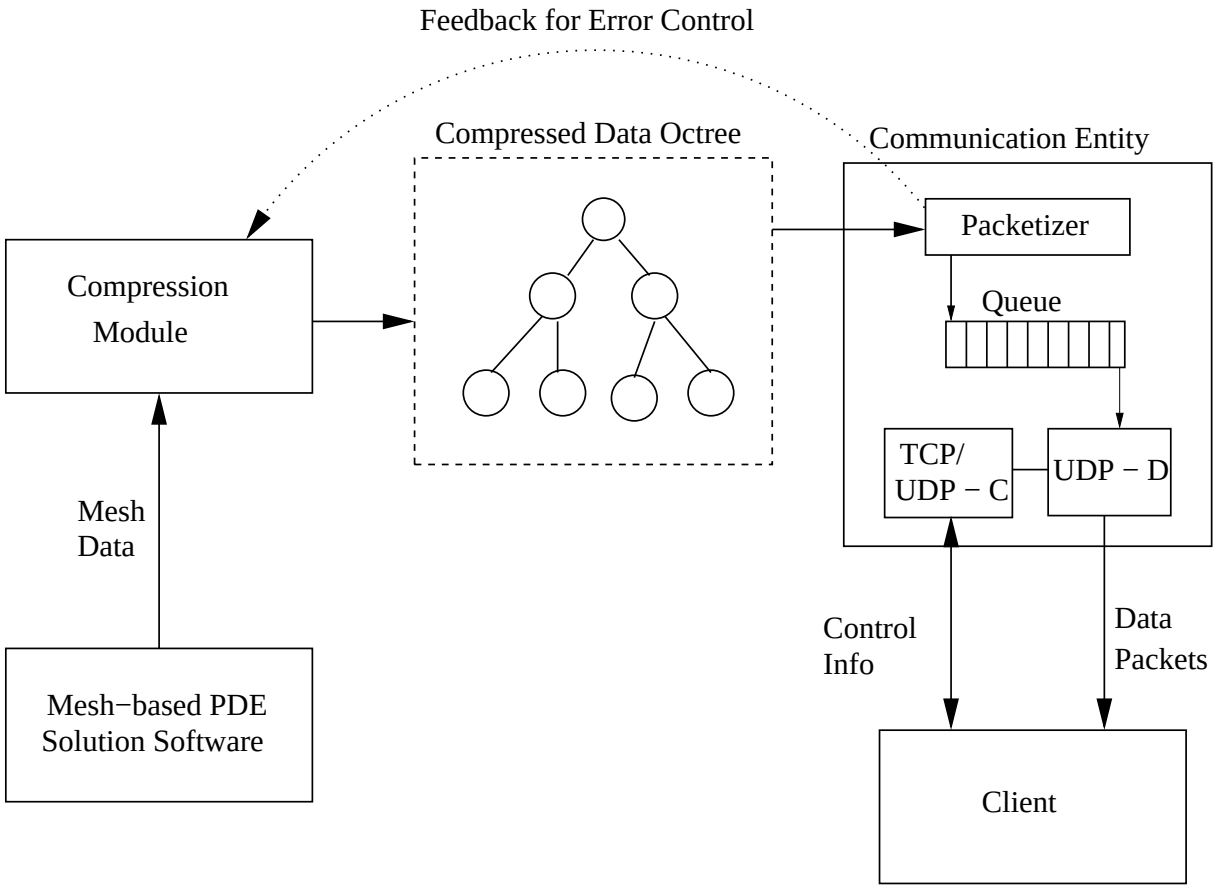

Figure 3.2: Server components and shared data structures with interface to the client

each leaf of the octree has an upper limit on the number of points it contains. High complexity regions with more number of points are split more as compared to regions with fewer points. Polynomial approximation [39] is performed on data in each octree leaf. The leaves of the octree then contain compressed data, with the position of the octant indicating the position of data in space. The degree of compression can be altered by changing the acceptable error bound, λ. As the PDE solution solver outputs time varying data, new nodes are added to the octree, while old ones may be modified or deleted. The basic structure of the tree is maintained, thus avoiding the cost of re-creating it for every time step. Another advantage of

such a technique is that each octant may have data corresponding to a different time-range, thus allowing efficient compression depending on the rate at which various regions of the mesh change. Once compressed data is ready for a particular time step, a signal is sent to the communication block.

3.1.3 Communication Block

The communication block implements the server side application level network protocol. The server is accessible to the outside world only through its communication block.

The functions of the server's communication block are handled by three threads. The first thread waits on a TCP port for data request from the client. Once a request is received, exchange of control data completes the handshake. The server then creates two new threads and opens a UDP socket for sending compressed data to the client. Another UDP socket may also be initiated in the first thread for control functions. This is indicated to the client during handshake.

The TCP connection is kept open for exchange of control information between the server and the client. This may include information regarding change in precision, UDP packet format, and specific user requests.

The second thread, called *packetizer* reads data off the octree, packets it and queues the packets for sending. As shown in Figure 3.2, the shared data structure between the compression module and the packetizer is the octree. In order to have a standard format, the packetizer converts all data and time information into IEEE format with the specified precision. Precision information is conveyed to the client during handshake.

The UDP-Data (UDP-D) channel is used only for the purpose of sending application data.

As long as the packet queue has packets to send, a regular flow of packets is maintained between the server and the client. The UDP-D thread shares the packet queue (Figure 3.2) with the packetizer thread. It extracts packets from the packet queue and sends them at a specified rate by controlling the inter-packet spacing [23].

UDP-Control (UDP-C) channel is not required by the application itself, but is useful for monitoring performance and performing rate control functions. It may be used for exchanging information about the number of packets received during a certain time interval, thus providing an indication of packet loss. Such feedback can be useful in controlling the sending rate. The receiver reports may also give an estimate of the round trip time (RTT), facilitating network monitoring. Chapter 4 presents a rate control scheme that is used to determine ideal sending rate as the simulation proceeds. This sending rate is used by the UDP-D thread to change inter-packet spacing. The inter-packet spacing determines the rate at which the packet queue empties. The rate control algorithm is closely tied to the system through the length of the queue and the error bound, λ. The basic idea is that the degree of compression can be altered based on availability of bandwidth. This scheme is detailed in Chapter 4.

Thus, the communication block is multi-threaded with one thread each for UDP-D channel and data packetization, and another for TCP-C and UDP-C channels. Presently, it does not support connections with multiple clients.

3.2 Client

The client software can be deployed over a desktop machine. Performance of the client may be enhanced with the use of high performance hardware graphics cards and high system resources. The client is composed of three blocks as described in the following sub-sections

and shown in Figure 3.3.

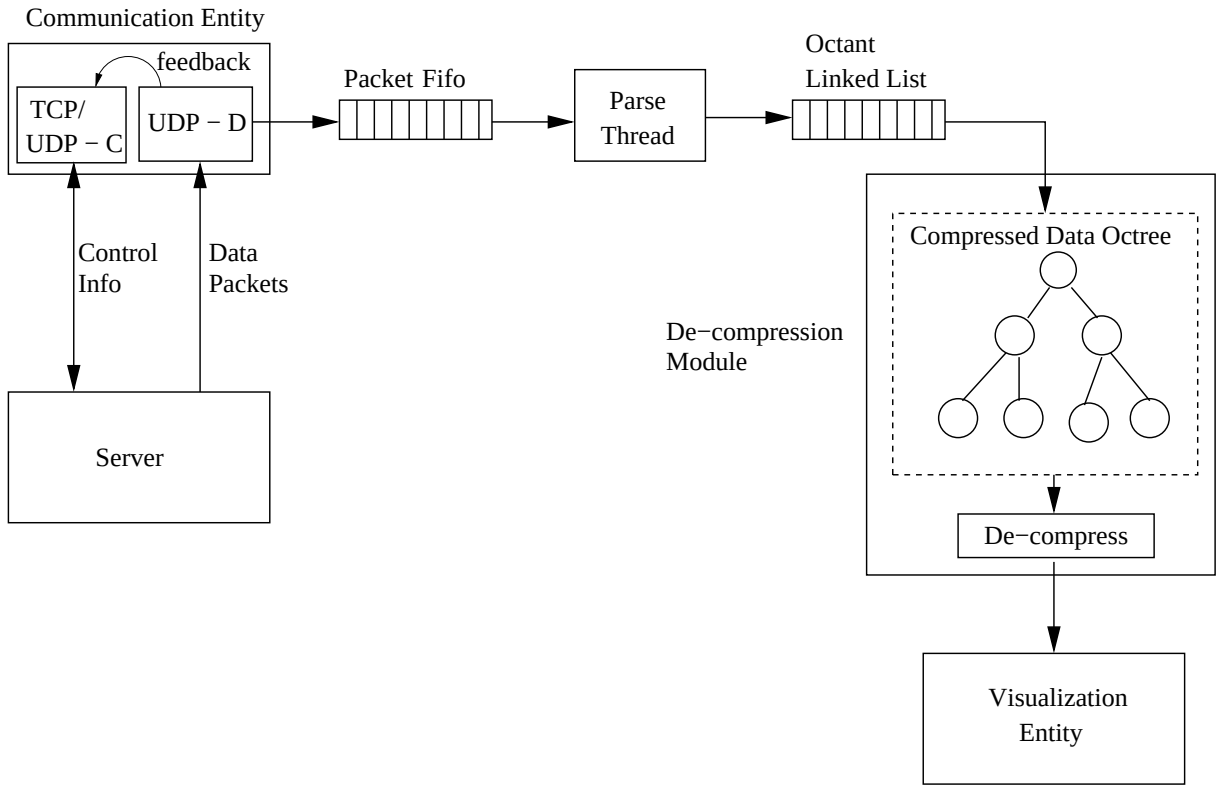

Figure 3.3: Data flow through client components and the data structures shared among them

3.2.1 Communication Block

The client communication block initiates the application and the interaction with the server. Like the server's communication block, this is also a multi-threaded application.

On start up, a TCP socket is created that sends a request for data to the server. During handshake, a UDP socket is created that waits for data to arrive. Handshake also includes the sending of the UDP-D port number of the client and receipt of data regarding packet

format. Packet format information is used by the client to interpret UDP packet data. It conveys the field arrangement and the length of each field in a packet.

The TCP connection and UDP-D channels are maintained by separate threads. Once the application is up, the TCP thread periodically waits for an interrupt to send user feedback to the server, and polls the socket for any data that the server might send. Every instance of information exchange over the TCP connection begins with a control signal that is used for receiving and interpreting the following information.

If the server so indicates, another UDP socket is created for the UDP-C channel and performance reports are sent to the server through this socket. This channel is also maintained by the UDP-D thread.

As shown in Figure 3.3, the UDP-D thread receives data packets and buffers them in a FIFO. As soon as the first packet is inserted into the FIFO, a *parse thread* is created. This thread extracts packets from the FIFO and parses them based on the packet format received from the server during handshake. Individual octants are extracted from the packets and placed in a linked list based on the time they are valid for. The decompression module has access to this linked list.

3.2.2 Decompression Module

As the name indicates, the basic function of the decompression module is to decompress polynomial data sent by the server and make it available for viewing.

As can be seen in Figure 3.3, the de-compression module functions as a separate thread, and shares the octant linked list with the parse thread. It extracts octants from the list based on their time stamps and uses them to create a replica of the server's octree (Figure 3.3).

Each octant has information on whether it is new or modified. The list also contains octants that need to be deleted from the tree. As simulation time proceeds and octants become outdated, more data arrives from the server in the form of octree updates which are then incorporated in the tree. Once the octree is created or updated, the polynomial data in each leaf of the tree is decompressed. Decompression of data includes evaluation of the function values at octant vertices. The decompression may be done through a software function call or in hardware. Presently, the de-compression is performed in software. The use of hardware for this functionality can potentially offer a high speed advantage. To gain an idea of the gain in speed attainable with the use of hardware for de-compression, a four-term Taylor series evaluation circuit was developed. The circuit used full precision pipelined constant multipliers and was found to work at a maximum speed of 121 MHz with 6% chip utilization on the Virtex II 6000 FPGA. As decompression proceeds across the leaves of the tree, point data is inserted into structures that the visualization entity uses for display. A signal is then sent to the visualization entity indicating that the data is ready for display.

3.2.3 Visualization Entity

The visualization entity uses VTK3.2 and the X-Windows interface to provide an interactive environment to the user. It shares certain VTK visualization structures with the decompression module for display of the octree data. The display is refreshed as and when the octree changes and a signal is received from the decompression module. The user may interrupt the refresh cycle in order to explore the data. VTK3.2 provides facilities for easy zooming and data picking. The dashed arrow in Figure 3.1 shows the option of user requirements being sent back to the server through the communication block and TCP-C connection. The motivation for this provision is derived from the capability of the server to adapt compression to user requirements.

Chapter 4

Networking Issues

A high-speed network is an important component of the visualization system presented in Chapter 3, but just having a such a network may not be sufficient. The application protocol should be designed in such a way that it is able to take advantage of the available bandwidth. Because the system requires fast delivery, the application was designed to send data over a UDP channel.

Even though UDP has lower overhead as compared to TCP, its unreliable nature introduces problems of packet loss, duplication and re-ordering. A UDP-based application should be able to handle these issues and also have some sort of congestion control mechanism in order to maximize data delivery. Some of the other important design considerations are byte ordering between machines of different architectures and choice of an appropriate UDP packet size. This chapter describes how the proposed system handles such issues.

4.1 Packet Duplication, Re-ordering, and Delay

The problem of duplicated packets is handled at the time of inclusion into the octree. Typically, such problems are detected using packet sequence numbers. The application does not insert a sequence number in the packet header. Therefore, the communication block (Figure 3.1) has no way of detecting such an occurrence. The duplicate packet is parsed and the octants are included in the linked list for insertion into the octree. The life span associated with the octant and its ID, uniquely identify the octant. This is used by the insertion/deletion/modification algorithm for the octree to detect a duplicate octant and discard it.

