## Abstract

CHANDRASEKHAR, VINAY. A Framework for Quality of Service Analysis of IP Based Video Networks (Under the direction of Dr. Mladen A. Vouk).

Applications that use real time video are becoming increasingly necessary for effective communication over the Internet. Their popularity is increasing in areas such as distance learning, distributed research and video conferencing. Since real time video has special Quality of Service (QoS) requirements for it to be acceptable to end users, administrators are faced with challenges that involve the end-to-end ability of participating systems to support real time video communication. Unfortunately, the tools that exist in the market today are either expensive and closed source, or are generic and do not explicitly consider the characteristics of real time video.

This work proposes a framework for assessment of end-to-end Quality of Service capabilities for support of real-time Variable Bit Rate (VBR) compressed video communication. This framework evaluates the endpoints and their inter-connectivity and determines the extent of their ability to support compressed video over User Datagram Protocol (UDP). The framework generates VBR traffic that mimics compressed low bit rate video, maintains and reports detailed and summary statistics of each test. User configurable options include specifying activity levels and bandwidth upper bounds. The framework itself is modular and extensible, allowing for functionalities to be added or replaced as the testing requirements change.

The framework defines a methodology to conduct video QoS tests, and describes how the test cycles are to be conducted for determining the Quality of Service support. Various metrics that are indicative of the Quality of Service for real time video are listed and described. Traffic generation and transmission requirements for mimicking video traffic are explained. At each participating end point, the values of the various QoS metrics are maintained in real time using specific Simple Network Management Protocol (SNMP) Management Information Bases (MIBs) designed for use with this framework.

A tool is implemented based on this framework, and its performance as a video QoS measurement tool is studied. The tool uses an Iperf based traffic generation module, that reads from trace files in order to generate test traffic. The trace files contain datagram size and timing information required to generate video-like VBR traffic. An SNMP MIB implementation is provided to record the QoS metrics determined during the tests. A separate monitoring program queries the endpoint MIBs, tabulates and displays the QoS information. This information can then be used to determine end-to-end support.

# A FRAMEWORK FOR QUALITY OF SERVICE ANALYSIS OF IP BASED VIDEO NETWORKS

BY
VINAY CHANDRASEKHAR
DEPARTMENT OF COMPUTER SCIENCE
NORTH CAROLINA STATE UNIVERSITY
RALEIGH, NC 27695

A THESIS SUBMITTED TO THE GRADUATE FACULTY OF
NORTH CAROLINA STATE UNIVERSITY
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

RALEIGH
JULY 2003

APPROVED BY:

DR. DAVID THUENTE

DR. KHALED HARFOUSH

DR. MLADEN A. VOUK
CHAIR OF ADVISORY COMMITTEE

# Biography

Vinay Chandrasekhar was born in Mysore, India, the city of royal splendor. After completing his Bachelor of Engineering degree in Electronics and Communications at S.J.C.E, Mysore, he worked with Mascot Systems, Bangalore as a systems analyst and software developer for about three years. He then joined the master's program in Computer Science at North Carolina State University.

While doing his gradaute studies, he initially worked as a teaching assistant, and later started his research work as part of the end-to-end video quality analysis project with NCNI, under the guidance of Dr. Mladen Vouk. During this time, he also assisted Dr. Vouk in other assignments involving the setup and maintenance of various video endpoints, equipment and networks.

# Acknowledgments

I am fortunate to have had the opportunity to work with Dr. Mladen Vouk. His passion for knowledge and his hard work are an inspiration. I thank him for his guidance and support. I thank Dr. David Thuente and Dr. Khaled Harfoush for their suggestions and feedback.

I would like to express my gratitude to John Streck and Tyler Johnson for their valuable comments and advice at various stages of my research. It was a learning experience working with Wesley and Dr. Ketan Mayer-Patel of UNC, Chapel Hill. I thank the NCNI folks for their feedback on this work. I appreciate the help of Michael Bugaev for his prompt assistance with the hardware required for the research. I also thank Marhn Fullmer and the networks staff at NC State for their help. My thanks to Sandeep, and to my sister, for reviewing this document.

I wish to sincerely thank my parents and my sister for their unending love, support and encouragement in all my endeavors. Finally, I am very grateful to be blessed with true friends who are always there for me.

This work has been supported in part by the North Carolina Networking Initiative.

# Table of Contents

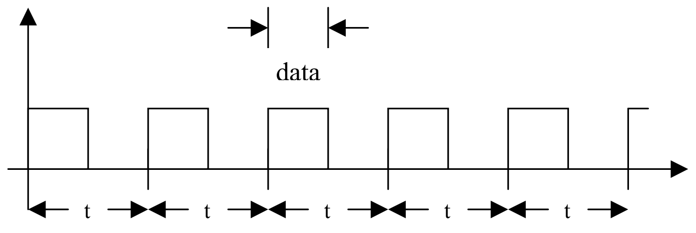
# List of Figures

# List of Tables

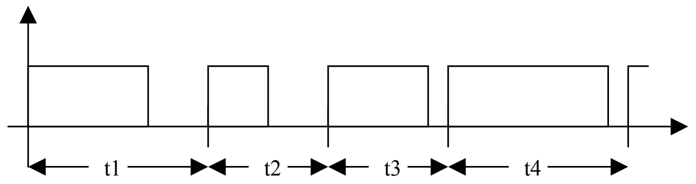
## Chapter 1

# Introduction

When multimedia is chosen to assist real time exchange of information, it enhances the overall user experience and adds significant value to the communication. With advances in multimedia sensitive networking, the strength of real time multimedia communications, such as in video conferencing and in distance education, are reaching new heights. Many of today's human-interactive communication solutions support video. For real time video communication over a packet network, video is digitized, often compressed, and then sent as packetized chunks of information. In IP networks, the transport layer protocol of choice is UDP [1]. Real time Transport Protocol (RTP) [2] is an Internet Engineering Task Force (IETF) protocol that provides end-to-end functionality for transport of real time audio and video data, and has been widely accepted. Sources transmitting video information can have either a Constant Bit Rate (CBR) or a Variable Bit Rate (VBR). CBR sources generate traffic at a constant bandwidth, where as VBR sources inject data into the network with a variable instantaneous bandwidth, as shown in Figure 1.1. CBR is the mode of choice for Voice over IP solutions because of the way voice is sampled and coded in real time. For compressed real time video, VBR transmission is preferred to CBR because it can ensure constant video quality over time, while conserving instantaneous bandwidth [46]. Two



(i) CBR: Amount of data sent over time is constant



(ii) VBR: Amount of data sent varies over time

**Figure 1.1**: CBR sources vs. VBR sources

popular series for compression of digitized video are the MPEG-x series by the Moving Picture Experts Group of the International Organization for Standardization (ISO), and the H.26x series by the International Telecommunications Union - Telecom (ITU-T). All these standards define different types of video units called frames that contain information about the video. For VBR video, the sizes of the frames and the time between datagrams vary depending on the video information they contain.

An acceptable video experience imposes certain constraints on the factors affecting video communication. These factors include delay, delay variation known as jitter, and packet and information loss. Given these restrictions, it is useful to be able to determine the extent to which video capable endpoints can support video communication. Network and system administrators often face the above task of evaluating end-to-end support for real time video communication. For example, an administrator has a new requirement that needs video communication between the two end-points, or is trying to debug an existing video communication setup between two end-points that does not appear to work well. In either case, a system that enables the administrator to actively inject video-like data into the network, collect and analyze the end-to-end quality metrics, is needed. The quality assessment needs to encompass both endpoint readiness for video, and the support of the underlying network. This support for video can be also tested and categorized for different types of video, in terms of video characteristics such as resolution and encoding method.

## 1.1 Existing Tools

**Videoconferencing tools:** Solutions from Polycom [47] and VCon [50] are some of the popular videoconferencing products. The Polycom ViaVideo desktop videoconferencing product release 2.2 reports packet loss, bandwidth utilization, frame rates and jitter values for ongoing video communication sessions. However, the values are instantaneous readings, and there is no mechanism to access or record the readings, or for determining average values. In addition, the load on the end points are not reported. VCon Vigo v4.51, another desktop solution, does not provide detailed diagnostic information about ongoing or previous video sessions. It has a LAN conference state monitor that displays the video and audio bandwidth allocation information, and the video and audio codecs being used by the incoming and outgoing signals for the ongoing call. These products are expensive, and they do not provide any Application Programming Interfaces (APIs) to access statistical information about ongoing or previous video sessions, and thus cannot be considered as comprehensive testing tools.

**Network performance tools:** Open Source tools such as Iperf [38], on the other hand, can be used to measure network parameters that affect video QoS such as UDP datagram losses and jitter. Iperf provides both periodic and summary reports of each test. But Iperf is only capable of CBR transmission. Also, this would not form a complete solution since end point support remains untested. Iperf is explained in more detail in Section 2.4. NetSpec [51], an Open Source network experimentation and performance measurement application, concentrates on network testing as opposed to end-to-end testing. It provides a wide range of testing categories, by categorizing the different traffic types. The emulated traffic models include voice, MPEG and video teleconference traffic streams. The user can specify video traffic characteristics such as frame rate and frame size. Individual sizes are then generated by using a gamma distribution [52]. The first drawback of NetSpec is that it has been designed mainly for Asynchronous Transfer Mode (ATM) networks, and all calculations and sizes are calculated based on ATM cells. Also, it focuses just on the network, rather than end-to-end. Finally, it does not monitor the various QoS metrics required to evaluate video performance.

**Delta Videoconferencing Emulator:** Delta [55] Protocol Test Solutions provides a videoconferencing test system to emulate and analyze various real time videoconferencing protocols, including H.263 [8] analysis. It is ideally used to troubleshoot network and interoperability problems by having the test tool communicate with an existing videoconferencing endpoint and debug the communication stream. The test tool is fully standards compliant and can code and decode all standard videoconferencing protocols. It can also provide test sequences and send multiplexed bit-streams containing audio, video and data to the remote endpoint. The metrics for video tests include RTP and RTCP [2] packet counts, bandwidth, jitter and picture rates, and total test data count. But the emulator is expensive and closed source. The endpoint being tested must already have a valid videoconferencing protocol stack itself. Finally, it does not measure host resource usage along with the network measurements that it provides.

**VideNet Scout and Chariot:** VideNet Scout [53] is a web based distributed network performance analyzer for video and voice over IP, developed and maintained by VideNet. It is an enhancement to an existing closed source solution called Chariot from NetIQ Corporation [54]. Scout is currently being used to analyze end-to-end video and voice capabilities, and in turn determine video conferencing support. The range of testing capabilities provided by Scout include bandwidth probes, port scan, and video and audio stream tests at different rates. Test streams may be bidirectional and asymmetric, and many test streams may be run in parallel. Tests between remote endpoint pairs are configured and run using console software, which may reside elsewhere. Tests are script based, and the test administrator can specify when and how the tests are to be initiated, number of tests to run and at what rate the information is recorded. The results of each test are archived and hosted on a web server. Various flavors of Unix, Linux and Windows Operating Systems are supported. The enhancements that VideNet Scout provides to the Chariot tool include web interfacing, a library of predefined performance tests, test scheduling and result archiving services. In order to test with the VideNet Scout, users begin by downloading and installing endpoint software. The users then register their endpoints with VideNet Scout using a web based registration form. The required tests are executed, and the results are displayed on the Scout web page [53]. The metrics that are collected during the tests include maximum throughput achieved, lost and duplicate datagrams, maximum, minimum and average jitter and endpoint CPU utilization. A summary of the entire test, as well as data for each iteration is displayed. For video tests, the user can specify the rate at which to send test traffic. The test traffic however, is sent at a constant bit rate, and the packet sizes are equal. Packet size has been determined by averaging packet sizes found in actual video traffic. Different activity levels, picture resolutions and varying traffic rates are thus not supported in Scout. Also, since Chariot itself is expensive and closed source, enhancements to Scout are difficult.

Thus, the existing tools are either closed source and expensive solutions, or are not comprehensive in their reporting of the various metrics that describe end-to-end QoS. This motivated the design of an open source testing solution that generates video-like test traffic, records and reports parameters that affect video QoS at the application, operating system and network levels.

## 1.2 Goals

The video QoS framework is designed to give the user an opportunity to test the extent of end-to-end video support for different video attributes, and to be modular so that components can interact without being closely bound. This work achieves the following objectives:

- Identify the various metrics at the application, Operating System and network levels, that contribute to end-to-end QoS in video communication
- Design and evaluate a traffic generator that generates video-like VBR traffic
- Design and evaluate an SNMP MIB to record and maintain QoS information locally at the end points
- Design and evaluate a monitor to query the end points, retrieve QoS data, organize and display the data to the user

- Make the implementation easy to distribute, setup and execute.

Initially, the goal was to study the characteristics of compressed real time video by varying attributes such as maximum encoding rate and picture resolution. These experiments were conducted for different kinds of video content, some with more motion in the video than others. The video content was then categorized based on the amount of motion and frame changes in the video content. This information was used to design a framework that would mimic video traffic. The effects of transmission of this video like data between the end points were analyzed. SNMP MIB storage mechanisms for QoS information, as well as procedures for conducting tests with the framework were defined. Finally, an Iperf [38] based implementation was built based on this framework, and the results analyzed.

## 1.3 Current Research

Many research communities and organizations are focusing on QoS initiatives for real-time video communication today. End-to-end QoS mechanisms that involve both the network and host level metrics have been stressed in earlier works such as [25][20].

The Internet2 community is actively involved in the fields of end-to-end performance and digital video measurements [56][21]. One initiative of the Internet2 community is Internet Commons, which provides services for collaborative research, and is currently focused on services related to videoconferencing. This effort is also supported by VideNet. VideNet currently uses the Scout system for video QoS measurement, as explained in Section 1.1. A survey [22] by the Internet2 QoS working group illustrates the several requirements that video imposes on both the network and the end host resources. It also discusses current efforts to standardize measurement and quality analysis techniques for video and audio.

Sharif and Chen [19] describe a set of experiments conducted over the Internet2 high performance network and discusses the end-to-end QoS requirements as well as the results.

Other works [12] also describe both the QoS metrics and mechanisms for QoS control in networks, as well as findings of video experiments conducted over the public Internet. For example, Wu and Hou [24] describe the QoS problems faced in real time video communication over the Internet, and propose control mechanisms to address the problems. The works by Loguinov and Radha [26][27] describe the results of video traffic experiments conducted over the Internet, and describe the different metrics studied and measured for the flow.

Another area in the research of video quality is perceptual or visual quality and measurement. Organizations such as the Video Quality Experts Group [57] strive to develop standards and recommendations for objective perceptual video quality measurements, based on video attributes such as blurring, tiling, and distortion. Wang and Bovik [23] propose a universal image quality index that compares two pictures and outputs a quality index between 0 and 1 that indicates degradation. A system that compares two motion sequences and provides a qualitative analysis similar to [23] is available with the Video Quality Experts Group [57].

The framework described in this work deals with quantitative metrics that indicate network and host loads that affect the video communication.

## 1.4 Thesis Layout

Chapter Two describes the various technologies that are involved in the design of this framework. Tools which have played a part either directly or indirectly in the development of the framework are also described. Chapter Three deals with the architectural aspects of the framework, and describes the different components that comprise the framework. Chapter Four presents an implementation based on the framework architecture described in Chapter Three. Some tools used in the implementation are replaceable with other tools that accomplish the same task. Chapter Five is based on the implementation of the framework as described in Chapter Four, and provides an analysis of the implementation and its components. The results of the tests performed with the implementation are also studied. Chapter Six concludes by highlighting the usefulness of the framework in analysing video QoS capabilities, and enumerates the enhancements to the framework and suggests improvements to the existing design. A glossary of the acronyms used in this work are listed in Appendix A. The SNMP MIB that has been defined for use with the framework is given in Appendix B.

Chapter 2

# Technologies and Tools

This Chapter describes the various tools and technologies that have played a part in the design of this framework. The sections on RTP and SNMP explain the respective protocols and their functions. The section on video compression describes how video sequences are broken down and compressed. The last section illustrates the use of Iperf as a traffic generation tool, and lists the protocol analyzer programs used in this work.

## 2.1 RTP

Real time Transport Protocol (RTP) [2] is a protocol useful for transporting video, voice and any other real-time data at an application level, over the underlying transport and network layers. It does not guarantee QoS, and provides only minimal control functionality to support the flow of data. It supports transfer of data to multiple recipients if the underlying network is multicast enabled. Though the protocol is designed to work independent of the underlying transport and network protocol, it is dependent on the transport protocol to fulfill RTP's multiplexing and checksum requirements. RTP is typically run over the UDP layer, and carries real time data. This protocol is used in conjunction with the Real time

Transport Control Protocol (RTCP) for monitoring data delivery, as well as conveying user information pertaining to a session. Therefore two UDP ports are used, one to transmit the data and the other for the RTCP control traffic. If both audio and video are used in a conference, then two different UDP port pairs are used, one pair for video, and the other for audio. Video transmission is broken into chunks of video data based on the size and fragment limitations of the underlying medium, as well as logical block boundaries of the video content. For every chunk of video data, an RTP header is generated and sent, enclosed inside a UDP datagram. Several RTP packets may also be carried in one lower layer protocol data unit using a framing mechanism. This mechanism itself is outside the scope of RTP. The RTP header includes information that is necessary to identify the source, as well as datagram information required to reconstruct the video. Specifically, the datagram information consists of a sequence number for datagram ordering, a timestamp, and a source identifier. In addition, there are fields within the header that have not been defined. These fields are for the use of the application [3] that runs over RTP, and can have any interpretation. This particular quality makes the protocol flexible and application-friendly, and in turn cannot be isolated as a separate layer.

## 2.2 SNMP

The Simple Network Management Protocol [9][10] is a widely accepted protocol for network management. SNMP provides the ability to query disparate devices in a network and obtain network management information from them. The management architecture consists of an extensible framework where new information can be added to an existing information base. There are three components that make up the SNMP protocol architecture. They are the SNMP agent, the network manager, and the Management Information Base.

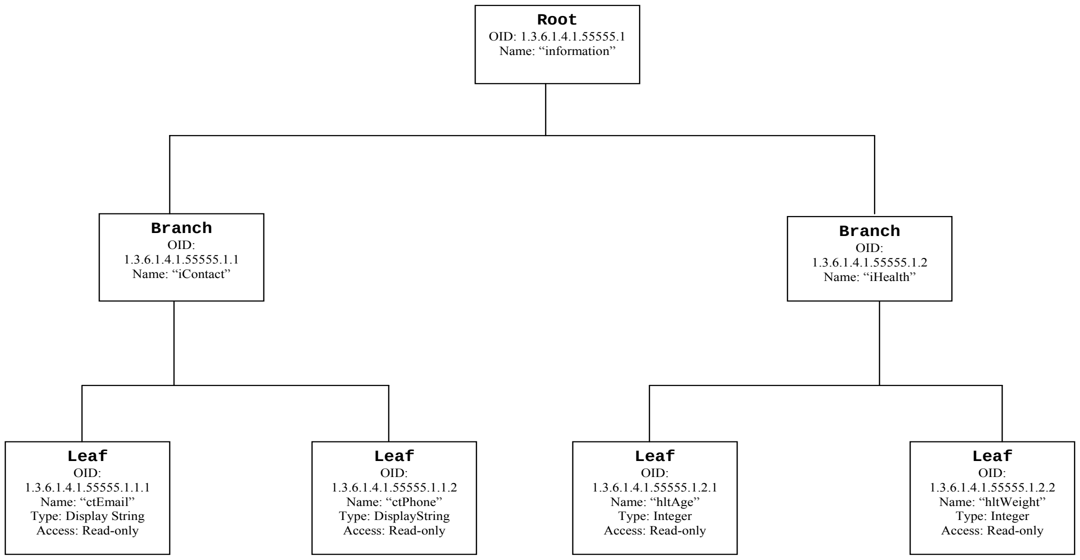


**Figure 2.1**: An example MIB tree hierarchy

The SNMP Agent maintains an online database of management information, and receives and processes requests for this information. The Network Manager, on the other hand, is capable of querying SNMP agents for information, and setting information that the agents maintain. The Management Information Base (MIB) defines a hierarchical collection of variables called objects and their properties that are required to represent and manage a particular device on the network. The MIB itself is not a database. Rather, it defines the way this database looks. The database is then implemented based on these definitions, and in turn used by the agent to maintain the actual values corresponding to the device. Each object in the Management Information Base is defined with an Object Identifier (OID), a unique name that identifies the variable, the data type that this variable represents, and its access mode (read only, read and write, no access and so on). These objects are organized into a hierarchical tree format, where each subtree represents a particular group, and the leaves of the trees represent individual objects. This is illustrated by an example in Figure 2.1. Companies or Organizations may request a unique OID for their use from the Internet Assigned Numbers Authority (IANA). The IANA maintains a web page [45] that lists the currently allocated OIDs.

Object definitions are based on the Abstract Syntax Notation (ASN.1) [13] standards. An example of an object defined using ASN.1 is shown below.

```
Object Name:  ctEmail
OID:          vqContact.2
Object Type:  DisplayString
Access Mode:  read-only
Status:       current
Description:  This object is used to display the email of
              the group contact
```

Either the Object name or the OID could be used to query the agent about this particular object. A dotted notation is used for both the object name and the OID. The above object can be denoted as `iso.org.dod.internet.private.enterprises.vidqos .vqContact.ctEmail` which specifies the hierarchical location of this object starting at the root, which is "iso". This object can also be referred by its OID as `1.3.6.1.4.1 .55555.1.2`, assuming vidqos is attached to the enterprises subtree at position 55555. The object type specifies the data type of the value defined by this object. In this example, the object type is a `DisplayString` of zero or more octets, the value of each octet being between 0 and 255. The access mode indicates how the object appears to the external world. In the example, network managers may just read the value contained in the object, but may not set or update it. The status of the object indicates whether the object is currently available within an agent implementation. Finally, the description field just a string that describes the object and its functionality.

The SNMP protocol consists of three basic commands: get, set and trap. Except for the trap service, the protocol follows a request response mechanism, where the network manager sends a request to the SNMP agent, and the SNMP agents responds to the request. A set command is used to set or update values of the objects maintained by an agent. A get command is used to obtain information contained within the objects, via the agent. Traps are generated by the agent itself without a request from a management station, and is used to indicate to the manager an alert of a special event. Traps are outside the scope of this discussion. There also exist other helper commands like getNext and getBulk that assist in getting more than one value. Since the only functions that are possible with the protocol are setting values and getting values, the protocol is termed “simple”.

