Performance Analysis of Java as a Streaming Digital Media Platform

By

Robert Christopher Norris
B.Sc., University of Victoria, 1999

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

We accept this thesis as conforming
to the required standard

[redacted]

Dr. D. M. Miller, Supervisor (Department of Computer Science)

[redacted]

Dr. G. Shoja, Departmental Member (Department of Computer Science)

[redacted]

Dr. F. Gebali, Outside Member (Department of Electrical and Computer Engineering)

[redacted]

Dr. W. Rutherford, External Examiner (BCIT Technology Centre)

University of Victoria

Supervisor: Dr. D. M. Miller

ABSTRACT

The IEEE 1394 high-speed serial bus provides time and bandwidth guarantees suitable for real-time streaming media applications. The Java platform is a language and runtime environment for heterogeneous, network-centric computing. The Java platform's streaming media performance for the relatively new IEEE 1394 bus and the *de facto* standard IEEE 802.3 network is considered. The identified performance criteria metrics are inter-frame delay, transmit lag, and change in burstiness. Two types of buffers are evaluated: fixed size in space and fixed size in time. The key finding of this research for both buffer types is that even the most performance oriented application development cannot overcome improper choices of application frame size and protocol buffer size.

The experimental runtime environment uses modern commodity PC computers and the GNU/Linux operating system. The current, at the time of this thesis, Java 2 Standard Edition SDK is used as the Java platform. Raw performance data is collected from a client server software framework designed and built for this research. This thesis also required the construction of a Java bridge to the IEEE 1394 bus. The statistical analysis phase of this thesis employs several statistical techniques to examine the correlation structure and descriptive statistics of raw performance data. Analysis of variance and linear regression statistical techniques determine individual contributions of repeated runs of experiment blocks. Two Hurst parameter estimation techniques, the variance-time statistic and the rescaled adjusted range statistic, examine the self-similar correlation structure of movie related time series.

Examiners:

[redacted]

Dr. D. M. Miller, Supervisor (Department of Computer Science)

[redacted]

Dr. G. Shoja, Departmental Member (Department of Computer Science)

[redacted]

Dr. F. Gebali, Outside Member (Department of Electrical and Computer Engineering)

[redacted]

Dr. W. Rutherford, External Examiner (BCIT Technology Centre)

CONTENTS

CONTENTS iii

LIST OF TABLES vii

LIST OF FIGURES ix

1. Introduction 1

2. Background 3

2.1 IEEE 802.3: Ethernet 3

2.1.1 Description of the IEEE 802.3 3

2.1.2 IEEE 802.3 Addressing 4

2.1.3 IEEE 802.3 Contention 4

2.1.4 IEEE 802.3 Transmission and Reception 5

2.1.5 IEEE 802.3 Network Configuration 6

2.1.6 IEEE 802.3 Summary 6

2.2 IEEE 1394: FireWire 7

2.2.1 Description of IEEE 1394 7

2.2.2 IEEE 1394 Addressing 8

2.2.3 IEEE 1394 Arbitration 11

2.2.4 IEEE 1384 Transmission Types 12

2.2.4.1 IEEE 1394 Isochronous Transmission 12

2.2.4.2 IEEE 1394 Asynchronous Transmission 13

2.2.4.3 IEEE 1394 Transmission Type Summary 15

2.2.5 IEEE 1394 Node Implementation Options 15

2.2.6 IEEE 1394 Transmission and Reception 16

2.2.7 IEEE 1394 Bus Configuration 16

2.2.8 IEEE 1394 Summary 17

2.3 Comparison of the IEEE 802.3 and the IEEE 1394 17

2.4 Software Protocols 20

2.5 Data Types 22

2.5.1 Digital Video 23

2.6 An Introduction to Self-Similarity 25

2.6.1 Self-Similarity in Computer Communication 26
2.6.2 The Effects of Self-Similarity on Computer Communications . 27
2.7 Streaming Variable Bit Rate Video 30
2.8 Summary 31