Considering the nature of the octree, packet re-ordering and delay do not manifest as potential problems. As long as the packet arrives in time to be played back, it can be included in the octree. If an octant arrives after its playback time has expired, it is not inserted into the linked list.

4.2 Packet Loss and Congestion Control

The proposed system is tolerant to packet loss and does nothing to recover a lost packet. Since the quality of the image depends on the amount of loss encountered, it is desirable to obtain a substantial amount of effective bandwidth. The amount of required bandwidth depends on the size of the compressed data set and how it changes over time. Thus, depending on bandwidth availability, the degree of compression may be increased or decreased. The following sub-section discusses an adaptive control theory approach that attempts to maximize the effective bandwidth of the application.

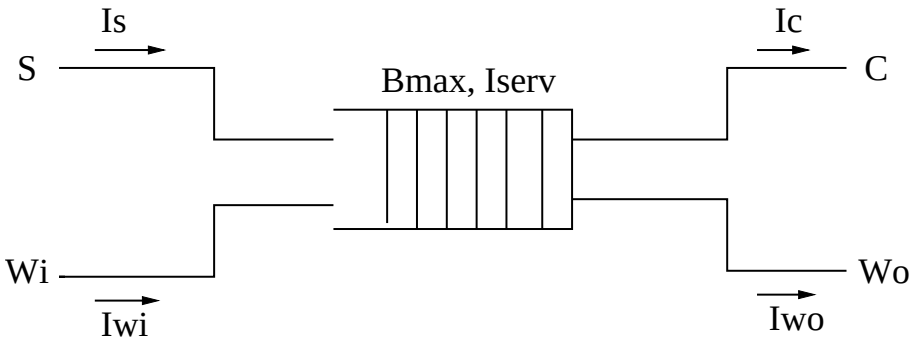

Figure 4.1: Network Model

4.2.1 An Adaptive Control Theory Scheme for Maximization of Effective Throughput

This scheme models the path between the source and sink as a single bottleneck link. The link receives traffic not only from the source of interest, but also from the rest of the network. In the proposed model, all other traffic generators are emulated by a single combined source.

The bottleneck link is completely defined by the buffer capacity (B_{max}) and service rate (I_{serv}). Figure 4.1 shows the model with S as the source of interest, W_i as the cumulative source, C as the sink for traffic generated by S and W_o as the receiver for traffic generated by W_i.

Let I_s = Instantaneous sending rate of S,
I_{wi} = Instantaneous sending rate of W_i,
I_{serv} = Service rate of the link,
I_c = Receiving rate of C,
I_{wo} = Receiving rate of W_o, and
$B_{max} = 1$
The underlying assumption for the model is that the ratio of I_c and I_{wo} in the output stream

is the same as the ratio of I_s and I_{wi}. In other words, none of the streams is given preferential treatment and packet loss is experienced by all streams at the bottleneck link.

The following set of equations describe the system behavior:

It may be noted that when $I_s + I_{wi} <= I_{serv}$, the buffer does not overflow, and therefore there is no loss. Thus,
when $I_s + I_{wi} <= I_{serv}$,

$$\text{Total Packet Loss Rate, } T_{pl} = 0, \text{ and} \tag{4.1}$$

$$\text{Packet Loss Rate for S, } P_l = 0. \tag{4.2}$$

When $I_s + I_{wi} > I_{serv}$,

$$T_{pl} = I_s + I_{wi} - I_{serv}. \tag{4.3}$$

As per the assumption stated above,

$$P_l = \frac{I_s}{I_s + I_{wi}} \times (I_s + I_{wi} - I_{serv}), \text{ and} \tag{4.4}$$

and

$$\text{Effective Bandwidth for S, } I_{se} = I_s - P_l. \tag{4.5}$$

Then, we may define a quality metric, Q, such that it provides a measure of the quality of the output:

$$Q = I_{se} - \beta \times P_l, \tag{4.6}$$

where β is a constant that determines a weight for P_l. A maximum value of Q would imply maximization of effective throughput and minimization of loss.

When $I_s + I_{wi} > I_{serv}$, we get

$$Q = I_s - (\beta + 1) \times P_l. \tag{4.7}$$

When $I_s + I_{wi} <= I_{serv}$, Equation 4.7 reduces to

$$Q = I_s. \tag{4.8}$$

Substituting P_l from Equation 4.4 in Equation 4.7,

$$Q = -\beta \times I_s + \frac{(\beta + 1) \times I_s I_{serv}}{(I_s + I_{wi})}. \tag{4.9}$$

Equation 4.7 shows that as β is increased, higher weight is given to P_l, leading to lower Q. Thus, the parameter β may be used to regulate the amount of acceptable packet loss, while maximizing the effective throughput.

Figure 4.2 shows Q plotted as a function of the sending rate, I_s for $\beta = 1$, and constant I_{wi} and I_{serv}. It can be seen that as the sending rate, I_s is increased, Q increases to a maximum value before dropping down. The maximum value on the curve gives us an optimal point of operation in terms of an ideal sending rate for maximized effective bandwidth and minimized loss.

The plot in Figure 4.2 shows the case when $I_{wi} = I_{serv}$. Figure 4.3 shows the plot for Q vs. I_s for $I_{wi} > I_{serv}$. In this case, the link is completely choked by external traffic, but we are still able to obtain a maximum value of Q greater than 0. Thus, even when the link is choked, we are able to attain an effective throughput that is greater than 0. Figures 4.4 and 4.5 are plotted for $I_{wi} < I_{serv}$. The linear increase in both these plots depicts the condition $I_s + I_{wi} <= I_{serv}$ and follows the Equation 4.8. The remaining portion of the plot follows from Equation 4.9. Clearly, Q reaches a maximum value in each case.

Thus, we can vary I_s so as to maximize the effective bandwidth with minimal cost in terms of packet loss. This gives us a control equation for I_s that allows us to change I_s iteratively, such that Q is maximized [40]:

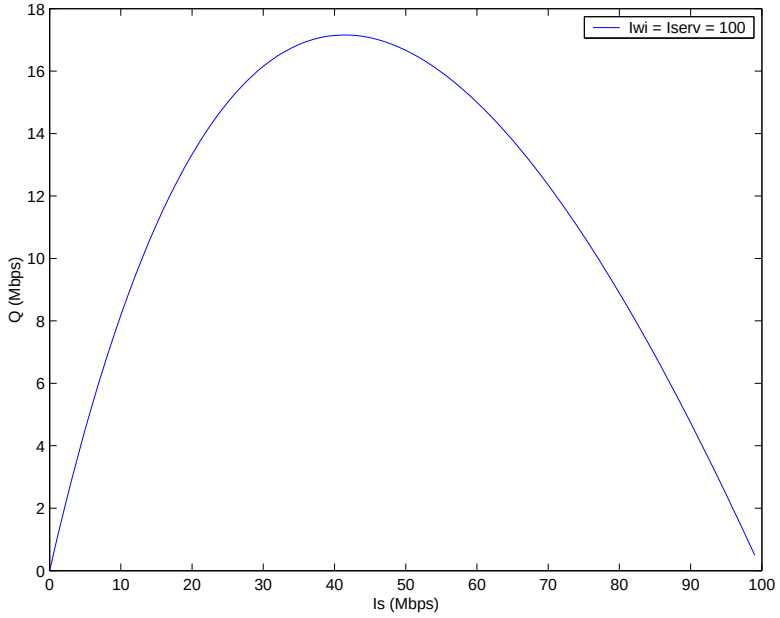

Figure 4.2: Plot of Q vs. I_s as I_s is varied from 0 to I_{serv} and I_{wi}=I_{serv} = 100 Mbps. Maximum Q = 17.16 Mbps; $\beta = 1$.

$$I_s(i) = I_s(i-1) + \frac{dQ}{dI_s} \tag{4.10}$$

$\frac{dQ}{dI_s}$ may be estimated with center difference as:

$$\frac{dQ}{dI_s} = \frac{Q(i) - Q(i-2)}{I_s(i) - I_s(i-2)} \tag{4.11}$$

Figures 4.2 and 4.3 show that Q is always a convex function, and therefore has only one maximum value. Because Q remains convex for any value of I_{wi}, it is always possible to find a maximum value of Q, even if I_{wi} changes. This makes the control algorithm adaptive to changing network load.

In order to test the control algorithm, several simulations were run using *Matlab version 6.1.0.450*. Simulation results showed that, based on Equation 4.10, iterative changes in I_s

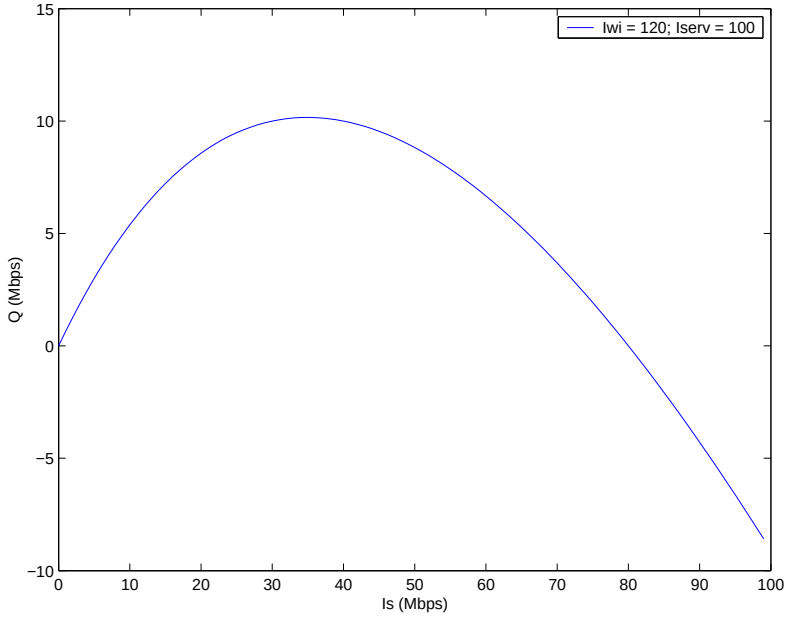

Figure 4.3: Plot of Q vs. I_s as I_s is varied from 0 to I_{serv}, $I_{wi} = 120$ Mbps and $I_{serv} = 100$ Mbps. Maximum $Q = 10.16$ Mbps; $\beta = 1$

introduce a high degree of variance in $\frac{dQ}{dI_s}$, making it impossible for the system to reach a steady value of I_s at maximum Q. Therefore, $\frac{dQ}{dI_s}$ in Equation 4.10, was replaced by a constant a. The value of constant, a, determines the rate of convergence as well as the variation in I_s. A high value would increase the variation in sending rate, but may lead to faster convergence for the system. The iterative control equation may now be re-written as:

$$I_s(i) = \begin{cases} I_s(i-1) + a & \text{when } \frac{dQ}{dI_s} > 0 \\ I_s(i-1) - a & \text{when } \frac{dQ}{dI_s} < 0 \\ I_s(i-1) & \text{otherwise} \end{cases} \tag{4.12}$$

Simulation results show that a high effective bandwidth can be achieved using the above control algorithm for the sending rate. The simulations are run assuming that the receiver (C) sends packet loss notification to the sender, S. The sender then calculates Q and changes the sending rate according to Equation 4.12.

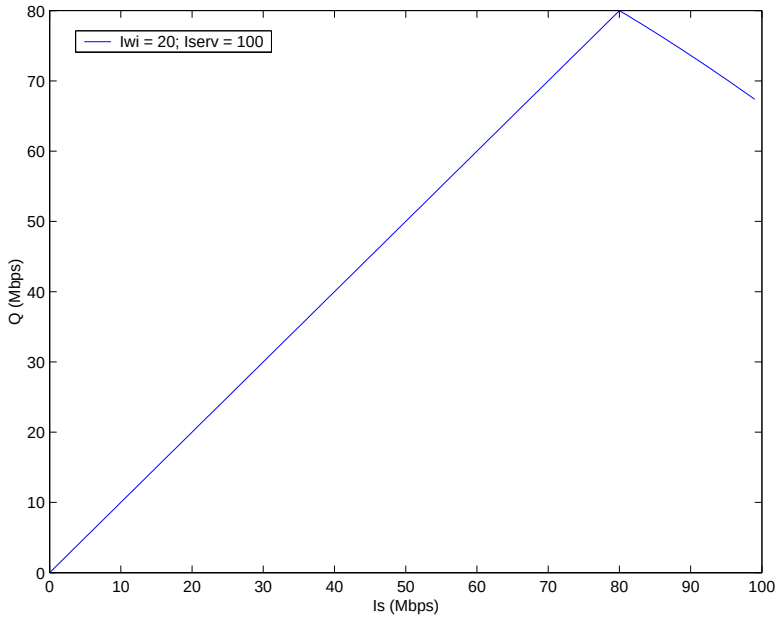

Figure 4.4: Plot of Q vs. I_s as I_s is varied from 0 to I_{serv}, $I_{wi} = 20$ Mbps and $I_{serv} = 100$ Mbps. Maximum $Q = 80$ Mbps; $\beta = 1$

Because the only observable value in the network is P_l, the simulation uses Equation 4.7 to calculate Q. Figures 4.6, 4.7 and 4.8 show I_s and Q as I_{wi} changes over time.

It may be noted that I_s tends to reach a steady state as I_{wi} becomes constant over time and thus displays controlled behavior. In the simulation results, Q attains steady-state values of 80 Mbps and 27.02 Mbps for $I_{wi} = 20$ Mbps and $I_{wi} = 80$ Mbps respectively. These are also the maximum values of Q in Figures 4.4 and 4.5.

The results of another simulation are shown in Figures 4.9, 4.10, 4.11, and 4.12 with $I_{serv} =$ 10 Mbps and $I_{wi} = 4$ Mbps. Plots in Figures 4.9, 4.10 and show I_s and P_l plotted against time for different values of β. It can be seen that as β is increased from 1 in Figure 4.9 to 20 in Figure 4.10, the peak packet loss rate reduces from 0.25 Mbps to 0.025 Mbps. Figures 4.11 and 4.12 show I_s and I_{se} for the same simulation runs with $\beta = 1$ and 20 respectively.