## 2.3 Video Compression

Uncompressed digital video requires lots of memory space, and in turn requires large bandwidth for transmission. For example, in a simple experiment, we found that recording 5 seconds of NTSC video with a resolution of 640 x 480 at 30 frames per second required about 500 MB of hard disk space. Thus, transmission of uncompressed video is currently not a feasible solution for video conferencing and other applications that need real time video transmission over existing IP networks. This problem is overcome by exploiting redundancies in the digital video content. This exploitation however, results in some loss of information content in the resulting video that cannot be recovered. Video information is redundant in two aspects. Spatial redundancy is based on the fact that pixels considered in a very small area of a frame all contain almost the same information. Temporal redundancy, on the other hand, occurs because two successive frames in a video contain similar information. Both these redundancies are utilized during video compression.

For the purpose of compression, a video sequence is broken into a Group of Pictures (GOP). Each such group consists of temporally separated frames. A frame is a unit of video containing spatial video information pertaining to a particular time instant of the sequence. Three types of frames have been defined for use in video compression. These frames often use transform coding such as Discrete Cosine Transform (DCT) coding to encode the video data. "I" or Intra-coded frames are coded in isolation from other frames, and contain all the information required to reconstruct the frame without having to refer to any other frame. "P" or Predictive frames are coded with the difference information between a temporal prediction from a previous reference frame and the actual frame. In other words, a prediction for this time instant is made with the help of a previous reference frame, and the difference in the information between the predicted frame and the actual frame is then encoded. Lastly, "B" or Bi-directional coded frames are similar to P frames but use a previous and a future reference frame. In this case, the prediction is made using a reference frame that occurred previously, and another reference frame that is to occur at a later instant of time. The difference between the actual frame and the predicted frame is then encoded. The breakup of individual frames is in terms of number of pixels per line, number of lines per frame, an aspect ratio and some inter-picture timing information. The components that express the picture content are Luminance and color-difference components describing Chrominance. Luminance refers to the amount of brightness associated with the picture, and chrominance contains the color information pertaining to the picture. Luminance and chrominance values for a picture are defined in a two-dimensional matrices of size ”X x Y”, where X denotes a multiple of the number of pixels per line, and Y denotes a multiple of the number of lines per picture.

Two popular standards for real time coding of digital video that co-exist in the market today are the H.26x series proposed by the International Telecommunications Union - Telecommunications (ITU-T), and the MPEG-x series, brought forth by the Motion Pictures Experts Group committee of the International Organization for Standardization (ISO). H.263 [8], released in may 1996, was aimed at low bit rate video communication over the Public Switched Telephone Network (PSTN) and has been used in the experiments conducted for this work. Of the MPEG series, MPEG2 focused on digital television and DVD coding, and the more recent MPEG4 is aimed towards solving Internet based multimedia needs [37].

## 2.4 Traffic Generation and Analysis

Traffic generators perform the task of actually injecting data onto the network for the purposes of testing and measurement. The type of data flow is usually specified or selected from a list of options before executing the generator. The generator module itself usually works in a client server environment, where the client pumps data into the network, and the server sinks the data, and performs useful analysis on it and outputs traffic characteristics that it observed.

**Iperf:** Iperf [38] is a multicast capable open source tool that uses active measurement in order to determine TCP and UDP characteristics. Iperf is based on a client-server model, and the user can run the client with configurable options and receive and report measurement data at the server. For UDP traffic, Iperf measures datagram losses and delay jitter for

```
iperf-1.6.5>iperf -s -u -i 1
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
------------------------------------------------------------
[136] local 127.0.0.1 port 5001 connected with 127.0.0.1 port 2353
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[136]  0.0- 1.0 sec   128 KBytes  1.05 Mbits/sec  0.000 ms    0/   89 (0%)
[136]  1.0- 2.0 sec   128 KBytes  1.05 Mbits/sec  0.000 ms    0/   89 (0%)
[136]  2.0- 3.0 sec   128 KBytes  1.05 Mbits/sec  0.000 ms    0/   89 (0%)
[136]  3.0- 4.0 sec   129 KBytes  1.06 Mbits/sec  0.000 ms    0/   90 (0%)
[136]  4.0- 5.0 sec   128 KBytes  1.05 Mbits/sec  0.626 ms    0/   89 (0%)
[136]  5.0- 6.0 sec   128 KBytes  1.05 Mbits/sec  0.626 ms    0/   89 (0%)
[136]  6.0- 7.0 sec   128 KBytes  1.06 Mbits/sec  0.626 ms    0/   89 (0%)
[136]  7.0- 8.0 sec   128 KBytes  1.05 Mbits/sec  0.626 ms    0/   89 (0%)
[136]  8.0- 9.0 sec   128 KBytes  1.05 Mbits/sec  0.626 ms    0/   89 (0%)
[136]  9.0-10.0 sec   129 KBytes  1.06 Mbits/sec  0.626 ms    0/   90 (0%)
[136]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec  0.626 ms    0/  893 (0%)
```

**Figure 2.2**: Iperf server report for a UDP test

CBR traffic. Apart from specifying the target IP address and port number, user configurable options for UDP measurements in Iperf v1.6.5 include the bandwidth to send traffic at, and the amount of time to transmit for. Alternatively, the user can specify the datagram size and the number of such datagrams to transmit. Periodic and summary reports can be generated to report datagram losses and delay jitter, as shown in Figure 2.2. Bandwidth numbers can be obtained in various formats. Iperf v1.7.0 has useful new features including the ability to run Iperf in bidirectional mode causing the server to connect back to the client and transmit test data with the options specified, and the ability to report server side statistics at the client. In this work, a modified version of Iperf v1.6.5 is used as a traffic generation and analysis tool that injects test data into the network, and receives and analyzes the data at the target end point. The modification transforms Iperf into a VBR traffic tool, enhancing its current CBR-only functionality.

**Ethereal:** Ethereal [40] is a GUI protocol analyzer that examines real time network data and displays relevant protocol header and protocol data information for each packet or datagram that it encounters. In addition, Ethereal also displays the time instant at which it captured the particular datagram or packet. Ethereal is run on the target host where the network data is to be captured and analyzed. An existing interface on the host is selected, and the capture process is started. Once the capture process is terminated, the results of the capture are displayed. Filters can be applied both before and after capture to filter out unwanted information. The captured data can be saved for future use either in Ethereal or an other packet analyzer program like tcpdump (see below). Ethereal is used in this work to capture video source traffic and analyze its characteristics.

**Tcpdump:** Tcpdump [41] is a text based packet analyzer program that outputs in real time a dump of the packet header information for a given network interface. Optionally, a boolean expression can be specified and the output will contain information of only those packets that match the boolean expression. Tcpdump can also read from a stored file and output packet information, instead of in real-time from a network interface. Windump [42] is a port of this program for the Windows platform. Windump claims full compatibility with tcpdump, and uses a packet capture library built for Windows, in order to analyze network traffic. In addition to its use as a traffic analyzer, Windump is used in this work as a filter to parse required video datagram information from the trace files saved using Ethereal.

**Chapter 3**

# Framework Architecture

The framework is designed to generate, transmit, and receive video-like traffic over UDP/IP in the participating endpoints, and record and display the various metrics that assist in QoS analysis. The framework is modular. Different components in the framework are responsible for the tasks mentioned above. Section 3.1 describes these framework components in detail, and Section 3.2 provides the rationale for choosing the various QoS metrics and explains the metrics that are considered by the framework. The framework is designed to perform as a testing tool, and the procedure for conducting tests with this framework is described in Section 3.3.

The following is a list of terms used henceforth in this document:

- Frames - A frame refers to a video picture frame whose information is contained in one or more UDP datagrams. In the context of the framework and its implementation, the frame is conceptual since the datagrams that make up the frame do not contain valid video information.
- Tool/Implementation - These have been used interchangeably and both refer to an actual implementation based on the framework discussed in this Chapter.

- Picture resolution - This term refers to the picture sizes in terms of pixels. For example, a picture resolution of QCIF refers to video of size 176 pixels by 144 pixels. The terms picture resolution and picture format have been used interchangeably.
- Participating end points - The computers that are actually participating in the testing process. The end point that injects the test data into the network is called the sending end point or the sender. The end point that receives test data from a sender is called the Receiving end point or the receiver.

## 3.1 Framework Layout

Quality of Service capabilities are determined through the process of active measurement. We inject our own test data between the end points, collect measurements, and use this data to assess end-to-end QoS. The framework consists of four individual subsystems:

- Data generation
- Traffic generation
- Analysis and display
- Information management

A diagram that illustrates the relationship among the components and their interaction is given in Figure 3.1. The following steps occur during the testing process:

1. The user starts by specifying the way in which the tests are to be conducted.
2. The data generation module specifies the traffic characteristics, and the traffic generator generates traffic according to these requirements.
3. At both the sending and the receiving end points, datagrams are analyzed, and QoS

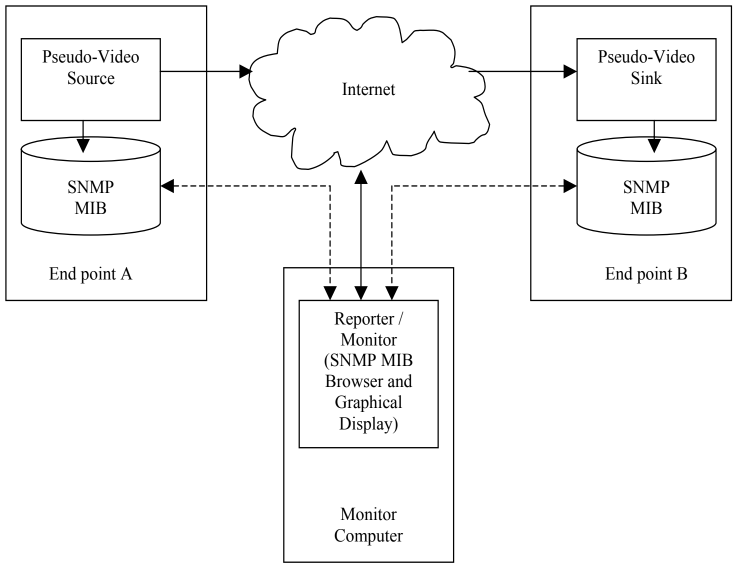


**Figure 3.1**: Layout of the framework components

relevant information is filtered out and maintained in the SNMP MIB.

4. A monitor may choose, independently, to query the Information management module and retrieve data pertaining to one or more such tests.

### 3.1.1 Data and Traffic Generators

The data and traffic generator modules are responsible for generating video-like test data and transmitting them between the participating end points according to a set of user specifications. Data generation involves determining the size of each frame and the time instant at which to transmit the video, as well as sizes of individual datagrams that make up each frame. This timing and datagram size information are determined from the picture format, motion scene activity level, and maximum bandwidth information input by the user. In addition, all necessary information that may be required for QoS analysis is contained within every transmitted datagram so that there is no additional end-to-end load generated by the framework. Data generation can be achieved either by means of traffic modeling, or by the use of trace files.

One approach would be to use one of several empirical or stochastic video traffic models. A survey of different techniques for VBR video traffic modeling is given in [28]. This survey also gives recommendations on which models to choose, depending on the requirements. When choosing a model for use with this framework, preference must be given to models with a fixed GOP structure, whose activity level can be controlled. This is because in video-conferencing applications, there is not much interframe or inter scene movement (For example, like rapid changes and inconsistent scene lengths in an action movie). Also, a GOB level break-up of video data should be preferred over frame level break up, to keep the datagram sizes similar to RTP fragmentation [3]. The model should also be able to generate video datagram information for varying levels of activity. For example, The Doulamis model [31] considers activity level, and assumes a fixed GOP structure, but the number of input parameters required for this model is quite high [30]. Some characteristics of traffic sources, including Short Range Dependence (SRD) and Long Range Dependence (LRD), have been explained in [33]. An SRD model for VBR video has been described in [29].In order to maintain activity level, short term correlation must be considered while choosing the model.

Another approach to data generation can be through the use of trace files. Timing and datagram size values can be generated based on the information contained in a trace file that contains data corresponding to the options that the user has selected. These trace files contain static information either gathered during actual video conferences, or created using a modeling scheme. Several trace files corresponding to the different combinations of user inputs can then be collected for varying activity levels, encoding schemes and resolutions.

In any case the traffic generator should display best-effort behavior. In other words, the traffic generator should make a *best effort* in transmitting the data as specified by the data generator. This helps in gaining useful insight into the load on the processor on which the traffic generator is being executed. This effect is explained and demonstrated in Section 5.2. Transmission involves interpreting the data provided by the traffic generation scheme, and actually creating and transmitting the UDP datagrams of specified size at the given time instants. In addition to transmitting the datagrams, the sender also locally records information regarding its transmission. The receiving end point closely interacts with an analysis unit to extract and determine the required Quality of Service metrics from the received datagrams, and records the information.

### 3.1.2 Analysis and Display

Each transmitted test datagram consists of the following fields at a minimum to assist in determining the various Quality of Service metrics:

i. A datagram sequence number

ii. A frame sequence number indicating the frame to which this datagram belongs and

iii. A time stamp that is necessary for calculating delay and jitter.

The module that receives the test data records the time instants at which it received each datagram, and uses it to calculate the received (effective) bandwidth, frame rate and jitter

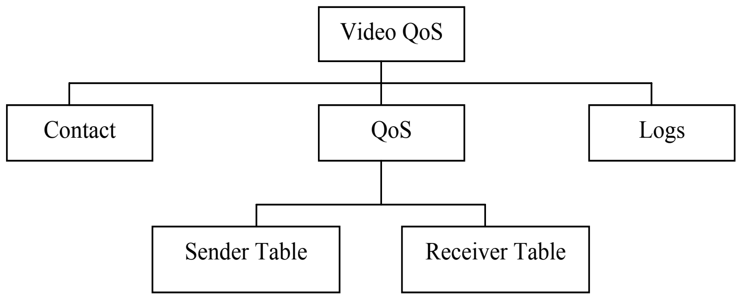


**Figure 3.2**: Video QoS MIB components

values. The various metrics that determine QoS are discussed in Section 3.2. The display module consists of a monitor is designed to query and display the Quality of Service metrics and data in a user-readable graphical format. The monitor is asynchronous. It requests the sending and receiving end points using the SNMP protocol to provide the recorded data upon user request, irrespective of the state the testing process is in. The monitor then formats the data to display test results in a graphical format, highlighting anomalies and extremities, and indicating hardware resources present on the end-systems.

### 3.1.3 Information Management

The QoS information pertaining to the tests are stored in an SNMP MIB implementation, called the Video QoS MIB. The motivation to design a new MIB for use with this framework arose because existing MIBs focus on the control data of a video communication rather than the video payload, or others, like the RTP MIB, do not contain frame level information. Our MIB has been designed specifically for use with our framework. Table 3.1 compares the existing MIBs and specifies the motivation for designing a new MIB. Extensions to the already existing MIBs can be an alternative to consider.

| | | |
|---|---|---|
| H.341 MIBs [14] | Defines various control metrics, H.323 [15], H.320 [17], H.245 [16] and Gateway information. Also contains System capabilities, line rates etc. | Does not contain actual multimedia data information |
| RTP MIB [18] | Defines RTP Sessions and session entries. Sender/Receiver tables contain number of packets and bytes | Does not contain video specific information such as frame rates and frame losses. Does not contain data rate information. |
| Video QoS MIB | All missing information, along with other required information maintained in one place. Extensible design. Designed for use with the framework | Does not concern itself with control information. Could use H.341/RTP MIBS in conjunction for control info. |

**Table 3.1**: Video QoS MIB comparison

The QoS subtree in Figure 3.2 is divided into two parts. The Sender table subtree contains entries which record the different options selected for each test at the sending end point, including activity level and maximum encoding rate. It also maintains the frame rate, bandwidth and datagram information observed at the sender. The receiver table subtree has entries that contain frame and datagram information, delay, jitter and losses observed at the receiving end point corresponding to each test. The Logs subtree consists of a log table that records event and application logs. Information about the number of processes, physical, extended physical memory, and CPU usage on the sending and receiving end points could be obtained from an existing host system MIB implementation such as the Host Resources MIB [6]. The values would indicate the present load on the system and the extent to which that system is ready for a video communication application, in terms of video coding and display.

It is necessary to have a mechanism that interfaces the traffic generator module with the SNMP agent in order to update the values of the QoS metrics stored in the MIB implementation. In the best case, the stored values are updated for each datagram that is transmitted or received. Otherwise, the MIB values are updated at regular intervals with the data that is transmitted or received. This interval is independent from the intervals at which the display module queries the SNMP agent to retrieve the latest values.

## 3.2 End-to-end QoS Metrics

The human workflows that use real time video communication impose certain restrictions on the underlying network, and the sending and receiving end points, for the resulting video to be acceptable to the end user. For example, a low frame rate of less than 10 frames per second appears as discontinuous video and is thus frustrating to the user. Thus, Quality of Service refers to the capability of participating end points and the underlying network to provide real time compressed video service that meets end-user needs.

QoS metrics fall into one of three categories. Those metrics that are indicative of the QoS support provided by the network are called Network QoS metrics. Those that describe the QoS support provided by the end point host Operating System and Platform are called Operating System QoS metrics. Finally, those metrics that describe QoS support at the application level are known as Application QoS metrics. The list of the Quality of Service metrics that were considered in the design of this framework are given in Table 3.2, and a description of how the different layers impact the QoS metrics is given in Table 3.3. For example, the number of frames is directly determined by the application level, and not by the network or the host Operating System. Latency, on the other hand, is affected by the application, Operating System and network levels.

The rationale for choosing the video QoS metrics for this framework is based on the host and network resource requirements of real time compressed video. At the host level, the generation and transmission of digital video passes through various stages and utilizes different resources [25] at each stage. Input/Output and memory buffers are required for information storage before transmission, and sufficient Processor speed is required by the video codec to perform the necessary real time computation involved in the encoding of the video signal. Thus physical memory and CPU utilization become important metrics that determine host resource support. Buffering at the end point hosts and the network nodes introduces bandwidth variation, losses, delay and jitter [22][36], which in turn impact the quality of the received video seen by the end user. Finally, components such as frame losses and frame rates of the video itself contribute significantly to providing perceptual metrics that can either be used directly, or can be input to a perception analysis system for qualitative analysis (see Section 5.6).

### 3.2.1 Bytes

The framework maintains the amount of total bytes transfered during each test. This would be indicative of the amount of compressed real time video content transfered in the duration of the test. This amount varies with the different configurable options selected by the user prior to the start of the test. The actual bytes neither contain valid video information nor valid headers. But for the purposes of calculation, the number that is represented in this QoS variable assumes that the the following are included in the pseudo-video data:

- RTP header

| Metric | Metric directly impacted by | | |
|---|---|---|---|
| | Network | Application | Operating System |
| Number of Bytes | No | Yes | No |
| Number of Datagrams | Yes | Yes | Yes |
| Number of Frames | No | Yes | No |
| Bandwidth | Yes | Yes | Yes |
| Frame Rate | No | Yes | No |
| Latency | Yes | Yes | Yes |
| Datagram Jitter | Yes | No | Yes |
| CPU Usage | No | Yes | Yes |
| Memory Usage | No | Yes | Yes |

**Table 3.2**: List of video QoS metrics

- video content (RTP Payload header, H.263 or other encoding format header and encoded video)

The number does not include any of the lower level headers like UDP and IP.

### 3.2.2 Datagrams

In the context of the framework, datagrams refers to the number of UDP datagrams transfered during a test. Each UDP datagram may be of different size, and may be sent at different instants of time, not necessarily at regular intervals. The UDP datagrams are generated according to the timing and size information specified by the data generator. In addition to the number of datagrams transfered, the number of datagrams lost during a unit test are also maintained at the receiver. Datagrams may be lost either because of a load on a network node, or due to a load on the the end point that receives the test data.

| Metric | Description |
|---|---|
| Number of Bytes | The application level determines the amount of data generated in terms of bytes, depending on various video data factors like motion and picture format. Datagram losses in turn implies a loss in the number of bytes received. |
| Number of Datagrams | At the application level, protocols such as RTP fragmentation determine number of datagrams. Also lower layer framing at the operating system level, and datagram losses and data corruption at the network level determine the number of datagrams received. |
| Number of Frames | The application level determines the number of frames generated. Datagram losses indirectly impacts number of frames received. |
| Bandwidth | Bandwidth may be deliberately restricted or specified at the application level. Bottlenecks at the operating system level and the network level may also affect the bandwidth value. |
| Frame Rate | Frame rate may be specified at the application level. It is also affected by various video data factors like motion and picture format. Datagram losses and bandwidth values indirectly impact frame rate. |
| Latency | Buffering at the application, operating system and network levels, as well as propagation delays all contribute to latency. |
| Jitter | Variation in the servicing of datagrams at the operating system and network levels contribute to datagram jitter. |
| CPU Usage | The number of applications running on the operating system, and the amount of processor time consumed by the applications in turn determine the percentage of CPU time allocated to the video application. CPU usage indirectly affects all other metrics (except memory usage). |
| Memory Usage | Similar to CPU usage, the number of applications executing simultaneously determine the amount of memory space allocated to the video application. |

**Table 3.3**: Description of how the video QoS metrics interact

### 3.2.3 Frames

As a Quality of Service metric, each frame denotes an encoded video frame. Each individual frame may span one or more UDP datagrams. This span is determined by two factors for actual video - the total size of the frame, and the positions at which RTP packetizes the frame. For this framework, it is assumed that one UDP datagram may contain not more than one frame. Each frame in a test is identified by a frame number starting at 1. The framework does not take the type of frame (“I”, “P” or “B”) into account for any of its QoS analysis. In the context of this framework, a frame is said to be lost if *all* the UDP datagrams that comprise this frame are lost. Frame losses can be compared with datagram losses as follows. When frame sizes are small and every datagram contains a different frame, frame loss numbers would be equal to datagram losses. However, as frame sizes increase, datagram losses tend to be far greater than frame losses, because in this case many UDP datagrams make up one frame.