3. Software and Runtime Environment 33
3.1 Runtime Environment 33
3.1.1 Hardware 35
3.1.2 Operating System 36
3.1.3 Application 38
3.1.4 Java 38
3.1.4.1 JavaBeans 39
3.1.4.2 Java Native Interface 40
3.1.5 Summary 43
3.2 Custom Software 44
3.2.1 Java IEEE 1394 Interface 44
3.2.1.1 IEEE 1394 Isochronous Resource Management 45
3.2.1.2 IEEE 1394 Transactions 46
3.2.1.3 Summary 47
3.2.2 Client Server Framework 47
3.2.2.1 Client Server Modules 48
3.2.2.2 Generation Module 49
3.2.2.3 Observation Module 51
3.2.2.4 Transmission Module 52
3.2.2.5 Server Application 53
3.2.2.6 Client Application 54
3.2.2.7 Summary 55
3.2.3 Software Design Goals 55
3.3 Java Performance 57
3.4 Supporting Software 59
3.5 Summary 60

4. Maximum Utilization Experiment and Results 62
4.1 Experiment Description 62
4.1.1 Channel Performance 63
4.1.2 Maximum Utilization Experiment with Default Buffers 64

4.1.3 Maximum Utilization Experiment with Reduced Buffers . . . 65
4.1.4 Frame Size . . . 65
4.1.5 Summary . . . 67
4.2 Runtime Environment Contribution to Response Variables . . . 68
4.2.1 Systematic Contributions . . . 69
4.2.2 Random Contributions . . . 70
4.2.2.1 Reducing Random Contributions . . . 70
4.3 Preliminary Statistical Analysis . . . 72
4.3.1 Inter-Frame Delay . . . 74
4.3.2 Transmit Lag . . . 76
4.4 Descriptive Statistics . . . 78
4.4.1 Inter-Frame Delay . . . 79
4.4.2 Transmit Lag . . . 83
4.4.3 General Trends of Inter-Packet Delay and Transmit Lag . . . 90
4.5 Garbage Collection . . . 91
4.5.1 Results . . . 93
4.6 Conclusion . . . 94
4.6.1 Recommendations . . . 95
5. Movie Experiment and Results . . . 101
5.1 Hurst Parameter . . . 101
5.2 Methods of Estimating the Hurst Parameter . . . 104
5.2.1 Auto-Correlation Function . . . 104
5.2.2 Short-Range and Long-Range Dependence . . . 106
5.2.3 Rescaled Adjusted Range Statistic . . . 107
5.2.4 Variance-Time Statistic . . . 108
5.2.4.1 Unsuitability of Variance-Time Plots for Some Self-Similar Time Series . . . 109
5.2.5 Periodogram . . . 110
5.2.6 Summary . . . 111
5.3 Experiment Description . . . 111
5.3.0.1 Buffering Client Input . . . 113
5.3.1 Statistical Measures of Burstiness . . . 113
5.3.2 Performance Metrics . . . 115

5.3.2.1 Unsuitability of Inter-Frame Delay and Transmit Lag Channel Performance Metrics 117
5.3.3 Interpreting Variance-Time and Rescaled Adjusted Range Plots 119
5.3.4 Movie Trace Description 120
5.3.4.1 Movie Trace Statistical Properties 122
5.3.5 Summary 128
5.4 Preliminary Investigation of Client Movie Lag 129
5.4.1 Movie to Client Lag Density 130
5.4.2 Client Buffer Size 132
5.5 Estimates of Burstiness 133
5.6 Trend Analysis 138
5.6.1 Trends 141
5.6.2 Comparing Trends 142
5.7 Conclusion 144
6. Conclusion 146
6.1 Improved Runtime Environment 148
6.2 Improved IEEE 1394 Device Driver 148
6.3 Heterogeneous Computing Performance Evaluation 149
6.4 Increased Resolution of Factor Levels 149
6.5 Garbage Collection Evaluation 150
6.6 Examiniation of Nagle's Algorithm 150
6.7 Validation of Results 150
6.8 Simulation of Results 150
Bibliography 152
A. Acronyms 157