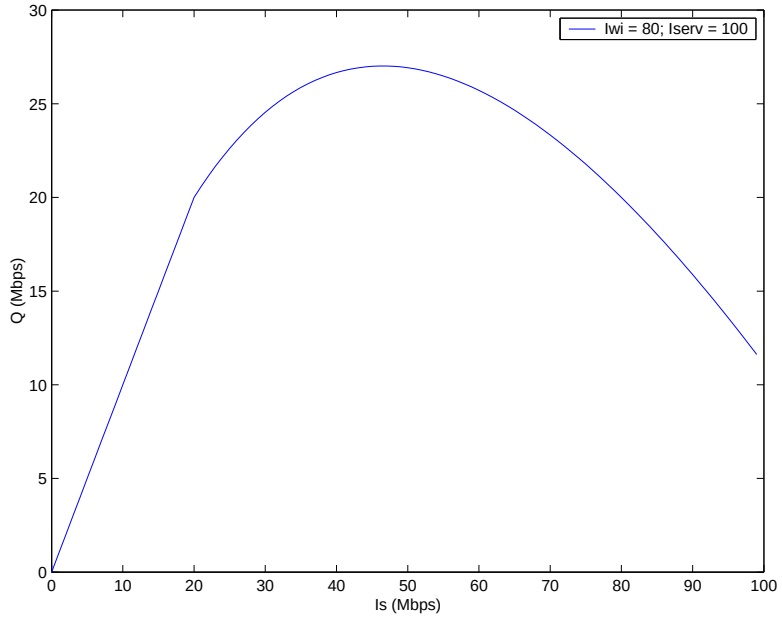

Figure 4.5: Plot of Q vs. I_s as I_s is varied from 0 to I_{serv}, $I_{wi} = 80$ Mbps and $I_{serv} = 100$ Mbps. Maximum $Q = 27.02$ Mbps; $\beta = 1$

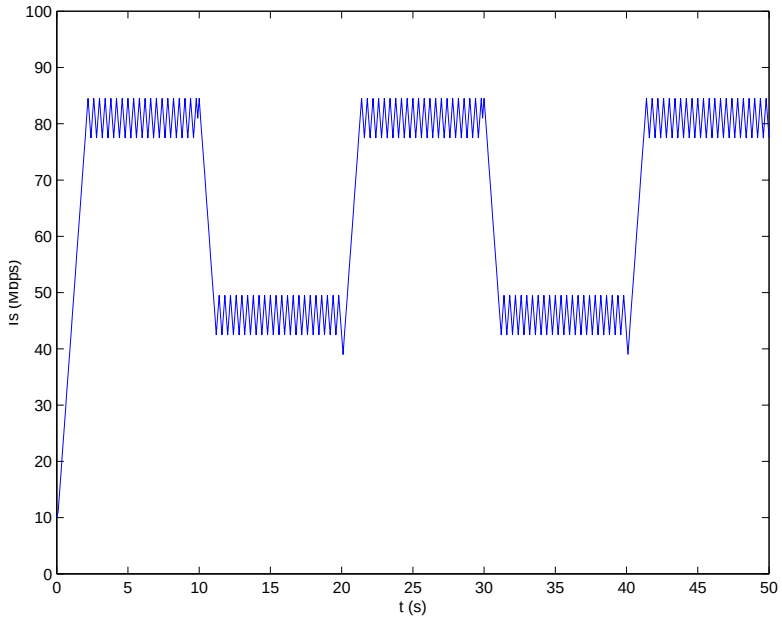

Figure 4.6: Plot of I_s vs. time with $I_{serv} = 100$ Mbps; $\beta = 1$

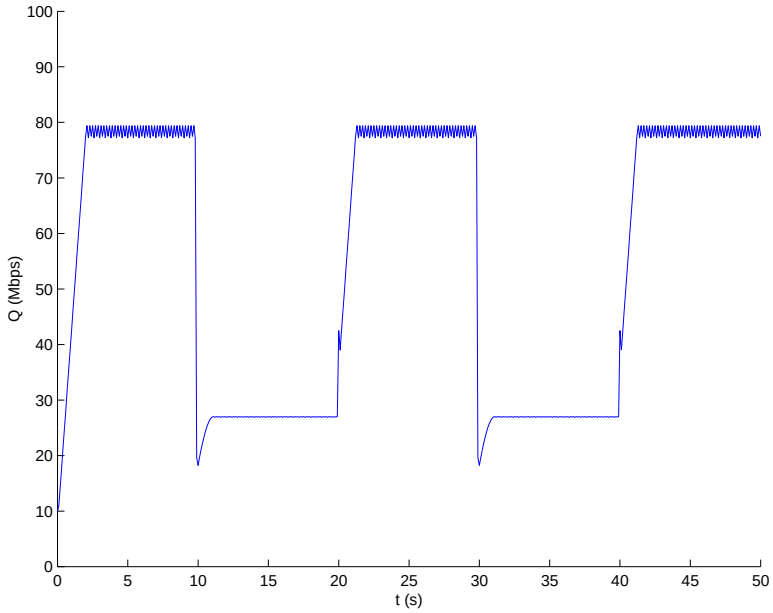

Figure 4.7: Plot of Q vs. time with $I_{serv} = 100$ Mbps; $\beta = 1$

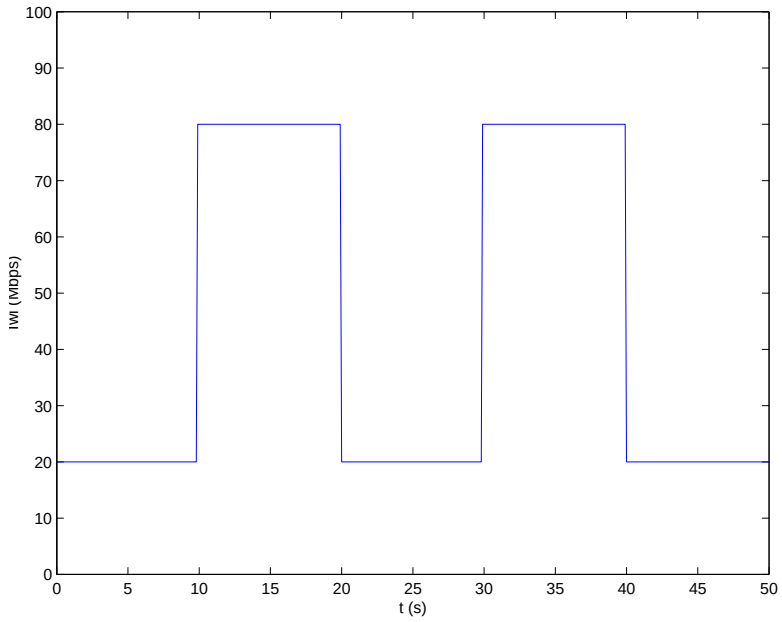

Figure 4.8: Plot of I_{wi} vs. time with $I_{serv} = 100$ Mbps; $\beta = 1$

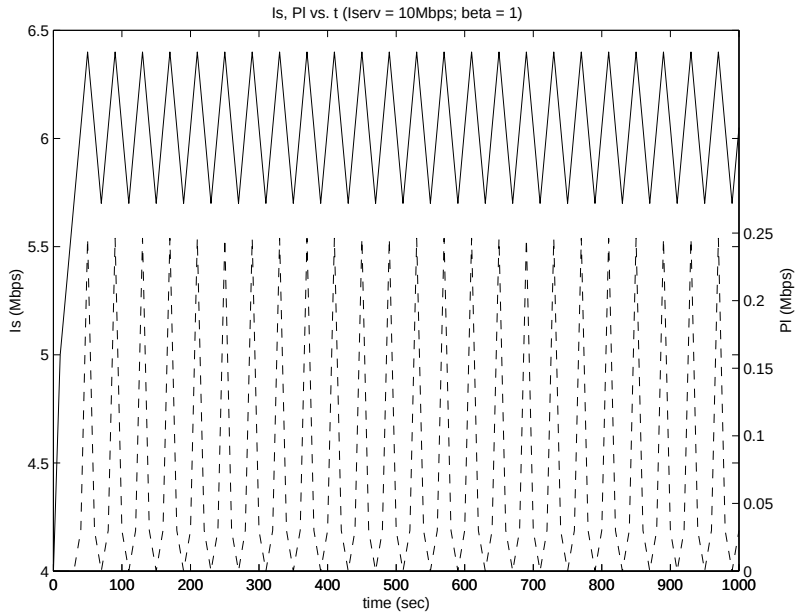

Figure 4.9: Plot of I_s and P_l vs. time with $I_{serv} = 10$ Mbps; $I_{wi} = 4$ Mbps; $\beta = 1$

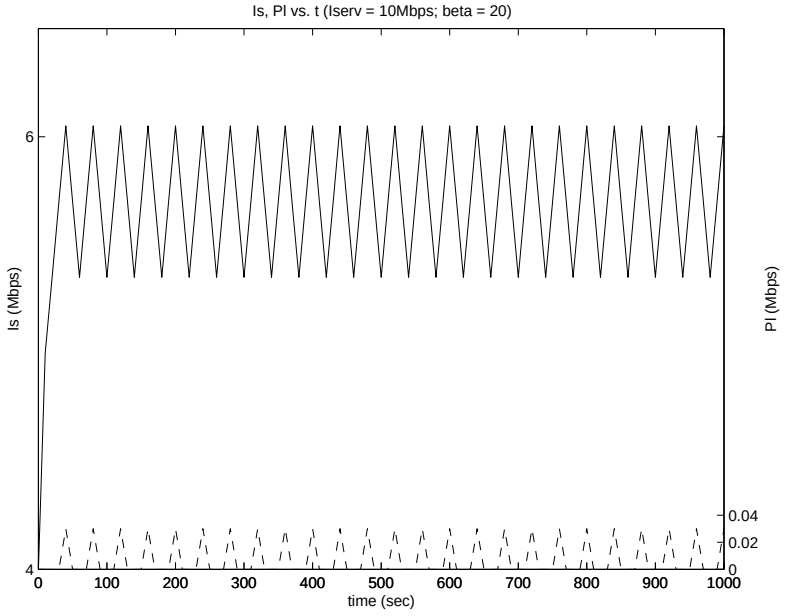

Figure 4.10: Plot of I_s and P_l vs. time with $I_{serv} = 10$ Mbps; $I_{wi} = 4$ Mbps; $\beta = 20$

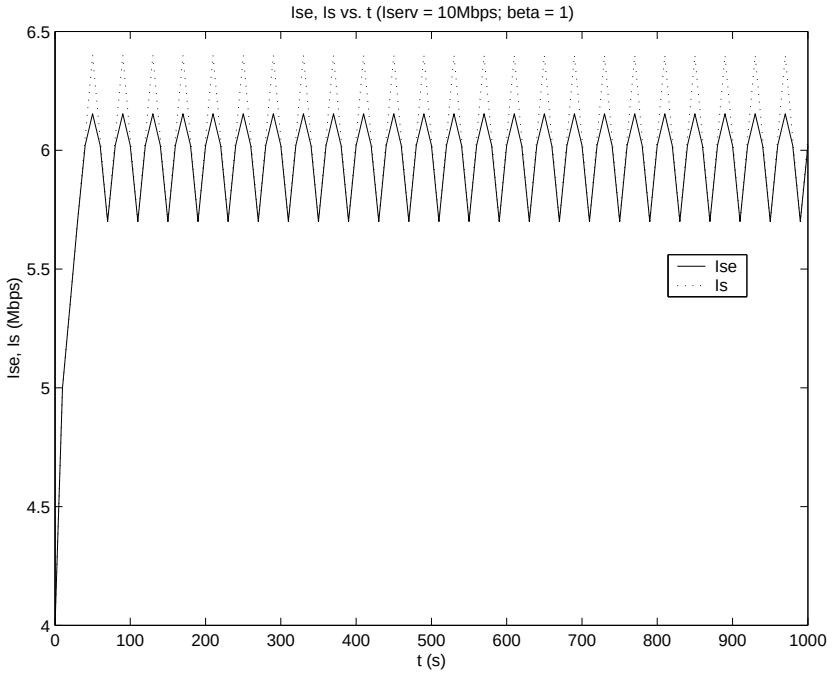

Figure 4.11: Plot of I_s and I_{se} vs. time with $I_{serv} = 10$ Mbps; $I_{wi} = 4$ Mbps; $\beta = 1$

It can be seen that an increase in β reduces the packet loss rate while still maximizing the effective throughput.

Thus, the above scheme allows us to control the sending rate such that the effective bandwidth achieved is high even when the link is congested. The following sub-section details how this adaptive rate control scheme is used in the proposed system.