### 3.2.4 Bandwidth

The bandwidth metric is used to maintain the effective bandwidth utilized during each test. This metric recorded at the sender indicates the ability of the sending end point to support the bandwidth of the data that is generated by the data generator. Since the pattern of test data generated by the data generator can be predetermined for a particular set of input parameters the user selects, the bandwidth of the data generator can be compared with the actual bandwidth of test traffic at the sender to serve as a measure of the sending end point support for video. On the other hand, bandwidth recorded at the receiver is a useful measure of the ability of the underlying network and the receiving end point to support the selected type of video data traffic.

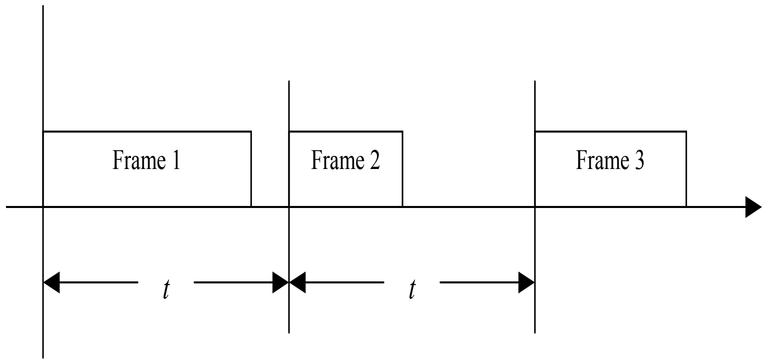


**Figure 3.3**: Constant frame rate from a VBR video source

### 3.2.5 Frame Rate

The frame rate metric maintains the number of frames transfered per second. As given in Figure 3.3, VBR video sources tend to keep a constant frame rate, while the size of each frame may vary. This value is determined both by the number of frames being generated by the data generator, as well as delays in the underlying network. Even though frame rate tends to vary similar to bandwidth in most cases (see Section 5.4.2), this value is useful as it determines user perception, and can be used as input to other systems such as VQM [57], that analyze user perception.

### 3.2.6 Latency

The latency metric captures the one way delay encountered by a datagram from the instant it is transmitted by the framework at the sending end point, to the time instant it is received by the framework in the receiving end point. For the one way delay measurement to be accurate, a synchronization mechanism such as NTP [34] is required. In real time communication, it is important for one way latency to be minimal for the communication

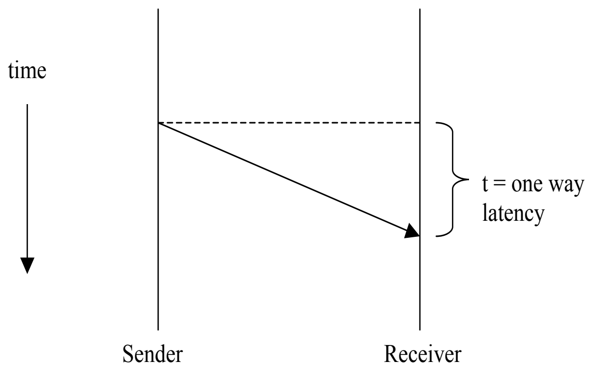


**Figure 3.4**: One way latency

experience to be acceptable. In a video conference for example, as the latency increases, the user has to wait for longer periods of time for responses from the other end point. When the wait duration becomes noticeable for humans, the resulting experience is less pleasurable or in some cases unacceptable. Figure 3.4 outlines the one-way latency between the two participating end points.

### 3.2.7 Jitter

Datagram jitter is a measure of the variation in the end-to-end delay times encountered by datagrams as they are received by the target end point. The expression for calculating jitter that is used by this framework is the one specified in the RFC 1889 (RTP) recommendation [2]. The formula is:

$$J_{curr} = J_{prev} + (|D_{(i-1,i)}| - J_{prev})/16$$

Where $J_{curr}$ = Current value of inter arrival jitter,

$J_{prev}$ = Previous value of inter arrival jitter,

$D_{(i-1,i)}$ = time difference in packet spacing between packets i-1 and i.

| Datagram No. | Sent time instant $s$ | Recd time instant $r$ | $D_{i-1,i} = (r_i - r_{i-1}) -(s_i - s_{i-1})$ | Jitter $J_{curr}$ |
|---|---|---|---|---|
| 1 | 0 | 2 | N.A | N.A. |
| 2 | 4 | 6 | $(6-2)-(4-0)=0$ | 0 |
| 3 | 8 | 11 | $(11-6)-(8-4)=1$ | $0+(1-0)/16=0.0625$ |

**Table 3.4**: Jitter calculation example

The RFC states that the algorithm is a first order estimation and that the 1/16 factor is a gain parameter that provides good noise reduction ratio while maintaining convergence. The definition of Jitter and explanation of the calculation mechanism for Jitter provided in the RFC [2] has been attached in Appendix C. The validity or effectiveness of this expression is outside the scope of this work. The expression is used as stated in the RFC so as to maintain compliance. The variance in the time spacing between datagrams, irrespective of which frame they belong to, is considered for calculating datagram jitter. An example of jitter calculation using this expression is illustrated in Table 3.4. For this framework, out-of-order datagrams and lost datagrams are left out for the purposes of jitter calculations.

**Intra frame vs. Inter frame Jitter:** Consider two frames $F_1$ and $F_2$ as given in Figure 3.5. The $D_x$ notations indicate datagrams belonging to each frame. For example, $F_1 - D_1$ indicates the first datagram of the first frame. The variance in the time spacing $t_1$ between datagrams of the same frame is considered for calculating the Intra frame Jitter. The variance in time spacing $t_2$ (where $t_2$ is usually much greater than $t_1$) between the starting datagram of one frame and the starting datagram of the next frame is considered for calculating the Inter frame Jitter, also known as frame inter-arrival jitter [48]. At the source, time spacing between datagrams of the same frame is typically in the order of a few hundred microseconds. On the other hand, the time spacing between the starting datagram

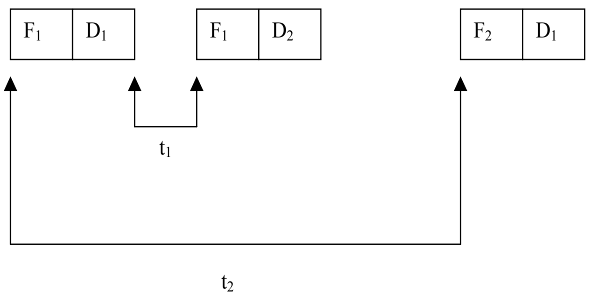


**Figure 3.5**: Inter frame jitter vs. Intra frame jitter

of one frame and the starting datagram of the next frame is in the order of tens of milliseconds. For example, for a 30 fps picture, ideally inter frame spacing would be 1/30 seconds. This difference in the time spacing between intra and inter frame datagrams could affect how network nodes service the datagrams. There is no inherent relationship between inter frame jitter and intra frame jitter. However, they affect the way in which buffers must be designed at the receiver in order to render the frames. This makes them important metrics to consider for codec buffer choice at the receiver.

### 3.2.8 CPU Usage

The percentage of processor time that is taken up by the different processes that are currently executing, is indicative of the load on the CPU. This can be determined using the amount of time the processor spends executing an Idle thread, in comparison to the amount of time it spends executing all the other threads, as follows:

$$CPU = (1 - \frac{t_{idle}}{T}) * 100$$

where $CPU$ = Percentage of time spent by the processor executing non-Idle threads,

$t_{idle}$ = Amount of time in current sampling interval spent on Idle thread,

$T$ = Total amount of time in the sampling interval.

A busy CPU results in less CPU time available for a real time video application to execute, resulting in a reduction in the frame rate and thus the quality of video. Both the sending end point and the receiving end point can be affected by a busy or overloaded CPU.

### 3.2.9 Memory Usage

The amount of Physical and any extended physical memory, such as Virtual Memory in the Windows Operating System, used by currently executing processes together indicate the memory load on the participating end point. As in the case of CPU Usage, memory usage is ideally a percentage value of the total memory currently available on this end point. Physical memory is the actual amount of Random Access Memory available on the end point, and extended memory is the amount of memory (greater than the actual physical memory) *seen* by applications, and consists of the physical memory in combination with secondary storage such as hard disk space. The amount of paging or other memory swapping activity on the host also indicates whether the currently executing processes are continually starved for memory. Less memory available to a real time video application translates to a reduction in the resulting video quality, similar to the affect of an overloaded CPU.

## 3.3 Tests and Sessions

At user level, the framework determines Quality of Service based on a series of tests conducted at user specified time intervals. The number and variety of tests performed determines the stability and accuracy of the QoS results obtained.

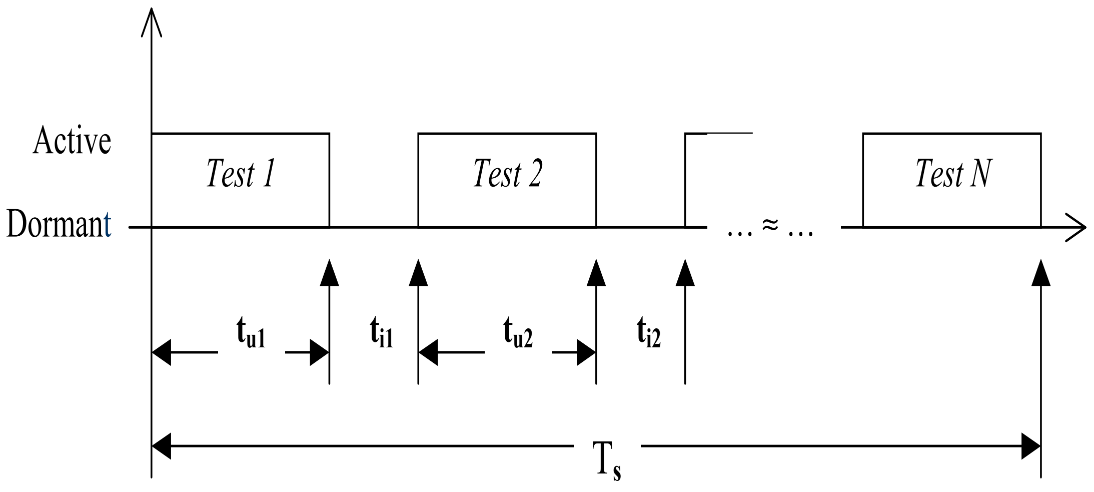


**Figure 3.6**: Testing cycle

A Unit Test, in our framework, refers to an individual test of time duration $t_u$ performed with one set of configured options selected by the user. The user may select N unit tests $(N \geq 1)$, to be performed at specified intervals of time. This cluster of n unit tests for the duration of which the user selected options remain constant are then considered as one "session". Each session is identifiable through a unique Session Identifier. Tests within a particular session are numbered starting at 1. In Figure 3.6, the total time for one session, $T_s$, can be given as follows:

$$T_s = \sum_{x=1}^{N} t_{u_x} + \sum_{x=1}^{N-1} t_{i_x}$$

For example, the user may request ten unit tests of duration twenty seconds each, to be performed at regular intervals of one hour between unit tests. In this case, this one session lasts for a duration of ten hours and twenty seconds. The set of configurable options selected by the user is constant for the duration of the session. The user may run several such sessions, each with a different set of configurable options, to determine the behavior of the tests for those sets of options. The Quality of Service metrics that result from the tests indicate the extent to which video is supported between the two end points, for the specified interval of time. Thus more tests performed over extended periods of time tend to give a clearer picture of actual end-to-end support for video communication.

## 3.4 Framework Output and Usage

The output of the framework is a quantitative description of the results of the tests it conducts to determine the extent of support for video between the participating end points. This description consists of the various QoS metrics and host resource information and is displayed in a human consumable format. An administrator may gather several sets of such readings using the framework, and then use the readings to determine possible pictorial degradation and other qualitative metrics either manually or by other means (see Section 5.6). The framework alone does not attempt to do any qualitative or perceptual analysis based on the QoS information it derives from the tests.

Chapter 4

# Implementation

This Chapter describes an implementation of the architecture presented in the Chapter Three. The implementation of each architectural component is explained in detail, and their functional traits discussed. All the results and the analysis of the results of the framework described in Chapter five are based on this particular implementation. The modules are implemented so that they remain extensible, and additional functionality may be added with minimal changes to existing functionality. For most part, the modules are loosely coupled so that modules may be replaced with others providing similar functionality, with minimal changes to the code.

## 4.1 Target Environment

The target Operating System chosen for implementing the framework is Microsoft Windows. The Windows Operating System is chosen specifically because most of today's common Video Conferencing solutions have been developed for this Operating System. This facilitates the administrators to determine video QoS capabilities on the same end points that are or will be used to run the actual video conferencing products. Our implementation is disparate in terms of the development platforms used. User interfaces have been created using Java version 1.4.0 Swing classes. Timing and information management modules have been created using C and C++ languages, compiled with Microsoft Visual Studio version 6.0.

## 4.2 Data and Traffic Generation

For our implementation, the traffic generator was built using Iperf [38] version 1.6.5. Since Iperf is an Open Source tool, its code could be modified to generate VBR traffic. Iperf also provides built-in support for datagram loss and Jitter calculations that are vital metrics for determining video QoS. A diagram that shows this Iperf based implementation is given in Figure 4.1. The tool consists of the following components that assist in test traffic generation as well as QoS information maintenance:

- A Graphical User Interface is used to accept test cycle and video detail information from the user.
- The modified version of Iperf is started either in Client mode or in Server mode depending on user selection. The Iperf execution is handled in a separate thread. If the tool is being run in “client” or sender mode, the thread also utilizes a test cycle timer program to control the generation and transmission of test data as per user specification.
- The Iperf client code reads a particular trace file that contains the timing and datagram size information necessary to generate and transmit test data to the receiver. The trace file that is chosen is determined by the combination of input parameters chosen by the user.

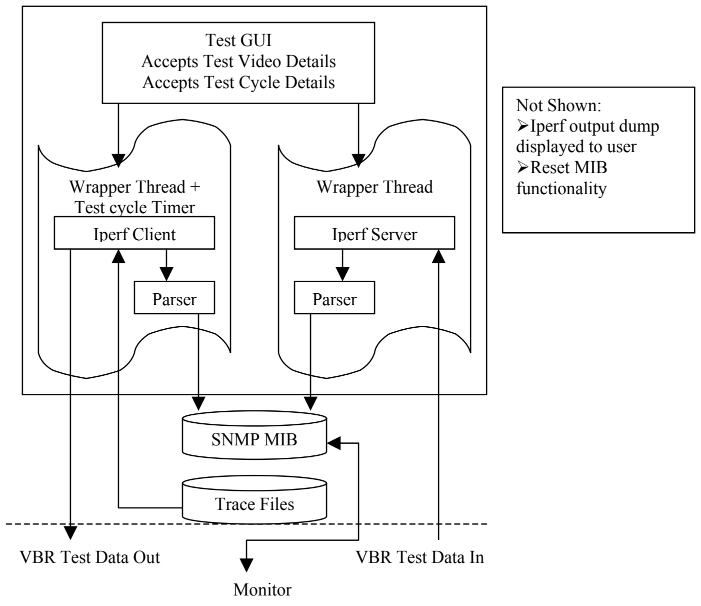


**Figure 4.1**: Iperf based tool implementation

- The Iperf server listens on the specified port for incoming test data.
- The output from the Iperf program is fed to a Parser object. The parser filters out the necessary QoS information and formats it to be sent to the SNMP MIB implementation for records-keeping.
- A monitor program may choose at any time to query the MIB on the participating end points and retrieve the QoS information.

### 4.2.1 Data Generation

A set of experiments were conducted to characterize VBR video traffic generated at the source. The test was based on studying RTP datagrams generated at the source with an H.263 encoder. The RTP output was collected for different combinations of the set of {Activity Level, Maximum Encoding Rate, Picture Resolution}. The resulting timing, datagram size and frame number information was dumped to a file. The data was then filtered out and analyzed to produce characteristics charts. The characteristics of H.263 video traffic and the details of the experiment are outlined in Section 5.1.

### 4.2.2 Traffic Generation

The Iperf code was modified for incorporating video VBR traffic generation by inserting an additional module to read the trace file corresponding to the options the user has chosen. The datagram sender module in Iperf was also modified to send datagrams according to the trace file entries. The timing calculations in Iperf were modified[1] to maintain better timing accuracy through the process of time compensation. The time for execution of one iteration of the datagram transmission loop is subtracted appropriately from the time instant at which the next datagram is sent, thus compensating for the time required by the host to execute that part of the code.

The sizes of individual trace files are kept at a minimum by including the trace information for a short duration, and having the trace file reader loop through the same file using a wrap-around method. The last entry of the trace file is assumed to be the same as the first entry in the trace file, except for the time instant at which it is sent. If the trace file looks like this:

| | |
|---|---|
| Time instant: 0.0 | Datagram: $D_0$ |
| Time instant: $t_1$ | Datagram: $D_1$ |

...

[1]The timing mechanism in Iperf 1.6.5 for Windows had a resolution of about 10 milliseconds (ms). In other words, updates to the timer function used in the Iperf code were of the order of 10 ms. For VBR data generation, this was highly insufficient. So the timing function in Iperf had to be rewritten for microsecond resolution in Windows.

...

...

Time instant: $t_{n-1}$ Datagram: $D_{n-1}$

Time instant: $t_n$ Datagram: $D_0$

The next instant of time is calculated as $t_n + t_1$ and the datagram to be sent is $D_1$. So the program just wraps around after the end of the trace file and starts again from the beginning. This method is also useful to maintain the activity level for the duration of a test. All the necessary information that needs to be transfered from the sender to the receiver is contained within each datagram so as not to introduce additional control information on the network when the test is in progress. The header for each UDP test datagram is designed to carry the control information as shown in the header diagram Table 4.1.

*Datagram ID:* A 32 bit number starting at 1 that uniquely identifies a datagram in a unit test. This number is used at the receiver to calculate the total datagrams received, as well as the number of lost datagrams.

*Sender time:* a 64 bit number with the first 32 bits representing the number of seconds and the next 32 bits representing the number of microseconds of the time instant at which this datagram is sent. The time stamp is obtained from the underlying Operating System.

*Frame ID:* A 32 bit number starting at 0 that identifies the frame that this datagram belongs to. More than one test datagram may have the same frame ID depending on whether the information about one frame is being carried by multiple datagrams. This number is used by the receiver to calculate the number of frames received and lost, as well as the frame rate observed at the receiver.

| 1 | 32 |
|---|---|
| Datagram ID | |
| Sender Time instant (Seconds) | |
| Sender Time instant (Microseconds) | |
| Frame ID | |
| Test Session ID | |

**Table 4.1**: Test datagram header

*Test Session ID:* A 32 bit number that is randomly generated at the sender and uniquely identifies datagrams that belong to a particular test session. The session ID is used by the receiver to maintain and differentiate among the information generated by several test sessions.

Since the total size of the test datagram header is 20 bytes, the minimum size of a test datagram is 20 bytes. If the trace file specifies a datagram size less than 20 bytes, the tool still sends a 20 byte datagram in order to maintain all the information necessary to identify the datagram at the receiver.

### 4.2.3 Analysis

Section 4.2.2 explained the test datagram header. The information in the header is used at the receiver to analyze and record the various metrics that quantify QoS. At the sender, as the test data is being injected into the network, requisite information is continuously recorded and periodically output from the Iperf module. This information includes number of bytes, datagrams and frames transmitted, and bandwidth and frame rate determined at the sender. Host resource usage metrics monitored include peak values and last determined values of CPU usage, physical memory and committed memory as a percentage of their respective maximum values on the end point. Committed Memory is defined in Windows PerfMIB [49] as follows:

> Committed memory is physical memory for which space has been reserved on the disk paging file in case it needs to be written back to disk.

This information is stored in the local MIB, along with the configurable options set by the user for this test. At the receiver, datagram headers are filtered from the received datagrams. The datagram numbers indicate the number of total datagrams received and the number of lost datagrams. For example, if the last datagram received during a test has the datagram number as 200, then a total of 200 test datagrams were received at the target end point in the duration of the test. Non-sequential datagram numbers in the received datagrams indicate lost or out-of-order datagrams. Out-of-order datagrams are also counted as lost datagrams. The number of total frames received and frame loss are also calculated similar to datagrams, but using the Frame numbers contained within the received datagrams. In this case, out-of-order frames are not reported at all, and any frames that are out of sequence are automatically considered as lost frames. Jitter is calculated based on the time stamp contained within the received datagram header and the time instant at which the datagram was received, as given in Section 3.2.7. The maximum value of jitter that was determined for the duration of the test is also recorded, and indicates an anomaly that may have occurred during the test. Latency is not calculated due to the absence of a time synchronization mechanism in this implementation.

| No. | Variable | Description |
|---|---|---|
| 1 | Session ID | Session ID that this test belongs to |
| 2 | Test Number | Test number of this test, starting at 1 |
| 3 | Test Duration | Duration of the test, in seconds |
| 4 | Activity Level | Activity level selected for this test |
| 5 | Picture Format | Picture Format selected for this test |
| 6 | Maximum Encoding Rate | Max. Encoding rate selected for this test |
| 7 | Bytes Sent | Number of bytes sent during this test |
| 8 | Bandwidth at the sender | Bandwidth at the sender for this test, in bytes per second |
| 9 | Datagrams Sent | Number of Datagrams sent during this test |
| 10 | Frames Sent | Number of video frames sent during this test |
| 11 | Frame Rate at the Sender | Frame rate for this test, in frames per second |
| 12 | Test Status | States whether the test is in progress, or is complete |

**Table 4.2**: Video QoS MIB Sender Table entries

## 4.3 SNMP Module

The QoS subtree in the SNMP MIB described in Section 3.1.3 consists of a Sender Table and a Receiver Table, and maintains all the QoS metrics determined with the traffic generator tool. The entries within these tables are given in Tables 4.2 and 4.3. The MIB is implemented to maintain these values, and is attached to an existing SNMP Service, which is the standard SNMP Service that is part of the Windows Operating System. The tables are internally implemented as singly linked lists, utilizing additional memory as required to grow the table and incorporate new entries. They are initialized at SNMP service start up, and cleared at SNMP service shutdown.