LIST OF TABLES

2.1 MPEG standards for digital video encoding and compression. 25

3.1 Specific components of the thesis runtime environment. 43

4.1 Analysis of variance of inter-frame delay for the maximum experiments. 74

4.2 Linear regression of TCP/IP inter-frame delay of 992-byte frames, and with reduced buffers. 75

4.3 Revised analysis of variance of TCP/IP inter-frame delay of 992-byte frames, with reduced buffers, and with run 4 excluded. 75

4.4 Linear regression of UDP/IP inter-frame delay of 992-byte frames, and with reduced buffers. 76

4.5 Revised analysis of variance of UDP/IP inter-frame delay of 992-byte frames, with reduced buffers, and with run 4 excluded. 76

4.6 Analysis of variance of transmit lag for the maximum experiments. . . . 77

4.7 Quantile breakdown of inter-frame delay for the maximum experiments. 80

4.8 Decriptive statistics of inter-frame delay. 81

4.9 Quantile breakdown of transmit lag for the maximum utilization experiments. 84

4.10 Decriptive statistics of transmit lag. 85

4.11 Mean and total of garbage collection time for each frame size factor level. 93

4.12 Linear regression of isochronous IEEE 1394 transmit lag. 96

4.13 Revised analysis of variance of isochronous IEEE 1394 transmit lag. . . 96

4.14 Linear regression of TCP/IP transmit lag with default buffers. 97

4.15 Revised analysis of variance of TCP/IP transmit lag with default buffers. 97

4.16 Linear regression of UDP/IP transmit lag with default buffers. 98

4.17 Revised analysis of variance of UDP/IP transmit lag with default buffers. 98

4.18 Linear regression of TCP/IP transmit lag with reduced buffers. 99

4.19 Revised analysis of variance of TCP/IP transmit lag with reduced buffers. 99

4.20 Linear regression of UDP/IP transmit lag with reduced buffers. 100

4.21 Revised analysis of variance of UDP/IP transmit lag with reduced buffers. 100

5.1 Descriptive statistics of original and transformed movie traces. 122

5.2 Burstiness statistics of the MPEG and Starwars movie traces. 123

5.3 Client buffer size levels of the movie experiments. 133

5.4 Burstiness quantification estimates of the client buffer occupancy of the MPEG movie trace. 138

5.5 Burstiness quantification estimates of the client buffer occupancy of the Starwars movie trace. 139

LIST OF FIGURES

2.1 IEEE 1394 bus cycle and isochronous channel transmission. 13

2.2 ISO OSI network model. 21

2.3 a) IEEE 802.3 protocol stack, b) ISO OSI network model, and c) IEEE 1394 protocol stack. 22

2.4 Box and whiskers plot of estimated mean with 95% and 99% confidence intervals calculated from the first n frame observations of the MPEG movie trace (top) and Starwars movie trace (bottom). 29

3.1 Runtime environment conceptual hierarchy. 34

3.2 Two views of the Java platform and its runtime environment. 40

3.3 Collaboration diagram of the client server framework and modules. . . . 49

4.1 Performance metrics of the maximum utilization experiments. 63

4.2 Transmit lag of the isochronous IEEE 1394 for a sequence of 1000 24-byte frames. 86

4.3 Transmit lag (upper - solid line) and inter-frame delay (lower - dashed line) plot of the TCP/IP with default buffers for a sequence of 1000 992-byte frames. 87

4.4 Transmit lag (upper - solid line) and inter-frame delay (lower - dashed line) plot of the TCP/IP with reduced buffers for a sequence of 100 992-byte frames. 88

4.5 Transmit lag (upper - solid line) and inter-frame delay (lower - dashed line) plots of the UDP/IP with default (upper plot) and reduced (lower plot) buffers for a sequence 1000 24-byte frames. 89

5.1 Pictorial view of a self-similar process, left column, and a short-range dependant Poisson process, right column, at five time scales. Adapted from Taqqu *et al.*[1]. 103