4.2.1.1 Rate Control for the Visualization System

The degree of compression is determined by the error bound, λ. A high λ would result in a coarser image with less number of octants as compared to a lower λ that would increase the image resolution as well as the amount of data to send to the client. The relation between λ and number of octants is shown in Figures 4.13, and 4.14 for two 3-D data sets and zero

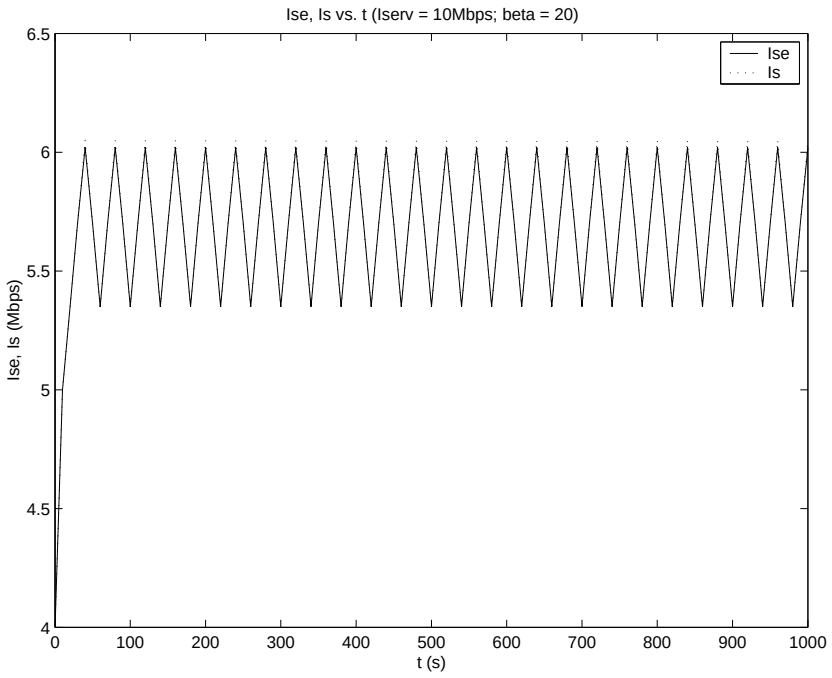

Figure 4.12: Plot of I_s and I_{se} vs. time with $I_{serv} = 10$ Mbps; $I_{wi} = 4$ Mbps; $\beta = 20$

degree polynomials evaluated for each octant. It can be seen that an increase in the error bound, reduces the number of octants. This is due to the increased compression achieved by acceptance of higher error. Though the values in the the two figures are specific to their respective data sets, they reflect a general trend any data set would have.

Thus, the system can be made to adapt to changing bandwidth by tuning λ to regulate the amount of data being sent. The sending rate determines how fast the packet queue (Figure 3.2) gets emptied. Therefore, the length of the queue gives an estimate of the current sending rate.

For the purpose of adaptively determining λ, the target packet queue length is set to be half the maximum queue length. For every time step, the current queue length is compared to the target queue length. Also, the derivative of the curve at the current lambda ($\frac{dO}{d\lambda}$) is

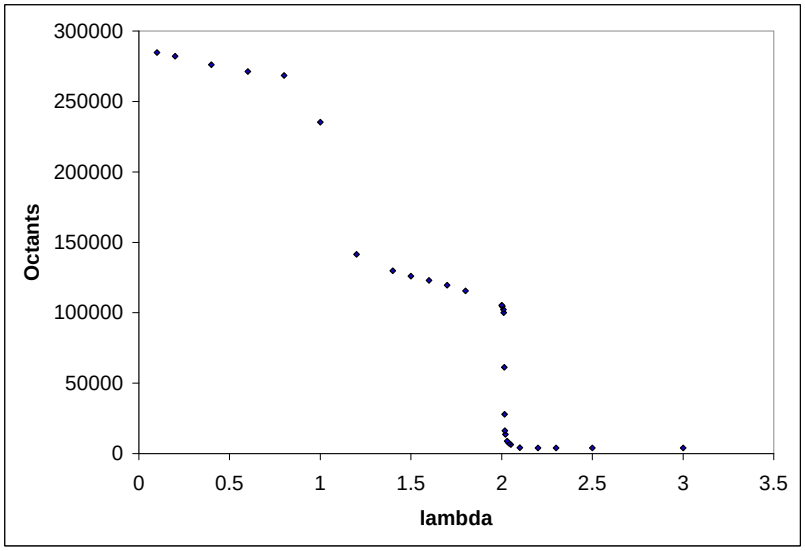

Figure 4.13: Number of octants vs. λ for a 3-D Raleigh Taylor series simulation data

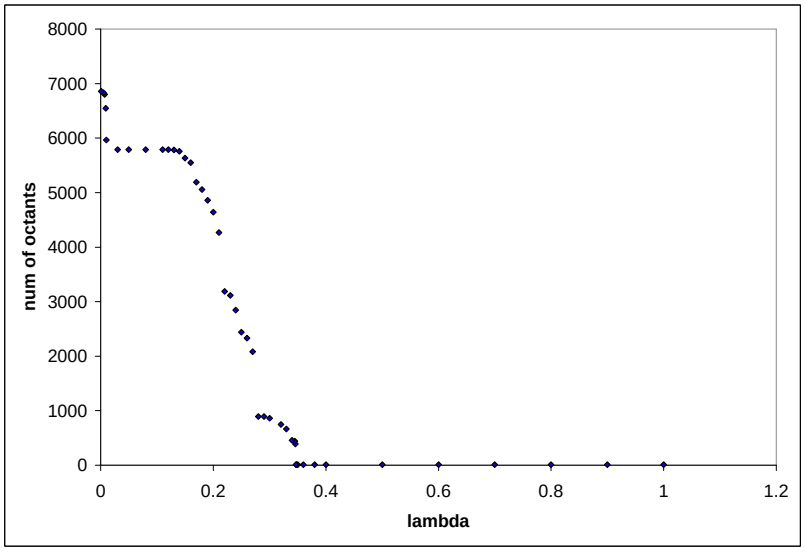

Figure 4.14: Number of octants vs. λ for a 3-D Gaussian data set

calculated using the previous three values of the derivative. If the target queue length is greater than the current queue length, target number of octants is calculated as:

$$target\_oct = (target\_queue\_length - cur\_queue\_length) \times octants\_per\_packet, \text{and} \quad (4.13)$$

λ is decremented by $d\lambda$, where $d\lambda$ is found as:

$$d\lambda = |\frac{(target\_oct - octs\_prev\_timestep)}{\frac{dO}{d\lambda}}| \quad (4.14)$$

where *octs_prev_timestep* is the number of octants sent in the previous time step.

If the target queue length is less than the current queue length, λ is incremented by the previous value of $d\lambda$.

This algorithm allows us to find the derivative of the curve in Figures 4.13, and 4.14 at each point and enables fine λ control. The system performance achieved using this control strategy is shown in Chapter 5.

4.3 Byte Ordering

Different machines store data differently. There is no standard specifying whether two bytes of data should be stored in little- or big-endian format. If the client and server applications are deployed on different architectures, it is possible that they store data differently and communication of anything except individual bytes would be meaningless. There are two ways of handling this problem:

- Symmetric Data Conversion
- Asymmetric Data Conversion

Symmetric Data Conversion means that the data is converted to a standard network format at the sender's side, and the receiving application converts it from the network format to the local machine format. External Data Representation (XDR) [41] is a standard used by most appplications. Such a system encounters processing overhead at both ends, irrespective of whether the architectures are actually different.

Asymmetric Data Conversion involves data conversion only at one end point. This requires that either the client or the server know the architecture at the other end. It is impractical to have the application configured to convert between all possible architectures. Therefore, the proposed system involves exchange of a known value between the client and server during handshake. If the received value does not match the expected value, both machines recognize the incompatibility. Thereafter, no conversion is done by the sender, but the receiver inverts byte ordering before interpreting the data. This technique requires data conversion only at one end and only if the machines are identified to store data differently.

4.4 Packet Size

Choosing the appropriate packet size is important to a time-critical application. Small packets would increase the overhead in terms of the ratio of header size to useful data size. On the other hand, large packets might lead to packet fragmentation and re-assembly increasing the processing overhead. In particular, an IP packet is fragmented if its size exceeds the supported Maximum Transfer Unit (MTU) of the data link connection. Therefore, in order to avoid fragmentation, the packet size should be chosen such that it is less than or equal to the smallest MTU of the path. Most present day networks, specifically Ethernet, support an MTU of 1500 Bytes. This can also be estimated at run time using socket level calls.

By default, the application packet size is chosen to be 1472 bytes to accommodate 8 and 20

bytes of UDP and IP headers respectively. The total packet size at the IP level then becomes 1500 bytes, and avoids packet fragmentation. A packet size of 1472 bytes was found to be sufficiently large to accommodate a substantial amount of node information. At the same time, it is small enough to not lose a lot of information in case of packet loss. The amount of data actually contained in the packet depends on the length of compressed polynomials and the required precision.

Chapter 5

Experimental Results

This chapter details results from experimental runs of the rate control scheme, presented in Chapter 4, and the visualization system.

A simple UDP channel was used to test the rate control algorithm. Tests conducted between various end-points showed that the rate control works effectively. This method of rate control was then tied to the proposed system, and the system was found to work successfully over actual networks. The system is seen to be robust, low on its resource utilization, and adaptive to changing network conditions. The following subsections show illustrative rate control and system performance results.

5.1 Rate Control Results

To test the rate control algorithm, a server was created to send fixed size UDP packets (1472 bytes) at specified rates to the client. The server then recorded the number of packets sent

Path	Source	Destination	Max. UDP Bandwidth	TCP Bandwidth	RTT	Number of Hops
1.	128.173.52.7	140.221.11.103	9.0 Mbps	3.1 Mbps	44.9 ms	17
2.	128.173.52.7	130.203.8.38	9.0 Mbps	815 Kbps	41.2 ms	17
3.	140.221.11.103	130.203.8.38	70.4 Mbps	2.3 Mbps	19.2 ms	12
4.	128.173.52.7	66.37.66.32	523 Kbps	444 Kbps	56 ms	19

Table 5.1: Characterization of paths between various source and destination pairs used for testing the rate control algorithm

every 1 sec. Once the client started receiving the packets, it sent back a report containing the number of packets it received every second. The reports contained sequence numbers enabling the server to match the number of received packets against the number of packets sent during a particular interval. Once the server received 10 reports, it calculated packet loss rate based on the difference between number of packets sent and received. As per the algorithm outlined in Chapter 4, a new sending rate was determined based on the packet loss rate, Q value, and current sending rate. Inter-packet spacing was then changed so as to conform to the new sending rate.

This test program was deployed on four sets of network endpoints. The tests were performed when the least traffic was expected on the links and Iperf version 1.1.1 [42] was used to control I_{wi} (Figure 3.1) deterministically. Table 5.1 lists the four test paths along with the maximum TCP and UDP bandwidths observed through Iperf, average round trip times (RTT) and the number of hops between the source and destination pairs as obtained from the system utilities, *ping* and *traceroute* respectively, and averaged over 5 runs. Detailed node information between this endpoints, obtained from the system utility *traceroute*, is given in Appendix A.

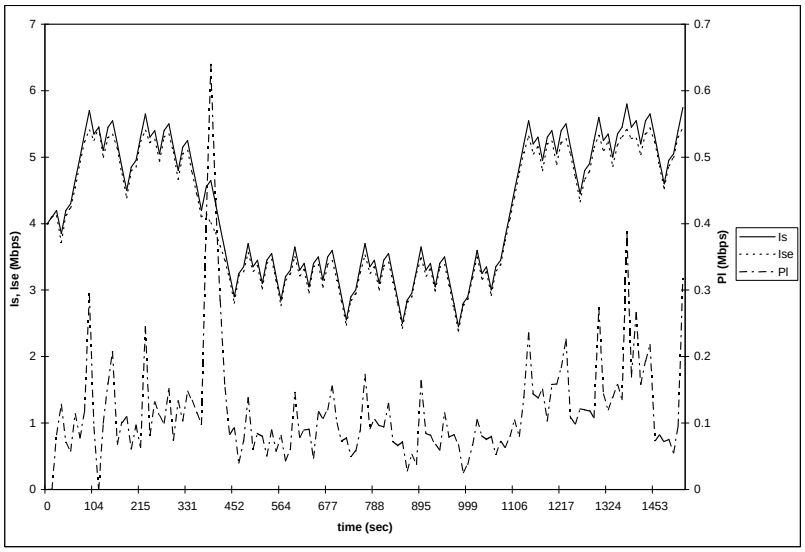

Figure 5.1: Plot of I_s, I_{se} (left axis) and P_l (right axis) vs. time as I_{wi} changes from 4 Mbps to 6 Mbps and back to 4 Mbps along path 1; $\beta = 9$

Figure 5.1 shows a test run result on path 1 (Table 5.1) with initial sending rate, $I_s = 4$ Mbps and $\beta = 9$. The plot shows the sending rate, I_s and effective throughput, I_{se} on the left axis as they change over time. The right axis shows corresponding packet loss rate, P_l. As time proceeds, I_{wi} is changed from 4 Mbps to 6 Mbps and back to 4 Mbps. Figure 5.2 is a similar plot showing results of a test run along path 2 with parameters same as those in the previous case. In each of these plots, sending rate closely follows the effective throughput, and dynamically responds to a change in load.