To record the peak and last values of end point host resource QoS metrics such as CPU usage and Memory usage as given in Table 4.4, the tool queries the PerfMIB subtree maintained by the Windows Operating System. The peak and last determined values are then

| No. | Variable | Description |
|---|---|---|
| 1 | Session ID | Session ID that this test belongs to |
| 2 | Test Number | Test number of this test, starting at 1 |
| 3 | Test Duration | Duration of the test, in seconds |
| 4 | Bytes Received | Number of bytes received during this test |
| 5 | Bandwidth at the receiver | Bandwidth at the receiver for this test, in bytes per second |
| 6 | Total Datagrams Sent | Number of Datagrams sent during this test |
| 7 | Datagrams Lost | Number of Datagrams lost during this test |
| 8 | Total Frames Sent | Number of video frames sent during this test |
| 9 | Frames Lost | Number of video frames lost during this test |
| 10 | Frame Rate at the Receiver | Frame rate for this test, in frames per second |
| 11 | Jitter | Datagram Jitter recorded for this test, in microseconds |
| 12 | Maximum Jitter | Maximum Jitter recorded for this test, in microseconds |
| 13 | Test Status | States whether the test is in progress, or is complete |

**Table 4.3**: Video QoS MIB Receiver Table entries

| No. | Variable | Description |
|---|---|---|
| 1 | Peak CPU Usage | Percentage of time the processor is executing a non-idle thread, maximum value during this test |
| 2 | Last CPU Usage | Percentage of time the processor is executing a non-idle thread, last value determined during this test |
| 3 | Peak Physical Memory Usage | Physical memory used in bytes, maximum value during this test |
| 4 | Last Physical Memory Usage | Physical memory used in bytes, last value determined during this test |
| 5 | Peak Committed Memory Usage | Committed virtual memory used in bytes, maximum value during this test |
| 6 | Last Committed Memory Usage | Committed memory used in bytes, last value determined during this test |
| 7 | Peak Page accesses | Number of page accesses (hard page faults) , maximum value during this test |

**Table 4.4**: Host QoS metrics calculated from Windows PerfMIB

stored in a *Sender/Receiver Table* subtree implemented in the Video QoS MIB as the *System* subtree. The information in the System subtree is used in conjunction with the information given by the Host Resources MIB [3], discussed in Section 4.4.

There is an option provided in the MIB implementation for the tool user to clear the existing table entries without having to restart the SNMP service. This option is useful in several ways. Firstly, the user may clear test data entries in the MIB without having to lose SNMP information contained in subtrees other than the Video QoS MIB subtree. It also helps in clearing entries that were recorded during previous test sessions, before starting a new test session. This in turn frees up the memory occupied by all the entries in the previous test sessions. When the memory space occupied by the MIB entries becomes significantly large, it may contribute towards slowing down the system performance. The reset MIB functionality is not selective in its clearing of the values. All values that currently exist in the MIB are deleted when the user sends a reset MIB request.

## 4.4 Display

The Display module queries the participating end points, gathers QoS information, formats it so that it is human-consumable, tabulates and displays the information to the user. The user starts by selecting which target end points are to be queried for test information, and the rate at which the end points must be queried to get fresh data. For each end point, an end point handler object is created to query the SNMP MIBs. Separate threads are created to query the QoS and System Sender and Receiver tables. These threads populate the GUI display with the latest values parsed from the values obtained from the MIB. In addition to querying the Video QoS MIB implementation, the Host Resources MIB [3] implementation provided by the Windows Operating System is also queried, and the following information is obtained:

- Total amount of Random Access Memory on the host
- Number of processes running on the host
- Number of CPUs on the host
- Host system description

These above values are obtained just once, at the start of the query process. However, a manual “Refresh” button is provided to the user to refresh the values recovered from the Host Resources MIB on the specified end point. The steps that occur while gathering SNMP information from the end points are as follows:

1. Query the Host Resources MIB at the specified end point and get and display system hardware information

2. Start separate threads to query

    - QoS Sender table
    - QoS Receiver table
    - System Sender table
    - System Receiver table

3. Parse the information from the tables, and convert them to human-readable format. For example, convert Picture Resolution values obtained from the Sender table entries, from 0 and 1 to QCIF and CIF respectively

4. For the peak and last CPU and Memory usage values obtained from the System subtree tables, calculate percentage values from the absolute values

5. Display the latest values

| Video Test Type | Test Time Information |
|---|---|
| Activity Level | Duration of each test |
| Picture Format | Total number of unit tests to be performed |
| Maximum Encoding Rate | Time interval to wait in between unit tests |

**Table 4.5**: Test cycle input parameters expected from the user

An Open Source code that provides an SNMP client [43] was utilized and modified to do an SNMP "walk" in order to get all the entries in a table having specified the starting and ending Object Identifiers.

## 4.5 Tests and Sessions

The implementation assumes that all unit tests in a session are of the same duration, and that the interval of time between unit tests in a session is constant. Thus the values $t_{u_x}$ and $t_{i_x}$ are reduced to $t_u$ and $t_i$ respectively in the calculation of the total session time $T_s$ given in Section 3.3. Thus, session time $T_s$ can be calculated as follows:

$$T_s = N(t_u) + (N-1)(t_i)$$

All information for conducting the test is accepted from the user in a Graphical User Interface (GUI). The user starts a session of unit tests by specifying the following values describing the nature of the video input and the test duration as given in Table 4.5, along with the destination addresses and ports. Each session of such unit tests is started in a separate thread. The logic for execution of unit tests in a session is given below:

1. Start a separate thread for this session

2. Run the Iperf process with the video options chosen

3. Parse all the input from the output and error streams of the Iperf process, enter the parsed information into the SNMP MIB

4. if {number of unit tests not complete}

   - wait for the time interval specified, compensating for any time lost while running the Iperf process and parsing its output
   - decrement the number of tests to be performed
   - go to step 2

## 4.6 Inter-Module Communication

The parser program selectively filters out the information output by the Iperf based traffic generator module, and sends them to the MIB implementation to be updated. This communication happens through the use of a UDP socket. As soon as the SNMP service is started on the end point, the Video QoS MIB implementation starts a a single thread that listens to a port for incoming updates. The parser pushes values to be updated to this socket and the thread receives them. The Display module queries the SNMP agent at the end points through the well known SNMP UDP port 161, again through the use of sockets. The Iperf module outputs periodic and summary information to the output console. A wrapper application that runs the Iperf module opens a Java Output Stream, and receives all the output information that the Iperf module dumps on the console.

## 4.7 Implementation Availability

This implementation is owned by NCNI and is currently undergoing testing. It will eventually be made public-domain and Open Source using an appropriate license agreement. The setup file for the implementation can be found at

http://www4.ncsu.edu/ vchandr/videosite/progress/soft/setup.exe

# Chapter 5

# Tool Calibration and Performance

We start this Chapter by describing the experiments conducted for studying the traffic characteristics of a VBR video source. The second section explains the traits displayed by the traffic generation module of the tool. In the third section, we describe the results of experiments conducted with the tool and obtaining measurements under different network and host resource loads. Finally, we describe the affect of load on the human perception of actual video transmitted between the participating endpoints.

## 5.1 Source Traffic Characteristics

### 5.1.1 Experimental Setup

The objective of this set of experiments was to accurately characterize VBR video traffic generated at the source. The test was based on studying RTP datagrams generated at the source with a software H.263 encoder. The RTP output was collected for different input video sequences categorized according to the amount of motion they contained. Table 5.1 lists these well-known video test sequences. The RTP datagram information was collected and saved using Ethereal. The data was then filtered out using Windump and analyzed to

| Activity Level | Video Sequence | Length* (seconds) | Description |
|---|---|---|---|
| None | None | None | No input video. Camera switched off, and the resulting blue screen transmitted |
| Minimal | Miss America | 5 | Person talking against a dark background, minimal face and neck movement |
| Some | News | 10 | Two news anchors against a background that features a screen with a pair of ballet dancers |
| High | Foreman | 13.33 | Involves panning and rapid scene change |

* - All video sequences played at 30 frames per second

**Table 5.1**: Video sequences used to study traffic characteristics

produce characterization charts. Trace files were also created from the RTP dump files, and used to input datagram size and timing information to the tool.

A Canon Vizcam 1000 video camera focused towards a continuously looping video sequence being played at 30 frames per second on the monitor of a computer fed the signal to a frame grabber card. The video sequences were made to loop so as to maintain the activity level. The UCL/LBNL version 2.8ucl-1.1.3 of the Video Conferencing Tool (VIC) was used to encode and transmit this video. The experiments were performed using the H.263 compression scheme. Keeping the quality level on the VIC User Interface a constant at 10, RTP traces were recorded for different maximum bandwidth settings (64, 128, 384 and 1536 kbps), and picture format (small - corresponding to QCIF, and normal - corresponding to CIF). Each experiment was conducted for a duration of exactly 15 minutes and analyzed, so that the video sequence has looped sufficient number of times to yield stable values. The computer that was used to conduct the experiments had an Intel PIII 930 Mhz processor with 128 MB RAM, Matrox Millenium G550 video card 32 MB video RAM, running Windows 2000 Professional.

### 5.1.2 Characteristics Charts

The charts depicted in Figures 5.1 through 5.5 describe the characteristics of the video traffic at the source. For the purposes of this study, Intracoded (I) frames were not considered. Only Predictive (P) frames, and the datagrams that carry P frame information (henceforth called P datagrams) were considered. The following observations were made regarding the characteristics of compressed real time video traffic:

- It can be observed from figures 5.1 and 5.2 [1] that on an average for most activity levels, one datagram is sufficient to carry one P frame in case of QCIF. Even in the case of High activity level, the maximum number of datagrams that constitute a P frame is just 3. For CIF resolution, the number of datagrams required to carry one P frame is greater than one for most activity levels, and the maximum number of datagrams that constitute a P frame tends to increase with an increase in activity level[2].
- For QCIF resolution, average datagram sizes tend to vary and increase in proportion with increase in activity level. Variations in mean datagram sizes also tend to increase with increase in activity level. However, for CIF resolution, average datagram sizes tend to be *fuller* for any amount of activity level, including for minimal motion. But with increase in activity level, the variation in mean sizes actually decreases, indicating that there is not much difference between the maximum and minimum datagram sizes.
- For the following combinations of inputs, the frame rates at the source are

[1]Values for datagram sizes and number of datagrams at a particular activity level varied slightly for different maximum encoding rates specified. The variations seemed to be unrelated, but the values were close together. So the mean of the values at different maximum encoding rates was taken to generate the graphs in figure 5.1 and 5.2.

[2]Except for the High activity level, *scene changes* because of the looping of the test video sequence are not significant, by visual inspection. For example, in the Minimal activity (Miss America) sequence, there is not much change in luminance and background color between the last and the first frames

successfully maintained at 30 fps irrespective of the activity level of the video, which is the ideal case in our study:

1. QCIF video with maximum encoding rate specified to be 384 kbps
2. QCIF video at 1536 kbps
3. CIF video at 1536 kbps

This implies that with the given host (sending end point), the above three combinations of picture resolutions and maximum encoding rate can produce 30 fps for any of the given activity levels. The reason why other combinations fall below the expected ideal rate of 30 fps, as shown in Figure 5.3, can be attributed to two independent factors:

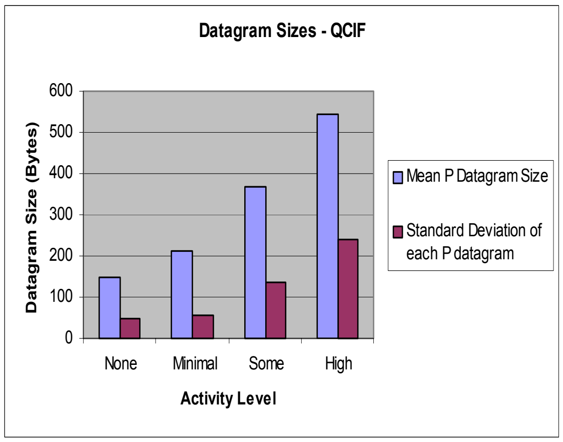
1. availability of host resources on the sending end point
2. The given input combination is less than perfect

The first point above is self explanatory, and implies that even though it is possible to generate 30 fps video for the given input, the CPU and memory resources on the source end point are insufficient and restricting the frame rate that is being generated at the source. The second point states that even though there may be enough resources on the host to generate 30 fps for the given input, either the maximum encoding rate specified is insufficient to generate 30 fps for the given activity level and picture resolution, or the picture resolution is too high for the given activity level and maximum encoding rate. It can be seen from Figure 5.3 that frame rates fall slower with increase in maximum encoding rate. Also, it can be seen that for CIF resolution video, 64 kbps and 128 kbps maximum encoding rates are highly insufficient for any amount of activity level.

- The charts in Figure 5.4 further supplement the observations made in the previous point. For a given combination of activity level and picture resolution, the output video is generated at a certain bit rate. If the maximum encoding rate is more than what is required, then the extra encoding bandwidth remains unused. However, in cases where the maximum encoding rate is specified to be less than what is actually required, then all the encoding bandwidth is used, but the output video frame rate is less than ideal. For example, in the CIF chart in Figure 5.4, we can see that at 64 kbps, the entire encoding bandwidth is always used (for all activity levels), but the frame rate reduces drastically with increase in activity level as seen in Figure 5.3. At the other extreme, the specified encoding rate of 1536 kbps is never fully utilized at any activity level or picture resolution, and the output frame rate is ideal at 30 fps.
- The average number of datagrams per second for maximum encoding rate specified at 1536 kbps generating 30 fps ideal video output is shown in Figure 5.5. The average number of datagrams remains almost the same irrespective of the activity level, in case of QCIF resolution. This is expected because for QCIF, the average number of datagrams per frame is 1 (Figure 5.1), and so a constant frame rate of 30 fps implies that the average datagrams per second is also 30. However, for CIF resolution, more than one datagram usually constitutes one frame, and so the average number of datagrams per second is far higher than 30, and the number tends to increase with an increase in activity level, because there is an increase in the number of datagrams that make up a frame.

Here is a summary of the observations made above:

- Datagram sizes tend to be *fuller* for CIF resolution video irrespective of activity



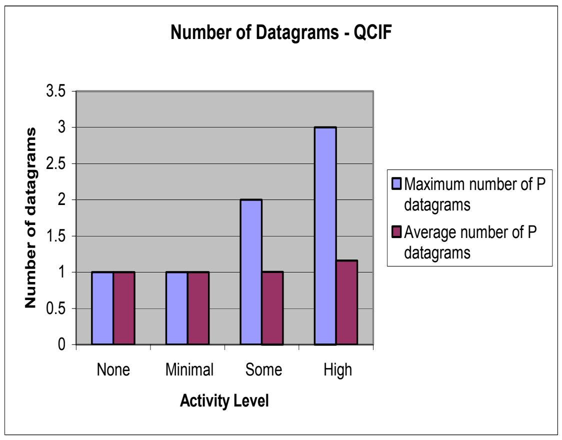


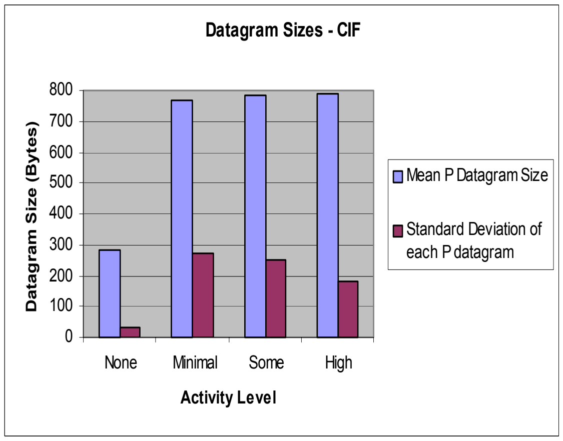
**Figure 5.1**: QCIF datagram characteristics

level, when compared to QCIF.

- The number of datagrams per frame for CIF resolution video are higher than for QCIF resolution, for a particular activity level
- Frame rates tend to drop with activity level if the picture resolution is high and/or a low maximum encoding rate is specified
- A particular combination of user inputs for activity level and picture resolution requires a certain maximum encoding rate to be specified in order to output 30 fps video.

## 5.2 Traffic Generator Characteristics

The goal of this experiment was to determine the effectiveness of the Iperf based traffic generator module by comparing the output from the tool with the information in the corresponding trace file that it uses as input. The output from the traffic generator was recorded using Ethereal, and the time instants of each datagram was used to produce the charts. The



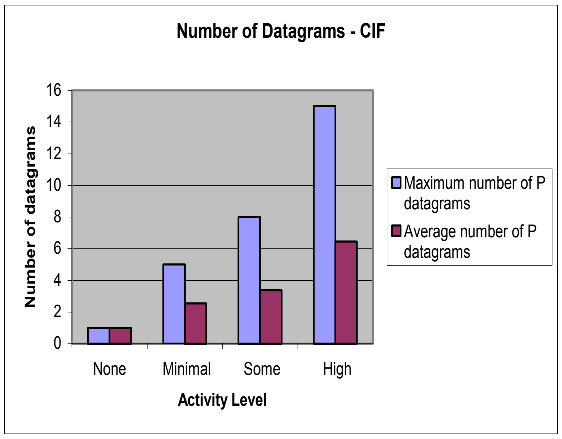


**Figure 5.2**: CIF datagram characteristics

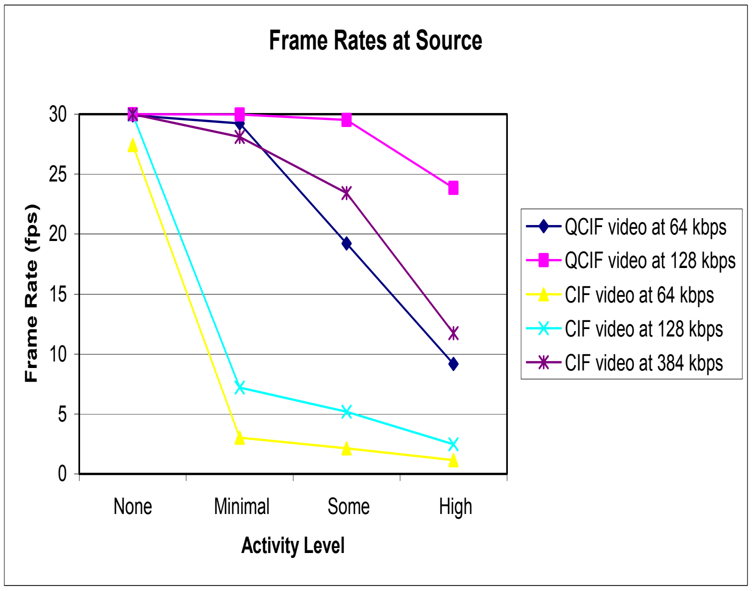


**Figure 5.3**: Frame rates at source

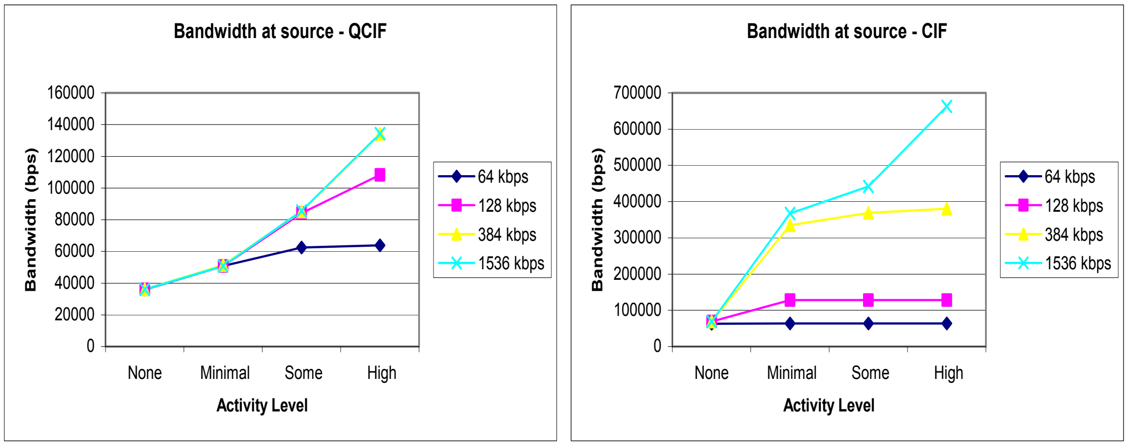


**Figure 5.4**: Maximum encoding rate vs. bandwidth at source

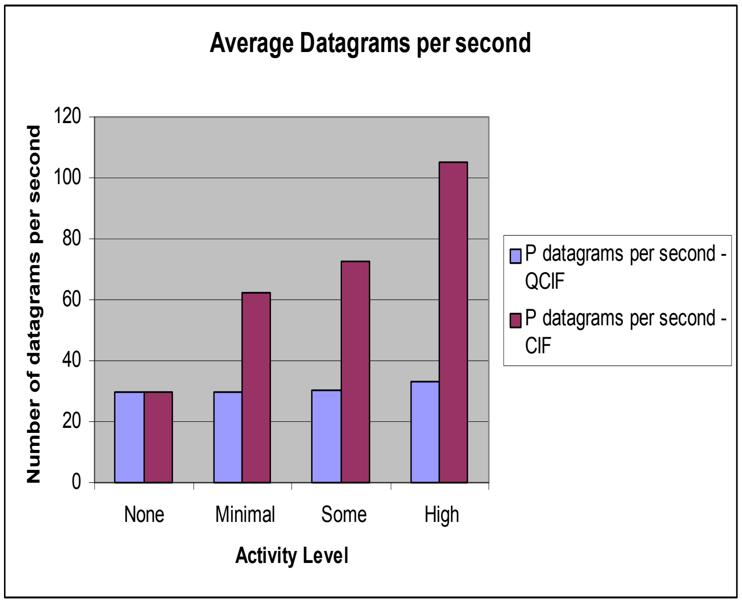


**Figure 5.5**: Average datagrams generated per second

combination of input parameters specified in the tool for these experiments was {Activity level = Some, Maximum Encoding Rate = 384 kbps, Picture Format = QCIF}. The outputs were compared with respect to the first 25 seconds. The observed frame rate in the actual trace and the tool output is 30 fps. The computer used for the experiment is an Intel PIII 930 Mhz, with 128 MB RAM, running Windows 2000 professional.