5.2 Original time series unevenly partitioned into seven aggregated time series. The aggregated time series excludes the horizontally stripped regions because they are smaller than the block size. 109

5.3 Movie to client lag performance measure used in the movie experiments. 115

5.4 Density plot of lag from the MPEG movie trace to server frame timestamp. The visible portion of the plot represents 99.6% of the MPEG movie trace frames. 118

5.5 Auto-correlation function: top) MPEG movie trace's frame size, middle) MPEG movie trace's group of pictures size, bottom) Starwars movie trace's frame size. 125

5.6 R/S pox plot of the MPEG movie trace. 126

5.7 Variance-time plot of the MPEG movie trace. 126

5.8 R/S pox plot of the Starwars movie trace. 127

5.9 Variance-time plot of the Starwars movie trace. 127

5.10 Movie to client lag of the MPEG movie trace. 131

5.11 Coefficient of variation of the MPEG movie trace at different buffer sizes. 134

5.12 Peak to mean ratio of the MPEG movie trace at different client buffer sizes. 135

5.13 Coefficient of variation of the Starwars movie trace at different client buffer sizes. 135

5.14 Peak to mean ratio of the Starwars movie trace at different client buffer sizes. 136

5.15 R/S Hurst parameter estimate of the MPEG movie trace at different client buffer sizes. 136

5.16 Variance-time Hurst parameter estimate of the MPEG movie trace at different client buffer sizes. 137

5.17 R/S Hurst parameter estimate of the Starwars movie trace at different client buffer sizes. 137

5.18 Variance-time Hurst parameter estimate of the Starwars movie trace at different client buffer sizes. 140

1. Introduction

EVALUATING the Java platform for streaming temporal media is an important pursuit. The Java platform is an excellent choice for streaming temporal media implementation, if it can meet the special performance requirements of streaming temporal media. Industrial standards groups such as *Home Audio Video Interoperability (HAVi)* are vigorously pursuing local distribution of digital media to home audio-visual devices using Java.

This thesis investigates the Java platform's network communication performance, with emphasis on the Java platform for streaming temporal media. This thesis contains relevant background information, a description of the software and runtime environment used for the experiments, and a thorough description of the two groups of experiments that evaluate the performance of the Java platform for streaming temporal media.

Chapter 2 introduces background material and previous research important to the remainder of this thesis. The chapter begins with the operation, configuration, and capabilities of the IEEE 802.3 network and the IEEE 1394 bus. This chapter describes the three types of computer communications, the isochronous IEEE 1394, and the TPC/IP and the UDP/IP transport layer software protocols over IEEE 802.3, used to evaluate the streaming performance of the Java platform. The chapter discusses previous research related to streaming temporal media, concentrating on the self-similarity of compressed variable bit rate video and its affects on computer communications.

Chapter 3 describes the software and runtime environment used to evaluate the streaming temporal media of the Java platform. The chapter begins with a description of the hardware, operating system, and application programs of the runtime environment. These components form the backbone of all streaming media applications and greatly affect their performance. The chapter continues to describe the two software

projects I implemented to facilitate the generation, transmission, and observation of streaming traffic across the isochronous IEEE 1394, the TCP/IP, and the UDP/IP. The first project, j1394, provides an implementation of a Java to IEEE 1394 bridge that is missing from all core or standard extension Java packages. The second project, the client server framework, provides a consistent streaming media framework for all the performance evaluation experiments. This chapter also describes the steps taken to ensure the software projects provide reliable and robust performance.

Chapter 4 describes the experiments that evaluate the performance of the Java platform under maximum channel utilization conditions. The inter-frame delay and transmit lag performance metrics evaluate the three computer communication types with several fixed application frame sizes and, when feasible, several protocol buffer sizes. This chapter also includes a thorough treatment of repeatability of experiment runs and the contributions of outlier values to the overall descriptive statistics.