Figure 5.3 shows the change in sending rate I_s, effective throughput I_{se} and packet loss rate P_l with respect to time as I_{wi} is kept constant at 30 Mbps along path 3. Here again, I_s is seen to closely follow I_{se} so as to keep P_l low. In this case, $\beta = 3$. Similar results were obatined along path 4. These are shown in Figure 5.4 for $I_{wi} = 0.2$ Mbps and $\beta = 9$. The destination host for path 4 is a home computer connected to the Internet through a 1.5 Mbps

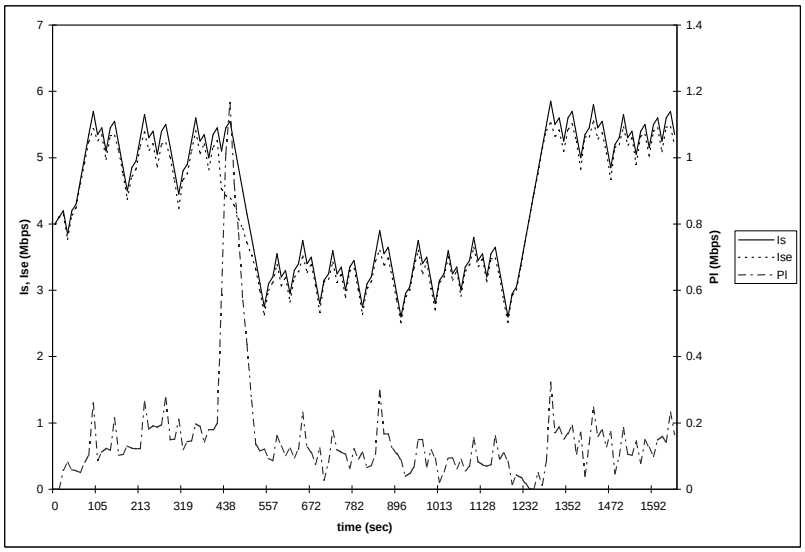

Figure 5.2: Plot of I_s, I_{se} (left axis) and P_l (right axis) vs. time as I_{wi} changes from 4 Mbps to 6 Mbps and back to 4 Mbps along path 2; $\beta = 9$

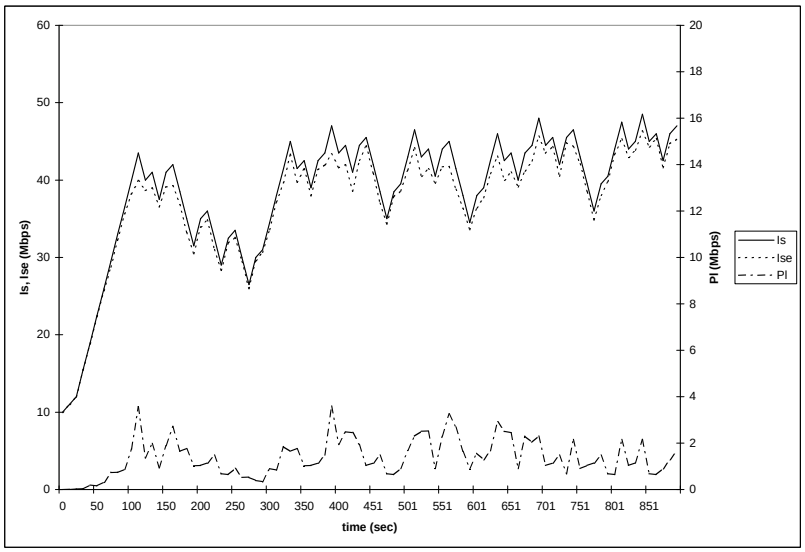

Figure 5.3: Plot of I_s, I_{se} (left axis) and P_l (right axis) vs. time with $I_{wi} = 30$ Mbps along path 3; $\beta = 3$

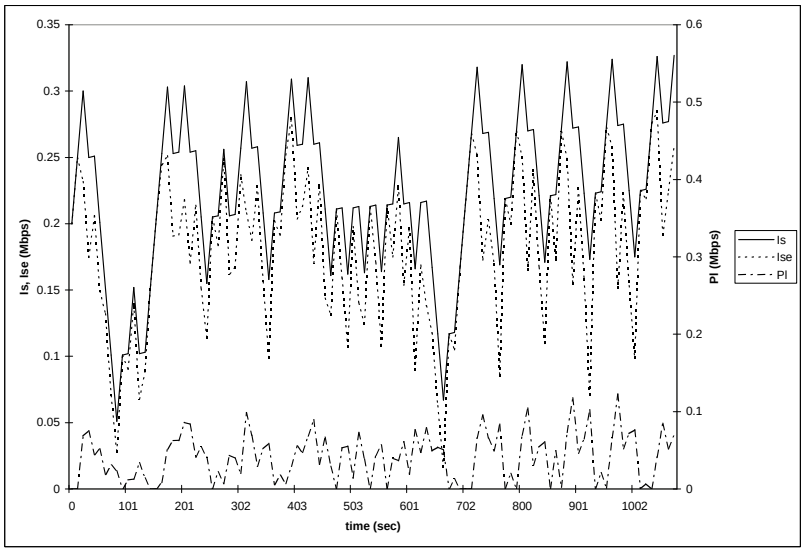

Figure 5.4: Plot of I_s, I_{se} (left axis) and P_l (right axis) vs. time with $I_{wi} = 0.2$ Mbps along path 4; $\beta = 9$

access link. The traffic along this path was found to be highly variable, accounting for the variation in I_{se}.

The parameter β, in Equation 4.9, was found to be effective in controlling the amount of tolerable packet loss. Figures 5.5 and 5.6 show the packet loss rates observed along path 1 with $I_{wi} = 4$ Mbps, and $\beta = 1$ and 9 respectively. The average packet loss rate for $\beta = 1$ was found to be 0.0958 Mbps, whereas it reduced to 0.0387 Mbps for $\beta = 9$. The peak packet loss rate dropped from 0.507 Mbps for $\beta = 1$ to 0.231 Mbps for $\beta = 9$.

In order to evaluate the usefulness of the control scheme, packet loss rates attained with the control mechanism were compared to those obtained when a constant data rate was maintained. Figure 5.7 shows I_s, I_{se} and P_l obtained along path 2 with constant $I_{wi} = 4$ Mbps and $\beta = 9$. The average sending rate obtained during this experiment was 4.95 Mbps. In order to compare this result with the case when no rate control is employed, a constant

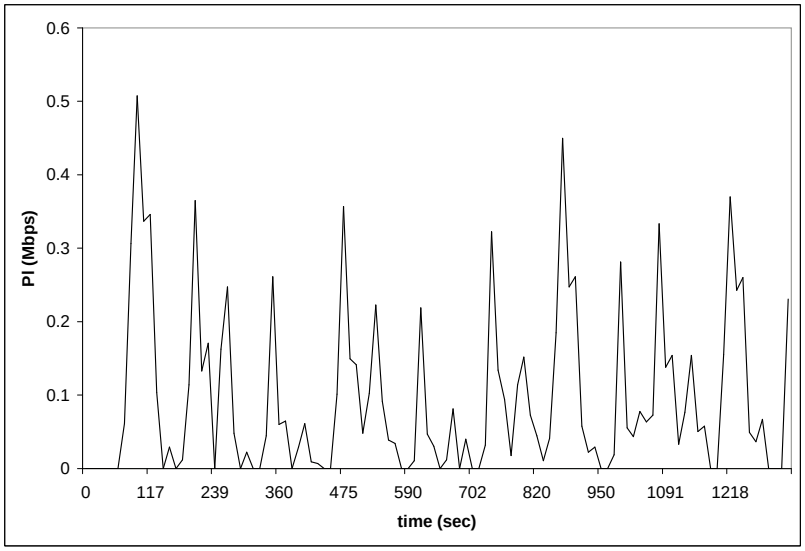

Figure 5.5: Plot of P_l vs. time with $I_{wi} = 4$ Mbps along path 1; $\beta = 1$

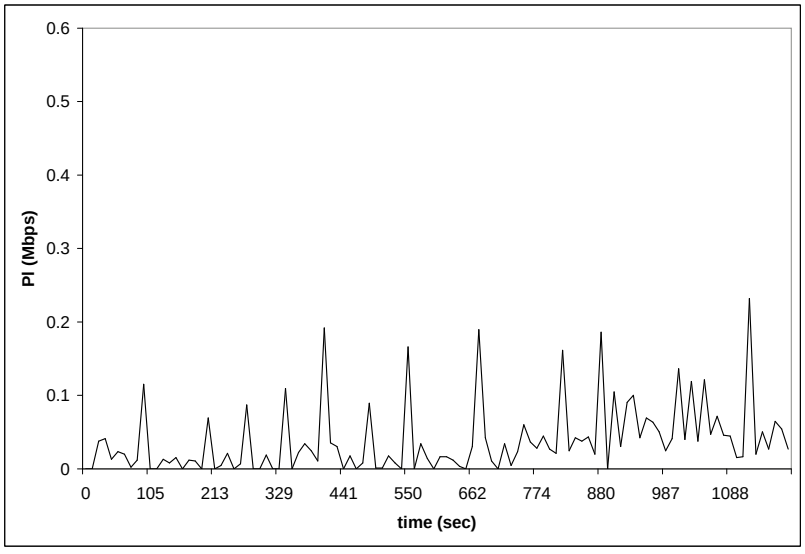

Figure 5.6: Plot of P_l vs. time with $I_{wi} = 4$ Mbps along path 1; $\beta = 9$

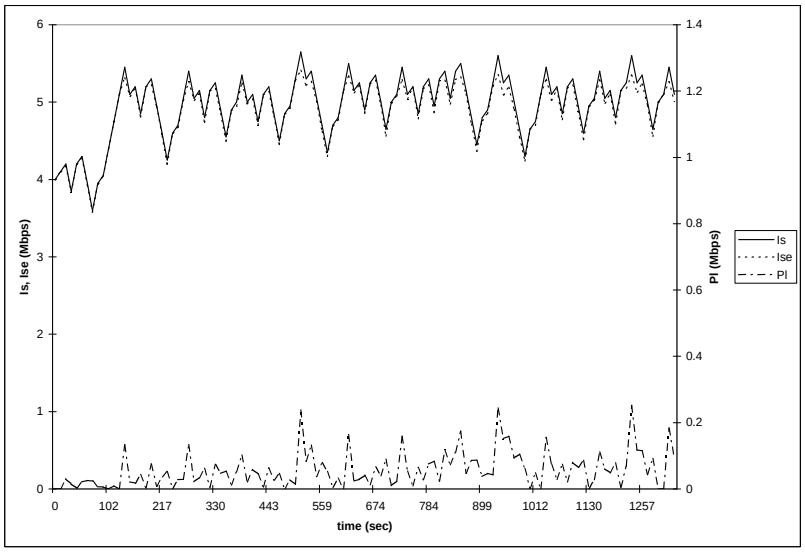

Figure 5.7: Plot of I_s, I_{se} (left axis) and P_l (right axis) vs. time with $I_{wi} = 4$ Mbps along path 2; $\beta = 9$

$I_s = 4.95$ Mbps was used to obtain results shown in Figure 5.8. As seen from Figures 5.7 and 5.8, the packet loss rate is much lower when no rate control is used as against the case when the rate control mechanism is in place. The advantage of using the rate control scheme becomes evident when the load on the link changes. This is clearly seen from Figure 5.2, where the rate control scheme is in place, and Figure 5.9 where a constant sending rate of 4.95 Mbps is maintained. In both the cases. I_{wi} is changed from 4 Mbps to 6 Mbps and back to 4 Mbps. Evidently, the packet loss rate is much higher if no rate control is performed and the load on the network changes.

The above graphs bring out the robustness and usefulness of the algorithm. The scheme is found to be very beneficial in an environment where load on the network is constantly changing. This, however, is not unusual in the Internet. The amount of tolerable packet loss may also be tuned using the parameter β making the algorithm highly useful for applications

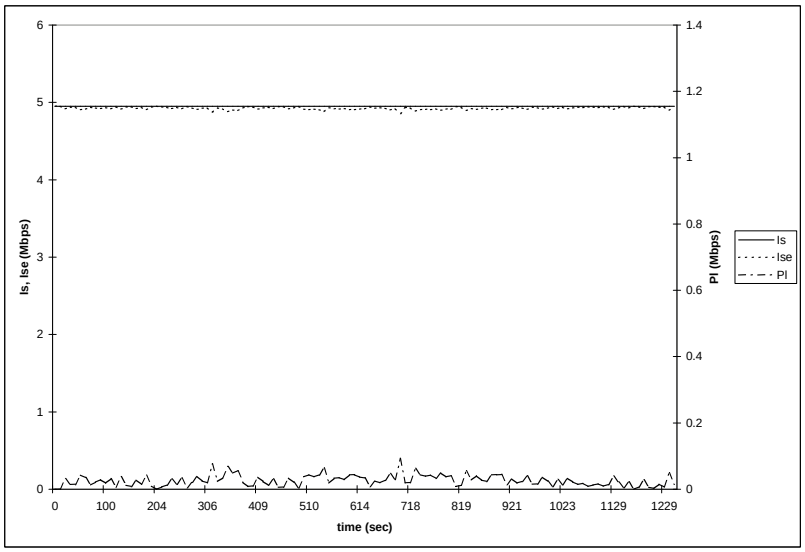

Figure 5.8: Plot of I_s, I_{se} (left axis) and P_l (right axis) vs. time with $I_s = 4.95$ Mbps and $I_{wi} = 4$ Mbps along path 2

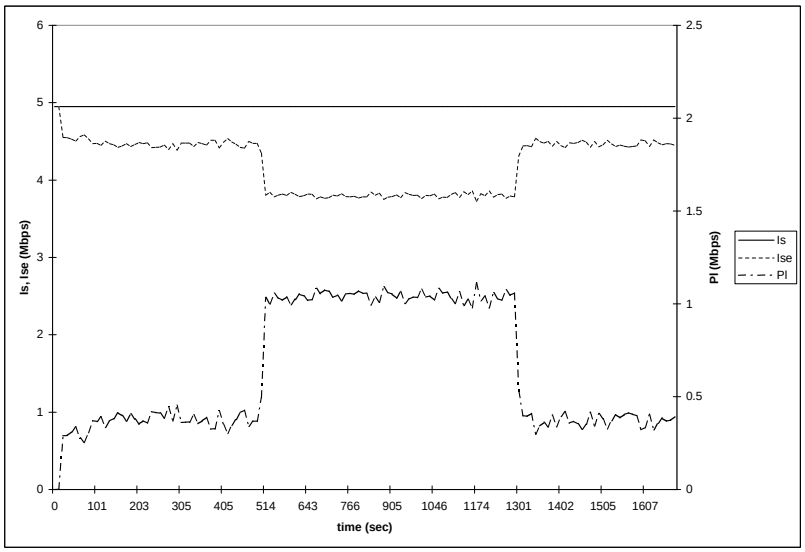

Figure 5.9: Plot of I_s, I_{se} (left axis) and P_l (right axis) vs. time with $I_s = 4.95$ Mbps and I_{wi} changing from 4 Mbps to 6 Mbps and back to 4 Mbps along path 2

that use UDP, send data for long periods of time and require high throughput. The next section shows how this algorithm was found to be of use in the proposed system.

5.2 System Performance

This section details the results of test runs performed for remote visualization of a 2239488 point 3-D unstructured mesh obtained from Raleigh-Taylor (R-T) simulations. The data set was produced by an adaptive tetrahedral mesh code using Discontinuous Galerkin Euler formulation developed at Rensselear Polytechnic Institute [43]. The application was deployed over the Internet along path 1 (Table 5.1).