Two experiments were conducted with the traffic generator, one in which there was no significant CPU and memory load on the end point host being used for the experiment, and the other in which there was significant load. For the first experiment with no load, there were 24 processes executing on the host, taking up 101 MB of the total 310 MB committed memory available, and 66 MB of the 130 MB physical memory space available. For the second experiment, there were 41 processes executing, taking up 247 MB of the 310 MB available committed memory, and 122.5 MB of the 128 MB of RAM available. There was no other significant activity on the source computer while the experiment was being conducted. Since the traffic generator reads from the actual trace file, records the datagram size and the time instant at which each datagram was sent, and mimics that behavior, datagram sizes will be the same in both cases (see Section 4.2.2) and thus do not need comparison.

The three charts each in Figures 5.6 and 5.7 display the datagram timing information in the actual trace file, the traffic generator with no memory and CPU load on the generating host, and the traffic generator with memory and CPU load on the generating host respectively. Figure 5.6 shows the behavior of the generator in a 25 second period, and Figure 5.7 is a 2-second zoom in on a portion of the trace shown in Figure 5.6. Part a. in the Figures is plotted by taking the information contained in the source trace files directly. These trace files are the files that the traffic generator module reads in order to obtain the information it needs in order to generate and transmit test traffic. Part b. in the Figures is plotted by monitoring the test traffic datagrams output by the traffic generator module, when there is no other load affecting the source end point (and in turn the traffic generator module). Part c. in the Figures is plotted by monitoring the test traffic datagrams output by the traffic generator module, but this time with memory and CPU loads affecting the source end point (and in turn the traffic generator module). The two charts Part a. and Part b. in Figures 5.6 and 5.7 indicate that the traffic generator does indeed mimic the actual trace. The comparison may seem trivial because the generator reads from the same trace file to generate its traffic, and that traffic output is compared with the trace file again. But the challenge is to maintain exact (or almost the same) time difference between successive datagrams in the generator as provided in the trace file, and this is usually in terms of microseconds. Since the timing mechanism in Iperf for windows [38] had to be modified to improve accuracy, as mentioned in Section 4.2.2 this comparison assumed importance. The output of the traffic generator measured at the network level as shown in Part b. of the Figures 5.6 and 5.7, when compared with Part a. of the Figures, indicate that the Operating System (under no additional memory or CPU loads) does not seem to have any adverse affect on the working of the traffic generator module.

The traffic generator is limited in its capability to mimic the actual trace file by the time difference between successive datagrams. For very low time differences, of the order of a few hundred microseconds, there is a time lag between the actual trace file and the traffic generator. This is because the generator/transmitter code loop takes more than this duration of time for one iteration. These numbers may also vary with different processor speeds. The traffic generator output with respect to time is also affected by the amount of load on the computer where it is being executed. This effect can be seen in the upward spikes in the chart Part c. in each of the Figures 5.6 and 5.7, indicating more time taken between successive datagram transmissions. When many tasks are executing simultaneously, or when there is increased context switching, there is a time lag between the when the datagram is actually sent and the timing information in the trace file. The presence of increased network traffic such as broadcast traffic on the Network Interface Card (NIC) of the source end point also affects the traffic generator module similar to Part c. in Figures 5.6 and 5.7. So in essence, the traffic generator does a *best-effort* in sending the test data and transmits at a lower frame rate (fps) in comparison to the actual trace file as the load on the executing machine increases. However, this behavior would be beneficial to the tool, because the frame rate at the source would be indicative of the load on the Operating System at the source. The sending frame rate could be compared with the frame rate in the trace file to determine the amount of load on the sending end point.

## 5.3 Tool Requirements

In this section we describe the basic scalability issues involved with the tool. The SNMP MIB implementation consists of tables that hold information gathered during each unit test conducted with the tool. The MIB subtree grows with the number of unit tests performed, and the growth can be given as O(N), where N is the number of unit tests performed with the tool. The user is given an option to clear all existing values in the MIB subtree and thus manually manage the amount of memory utilized by the MIB implementation. In addition to these memory requirements, the parser module also maintains one cumulative set of values for every unit test, for the duration of a unit test. This is equivalent to the amount of memory required for containing one unit test worth of information in the SNMP MIB implementation.

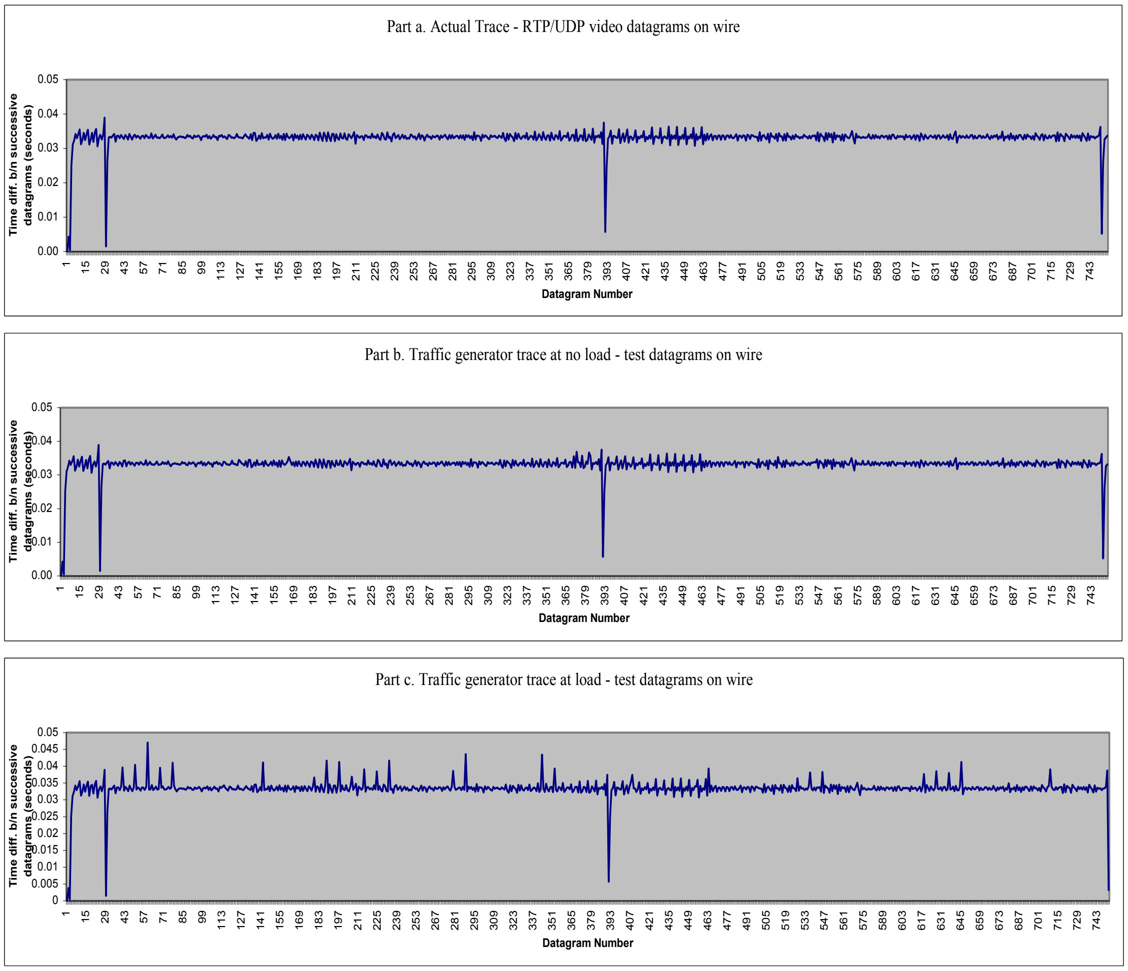


**Figure 5.6**: Traffic generator behavior for a 25 second sampling period

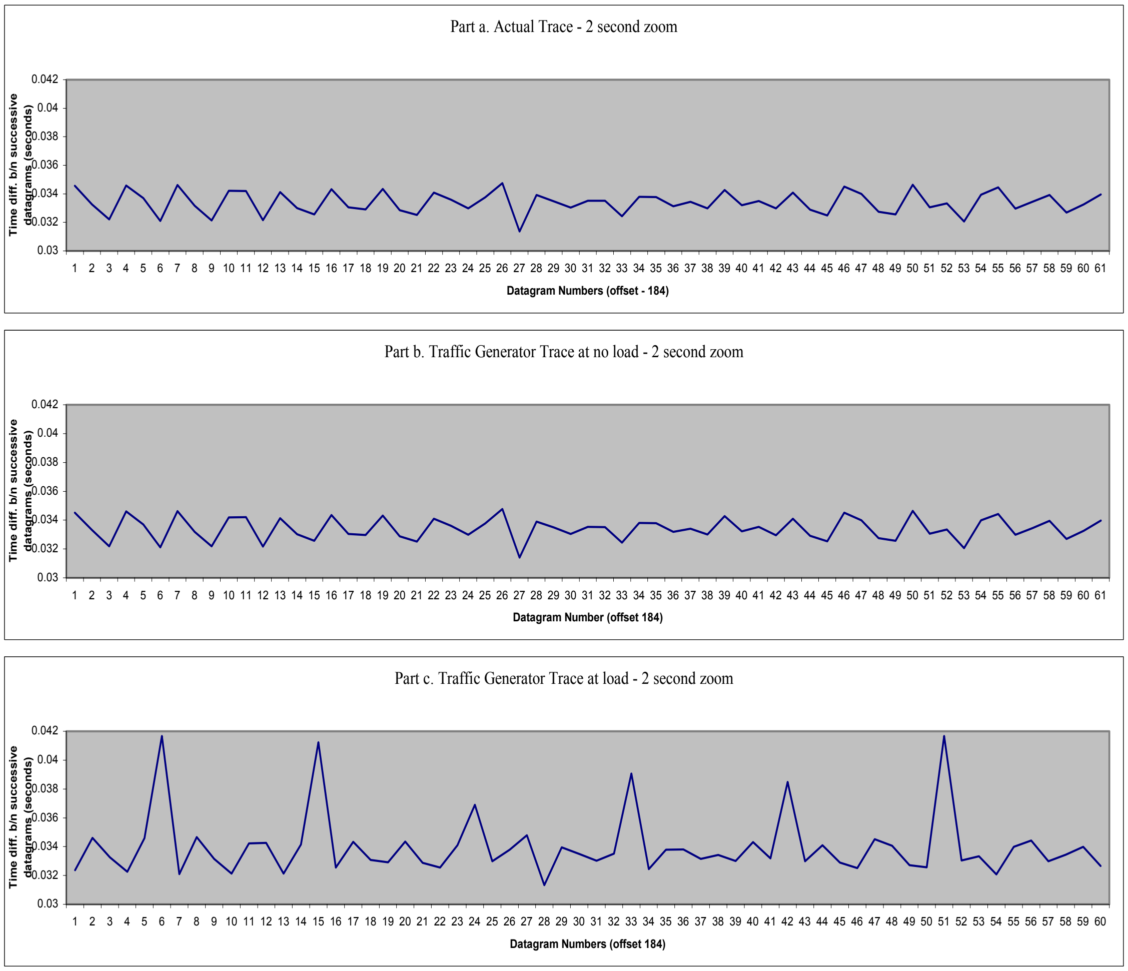


**Figure 5.7**: Traffic generator behavior for a 2 second zoom period

In terms of processor load, the sending end point places closes to a 100 percent load on the processor. This is because the tool internally uses an Iperf based traffic generator, and Iperf in turn utilizes 100 percent of CPU for generating and transmitting the traffic. Thus for the duration of every unit test, processor load on the sending end point is close to 100 percent. On the receiving end point, there is hardly any noticeable processor load. The Iperf traffic generator may be replaced by one of several other traffic generation tools, or one of the traffic modeling techniques discussed in Section 3.1, in order to improve the CPU utilization at the sending end point. The additional processes and threads created by the tool are two processes - the Graphical User Interface and the modified Iperf process, and one thread - the Iperf wrapper thread.

## 5.4 Tool Measurements

The objective in the following experiment was to execute the tool between two end points for varying network and host load conditions, and study the results obtained from the tool, in order to determine the affects of load on VBR test traffic.

### 5.4.1 Experimental Setup

The experimental setup for executing the tool and obtaining and analyzing the measurements it outputs, is given in Figure 5.8. The configurations of each of the machines used in the experiment are outlined in Table 5.2. For cross traffic creation (network load generation), the Iperf tool in its unmodified state was used in UDP mode, and CBR cross traffic utilizing different bandwidths was continuously sent for the duration of the tests. Two computers running the Redhat Linux Operating System were used as routers for packet

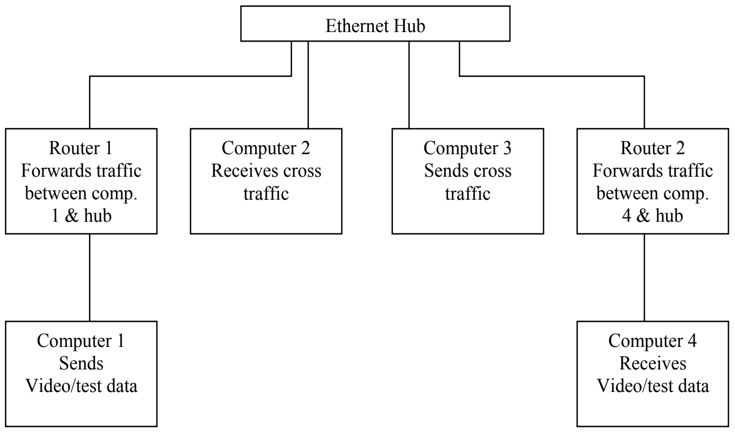

Ethernet Hub
Router 1 Forwards traffic between comp. 1 & hub
Computer 2 Receives cross traffic
Computer 3 Sends cross traffic
Router 2 Forwards traffic between comp. 4 & hub
Computer 1 Sends Video/test data
Computer 4 Receives Video/test data

**Figure 5.8**: Experimental setup diagram for tool measurements

forwarding, in order to effectively isolate the cross traffic at the hub from reaching the video test computers. The hub was used in order to force collisions to occur between cross traffic and test traffic.

Tool measurement experiments were conducted with the input parameter combinations {Activity level = Some, Maximum Encoding Rate = 384 kbps, Picture Format = QCIF} and {Activity level = Some, Maximum Encoding Rate = 384 kbps, Picture Format = CIF}. Ten unit tests of 30 seconds duration were conducted at 30 second intervals and the values were averaged out to obtain one set of results per test session. Test sessions were conducted for varying network traffic loads, as well as CPU and memory loads and the resulting values were plotted. All resulting values are determined by the tool residing on the receiving computer (computer 4).

| Component | Configuration |
|---|---|
| Ethernet hub | 3Com Super Stack II hub 10 (10 Mbps) |
| Computer 1 | Intel 933 Mhz PIII with 128 MB RAM running Windows 2000 |
| Computer 2 | Intel 399 Mhz PII with 128 MB RAM running Windows 2000 |
| Computer 3 | Intel 299 Mhz PII dual processor with 128 MB RAM running Red Hat Linux 7.2 |
| Computer 4 | Intel 200 Mhz with 64 MB RAM running Red Hat Linux 7.3 |
| Router 1 | Intel PIII 930 MHz with 128 MB RAM running Red Hat Linux 7.3 |
| Router 2 | Intel Pentium 199 MHz with 64 MB RAM running red Hat Linux 7.3 |

**Table 5.2**: Configuration of the various testing components for tool measurements

### 5.4.2 Measurement Charts

#### Variation with Network Load

For this set of experiments cross traffic with a specified bandwidth was continuously sent from computer 3 to computer 2 while the tool on computer 1 sent video test data to the tool residing on computer 4, in a setup as shown in Figure 5.8. The specified bandwidth for cross traffic was increased in stages from 0 to a maximum value at which most of the test datagrams failed to reach computer 4. The increments chosen for bandwidth values were non-uniform and successive increments depended upon how significant the difference in output was at those values. At each stage the output from the tool was recorded, and the information was used to plot the charts shown in Figures 5.9 to 5.14. The change in values were quite sharp and so plots with a logarithmic Y-axis were not very different from the plots with linear scales. So all the plots shown in this section are non-logarithmic.

Jitter values increased very slightly with increase in cross traffic until a certain threshold value, after which they increased drastically. This was true for both QCIF and CIF picture tests. In the case of CIF tests, the threshold value occurred at a lower value of cross traffic bandwidth (about 8000 Mbps) when compared to QCIF tests (about 9400 Mbps), as shown in Figures 5.9 and 5.10. At high cross traffic loads, average jitter was almost as high as maximum jitter for QCIF tests. At the same loads for CIF tests, maximum jitter values were far higher than average values.

Datagram losses were not encountered until the threshold value of cross traffic was reached. After the threshold value, losses shot up almost vertically. For QCIF tests, datagram losses were similar to frame losses, because each frame mostly consisted of just one datagram. On the other hand, datagram losses were far higher than frame losses for CIF tests, because many datagrams tend to constitute a single frame. The larger number of datagrams also explains the fact that percentage losses after the threshold value are far higher for QCIF tests when compared to CIF tests, as can be seen in Figures 5.11 and 5.12. The threshold value for losses in CIF tests occurred at about 9000 Mbps, where as that for QCIF tests remained at about 9400 Mbps.

The Y-axis on the frame rate and bandwidth plots denote the difference between the sent and the received frame rates. Frame rate and bandwidth values at the receiver decreased only after the threshold values were reached. Frame rate variations were very similar to variations in received bandwidth, as seen in figures 5.13 and 5.14. The threshold values for QCIF and CIF tests were the same as the respective values determined for losses due to cross traffic. For QCIF tests, frame rate decreased much more than in the case of CIF tests.

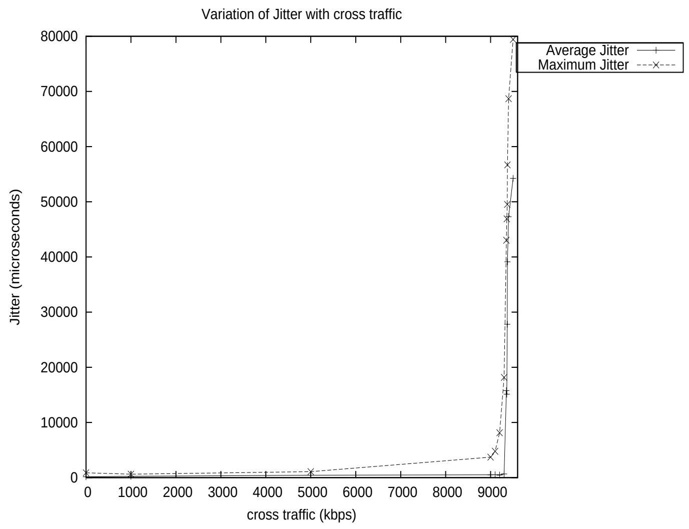


**Figure 5.9**: Jitter variation with network cross traffic (QCIF)

The reason for this is partly due the way frame rate calculations are made by the tool (see Section 3.2.3).

### Variation with Host Resource Load

The CPU and the virtual memory on the receiving end point (computer 4 in Figure 5.8) were varied to study the effect of host resource load on video communication. In the first set of experiments, CPU load on the receiving end point was increased in stages from 0 to 95 percent. Tests were conducted using the tool at each stage, and the resulting values were plotted graphically. These charts are given in Figures 5.15 to 5.20. In the second set of experiments, tests were conducted using the tool with the receiving end point running with low virtual memory. The values output by the tool for CPU load were used to plot the graphs shown in Figures 5.15 to 5.20. A discussion on the affect of memory load on the video communication is presented in Section 5.6.

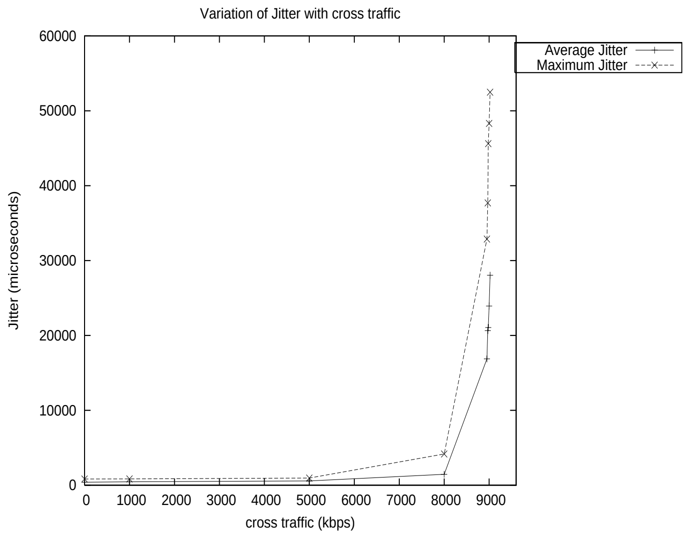


**Figure 5.10**: Jitter variation with network cross traffic (CIF)

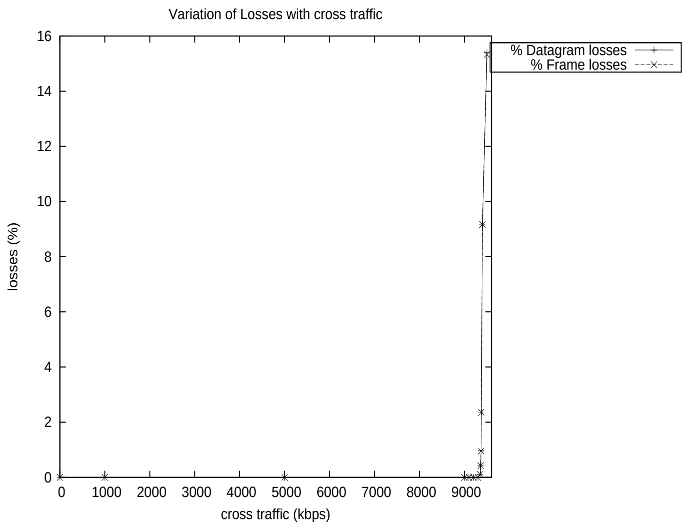


**Figure 5.11**: Loss variation with network cross traffic (QCIF)

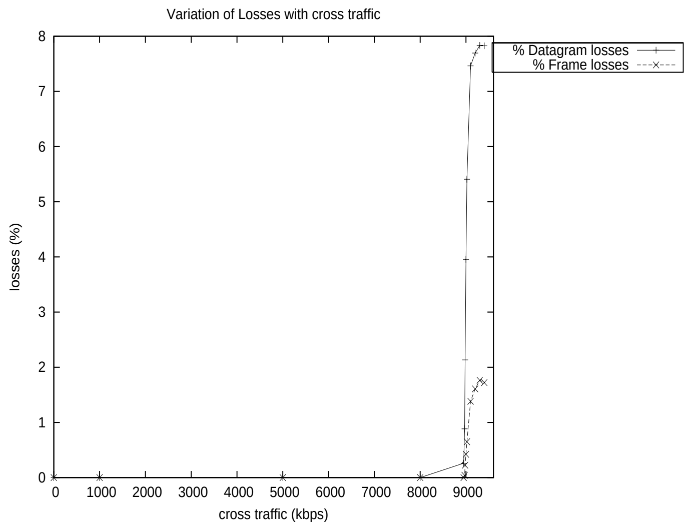


**Figure 5.12**: Loss variation with network cross traffic (CIF)

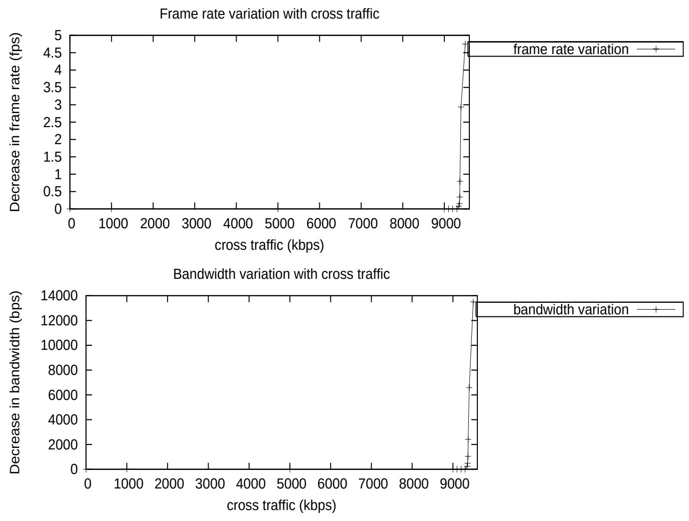


**Figure 5.13**: Frame and bit rate variation with network cross traffic (QCIF)

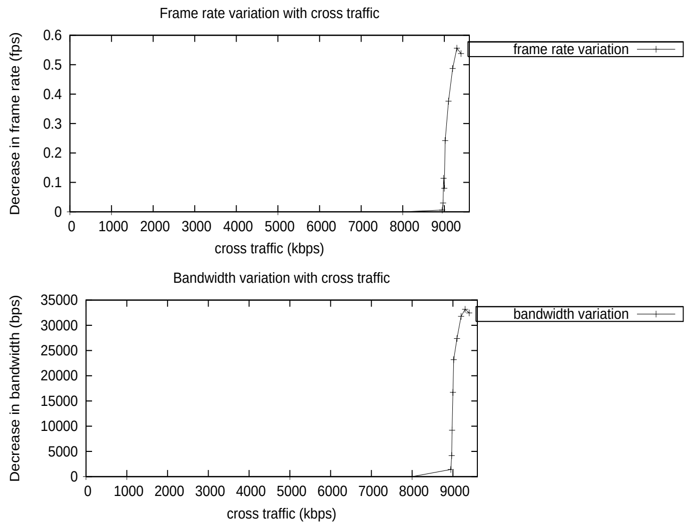


**Figure 5.14**: Frame and bit rate variation with network cross traffic (CIF)

Jitter values rose in proportion to CPU load, and jitter values were high at CPU loads less than 50 percent, as seen in Figures 5.9 and 5.10. Two interesting points to note are that irrespective of picture resolution (QCIF or CIF), maximum jitter values were always higher than average jitter values, and jitter variations with CPU load were very similar. This is different from jitter variation for QCIF and CIF tests with cross traffic loads discussed in Section 5.4.2.

Datagram and frame losses for QCIF occurred after a threshold value of CPU load (of about 60 percent) was reached. Since each frame mostly comprised a single datagram, datagram losses matched frame losses for QCIF tests as seen in Figure 5.17. For CIF tests, losses were encountered at much lower CPU loads, and increased almost linearly with CPU load, starting at about 20 percent CPU load. An important point to note here is that frame

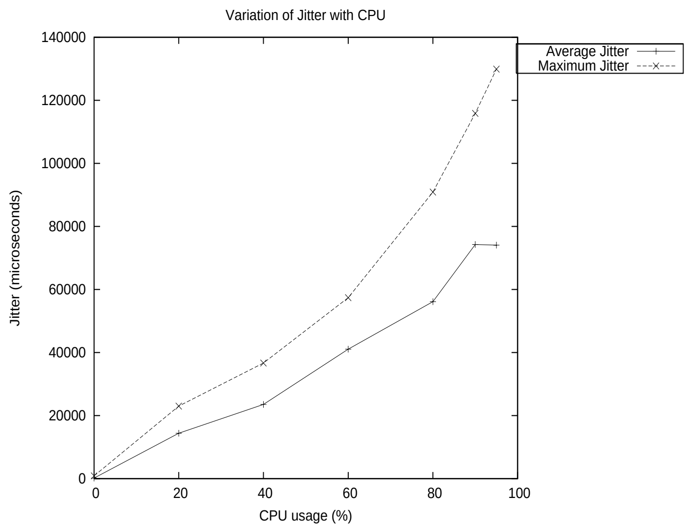


**Figure 5.15**: Jitter variation with CPU load (QCIF)

losses almost matched datagram losses, as can be seen in Figure 5.18, indicating that datagram losses were more bursty and thus whole frames were lost. Again, this is different from variation of datagram and frame losses with network cross traffic.

Frame rate and bandwidth remained constant until about 60 percent CPU load for QCIF tests, and 20 percent for CIF tests, and then began to decrease. However, the decrease was more linear for CIF tests when compared to QCIF tests, as shown in Figures 5.19 and 5.20. The decrease in frame rate for CIF tests with increase in CPU load (difference of about 16 fps at 90 percent CPU load) was far higher than in the case of QCIF tests (difference of about 3 fps at 90 percent CPU load).

A summary of the observations made with the tool measurements is given below:

- For network cross traffic, test data QoS metrics at the receiving end point varies minimally until a threshold value of cross traffic is reached. After this threshold

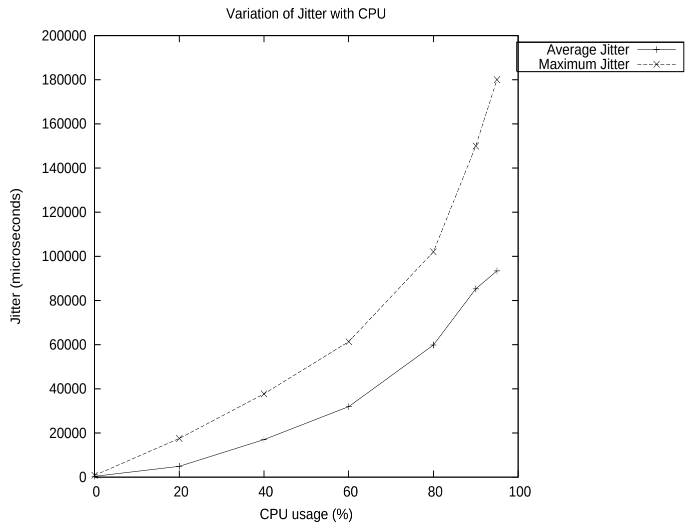


**Figure 5.16**: Jitter variation with CPU load (CIF)

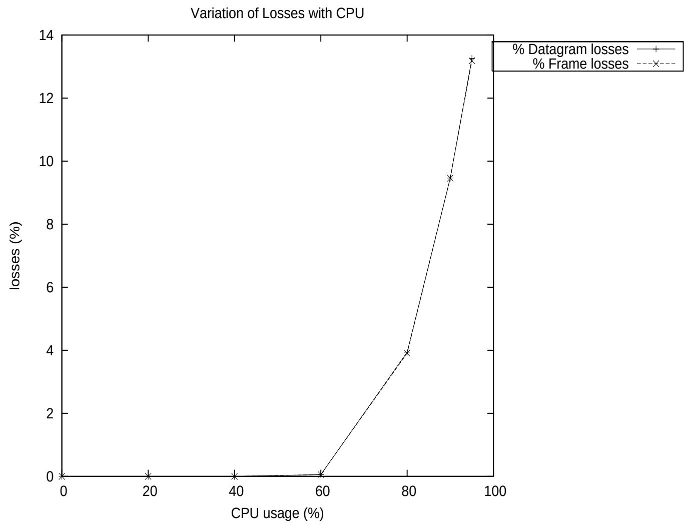


**Figure 5.17**: Loss variation with CPU load (QCIF)

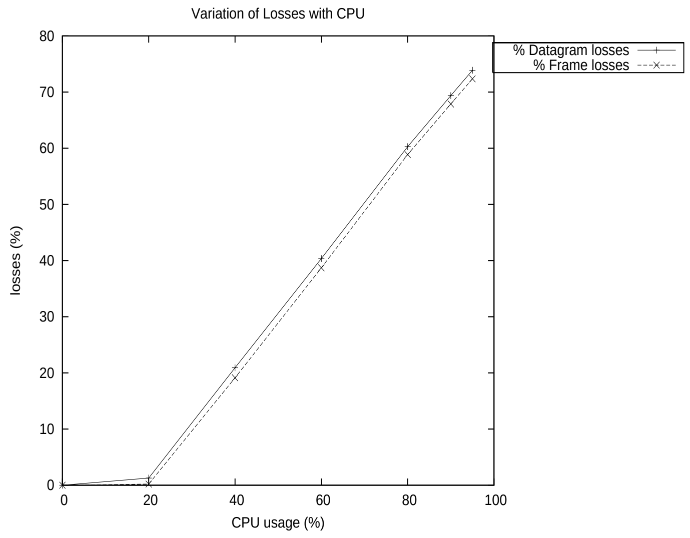


**Figure 5.18**: Loss variation with CPU load (CIF)

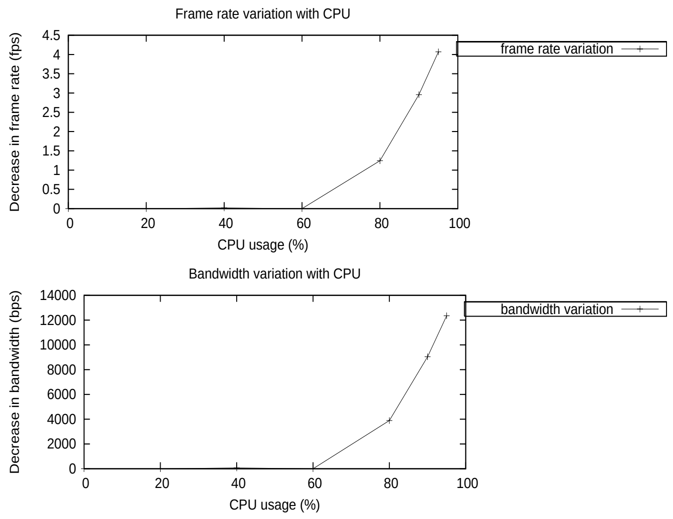


**Figure 5.19**: Frame and bit rate variation with CPU load (QCIF)

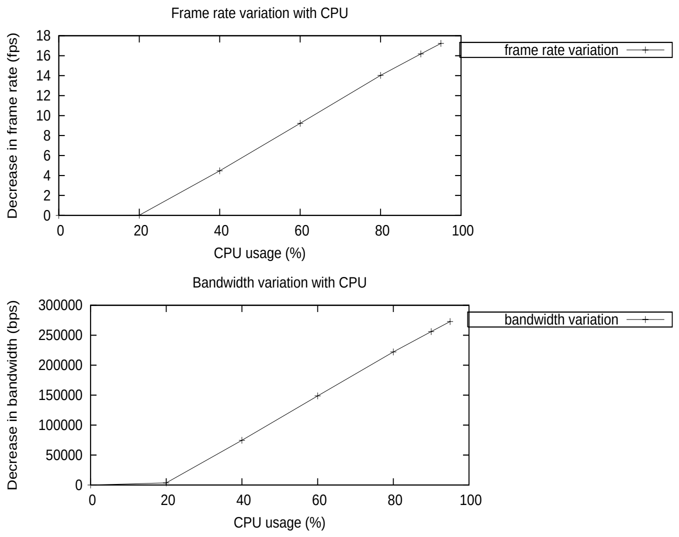


**Figure 5.20**: Frame and bit rate variation with CPU load (CIF)

value, any additional cross traffic drastically affects the communication

- The threshold value is higher for QCIF tests than for CIF tests, for network cross traffic
- The variation of QoS metrics with cross traffic for QCIF is similar to that for CIF, except in the case of frame losses
- For CPU loads, there is a threshold value after which losses and frame rates are affected by CPU load, similar to the affect of network cross traffic.
- unlike network cross traffic, jitter values are affected by small CPU loads. Also, jitter variations are similar in case of QCIF and CIF tests with varying CPU loads, unlike in the case of QCIF and CIF tests with varying network cross traffic
- The variation of QoS metrics with CPU load is more gradual when compared to the variation due to network cross traffic

- Frame losses are almost as high as datagram losses, and frame loss variation matches that of datagram loss, in case of CIF tests with varying CPU loads. Also, the loss numbers are significantly higher than in all other test scenarios

## 5.5 Effect of Load on Actual Video

In this section we discuss how an actual video communication is affected in terms of human perception by network and host resource loads, based on visual inspection of the resulting video. These experiments were performed with the same experimental setup as given in Figure 5.8. Instead of using the tool, the video software VIC v2.8ucl-1.1.3 was used to transmit actual video data between computers 1 and 4. The video sequence used for these tests was the "news" sequence. This sequence was played on the computer monitor and captured by the video hardware (similar to the setup explained in Section 5.1.1). The video at the receiving end point was observed, and the results obtained as perception quality metrics by visual inspection of both the video as well as the received frame rate, bit rate and loss values reported by VIC on its Graphical User Interface.

### 5.5.1 Effect of Network Cross Traffic

Cross traffic of 9400 kbps and 9200 kbps were applied for QCIF and CIF respectively, in order to determine the quality of the video output by VIC at the receiving end point. For QCIF video, VIC reported about 3 to 13 percent instantaneous packet losses. 8 to 9.5 percent losses were observed for CIF video. The other visual effects created by network cross traffic have been listed in Table 5.3, where all numbers (except frame rate) denote the extent of the effect, on a scale of 0 to 5, where 0 denotes no effect, and 5 denotes maximum effect. The values were observed by watching the video at the receiving end point.

| Test | Tiling | Encoding noise | Residue | Fr. rate decrease | Jerkiness | Others |
|---|---|---|---|---|---|---|
| QCIF, no load | 1 | 2 | 0 | 0 fps | 0 | |
| QCIF, with load | 3 | 4 | 3 | 2 to 5 fps | 3 | some blurring |
| CIF, no load | 1 | 2 | 0 | 0 fps | 0 | |
| CIF, with load | 2 | 4 | 3 | 1 to 4 fps | 4 | |

**Table 5.3**: Effect of network cross traffic on actual video

Even without traffic, there was noticeable encoding noise and some blurring at both the sending and the receiving end points. But the frame rate remained the same as the sending end point in the absence of cross traffic. With network cross traffic and QCIF video, the effects that stood out were that there was more tiling, remnant *ghost* residue, and some amount of jerkiness. For QCIF video, this jerkiness increased significantly, along with encoding noise. But there was not as much tiling in case of CIF video as in the case of QCIF video. Also, drop in frame rate was also not as much for CIF video, when compared to QCIF video.

### 5.5.2 Effect of Host Resource Load

Video software usually consumes significant amount of CPU resources, especially for CIF resolution picture. For example, on computer 4, it was found that the VIC software used about 35 to 40 percent of CPU for CIF video decoding and display, while ViaVideo utilized about 50 percent. So minor variations in CPU load (by other applications running on the

| Test | Tiling | Encoding noise | Residue | Fr. rate decrease | Jerkiness | Others |
|---|---|---|---|---|---|---|
| QCIF, no load | 1 | 2 | 0 | 0 | 0 | |
| QCIF, with load | 1 | 2 | 1 | 0 fps | 1 | high blurring, *slow-motion effect* |
| CIF, no load | 1 | 2 | 0 | 0 | 0 | |
| CIF, with load | 1 | 2 | 1 | 1 fps | 3 | less blurring, higher *slow-motion effect* |

**Table 5.4**: Effect of CPU load on actual video

host) tend to effect the video quality. Overall CPU load of 90 percent at the receiving end point was used to study the visual effects of CPU load on the received video. The effects have been listed in Table 5.4.

The most significant effect created by CPU load was the *slow-motion* effect. There was also a lack of increase in noise as well as a lack of increase in tiling and residue. There was a high amount of jerkiness but slightly less blurring in CIF video, when compared to QCIF video. Another significant effect in the case of CIF video was a drastic drop in the received instantaneous bit rate displayed by VIC. Surprisingly, VIC did not display any drop in frame rates for either QCIF or CIF video, even though the slow motion effect meant that the frame rate had to clearly be less than ideal.

## 5.6 End Notes

In this section, we discuss some issues with respect to video quality measurements, and conclude by suggesting a workflow application that connects the tool to a perceptual quality analysis tool.

The measurements and results discussed so far in this Chapter are all based on a single activity level. In general, as the activity level in the video increases, the number of datagrams and the sizes of individual datagrams tend to increase. Thus increased activity levels tend to be affected more by the same amount of load at the network, operating system or application levels.

We noticed for tool execution that the virtual memory loads on the receiving end point and its variations hardly affected the QoS metrics. Memory experiments on the tool were based on executing a memory loader application that gradually consumed virtual memory on the receiving end point. During this time, the tool was executed continuously, and the QoS metrics were monitored. Lack of virtual memory did not seem to affect the behavior of the tool or the results it displayed. However, other implications such as hard page faults and availability of dedicated video RAM were not tested, and these factors might impact the QoS metrics. When the memory loader application was executed to test its affect on actual video communication, a similar result was found. The received video picture did not seem to have any perceivable deterioration or defects.

The video perception tests with the Polycom ViaVideo videoconferencing solution yielded results that were dissimilar to the results obtained by VIC to some extent. The reasons for this can be attributed to the fact that ViaVideo uses the H.323 [15] umbrella standard and has built in mechanisms to measure and change the attributes of the sent traffic on the fly when it detects losses. One clear difference between the operation of VIC and ViaVideo in the presence of loads was the drastic drop in frame rates at the receiving end point with ViaVideo. At high loads, frame rates were reduced to near 1 fps. However, some other QoS metrics matched those indicated by the tool. For example, at high cross traffic loads, instantaneous jitter and loss values were reported to be about 65 milliseconds and 4 to 5 percent respectively, which is in the range of the values reported by the tool for similar cross traffic loads.

In the *real world*, the effect of load on the video tends to get compounded because the loads at different levels act in conjunction. This leads to more losses and jitter at lower individual loads than what was observed in this Chapter. Thus it becomes vital to gather all the different values and metrics for each individual test using the tool, in order to get the *big picture* and determine the reasons for a particular kind of video behavior.

The results and findings discussed in this Chapter are based on the H.263 encoding scheme. Video traffic that is generated using other encoding schemes may by affected differently by network and host loads, when compared to the H.263 encoding scheme. However, trace files may be generated using any encoding scheme and may then be used with the tool to test the end-to-end support. Thus the tool provides flexibility to test end-to-end support for various encoding schemes. To generate the trace files, a computer with adequate hardware resources may be used (see Section 5.1.2) in order to obtain consistent and optimal traces.

One advantage of obtaining results in quantitative form is that the frame rates and loss numbers provided by the tool may consequently be supplied as input to a perceptual quality analyzer model such as the Video Quality Metric (VQM) [57] software. The results obtained from the tool may be used to reconstruct a test video sequence, and the original and reconstructed video sequences then fed as input to the VQM software. VQM analyzes the sequences, and outputs an absolute number that denotes video quality on a scale of 0 to 1. It also indicates the probable root causes for the video degradation by specifying percentage blurring, jerkiness and block distortion. This interconnectivity between the tool and the VQM is underway. VQM integration and development is currently being researched in the department of Computer Science at UNC Chapel Hill.

Chapter 6

# Conclusion

This work studied the characteristics of compressed real time VBR video traffic, defined the various QoS metrics, and proposed a framework for end-to-end QoS analysis based on active measurement and testing. A tool was implemented based on the framework, and its effectiveness as an end-to-end QoS measurement test solution was studied. The results of the measurements of the tool under varying loads were described and analyzed. In this concluding Chapter, we present a summary of the achievements of this work, and suggest future enhancements that would improve and augment the capabilities of the tool.

## 6.1 Summary

The objectives for the development of the framework were to define the various QoS metrics that determine quality in real time video communication, and design a QoS measurement and analysis testing solution that would be modular and extensible. We began this work by categorizing video based on the amount of motion, and studying compressed real time video traffic characteristics by varying the amount of motion, the picture resolution and the maximum encoding rate. The study was based on the H.263 coding format, transmitted over the RTP/UDP/IP stack. This study provided useful insight into the how video traffic is generated and transmitted between endpoints. A framework was then designed to generate, transmit, receive and analyze test data that simulated real time VBR video traffic behavior. An SNMP MIB was specially designed for use with this framework to maintain QoS information that resulted from the tests. This MIB was used in conjunction with other standard MIBs to successfully provide all the metrics that determine QoS. A monitor program was designed and implemented to query end point MIBs, retrieve, collate and display the QoS information to the user. For the implementation of this framework, Iperf was chosen to generate traffic, and was modified to be able to generate video-like VBR test traffic. The tool was designed to allow the user to specify how the tests must be scheduled and run. The user could also vary test data traffic by specifying input parameters such as amount of motion, picture resolution and bandwidth. The tool was successfully executed under varying network and host loads, and test measurements were maintained locally at the end point MIB implementations. The monitor program was used to successfully query and retrieve the test measurements from the end points. The results from the test measurements indicated that the quality of the video is affected by the availability of both the resources on the network that connects the endpoints, as well as the resources on the endpoint hosts themselves, hence underlining the need for end-to-end measurement of the Quality of Service in order to determine support for compressed real time video traffic between end points.