The application was found to be stable and adaptive to changing network conditions. This is demonstrated in Figures 5.10 and 5.11 plotted for I_{wi} values of 2 Mbps, and 4 Mbps respectively. The figures show sending rate, I_s, effective throughput, I_{se} and packet loss rate, P_l as they change over time with $\beta = 1$. In both cases, the sending rate is seen to follow the effective throughput while maintaining a low packet loss rate.

Figure 5.12 shows the time lag encountered by the system for the two cases mentioned above. It is seen that in both cases the difference between display time and generation time is bounded. Once the system attains stability, the time lag does not vary substantially. When I_{wi} is 2 Mbps the sending rate achieved by the application is higher than that attained for the case of $I_{wi} = 4$ Mbps. This accounts for the lower time lag in the case when $I_{wi} =$ 2 Mbps. Figure 5.12 also shows the initial fluctuation in the display time before the system attains stability.

In order to quantify the quality of the image, error value was calculated for each octant assuming it being lost and reconstructed from its neighbors. The maximum error value

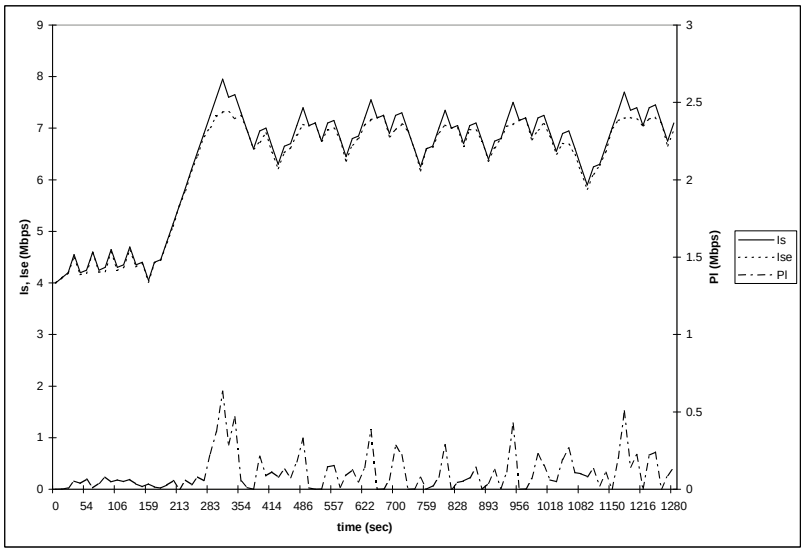

Figure 5.10: Plot of I_s, I_{se} (left axis) and P_l (right axis) vs. time with initial $I_s = 4$ Mbps and $I_{wi} = 2$ Mbps along path 1

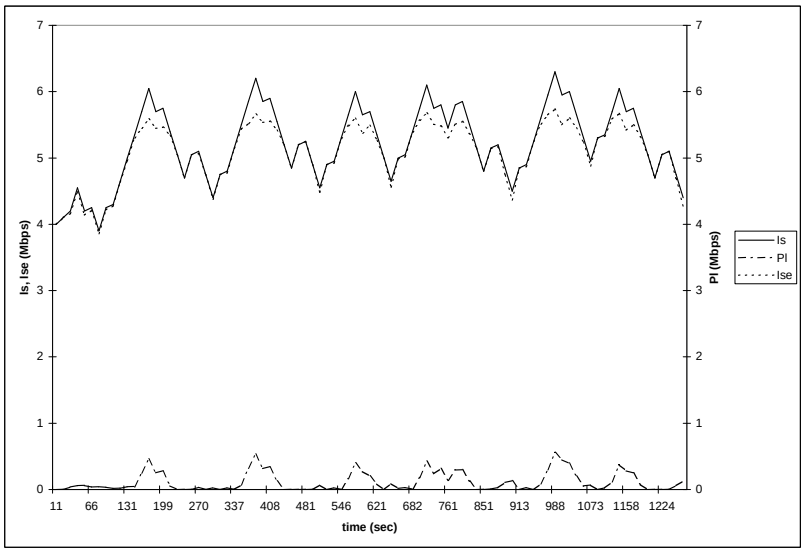

Figure 5.11: Plot of I_s, I_{se} (left axis) and P_l (right axis) vs. time with initial $I_s = 4$ Mbps and $I_{wi} = 4$ Mbps along path 1

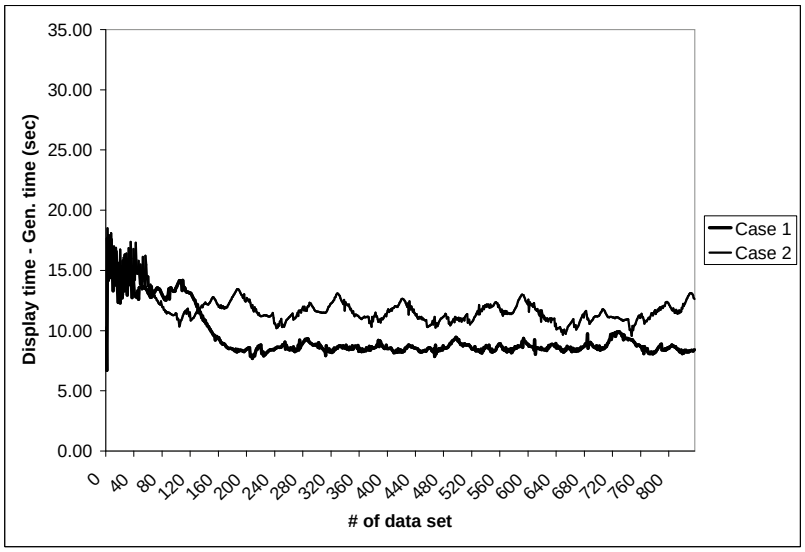

Figure 5.12: Plot of difference between the display time and generation time of a data set. Case 1, and 2 correspond to test cases with I_{wi} = 2 and 4 Mbps respectively

over the entire tree was assumed to be the worst case error ($\overline{\lambda}$) introduced per octant if an octant were lost. For the Raleigh-Taylor series data, $\overline{\lambda}$ was determined for various values of λ. Correspondingly, the average volume occupied by an octant (avg_volume) was also determined. A quality metric was then defined for the image as:

$$Q = \lambda \times avg\_volume \times octs\_received + \overline{\lambda} \times avg\_volume \times octs\_lost, \tag{5.1}$$

where $octs\_recvd$ is the number of octants received by the client, and

$octs\_lost$ is the number of octants lost.

Thus, the higher the value of Q, the lower the image quality. For the two test runs with I_{wi} = 2 and 4 Mbps, the quality of the images produced (Q_{recv}) is shown in Figures 5.13 and 5.14 respectively. The difference between the received image quality and the sent image quality (= Q_{recv} - Q_{sent}) is plotted along the right axes in the two figures. This gives an estimate of the degradation in image quality due to packet loss. The sent image quality (Q_{sent}) may be

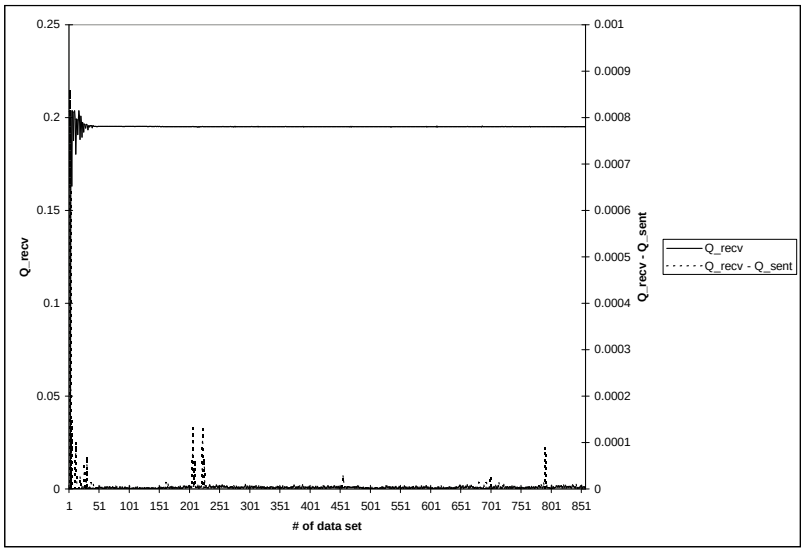

Figure 5.13: Plot of received image quality, Q_{recv} (left axis) and the difference between received and sent image qualities, Q_{recv} - Q_{sent} (right axis) corresponding to each data set with $I_{wi} = 2$ Mbps along path 1

deduced by adding the quantities plotted along the two axes. In both cases, once the system attains stability, the images are sent at a consistent quality level. It is also evident that the image is reproduced with almost the same degree of quality despite the encountered packet loss. Thus, the system tolerates packet loss without allowing a high degradation in image quality.

Appendix B contains plots showing resource usage of the system in terms of utilization of server side packet queue, client side packet fifo and octant linked list. In addition, the graphs show how the error bound, λ changes as the above mentioned simulations proceed. There is a high degree of initial fluctuation evident in all the graphs. This initial fluctuation is owed to the time that the system takes to adapt to the network conditions and accurately estimate the derivative of the curve shown in Figure 4.13. Once the derivative is found, the

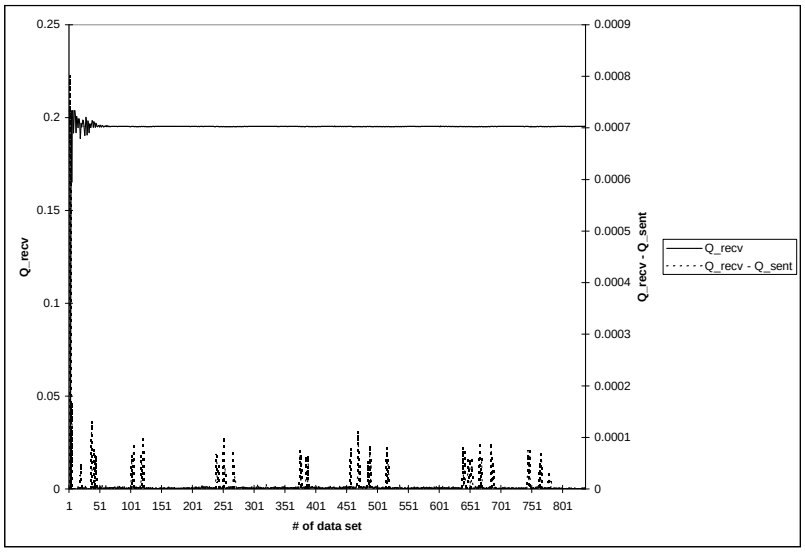

Figure 5.14: Plot of received image quality, Q_{recv} (left axis) and the difference between received and sent image qualities, Q_{recv} - Q_{sent} (right axis) corresponding to each data set with $I_{wi} = 4$ Mbps along path 1

rate control mechanism becomes effective in keeping λ considerably stable, yet adaptive to changing sending rate. The resource usage for the system is found to be low for both test cases.

Figure 5.15 shows the mesh as viewed at the client end. Here, $\lambda = 1.2$ and the number of octants is 141433. Figure 5.16 shows the mesh at a later time with much lower resolution (4096 octants) obtained with an error bound of 2.102497. As expected, the degree of compression is found to substantially affect the quality of the viewed image in terms of its resolution.

Thus, the system is found to be adaptive to changing network conditions and attempts to obtain the maximum possible effective throughput while minimizing packet loss.

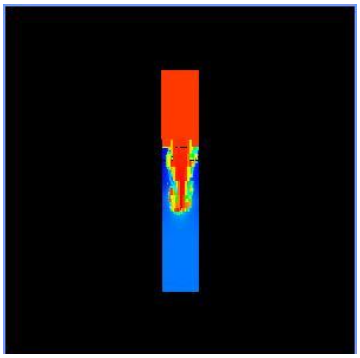

Figure 5.15: Image of 2239488 point Raleigh-Taylor simulation data with 141433 octants and $\lambda = 1.2$

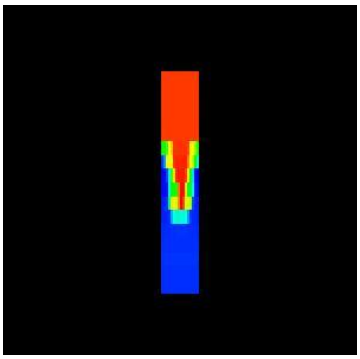

Figure 5.16: Image of 2239488 point Raleigh-Taylor simulation data with 4096 octants and $\lambda = 2.102497$

Chapter 6

Summary and Future Work

This thesis presents a remote visualization system for real-time viewing of scientific data. Such a system can be highly useful for viewing simulations of complex physical phenomena.

The system is developed as a client server application. Both, server and client are multi-threaded. The server resides on a multi-processor machine while the client may be deployed on a high performance desktop machine.

The server produces simulation data using mesh-based PDE solution software. For every time step, data is spatially distributed into an octree. The octree allows for efficient storage of data and enables easy updation and retrieval. Polynomial compression is performed on the data in each octant, and the compressed data is sent to the client for visualization. As time proceeds, various regions of the mesh change and the octree gets updated. The updated octants are compressed and sent to the client for every time step. Thus, the server is comprised of a PDE solution software, a compression module, and a communication entity. Once the data reaches the client, it is de-compressed and rendered. The server and client communication takes place through their respective communication entities. In addition to

the communication block, the client has a de-compression module and a visualization entity.

In order to maintain a regular sequence of images, the system has buffers at both ends. The server maintains a packet queue so it may generate a constant flow of packets at all times. The client buffers packets in a fifo. The packets are extracted and parsed to get individual data units that are then inserted in a linked list. When data needs to be (re-)rendered, it is extracted from the linked list and inserted into an octree. The leaves of this replicated octree then contain compressed data. Once an octree is updated, the data is decompressed, inserted into the visualization data structure, and rendered.