## 6.2 Future Developments

Eventually, client based agents that participate in QoS analysis tests and maintain the resulting QoS information can be developed. Data can be collected and analyzed by the agents to provide an estimate of end user QoS, and possibly recommend remedies. Periodic as well as on-demand reports on performance and availability can be generated to assist in choosing client access patterns [21].

One enhancement to the existing framework can be to have a Policy Controller that is based on a network policy server [35]. The policy controller can regularly probe the MIB to identify changes in resource requirements and dynamically change its policies for resource reservation. This helps dynamically alleviate resource problems inside the Autonomous System in which the Policy Controller is implemented. With a policy controller, tests with different combinations of inputs can be automatically performed to indicate the available (or expectable) Quality of Service for the given pair of end points.

Another enhancement is to consider multiplexing several video test traffic data and test end-to-end performance. This is similar to having a video conference with more than one video capturing the information at each participating end point. Also, audio test traffic capabilities may be included, so that audio and video traffic can be multiplexed to mimic video conferencing, and end-to-end support can be determined.

Finally, the system can be extended to include more than two participating end-points, in a multicast scenario, as opposed to the unicast design currently in place. Appropriate extensions will then have to be made to the Video QoS SNMP MIB to include additional information that may be necessary for a multicast scenario.

# Appendix A

# List of Acronyms

| | |
|---|---|
| ASN | Abstract Syntax Notation |
| ATM | Asynchronous Transfer Mode |
| CBR | Constant Bit Rate |
| CIF | Common Intermediate Format |
| DCT | Discrete Cosine Transform |
| fps | Frames per second |
| GOB | Group Of Blocks |
| GOP | Group Of Pictures |
| GUI | Graphical User Interface |
| I Frame | Intracoded Frame |
| IANA | Internet Assigned Numbers Authority |
| IETF | Internet Engineering Task Force |
| IP | Internet Protocol |
| ISO | International Organization for Standardization |
| ITU-T | International Telecommunications Union - Telecom |
| LRD | Long Range Dependence |

| | |
|---|---|
| MIB | Management Information Base |
| MPEG | Moving Picture Experts Group |
| NTP | Network Time Protocol |
| OID | Object Identifier |
| P Frame | Predictive-coded Frame |
| PSTN | Public Switched Telephone Network |
| QCIF | Quarter Common Intermediate Format |
| QoS | Quality of Service |
| RTCP | Real time Transport Control Protocol |
| RTP | Real time Transport Protocol |
| SNMP | Simple Network Management Protocol |
| SRD | Short Range Dependence |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |
| VBR | Variable Bit Rate |
| VIC | Video Conferencing tool |
| VQM | Video Quality Metric (software) |

# Appendix B

# Video QoS MIB

```
VIDQOS-MIB DEFINITIONS ::= BEGIN
IMPORTS
          Counter32, Counter64, Gauge32, Integer32,
          MODULE-IDENTITY,
          OBJECT-TYPE                FROM SNMPv2-SMI
          DisplayString              FROM RFC1213-MIB;

vidQoS MODULE-IDENTITY
          – format is YYYYMMDDHHMMZ
          LAST-UPDATED               "200325030000Z" – March 3, 2003
          ORGANIZATION
                                     "Video QoS Team, NCNI
                                     Email: h323@renoir.csc.ncsu.edu"
          CONTACT-INFO
                                     "Vinay Chandrasekhar
                                     Graduate Student
                                     Department of Computer Science
                                     NC State University
                                     Raleigh, NC - 27606.

                                     Tel:+1 919 515 0138
                                     Email: vchandr@unity.ncsu.edu"
          DESCRIPTION
                                     "The QoS metrics that must be managed for
                                     running end-to-end video capability tests
                                     between given endpoints. The MIB is comprised
                                     of four different parts:
                                     1. Contact Information - contains general contact
                                     information
                                     2. Host System Information - System parameters that
                                     describe the QoS metrics of the system that
                                     participates in the tests.
                                     3. QoS Information - QoS metrics that
                                     describe the behavior of various elements of video
                                     communication.
                                     4. Logs - Logs of the errors and events that occur during
                                     testing."
          REVISION                   "200325030000Z" – March 3, 2003
          DESCRIPTION                "Initial Release"

::= { enterprises 55555 }
–
– OBJECTS
–

vqContact        OBJECT IDENTIFIER ::= { vidQoS 1 }
vqSystem         OBJECT IDENTIFIER ::= { vidQoS 2 }
vqQoS            OBJECT IDENTIFIER ::= { vidQoS 3 }
vqLogs           OBJECT IDENTIFIER ::= { vidQoS 4 }

–
– Contact Information
–
ctGroup OBJECT-TYPE
          SYNTAX            DisplayString
          MAX-ACCESS        read-only
          STATUS            mandatory
          DESCRIPTION
                    "This object is used to display the name of the Group and / or Organization."
          ::= { vqContact 1 }

ctEmail OBJECT-TYPE
          SYNTAX            DisplayString
          MAX-ACCESS        read-only
          STATUS            mandatory
          DESCRIPTION
                    "This object is used to display the email of the group contact."
          ::= { vqContact 2 }

ctVersion OBJECT-TYPE
          SYNTAX            DisplayString
          MAX-ACCESS        read-only
          STATUS            mandatory
          DESCRIPTION
```

```
                    “This object is used to display the current version of the software system release.”
          ::= { vqContact 3 }
–
– Host System Information
–

– Sender Table
sysSenderTable OBJECT-TYPE
          SYNTAX              SEQUENCE OF sysSenderEntry
          MAX-ACCESS          not-accessible
          STATUS              current
          DESCRIPTION
                    “There’s one entry in the sysSenderTable for each individual test conducted by this
                    host. The test Number gives the serial number of the test, and the session ID tells which set of tests
                    belong to one session.“
          ::= { vqSystem 1 }

sysSenderEntry OBJECT-TYPE
          SYNTAX              sysSenderEntry
          MAX-ACCESS          not-accessible
          STATUS              current
          DESCRIPTION
                    “Data in sysSenderTable maintains HR metrics in a VidQoS test. A host agent
                    MUST create a read-only row for each test where datagrams are sent. Rows MUST be created by
                    the host Agent at the start of a test. The ssSendIndex will uniquely identify each test.”
          INDEX               { ssSendIndex }
          ::= { sysSenderTable 1 }

sysSenderEntry ::= SEQUENCE {
          ssSendIndex            Integer32,
          ssPkPhyMem             Gauge32,
          ssLastPhyMem           Integer32,
          ssPkCommMem            Gauge32,
          ssLastCommMem          Integer32,
          ssPkCPUTime            Gauge32,
          ssLastCPUTime          Integer32,
          ssPkPageFaults         Gauge32
}

ssSendIndex OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    “The index of the conceptual row which uniquely identifies the
                    test. Follows the values in the QoS subtree Send Index.”
          ::= { sysSenderEntry 1 }

ssPkPhyMem OBJECT-TYPE
          SYNTAX              Gauge32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    “The peak instantaneous physical memory used by this host
                    during the test, in bytes.”
          ::= { sysSenderEntry 2 }

ssLastPhyMem OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    “The Last instantaneous value of physical memory used by this
                    host during the test, in bytes.”
          ::= { sysSenderEntry 3 }

ssPkCommMem OBJECT-TYPE
          SYNTAX              Gauge32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    “The peak instantaneous Virtual memory used by this host
                    during the test, in bytes.”
          ::= { sysSenderEntry 4 }

ssLastCommMem OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    “The Last instantaneous value of virtual memory used by this
                    host during the test, in bytes.”
          ::= { sysSenderEntry 5 }

ssPkCPUTime OBJECT-TYPE
          SYNTAX              Gauge32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    “The peak CPU used by this host during the test, as a percentage
                    value.”
logTable OBJECT-TYPE
          SYNTAX              SEQUENCE OF LogEntry
          MAX-ACCESS          not-accessible
          STATUS              current
          DESCRIPTION
                    “There’s one entry in the logTable for each individual
                    log entered by the test application. This could contain both
                    errors generated while execution, as well as indicate events
                    that occur during the runtime of the system.”
          ::= { vqLogs 1 }

LogEntry OBJECT-TYPE
          SYNTAX              LogEntry
          MAX-ACCESS          not-accessible
          STATUS              current
          DESCRIPTION
```

```
                    "Data in logTable signifies errors and events during runtime of the
                    testing application. A host agent MUST create a read-only row for
                    each log. Rows created by the agent MUST be be retained until the
                    SNMP service is brought down. The logIndex will uniquely identify
                    each test."
          INDEX                { lgLogIndex }
          ::= { logTable 1 }

LogEntry          ::= SEQUENCE {
          lgLogIndex                 Integer32,
          lgLogTime                  DisplayString,
          lgLogTitle                 DisplayString,
          lgLogType                  Integer32,
          lgLogSessionTestNum Integer32,
          lgLogDescription           DisplayString
}

lgLogIndex OBJECT-TYPE
          SYNTAX               Integer32
          MAX-ACCESS           read-only
          STATUS               current
          DESCRIPTION
                    "The index of the conceptual row which uniquely identifies this log."
          ::= { LogEntry 1 }

lgLogTime OBJECT-TYPE
          SYNTAX               DisplayString
          MAX-ACCESS           read-only
          STATUS               current
          DESCRIPTION
                    "The time at which this error/event occured, as a string."
          ::= { LogEntry 2 }

lgLogTitle OBJECT-TYPE
          SYNTAX               DisplayString
          MAX-ACCESS           read-only
          STATUS               current
          DESCRIPTION
                    "The title of this log."
          ::= { LogEntry 3 }

lgLogType OBJECT-TYPE
          SYNTAX               Integer32
          MAX-ACCESS           read-only
          STATUS               current
          DESCRIPTION
                    "The log category to which this log belongs (Define different
                    categories here)."
          ::= { LogEntry 4 }

lgLogSessionTestNum OBJECT-TYPE
          SYNTAX               Integer32
          MAX-ACCESS           read-only
          STATUS               current
          DESCRIPTION
                    "The session test number to which this log applies."
          ::= { LogEntry 5 }

lgLogDescription OBJECT-TYPE
          SYNTAX               DisplayString
          MAX-ACCESS           read-only
          STATUS               current
          DESCRIPTION
                    "The description of the log."
          ::= { LogEntry 6 }

END

          ::= { sysSenderEntry 6 }

ssLastCPU OBJECT-TYPE
          SYNTAX               Integer32
          MAX-ACCESS           read-only
          STATUS               current
          DESCRIPTION
                    "The Last value of CPU used by this host during the test, as a
                    percentage value."
          ::= { sysSenderEntry 7 }

ssPkPageFaults OBJECT-TYPE
          SYNTAX               Gauge32
          MAX-ACCESS           read-only
          STATUS               current
          DESCRIPTION
                    "The peak number of page faults by this host
                    during the test."
          ::= { sysSenderEntry 8 }

– Receiver Table

sysReceiverTable OBJECT-TYPE
          SYNTAX               SEQUENCE OF sysReceiverEntry
          MAX-ACCESS           not-accessible
          STATUS               current
          DESCRIPTION
                    "There's one entry in the sysReceiverTable for each individual
                    test conducted with this system. The test Number gives the serial
                    number of the test, and the session ID tells which set of tests
                    belong to one session."
          ::= { vqSystem 2 }

sysReceiverEntry OBJECT-TYPE
          SYNTAX               sysReceiverEntry
          MAX-ACCESS           not-accessible
          STATUS               current
          DESCRIPTION
"Data in sysReceiverTable maintains HR metrics in a VidQoS test. A
                    host agent MUST create a read-only row for each test where datagrams
                    are received. Rows MUST be created by the host Agent at the
                    start of a test. The srRecvIndex will uniquely identify each test."
          INDEX                   { srRecvIndex }
          ::= { sysReceiverTable 1 }

sysReceiverEntry              ::= SEQUENCE {
          srRecvIndex                   Integer32,
          srPkPhyMem                    Gauge32,
          srLastPhyMem                  Integer32,
          srPkCommMem                   Gauge32,
          srLastCommMem                 Integer32,
          srPkCPUTime                   Gauge32,
          srLastCPUTime                 Integer32,
          srPkPageFaults                Gauge32
}

srRecvIndex OBJECT-TYPE
          SYNTAX                  Integer32
          MAX-ACCESS              read-only
          STATUS                  current
          DESCRIPTION
                    "The index of the conceptual row which uniquely identifies the test.
                    Follows the values in the QoS subtree Recv Index."
          ::= { sysReceiverEntry 1 }

srPkPhyMem OBJECT-TYPE
          SYNTAX                  Gauge32
          MAX-ACCESS              read-only
          STATUS                  current
          DESCRIPTION
                    "The peak instantaneous physical memory used by this host
                    during the test, in bytes."
          ::= { sysReceiverEntry 2 }

srLastPhyMem OBJECT-TYPE
          SYNTAX                  Integer32
          MAX-ACCESS              read-only
          STATUS                  current
          DESCRIPTION
                    "The Last instantaneous value of physical memory used by this
                    host during the test, in bytes."
          ::= { sysReceiverEntry 3 }

srPkCommMem OBJECT-TYPE
          SYNTAX                  Gauge32
          MAX-ACCESS              read-only
          STATUS                  current
          DESCRIPTION
                    "The peak instantaneous Virtual memory used by this host
                    during the test, in bytes."
          ::= { sysReceiverEntry 4 }

srLastCommMem OBJECT-TYPE
          SYNTAX                  Integer32
          MAX-ACCESS              read-only
          STATUS                  current
          DESCRIPTION
                    "The Last instantaneous value of virtual memory used by this
                    host during the test, in bytes."
          ::= { sysReceiverEntry 5 }

srPkCPUTime OBJECT-TYPE
          SYNTAX                  Gauge32
          MAX-ACCESS              read-only
          STATUS                  current
          DESCRIPTION
                    "The peak CPU used by this host
                    during the test, as a percentage value."
          ::= { sysReceiverEntry 6 }

srLastCPU OBJECT-TYPE
          SYNTAX                  Integer32
          MAX-ACCESS              read-only
          STATUS                  current
          DESCRIPTION
                    "The Last value of CPU used by this
                    host during the test, as a percentage value."
          ::= { sysReceiverEntry 7 }

srPkPageFaults OBJECT-TYPE
          SYNTAX                  Gauge32
          MAX-ACCESS              read-only
          STATUS                  current
          DESCRIPTION
                    "The peak number of page faults by this host
                    during the test."
          ::= { sysReceiverEntry 8 }
–
– QoS Information
–

– Sender Table

qosSenderTable OBJECT-TYPE
          SYNTAX                  SEQUENCE OF qosSenderEntry
          MAX-ACCESS              not-accessible
          STATUS                  current
          DESCRIPTION
                    "There's one entry in the qosSenderTable for each individual
                    test conducted by this system. The test Number gives the serial
                    number of the test, and the session ID tells which set of tests
                    belong to one session."
          ::= { vqQoS 1 }

qosSenderEntry OBJECT-TYPE
          SYNTAX                  qosSenderEntry
          MAX-ACCESS              not-accessible
          STATUS                  current
          DESCRIPTION
                    "Data in qosSenderTable uniquely describes a VidQoS test. A
                    host agent MUST create a read-only row for each test where datagrams
                    are sent. Rows MUST be created by the host Agent at the start of
                    a test. The sendIndex will uniquely identify each test."
          INDEX               { sndSendIndex }
          ::= { qosSenderTable 1 }

qosSenderEntry            ::= SEQUENCE {
          sndSendIndex              Integer32,
          sndSessionID              Integer32,
          sndSessionTestNum         Integer32,
          sndvidTestStartTime       DisplayString,
          sndVidTestDuration        Counter32,
          sndVidActivityLevel       Integer32,
          sndVidPictureFmt          Integer32,
          sndVidMaxBW               Integer32,
          sndVidDataBytesSent       Counter64,
          sndVidDataBWSent          Gauge32,
          sndVidDataDgramsSent      Counter64,
          sndVidDataFramesSent      Counter64,
          sndVidDataFrameRate       Gauge32,
          sndvidTestStatus          Integer32
}

sndSendIndex OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The index of the conceptual row which uniquely identifies the test."
          ::= { qosSenderEntry 1 }

sndSessionID OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The ID of the session. Several tests may be performed as part of one
                    session."
          ::= { qosSenderEntry 2 }

sndSessionTestNum OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "This object uniquely identifies a test within a given session."
          ::= { qosSenderEntry 3 }

sndvidTestStartTime OBJECT-TYPE
          SYNTAX              DisplayString
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The time on the local computer this test was started."
          ::= { qosSenderEntry 4 }

sndTestDuration OBJECT-TYPE
          SYNTAX              Counter32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The time duration of this test, in seconds."
          ::= { qosSenderEntry 5 }

sndvidActivityLevel OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The activity level chosen for this test. A Value 0 could
                    indicate a still picture, 1 could indicate a picture with
                    some activity and so on."
          ::= { qosSenderEntry 6 }

sndvidPictureFmt OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The picture format chosen for this test. 0 could indicate
                    QCIF, 1 could indicate CIF and so on."
          ::= { qosSenderEntry 7 }

sndvidMaxBW OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The maximum encoding bandwidth chosen for this test, in kbps. "
          ::= { qosSenderEntry 8 }

logTable OBJECT-TYPE
          SYNTAX              SEQUENCE OF LogEntry
          MAX-ACCESS          not-accessible
          STATUS              current
          DESCRIPTION
                    "There's one entry in the logTable for each individual
                    log entered by the test application. This could contain both
                    errors generated while execution, as well as indicate events
                    that occur during the runtime of the system."
          ::= { vqLogs 1 }

LogEntry OBJECT-TYPE
          SYNTAX              LogEntry
          MAX-ACCESS          not-accessible
          STATUS              current
          DESCRIPTION
                    "Data in logTable signifies errors and events during runtime of the
                    testing application. A host agent MUST create a read-only row for
                    each log. Rows created by the agent MUST be be retained until the
                    SNMP service is brought down. The logIndex will uniquely identify
                    each test."
          INDEX              { lgLogIndex }
          ::= { logTable 1 }

LogEntry            ::= SEQUENCE {
logTable OBJECT-TYPE
          SYNTAX             SEQUENCE OF LogEntry
          MAX-ACCESS         not-accessible
          STATUS             current
          DESCRIPTION
                    "There's one entry in the logTable for each individual
                    log entered by the test application. This could contain both
                    errors generated while execution, as well as indicate events
                    that occur during the runtime of the system."
          ::= { vqLogs 1 }

LogEntry OBJECT-TYPE
          SYNTAX             LogEntry
          MAX-ACCESS         not-accessible
          STATUS             current
          DESCRIPTION
                    "Data in logTable signifies errors and events during runtime of the
                    testing application. A host agent MUST create a read-only row for
                    each log. Rows created by the agent MUST be be retained until the
                    SNMP service is brought down. The logIndex will uniquely identify
                    each test."
          INDEX              { lgLogIndex }
          ::= { logTable 1 }

LogEntry            ::= SEQUENCE {
          lgLogIndex             Integer32,
          lgLogTime              DisplayString,
          lgLogTitle             DisplayString,
          lgLogType              Integer32,
          lgLogSessionTestNum    Integer32,
          lgLogDescription       DisplayString
}

lgLogIndex OBJECT-TYPE
          SYNTAX             Integer32
          MAX-ACCESS         read-only
          STATUS             current
          DESCRIPTION
                    "The index of the conceptual row which uniquely identifies this log."
          ::= { LogEntry 1 }

lgLogTime OBJECT-TYPE
          SYNTAX             DisplayString
          MAX-ACCESS         read-only
          STATUS             current
          DESCRIPTION
                    "The time at which this error/event occured, as a string."
          ::= { LogEntry 2 }

lgLogTitle OBJECT-TYPE
          SYNTAX             DisplayString
          MAX-ACCESS         read-only
          STATUS             current
          DESCRIPTION
                    "The title of this log."
          ::= { LogEntry 3 }

lgLogType OBJECT-TYPE
          SYNTAX             Integer32
          MAX-ACCESS         read-only
          STATUS             current
          DESCRIPTION
                    "The log category to which this log belongs (Define different
                    categories here)."
          ::= { LogEntry 4 }

lgLogSessionTestNum OBJECT-TYPE
          SYNTAX             Integer32
          MAX-ACCESS         read-only
          STATUS             current
          DESCRIPTION
                    "The session test number to which this log applies."
          ::= { LogEntry 5 }

lgLogDescription OBJECT-TYPE
          SYNTAX             DisplayString
          MAX-ACCESS         read-only
          STATUS             current
          DESCRIPTION
                    "The description of the log."
          ::= { LogEntry 6 }

END

          lgLogIndex             Integer32,
          lgLogTime              DisplayString,
          lgLogTitle             DisplayString,
          lgLogType              Integer32,
          lgLogSessionTestNum    Integer32,
          lgLogDescription       DisplayString
}

lgLogIndex OBJECT-TYPE
          SYNTAX             Integer32
          MAX-ACCESS         read-only
          STATUS             current
          DESCRIPTION
                    "The index of the conceptual row which uniquely identifies this log."
          ::= { LogEntry 1 }

lgLogTime OBJECT-TYPE
          SYNTAX             DisplayString
          MAX-ACCESS         read-only
          STATUS              current
          DESCRIPTION
                    "The time at which this error/event occured, as a string."
          ::= { LogEntry 2 }

lgLogTitle OBJECT-TYPE
          SYNTAX              DisplayString
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The title of this log."
          ::= { LogEntry 3 }

lgLogType OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The log category to which this log belongs (Define different
                    categories here)."
logTable OBJECT-TYPE
          SYNTAX              SEQUENCE OF LogEntry
          MAX-ACCESS          not-accessible
          STATUS              current
          DESCRIPTION
                    "There's one entry in the logTable for each individual
                    log entered by the test application. This could contain both
                    errors generated while execution, as well as indicate events
                    that occur during the runtime of the system."
          ::= { vqLogs 1 }

LogEntry OBJECT-TYPE
          SYNTAX              LogEntry
          MAX-ACCESS          not-accessible
          STATUS              current
          DESCRIPTION
                    "Data in logTable signifies errors and events during runtime of the
                    testing application. A host agent MUST create a read-only row for
                    each log. Rows created by the agent MUST be be retained until the
                    SNMP service is brought down. The logIndex will uniquely identify
                    each test."
          INDEX               { lgLogIndex }
          ::= { logTable 1 }

LogEntry          ::= SEQUENCE {
          lgLogIndex             Integer32,
          lgLogTime              DisplayString,
          lgLogTitle             DisplayString,
          lgLogType              Integer32,
          lgLogSessionTestNum Integer32,
          lgLogDescription       DisplayString
}

lgLogIndex OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The index of the conceptual row which uniquely identifies this log."
          ::= { LogEntry 1 }

lgLogTime OBJECT-TYPE
          SYNTAX              DisplayString
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The time at which this error/event occured, as a string."
          ::= { LogEntry 2 }

lgLogTitle OBJECT-TYPE
          SYNTAX              DisplayString
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The title of this log."
          ::= { LogEntry 3 }

lgLogType OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The log category to which this log belongs (Define different
                    categories here)."
          ::= { LogEntry 4 }

lgLogSessionTestNum OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The session test number to which this log applies."
          ::= { LogEntry 5 }

lgLogDescription OBJECT-TYPE
          SYNTAX              DisplayString
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The description of the log."
          ::= { LogEntry 6 }

END

          ::= { LogEntry 4 }

lgLogSessionTestNum OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The session test number to which this log applies."
          ::= { LogEntry 5 }

lgLogDescription OBJECT-TYPE
          SYNTAX              DisplayString
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The description of the log."
          ::= { LogEntry 6 }

END

sndVidDataBytesSent OBJECT-TYPE
          SYNTAX              Counter64
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The number of video data bytes sent in this test."
          ::= { qosSenderEntry 9 }

sndVidDataBWSent OBJECT-TYPE
          SYNTAX              Gauge32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The bandwidth recorded at the sender for this test, in kbps."
          ::= { qosSenderEntry 10 }

sndVidDataDgramsSent OBJECT-TYPE
          SYNTAX              Counter64
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The number of video data datagrams sent in this test."
          ::= { qosSenderEntry 11 }

sndVidDataFramesSent OBJECT-TYPE
          SYNTAX              Counter64
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The number of video frames sent in this test."
          ::= { qosSenderEntry 12 }

sndVidDataFrameRate OBJECT-TYPE
          SYNTAX              Gauge32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The rate of video frames sent in this test, multiplied by 100."
          ::= { qosSenderEntry 13 }

sndVidTestStatus OBJECT-TYPE
          SYNTAX              Integer32 {
                              InProgress(1),
                              Completed(2)
                              }
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The status of this test. 1 - indicates the test is still in
                    progress. 2 - indicates the test has ended."
          ::= { qosSenderEntry 14 }

– Receiver Table

qosReceiverTable OBJECT-TYPE
          SYNTAX              SEQUENCE OF qosReceiverEntry
          MAX-ACCESS          not-accessible
          STATUS              current
          DESCRIPTION
                    "There's one entry in the qosReceiverTable for each individual
                    test conducted with this system. The Test Number gives the serial
                    number of the test, and the session ID tells which set of tests
                    belong to one session."
          ::= { vqQoS 2 }

qosReceiverEntry OBJECT-TYPE
          SYNTAX              qosReceiverEntry
          MAX-ACCESS          not-accessible
          STATUS              current
          DESCRIPTION
                    "Data in qosReceiverTable uniquely describes a VidQoS test. A
                    host agent MUST create a read-only row for each test where datagrams
                    are received. Rows MUST be created by the host Agent at the start of
                    a test. The receiveIndex will uniquely identify each test."
          INDEX               { rcvsRecvIndex }
          ::= { qosReceiverTable 1 }

qosReceiverEntry          ::= SEQUENCE {
          rcvRecvIndex              Integer32,
          rcvSessionID              Integer32,
          rcvSessionTestNum         Integer32,
          rcvvidTestStartTime       DisplayString,
          rcvvidTestDuration        Counter32,
          rcvVidDataBytesRecd       Counter64,
          rcvVidBandwidth           Gauge32,
          rcvVidDataDgramsRecd      Counter64,
          rcvVidLostDatagrams       Counter64,
          rcvVidDataFramesRecd      Counter64,
          rcvVidLostFrames          Counter64,
          rcvVidDataIntraJitter     Gauge32,
          rcvvidMaxIntraJitter      Gauge32,
          rcvVidDataFrameRate       Gauge32,
          rcvvidTestStatus          Integer32
}
rcvRecvIndex OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The index of the conceptual row which uniquely identifies the test."
          ::= { qosReceiverEntry 1 }
rcvSessionID OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The ID of the session. Several tests may be performed as part of one
                    session."
          ::= { qosReceiverEntry 2 }
rcvSessionTestNum OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "This object uniquely identifies a test within a given session."
          ::= { qosReceiverEntry 3 }
rcvvidTestStartTime OBJECT-TYPE
          SYNTAX              DisplayString
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The time on the local computer this test was started."
          ::= { qosReceiverEntry 4 }
rcvvidTestDuration OBJECT-TYPE
          SYNTAX              DisplayString
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The time duration of this test in seconds."
          ::= { qosReceiverEntry 5 }
rcvVidDataBytesRecd OBJECT-TYPE
          SYNTAX              Counter64
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The number of video data bytes received in this test."
          ::= { qosReceiverEntry 6 }
rcvvidBandwidth OBJECT-TYPE
          SYNTAX              Integer32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The video bandwidth that was found at the receiver for
                    this test, in kbps."
          ::= { qosReceiverEntry 7 }
rcvVidDataDgramsRecd OBJECT-TYPE
          SYNTAX              Counter64
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The number of video data datagrams received in this test."
          ::= { qosReceiverEntry 8 }
rcvvidLostDatagrams OBJECT-TYPE
          SYNTAX              Counter64
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The number of video datagrams that were lost during this
                    test."
          ::= { qosReceiverEntry 9 }
rcvVidDataFramesRecd OBJECT-TYPE
          SYNTAX              Counter64
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The number of video frames received in this test."
          ::= { qosReceiverEntry 10 }
rcvvidLostFrames OBJECT-TYPE
          SYNTAX              Counter64
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The number of video frames that were lost during this test."
          ::= { qosReceiverEntry 11 }
rcvvidIntraJitter OBJECT-TYPE
          SYNTAX              Gauge32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The datagram jitter that was determined while
                    performing this test."
          ::= { qosReceiverEntry 12 }

rcvvidMaxIntraJitter OBJECT-TYPE
          SYNTAX              Gauge32
          MAX-ACCESS          read-only
          STATUS              current
          DESCRIPTION
                    "The maximum datagram jitter that was determined while
                    performing this test."
          ::= { qosReceiverEntry 13 }
```

```
rcvvidFrameRate OBJECT-TYPE
          SYNTAX            Gauge32
          MAX-ACCESS        read-only
          STATUS            current
          DESCRIPTION
                    "The frame rate at the receiver for this test, multiplied by 100."
          ::= { qosReceiverEntry 14 }

rcvVidTestStatus OBJECT-TYPE
          SYNTAX            Integer32 {
                            InProgress(1),
                            Completed(2)
                            }
          MAX-ACCESS        read-only
          STATUS            current
          DESCRIPTION
                    "The status of this test. 1 - indicates the test is still in
                    progress. 2 - indicates the test has ended."
          ::= { qosReceiverEntry 15 }
–
– Logs
–

– Log Table

logTable OBJECT-TYPE
          SYNTAX            SEQUENCE OF LogEntry
          MAX-ACCESS        not-accessible
          STATUS            current
          DESCRIPTION
                    "There's one entry in the logTable for each individual
                    log entered by the test application. This could contain both
                    errors generated while execution, as well as indicate events
                    that occur during the runtime of the system."
          ::= { vqLogs 1 }

LogEntry OBJECT-TYPE
          SYNTAX            LogEntry
          MAX-ACCESS        not-accessible
          STATUS            current
          DESCRIPTION
                    "Data in logTable signifies errors and events during runtime of the
                    testing application. A host agent MUST create a read-only row for
                    each log. Rows created by the agent MUST be be retained until the
                    SNMP service is brought down. The logIndex will uniquely identify
                    each test."
          INDEX             { lgLogIndex }
          ::= { logTable 1 }

LogEntry          ::= SEQUENCE {
          lgLogIndex             Integer32,
          lgLogTime              DisplayString,
          lgLogTitle             DisplayString,
          lgLogType              Integer32,
          lgLogSessionTestNum Integer32,
          lgLogDescription       DisplayString
}

lgLogIndex OBJECT-TYPE
          SYNTAX            Integer32
          MAX-ACCESS        read-only
          STATUS            current
          DESCRIPTION
                    "The index of the conceptual row which uniquely identifies this log."
          ::= { LogEntry 1 }

lgLogTime OBJECT-TYPE
          SYNTAX            DisplayString
          MAX-ACCESS        read-only
          STATUS            current
          DESCRIPTION
                    "The time at which this error/event occured, as a string."
          ::= { LogEntry 2 }

lgLogTitle OBJECT-TYPE
          SYNTAX            DisplayString
          MAX-ACCESS        read-only
          STATUS            current
          DESCRIPTION
                    "The title of this log."
          ::= { LogEntry 3 }

lgLogType OBJECT-TYPE
          SYNTAX            Integer32
          MAX-ACCESS        read-only
          STATUS            current
          DESCRIPTION
                    "The log category to which this log belongs (Define different
                    categories here)."
          ::= { LogEntry 4 }

lgLogSessionTestNum OBJECT-TYPE
          SYNTAX            Integer32
          MAX-ACCESS        read-only
          STATUS            current
          DESCRIPTION
                    "The session test number to which this log applies."
          ::= { LogEntry 5 }

lgLogDescription OBJECT-TYPE
          SYNTAX            DisplayString
          MAX-ACCESS        read-only
          STATUS            current
          DESCRIPTION
                    "The description of the log."
          ::= { LogEntry 6 }
END
```

Appendix C

# Datagram Jitter defined in RFC 1889

This appendix quotes the various sections in RFC 1889 [2] that define and explain the method of calculation of datagram inter-arrival Jitter.

## C.1 Section 6.3.1 - SR: Sender report RTCP packet

Interarrival jitter: 32 bits

An estimate of the statistical variance of the RTP data packet interarrival time, measured in timestamp units and expressed as an unsigned integer. The interarrival jitter $J$ is defined to be the mean deviation (smoothed absolute value) of the difference $D$ in packet spacing at the receiver compared to the sender for a pair of packets. As shown in the equation below, this is equivalent to the difference in the “relative transit time” for the two packets; the relative transit time is the difference between a packet’s RTP timestamp and the receiver’s clock at the time of arrival, measured in the same units.

If $S_i$ is the RTP timestamp from packet $i$, and $R_i$ is the time of arrival in RTP timestamp units for packet $i$, then for two packets $i$ and $j$, $D$ may be expressed as

$$D_{(i,j)} = (R_j - R_i) - (S_j - S_i) = (R_j - S_j) - (R_i - S_i)$$

The interarrival jitter is calculated continuously as each data packet $i$ is received from source SSRC_n, using this difference $D$ for that packet and the previous packet $i - 1$ in order of arrival (not necessarily in sequence), according to the formula

$$J = J + (|D_{(i-1,i)}| - J)/16$$

Whenever a reception report is issued, the current value of $J$ is sampled. The jitter calculation is prescribed here to allow profile- independent monitors to make valid interpretations of reports coming from different implementations. This algorithm is the optimal first- order estimator and the gain parameter $1/16$ gives a good noise reduction ratio while maintaining a reasonable rate of convergence [11]. A sample implementation is shown in Appendix A.8.

## C.2 Section 6.3.4 - Analyzing sender and receiver reports

The interarrival jitter field provides a second short-term measure of network congestion. Packet loss tracks persistent congestion while the jitter measure tracks transient congestion. The jitter measure may indicate congestion before it leads to packet loss. Since the interarrival jitter field is only a snapshot of the jitter at the time of a report, it may be necessary to analyze a number of reports from one receiver over time or from multiple receivers, e.g., within a single network.

## C.3 Appendix A.8 - Estimating the Interarrival Jitter

The code fragments below implement the algorithm given in Section 6.3.1 for calculating an estimate of the statistical variance of the RTP data interarrival time to be inserted in the interarrival jitter field of reception reports. The inputs are $r- > ts$ , the timestamp from the incoming packet, and arrival , the current time in the same units. Here s points to state for the source; $s- > transit$ holds the relative transit time for the previous packet, and $s- > jitter$ holds the estimated jitter. The jitter field of the reception report is measured in timestamp units and expressed as an unsigned integer, but the jitter estimate is kept in a floating point. As each data packet arrives, the jitter estimate is updated:

$$int\, transit = arrival - r- > ts;$$

$$int\; d = transit - s- > transit;$$

$$s- > transit = transit;$$

$$if(d < 0)d = -d;$$

$$s- > jitter+ = (1./16.) * ((double)d - s- > jitter);$$

When a reception report block (to which rr points) is generated for this member, the current jitter estimate is returned:

$$rr- > jitter = (u\_int32)s- > jitter;$$

Alternatively, the jitter estimate can be kept as an integer, but scaled to reduce round-off error. The calculation is the same except for the last line:

$$s- > jitter+ = d - ((s- > jitter + 8) >> 4);$$

In this case, the estimate is sampled for the reception report as:

$$rr- > jitter = s- > jitter >> 4;$$

# List of References

[1] J. Postel. *User Datagram Protocol*. RFC 768. August 1980

[2] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. RFC 1889. January 1996

[3] C. Zhu. *RTP Payload Format for H.263 Video Streams*. RFC 2190. September 1997

[4] J. Case, M. Fedor, M. Schoffstall and J. Davin. *A Simple Network Management Protocol (SNMP)*. RFC 1157. May 1990

[5] J. Case, K. McCloghrie, M. Rose and S. Waldbusser. *Introduction to version 2 of the Internet-standard Network Management Framework*. RFC 1441. April 1993

[6] P. Grillo and S. Waldbusser. *Host Resources MIB*. RFC 2790. March 2000

[7] M. Baugher, B. Strahm and I. Suconick. *Real-Time Transport Protocol Management Information Base*. RFC 2959. October 2000

[8] International Telecommunication Union - Telecommunications Standardization Sector (ITU-T). *Recommendation H.263. Line transmission of non-telephone signals: Video coding for low bitrate communication*. May 1996

[9] Dr. Sidnie Feit. *SNMP: A guide to Network Management.* McGraw-Hill Inc. International Editions, Singapore, 1995

[10] Steve Maxwell. *Red Hat Linux Network Management Tools.* McGraw-Hill Co. Inc., New York, NY, 2000

[11] James A. Cadzow. *Foundations of Digital Signal Processing and Data Analysis.* Macmillan, New York, NY, 1987

[12] Zheng Wang. *Internet QoS: Architectures and Mechanisms for Quality of Service.* Morgan Kaufmann, San Francisco, March 2001

[13] International Telecommunication Union - Telecommunications Standardization Sector (ITU-T). *Recommendation X.680. Abstract Syntax Notation One (ASN.1) & ASN.1 encoding rules.* June 2002

[14] International Telecommunication Union - Telecommunications Standardization Sector (ITU-T). *Recommendation H.341. Multimedia Management Information Base.* May 1999

[15] International Telecommunication Union - Telecommunications Standardization Sector (ITU-T). *Recommendation H.323. Packet based multimedia communication systems.* November 2000

[16] International Telecommunication Union - Telecommunications Standardization Sector (ITU-T). *Recommendation H.245. Control Protocol for multimedia communications.* July 2001

[17] International Telecommunication Union - Telecommunications Standardization Sector (ITU-T). *Recommendation H.320. Narrow-band visual telephone systems and terminal equipment.* May 1999

[18] M. Baugher et al. *Real-time Transport Protocol Management Information Base.* RFC 2979. October 2000.

[19] Hamid Sharif and Bing Chen. *End-to-End QoS requirements for real-time video streaming in Internet2.* The Information Society School, 2001

[20] M. Vouk, Z. Ortiz, A. Rindos, S. Woolet, D. Cosby, J. Sents and M. Aydemir. *Throughput and video performance of emerging LAN technologies: Switched Ethernet and LAN emulation over ATM.* Proceedings of the IEEE Southeastcon '97, pp. 131-134, Blacksburg VA, April 1997

[21] Richard Carlson, T. H. Dunigan, Russ Hobby, Harvey B. Newman, John P. Streck, Mladen A. Vouk. *End-to-End Performance on the Internet.* To appear in Network World, 2003

[22] Dimitrios Miras. *Network QoS Needs of Advanced Internet Applications: A Survey*. Internet2 QoS Working Group Document available at http://qos.internet2.edu/wg/apps/fellowship/Docs/Internet2AppsQoSNeeds.pdf, December 2002

[23] Zhou Wang and Alan C. Bovik. *A Universal Image Quality Index*. IEEE Signal Processing Letters, vol. 9, no. 3, pp. 81-84, March 2002

[24] Dapeng Wu, Yiwei Thomas Hou and Ya-Qin Zhang. *Transporting Real-time Video*

*over the Internet: Challenges and Approaches.* Proceedings of the IEEE, Vol 98, No. 12, December 2000

[25] Andrew Campbell, Cristina Aurrecoechea and Linda Hauw. *A Review of QoS Architectures.* Proc. 4th IFIP International Workshop on Quality of Service, Paris. March 1996

[26] Dmitri Loguinov and Hayder Radha. *Measurement study of low-bitrate Internet video streaming.* ACM SIGCOMM Internet Measurement Workshop, November 2001

[27] Dmitri Loguinov and Hayder Radha. *Large-scale Experimental Study of Internet Performance using Video Traffic.* ACM SIGCOMM Computer Communication Review (CCR), vol. 32, no. 1, January 2002.

[28] Michael R. Izquierdo and Douglas S. Reeves. *A Survey of Statistical Source Models for Variable Bit-Rate Compressed Video.* Multimedia Systems 7 Issue 3, pp 199-213, 1999

[29] E. Casilari, A. Reyes, A. Daz-Estrella and F. Sandoval. *Characterization and modeling of VBR video traffic.* Electronic Letters, Vol. 34, no. 10, pp. 968-969, May 1998

[30] Deepak S. Turaga and Tsuhan Chen. *Modeling of dynamic video traffic.* International Symposium on Circuits and Systems, June 2000

[31] N. Doulamis, A. Doulamis and S. Kollias. *Modeling and Adaptive Prediction of VBR MPEG Video sources.* IEEE Third Workshop on Multimedia Signal Processing, pp. 27-32, September 1999

[32] Mark Claypool and Jonathan Tanner. *The effects of jitter on the perceptual quality of video.* ACM Multimedia, Volume 2, Orlando, FL, November 1999

[33] Olivier Cappe, Jean-Christophe Pesquet and Athina Petropulu. *Long-range dependence and heavy-tail modeling for teletraffic data.* Vol. 19, no. 3, IEEE Signal Processing Magazine, May 2002

[34] David L. Mills. *Network Time Protocol (Version 3) Specification, Implementation and Analysis.* RFC 1305. March 1992

[35] Chidambaram Arunachalam. *Implementation and Validation of Network Policy Services.* Master's Thesis, NC State University, 2002

[36] E. Brent Kelly. *Quality of Service In Internet Protocol (IP) Networks.* Prepared for the International Communications Industries Association to support Infocomm 2002. Wainhouse Research 2001

[37] The Moving Picture Experts Group. *MPEG Home page.* http://mpeg.telecomitalialab.com/ ISO/IEC Working committees, Telecom Italia Lab, Italy

[38] Ajay Tirumala et. al. *Iperf home page.* http://dast.nlanr.net/Projects/Iperf/ 2003

[39] Network Research Group. *VIC home page.* http://www-nrg.ee.lbl.gov/vic/ 2002

[40] Gerald Combs et. al. *Ethereal home page.* http://www.ethereal.com/ 2002

[41] LBNL Network Research Group. *Tcpdump home page.* http://www.tcpdump.org/ 2003

[42] Loris Degioanni et. al. *Windump home page.* http://windump.polito.it/ 2002

[43] Jon Sevy. *Java SNMP query Open Source code.* http://edge.mcs.drexel.edu/GICL/people/sevy/snmp/snmp_package.html 2000

[44] Microsoft MSDN Development Network. *MSDN home page.* http://www.msdn.microsoft.com

[45] IANA Authority. *IANA SNMP Private Enterprise Numbers page.* http://www.iana.org/assignments/enterprise-numbers May 2003

[46] Daniel Forsgren. *Digital Video over IP.* http://www.online.kth.se/projects/csd/content/dv/dv.pdf KTH-IT, January 2001

[47] Polycom Videoconferencing solutions. *Polycom web page.* http://www.polycom.com

[48] PictureTel Corporation. *iPower Video Technology white paper.* http://www.polycom.com/common/pw_item_show_doc/0,1449,692,00.pdf April 2001

[49] Microsoft Corporation. *Using Performance Monitor Counters with SNMP.* Chapter 11 - Using SNMP for Network Management. http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol /winntas/reskit/net/sur_snmp.asp

[50] VCon Videoconferencing solutions. *Vcon home page.* http://www.vcon.com

[51] Roel Jonkman et. al. *Netspec: A tool for network experimentation and measurement, home page.* http://www.ittc.ku.edu/netspec/

[52] Beng Ong Lee, Victor S. Frsot, Roelof Jonkman. *Netspec 3.0 source models for telnet, ftp, voice, video and WWW traffic*. ITTC-TR-10980-19, University of Kansas, January 1997

[53] UNC-Chapel Hill. *Videnet Scout home page*. http://scout.video.unc.edu/

[54] NetIQ Corporation. *NetIQ Chariot Test tool home page*. http://www.netiq.com/products/chr/default.asp

[55] Delta Information Systems. *Delta Information home page*. http://www.delta-info.com

[56] Internet2 community. *Internet2 home page*. http://www.internet2.edu/

[57] Institute for Telecommunication Sciences. *Video Quality Experts Group Home page*. http://www.its.bldrdoc.gov/n3/video/Default.htm