At the base of this communication are one TCP and two UDP channels. While data is transferred over a UDP channel, the other two channels are maintained for exchanging control information. This thesis addresses issues involved in designing the communication protocol and presents a control theory based adaptive rate-control scheme for maximization of effective throughput between the two machines. The rate-control theory, proposed in this thesis, may be used by any UDP application that is tolerant to packet loss and can take advantage of available bandwidth.

The system is shown to be robust and adaptive to changing network conditions. As the rate control algorithm changes the ideal data sending rate, the compression module changes its degree of compression. The system thus takes advantage of the available bandwidth at any time, by changing the image resolution. This enables image generation with best possible quality and makes the application robust. The packet loss tolerance of the application also considerably affects its robustness. In addition, as shown by test results of the rate-control scheme in Chapter 5, the parameter β may be used to tune the results and obtain a lower packet loss. As in most real-time applications, UDP is used to avoid the overhead of re-transmissions. Further benefits are maximized by careful choice of packet sizes and packet formats.

6.1 Future Work

An intuitive direction for future work would be to explore the effects of the rate control scheme on other traffic streams - both UDP and TCP. While it was observed that the controlled stream does not excessively increase the loss experienced by another co-existing UDP stream, its effect on TCP streams has not been studied.

In addition, it would be interesting to observe if the rate-control is effective for multiple co-existent UDP streams that use the same rate control algorithm. Some preliminary tests were performed to this effect. It was found that if two such controlled UDP streams co-exist, one of them follows the desired pattern of sending rate, while the other's sending rate drops down to zero. Therefore, there probably is a need for signalling between the two sources. The rate control algorithm may then be modified so as to appropriately react to the situation.

During the development of the visualization system, it was found that it can greatly benefit from a fast and efficient rendering mechanism. This would reduce the processing burden on the client. Thus, a probable direction for future work would be to explore alternate methods of data rendering, including direct rendering without the use of a specialized software like the Visualization Toolkit that is presently used.

Bibliography

[1] G.L.Miller, D.Talmor, and S.H.Teng, "Optimal coarsening of unstructured meshes," *J. Algorithms*, vol. 31, pp. 29–65, April 1999.

[2] "ASCI Flash Center: Research." Website: http://flash.uchicago.edu/research.

[3] S. Lee, N. Piersol, W. Kalata, C. L. Skelly, M. A. Curi, P. F.Fischer, F. Loth, and L. B.Schwartz, "Numerical simulation of vein graft hemodynamics," in *Proceedings of the 2001 BioEngineering Conference*, pp. 733–4, June 2001. BED-Vol 50.

[4] P. Behling, "Supercomputing-based visualization systems used for analyzing output data of a numerical weather prediction model," in *Proceedings, 1990 International Conference on Supercomputing*, (Amsterdam), pp. 291–295, June 1990.

[5] a. E. S. Committee on Physical, Mathematical, "Grand Challenges 1993: High Performance Computing and Communications," Supplement the President's Fiscal Year 1993 Budget FY 1993 U.S. Research and Development Program, Office of Science and Technology Policy, 1993.

[6] T. Sterling, paul Mesina, and P. H. Smith, *Enabling Technologies for Petaflops Computing*. Cambridge, Massachusetts: The MIT Press.

[7] S. K. Graffunder, "Barrier-breaking performance for industrial problems on the CRAY C916," in *Proceedings of the Conference on Supercomputing '93*, pp. 516–519.

[8] J. Wells, V. Oberacker, A. Umar, C. Bottcher, M. Strayer, J. Drake, and R. Flanery, "The quantum structure of matter grand challenge project: Large-scale 3-D solutions in relativistic quantum dynamics," in *Proceedings of the Conference on Supercomputing '93*, pp. 44–53.

[9] E. Barczcz, "One year with an iPSC/860," in *Proceedings of the COMPCON Spring'91 Conference*, IEEE Computer Society Press, p. 213.

[10] E. Elmroth, C. Ding, Y.-S. Wu, and K. Pruess, "A parallel implementation of the TOUGH2 software package for large scale multiphase fluid and heat flow simulations," in *Proceedings of Supercomputing'99*.

[11] K. Pruess, "A general purpose numerical simulator for multiphase fluid and heat flow," Lawrence Berkley Laboratory Report LBL-29400, Lawrence Berkley Laboratory, Berkley, CA, May 1991.

[12] W. Schroeder, K. Martin, and B. Lorenson, *The Visualization Toolkit, An Object-Oriented Approach to 3D Graphics*. Upper Saddle River, New Jersey: Prentice Hall PTR, 1998.

[13] M. Woo, J. Neider, T. Davis, D. Schreiner, and O. A. R. Board, *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL, Version1.2*. Addison-Wesley Pub. Co., 1999.

[14] "AVS5." Website: http://help.avs.com/AVS5/.

[15] "SCIRun." Website: http://software.sci.utah.edu/scirun.html.

[16] "Integration of SCIRun, Genesis and Matlab." Website: http://www.sci.utah.edu/pubs/pdfs/scirun_gen_mat.pdf.

[17] "RealPlayer." Website: http://www.real.com.

[18] "Windows Media Player." Website: http://www.windowsmedia.com.

[19] J. Postel, "User Datagram Protocol," RFC 768, ISI, August 1980.

[20] J. Postel, "Transmission Control Protocol," Darpa Internet Program Protocol Specification, RFC 793, ISI, September 1981.

[21] S. Casner, H. Schulzrinne, and V. Jacobson, "RTP: A transport protocol for real-time applications," tech. rep., Network Working Group Request for Comments: 1889 Category: Standards Track Audio-Video Transport Working Group, January 1996.

[22] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[23] A. Kapdia, A. Feng, and W. chen Feng, "The effects of inter-packet spacing on the delivery of multimedia content," in *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS-2001)*, (Phoenix, Arizona), April 2001.

[24] Rahul, Balakrishnan, and Seshan, "An integrated congestion management architecture for internet hosts," Tech. Rep. MIT/LCS/TR-771, 1999.

[25] L. Freitag and R. Loy, "Comparison of remote visualization strategies for interactive exploration of large data sets," in *Proceedings 15th International Parallel and Distributed Processing Symposium*, 2001.

[26] C.-F. Chang, G. Bishop, and A. Lastra, "LDI tree: A hierarchical representation for image-based rendering," in *Siggraph 1999, Computer Graphics Proceedings* (A. Rockwood, ed.), (Los Angeles), pp. 291–298, Addison Wesley Longman, 1999.

[27] L. McMillan and G. Bishop, "Plenoptic modeling: An image-based rendering system," *Computer Graphics*, vol. 29, no. Annual Conference Series, pp. 39–46, 1995.

[28] J. W. Shade, S. J. Gortler, L.-W. He, and R. Szeliski, "Layered depth images," *Computer Graphics*, vol. 32, no. Annual Conference Series, pp. 231–242, 1998.

[29] R. Haimes, "pV3: A distributed system for large-scale unsteady cfd visualization," 1994.

[30] T. W. Crockett, "Beyond the renderer: Software architecture for parallel graphics and visualization," ICASE, 1996. Technical Report 96-75.

[31] H. Hoppe, "Progressive meshes," *Computer Graphics*, vol. 30, no. Annual Conference Series, pp. 99–108, 1996.

[32] B. Heckel, G. H. Weber, B. Hamann, and K. I. Joy, "Construction of vector field heirarchies," in *IEEE Visualization '99* (D. Ebert, M. Gross, and B. Hamann, eds.), (San Francisco), pp. 19–26, 1999.

[33] J. Roerdink and M. Westenberg, "Wavelet-based volume visualization," Technical Report IWI 98-9-06, Institute for Mathematics and Computer Science, University of Groningen, 1998.

[34] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner, "Real-time continuous level of detail rendering of height fields," *Proceedings of SIGGRAPH'96*, pp. 109–118, 1996.

[35] E. C. Lamar, B. Hamann, and K. I. Joy, "Multiresolution techniques for interactive texture-based volume visualization," in *IEEE Visualization '99* (D. Ebert, M. Gross, and B. Hamann, eds.), (San Francisco), pp. 355–362, 1999.

[36] "Early feedback on very High Speed Backbone Network (vBNS) Extremely Positive." Website: http://www.vbns.net/press/feedback.html.

[37] "Introduction to Grid Computing." Website: http://www.cs.ucsd.edu/classes/sp00/notes/fran/introweb.html.

[38] G. Cheng, Y. Lu, G. Fox, K. Mills, and T. Haupt, "An interactive remote visulaization environment for an electromagnetic scattering simulation on a high performance computing system," in *Proceedings, Supercomputing '93*, (Portland, Oregon), pp. 317–326.

[39] "Compression of scientific data," 2001. Ph.D. Proposal, Virginia Tech.

[40] N. Analysis, *B. D. Gupta*. Stosius Inc/Advent Books Division, 1989.

[41] "XDR: External Data Representation," RFC 1014, Sun Microsysyems, Inc, June 1987.

[42] "Iperf version 1.1.1," 2000. Website: http://dast.nlanr.net/Projects/Iperf1.1.1/.

[43] J. Flaherty, R. M. Loy, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz, "Adaptive local refinement with octree load balancing for the parallel solution of three-dimensional conservation laws," *Journal of Parallel and Distributed Computing*, vol. 47, no. 2, pp. 139–152, 1997.

Appendix A

Network Paths

This appendix contains results obatined from the system utility *traceroute* between the four test paths presented in Chapter 5, Table 5.1.

Path 1

Source: 128.173.52.7

Destination: 140.221.11.103

Traceroute Output:

1 localhost (10.0.0.1) 0.233 ms 0.172 ms 0.154 ms

2 tor-6509-1a.vl617.cns.vt.edu (128.173.55.254) 0.764 ms 0.685 ms 0.783 ms

3 sha-6509-1a.vl618.cns.vt.edu (128.173.2.17) 0.909 ms 1.534 ms 0.702 ms

4 bur-6509-1a.vl709.cns.vt.edu (128.173.0.73) 0.861 ms 0.875 ms 1.174 ms

5 isb-6509-1b.vl702.cns.vt.edu (128.173.0.21) 1.617 ms 1.781 ms 0.936 ms

6 isb-6006-1a.vl710.cns.vt.edu (128.173.0.82) 1.076 ms 0.888 ms 0.973 ms

7 isb-7507-2.gi1-0-0.cns.vt.edu (128.173.0.102) 1.000 ms 0.930 ms 0.858 ms

8 192.70.187.210 (192.70.187.210) 28.688 ms 28.427 ms 37.804 ms

9 192.70.138.22 (192.70.138.22) 38.909 ms 38.889 ms 44.966 ms

10 nycm-wash.abilene.ucaid.edu (198.32.8.46) 46.117 ms 48.517 ms 50.952 ms

11 clev-nycm.abilene.ucaid.edu (198.32.8.29) 57.615 ms 68.337 ms 62.766 ms

12 ipls-clev.abilene.ucaid.edu (198.32.8.25) 63.037 ms 67.333 ms 70.575 ms

13 chin-ipls.abilene.ucaid.edu (198.32.8.70) 65.477 ms 63.759 ms 62.241 ms

14 mren-chin-ge.abilene.ucaid.edu (198.32.11.98) 65.499 ms 50.518 ms 63.979 ms

15 anl-mren-gige.anchor.anl.gov (192.5.170.213) 65.496 ms 64.370 ms 79.399 ms

16 stardust-msfc-20.mcs.anl.gov (140.221.20.124) 70.234 ms 70.737 ms 77.094 ms

17 terra.mcs.anl.gov (140.221.11.103) 58.307 ms 45.991 ms 35.441 ms

Path 2

Source: 128.173.52.7

Destination: 130.203.8.38

Traceroute Output:

1 localhost (10.0.0.1) 0.237 ms 0.169 ms 0.161 ms

2 tor-6509-1a.vl617.cns.vt.edu (128.173.55.254) 1.828 ms 0.699 ms 0.675 ms

3 sha-6509-1a.vl618.cns.vt.edu (128.173.2.17) 0.825 ms 1.488 ms 0.704 ms

4 bur-6509-1b.vl709.cns.vt.edu (128.173.0.77) 0.863 ms 0.793 ms 0.722 ms

5 isb-6509-1b.vl702.cns.vt.edu (128.173.0.21) 1.295 ms 0.872 ms 0.834 ms

6 isb-6006-1a.vl710.cns.vt.edu (128.173.0.82) 2.204 ms 1.149 ms 1.130 ms

7 isb-7507-2.gi1-0-0.cns.vt.edu (128.173.0.102) 0.916 ms 0.948 ms 0.833 ms

8 192.70.187.210 (192.70.187.210) 34.052 ms 33.280 ms 45.707 ms

9 192.70.138.22 (192.70.138.22) 46.095 ms 45.369 ms 56.870 ms

10 nycm-wash.abilene.ucaid.edu (198.32.8.46) 57.814 ms 51.698 ms 46.052 ms

11 clev-nycm.abilene.ucaid.edu (198.32.8.29) 52.634 ms 38.070 ms 37.765 ms

12 abilene.psc.net (192.88.115.122) 39.795 ms 39.472 ms 49.690 ms

13 bar-beast.psc.net (192.88.115.17) 55.720 ms 57.761 ms 57.724 ms

14 psu-i2.psc.net (192.88.115.98) 50.626 ms 57.276 ms 62.307 ms

15 * * *

16 Pond-BLN-ARE1.cse.psu.edu (130.203.32.2) 51.400 ms 54.025 ms 54.370 ms

17 sumaa3d.cse.psu.edu (130.203.8.38) 40.597 ms * 61.108 ms

Path 3

Source: 140.221.11.103

Destination: 130.203.8.38

Traceroute Output:

1 stardust-msfc-11 (140.221.11.251) 0.373 ms 0.280 ms 0.331 ms

2 kiwi.anchor.anl.gov (140.221.20.97) 0.232 ms 0.198 ms 0.222 ms

3 mren-anl-gige.anchor.anl.gov (192.5.170.214) 0.924 ms 0.781 ms 0.811 ms

4 chin-mren-ge.abilene.ucaid.edu (198.32.11.97) 0.924 ms 0.751 ms 0.812 ms

5 ipls-chin.abilene.ucaid.edu (198.32.8.69) 4.679 ms 4.540 ms 4.566 ms

6 clev-ipls.abilene.ucaid.edu (198.32.8.26) 10.891 ms 10.869 ms 10.895 ms

7 abilene.psc.net (192.88.115.122) 14.063 ms 14.034 ms 14.071 ms

8 bar-beast.psc.net (192.88.115.17) 14.286 ms 14.271 ms 14.180 ms

9 psu-i2.psc.net (192.88.115.98) 18.047 ms 18.610 ms 18.398 ms

10 * * *

11 Pond-BLN-ARE1.cse.psu.edu (130.203.32.2) 19.999 ms 19.629 ms 19.924 ms

12 sumaa3d.cse.psu.edu (130.203.8.38) 19.696 ms * 18.979 ms

Path 4

Source: 128.173.52.7

Destination: 66.37.66.32

Traceroute Output:

1 localhost (10.0.0.1) 0.981 ms 5.774 ms 0.161 ms

2 tor-6509-1a.vl617.cns.vt.edu (128.173.55.254) 0.763 ms 0.689 ms 0.686 ms

3 sha-6509-1a.vl618.cns.vt.edu (128.173.2.17) 0.915 ms 0.758 ms 1.936 ms

4 cas-6509-1b.vl706.cns.vt.edu (128.173.0.53) 0.899 ms 0.928 ms 0.853 ms

5 isb-6509-1a.vl701.cns.vt.edu (128.173.0.9) 1.142 ms 1.070 ms 0.880 ms

6 isb-6006-1a.vl710.cns.vt.edu (128.173.0.82) 2.200 ms 0.961 ms 0.837 ms

7 isb-7507-1.gi1-0-0.cns.vt.edu (128.173.0.94) 1.003 ms 0.924 ms 0.845 ms

8 atm1-0.11.roa.networkvirginia.net (192.70.187.194) 2.791 ms 2.746 ms 31.055 ms

9 65.162.89.41 (65.162.89.41) 18.442 ms 8.028 ms 7.214 ms

10 65.162.89.33 (65.162.89.33) 7.196 ms 8.548 ms 8.306 ms

11 sl-gw21-rly-3-0.sprintlink.net (160.81.98.29) 8.577 ms 7.707 ms 7.799 ms

12 sl-bb21-rly-3-0.sprintlink.net (144.232.14.69) 8.079 ms 9.173 ms 8.080 ms

13 sl-bb20-atl-10-1.sprintlink.net (144.232.9.198) 21.886 ms 21.659 ms 21.547 ms

14 sl-gw10-atl-0-0-0.sprintlink.net (144.232.12.233) 21.824 ms 21.627 ms 22.313 ms

15 sl-smidat-5-0.sprintlink.net (144.232.244.10) 32.880 ms 32.737 ms 34.658 ms

16 207.42.149.126 (207.42.149.126) 44.642 ms 47.282 ms 41.156 ms

17 ct19.swva.net (66.37.69.19) 43.348 ms 48.791 ms 44.546 ms

18 packplace.swva.net (66.37.64.58) 91.121 ms 131.798 ms 48.974 ms

19 66.37.66.32 (66.37.66.32) 128.635 ms 45.826 ms 54.678 ms

Appendix B

System Resource Usage

This appendix contains graphs showing the system resource usage for two test runs along path 1 (Table 5.1) with $I_{wi} = 2$, and 4 Mbps.

The first set of graphs correspond to the test run with $I_{wi} = 2$ Mbps. Figure B.1 shows the change in error bound, λ, and utilization of packet queue at the server end. Figure B.2 displays the ratio between number of lost octants and number of octants generated for each data set. Corresponding λ is also plotted along the left axis of the graph. The resource utilizations of client packet fifo and octant linked list are shown in Figure B.3.

The same set of plots, corresponding to a test run with $I_{wi} = 4$ Mbps, are presented in Figures B.4, B.5, and B.6. It may be noted that in both cases, the system attains stability and the resource utilization remains low.

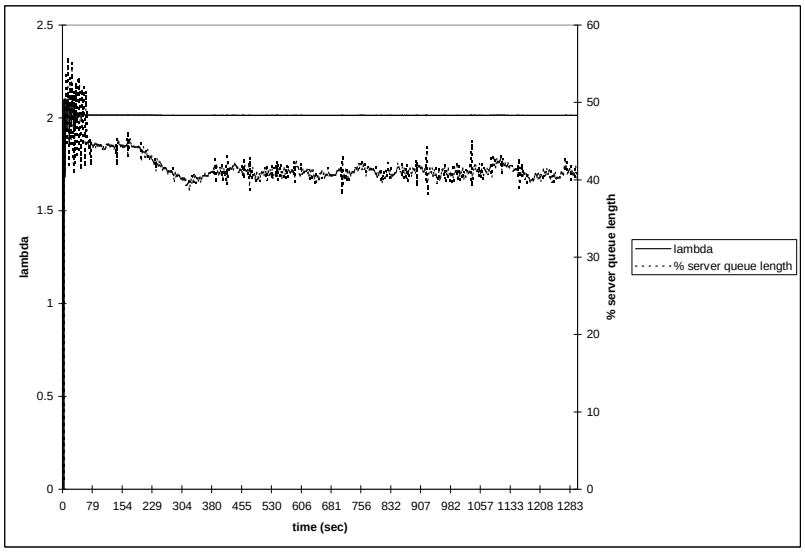

Figure B.1: Plot of λ (left axis) and server packet queue utilization (right axis) vs. time with $I_{wi} = 2$ Mbps along path 1

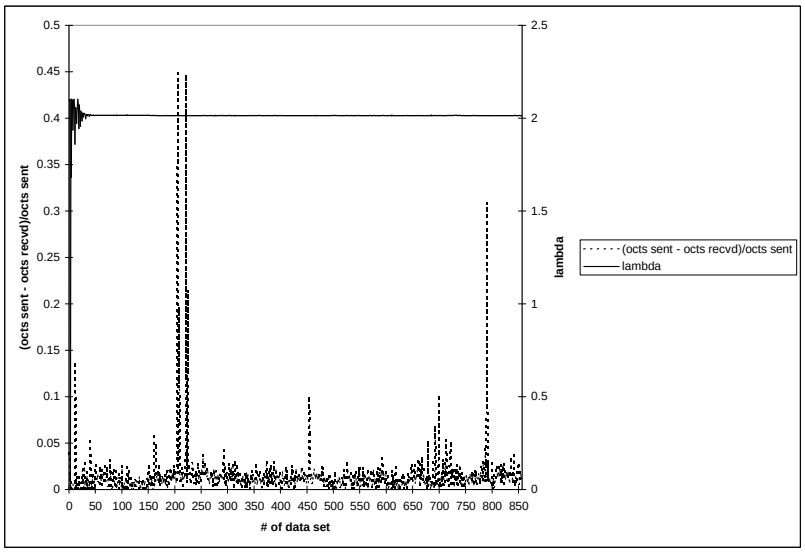

Figure B.2: Plot of ratio between lost octants and octants sent (left axis) and λ (right axis) vs. number of data set with $I_{wi} = 2$ Mbps along path 1

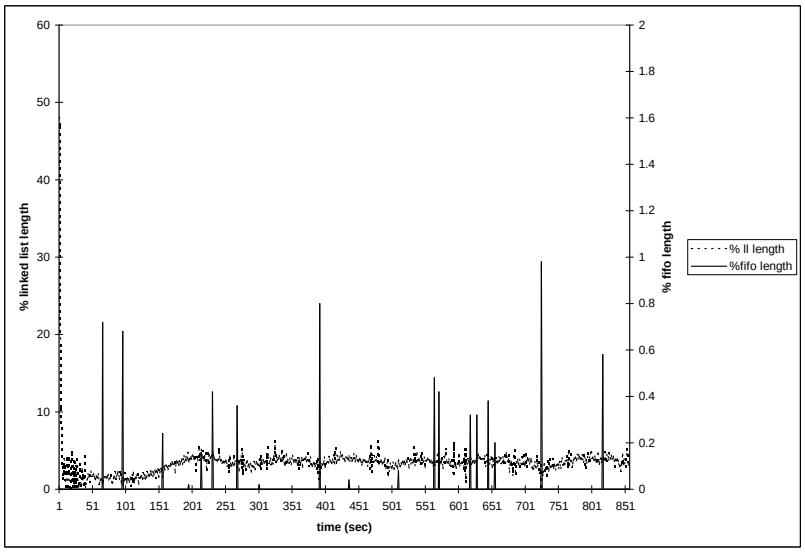

Figure B.3: Plot of client octant linked list (left axis) and fifo length (right axis) utilizations vs. time with $I_{wi} = 2$ Mbps along path 1

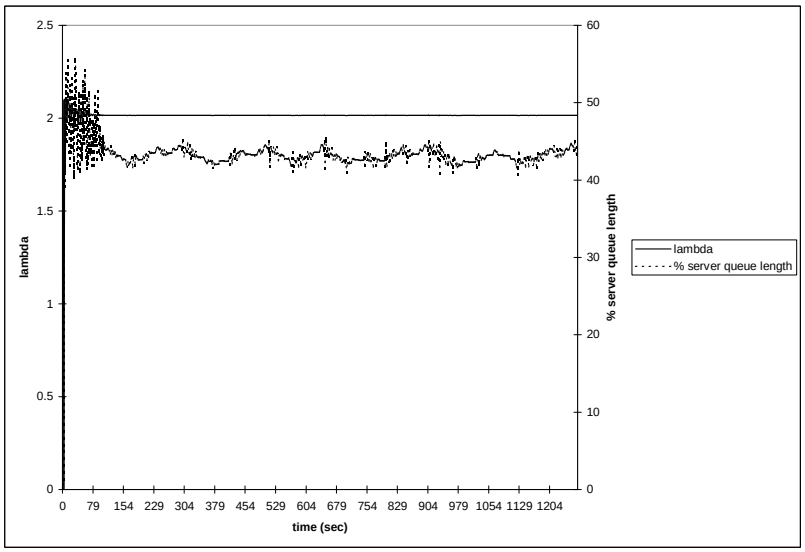

Figure B.4: Plot of λ (left axis) and server packet queue utilization (right axis) vs. time with $I_{wi} = 4$ Mbps along path 1

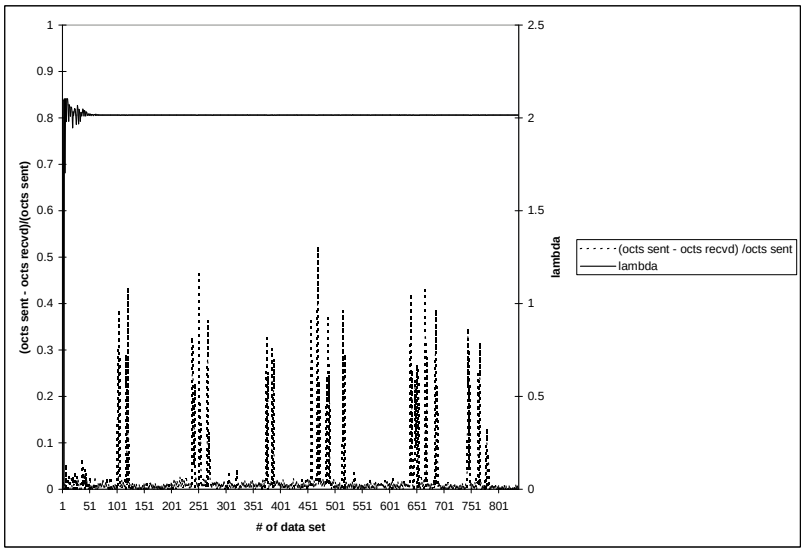

Figure B.5: Plot of ratio between lost octants and octants sent (left axis) and λ (right axis) vs. number of data set with $I_{wi} = 4$ Mbps along path 1

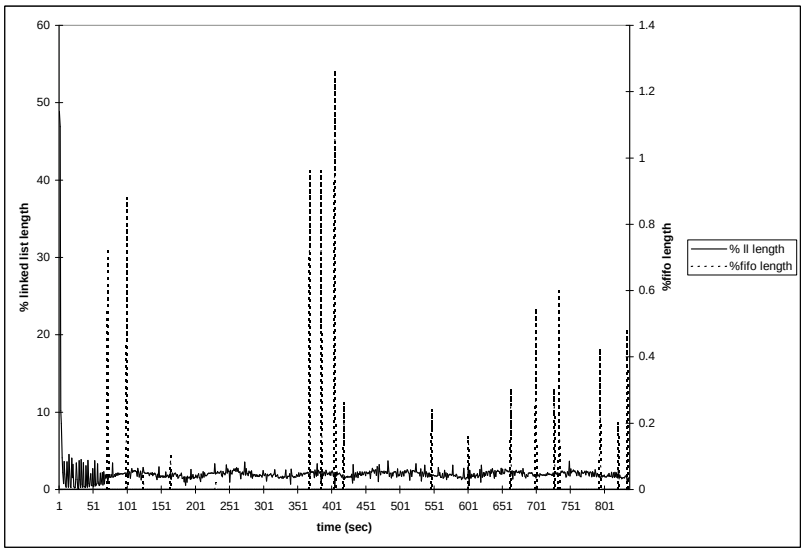

Figure B.6: Plot of client octant linked list (left axis) and fifo length (right axis) utilizations vs. time with $I_{wi} = 4$ Mbps along path 1

Vita

Mukta Nandwani was born and brought up in New Delhi, India. She joined Jamia Millia Islamia, Delhi for her technical education in electrical engineering in 1995. After completion of Bachelor of Technology in 1999, she joined Mascot Systems Limited, Chennai, where she worked as an Associate Consultant for an year. She joined Virginia Tech in 2000 for her Masters Degree in electrical engineering. In the summer of 2001, she worked as a Guest Graduate Researcher at Argonne National Labs, Illinois. Since then she has been involved in computer networking research. Mukta's hobbies are reading and music. Her technical interests include computer networks and software development.