Chapter 5 describes the experiments that evaluate the performance of the Java platform under long-running, self-similar, variable bit rate video. The chapter begins with a survey of self-similar time series and several Hurst parameter (the degree of self-similarity) estimation techniques. Two Hurst parameter estimation techniques and two traditional, non-correlation, burstiness estimation techniques examine the change in burstiness of self-similar data when streamed by the Java platform. The change in burstiness of the self-similar movie trace is evaluated for several fixed in time buffer sizes. The focus of these experiments shifts away from channel performance metrics based solely on frame timing to a performance measure that incorporates frame timing and frame size.

Chapter 6 discusses the key findings of this research and suggests several avenues of future research.

2. Background

This chapter covers background material and previous research relating to this thesis. Specifically, this chapter covers the following background material: the IEEE 1394, the IEEE 802.3, the TCP/IP, the UDP/IP, temporal data, and self-similarity. This chapter summarizes related research covering the self-similar property of some types of streaming data, self-similar structure in computer communication, and the additional challenges self-similarity introduces.

2.1 IEEE 802.3: Ethernet

The Institute of Electrical and Electronics Engineers (IEEE) 802.3 specification [2] defines a Local Area Network (LAN) commonly referred to as Ethernet [3]. Both the IEEE and the International Organization for Standardization (ISO) have adopted the IEEE 802.3 as a worldwide local area network specification. Ethernet began with the work of Robert Metcalfe and David Boggs at Xerox. Their seminal publication, "Ethernet: Distributed Packet Switching for Local Computer Networks" [4], was published in 1976. In 1985, the IEEE adopted the IEEE 802.3 specification and published "IEEE 802.3 Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications". The current IEEE 802.3 specification supports local area network transmission speeds of 10 megabits per second, 100 megabits per second, and 1000 megabits per second. Although 10 gigabits per second is the current maximum transfer rate of the IEEE 802.3, 10 gigabits per second is currently only available in service provider carrier equipment and not available for local area networks.

2.1.1 Description of the IEEE 802.3

An IEEE 802.3 physical implementation consists of nodes and a common signalling path. A *node* is a point on the IEEE 802.3 network that can transmit and

receive data. A node contains a single port to connect to the IEEE 802.3 network and is typically located in a computer host. The common signalling path, called the *media*, joins IEEE 802.3 nodes together forming a connected tree[1]. Each node becomes a leaf in the IEEE 802.3 network topology. There is exactly one path through the media between any pair of nodes. Signals propagate through the common signalling media without interruption to all nodes on the IEEE 802.3 network. There is no physical limit to the number of nodes connected to an IEEE 802.3 network, but in practice, large networks are replaced with several smaller networks joined together with bridges.

Charles Spurgeon [3] covers the basic operation of the IEEE 802.3 in greater detail in "Ethernet: The Definitive Guide". For complete coverage, the reader should consult the IEEE 802.3 specification [2] and specifications therein.

2.1.2 IEEE 802.3 Addressing

In every IEEE 802.3 network, each node has a unique 48-bit address. This creates a massive address space of hundreds of trillions of possible IEEE 802.3 addresses. To ensure that no two nodes in the same network have the same address, each IEEE 802.3 node has an immutable globally unique address assigned during the node's construction. An IEEE 802.3 node's address is read-only and never changes. No IEEE 802.3 address or IEEE 802.3 node has any special powers or responsibilities.

2.1.3 IEEE 802.3 Contention

The Media Access Control (MAC) protocol employed by the IEEE 802.3 is Carrier Sense Multiple Access with Collision Detection (CSMA/CD). The CSMS/CD protocol is comprised of three phases: carrier sense, multiple access, and collision detect.

Carrier sense requires each IEEE 802.3 node must wait until the media is idle before transmitting. This phases' name comes from the IEEE 802.3 specification,

[1] acyclic connected graph

where a data transmission on the media is called a *carrier*.

Multiple access states all IEEE 802.3 nodes have equal access to the media. An IEEE 802.3 node may only perform a single transmission after sensing an idle period on the media. Allowing a node to perform only a single transmission prevents a node from dominating the media with a continuous stream of transmission.

Collision detect requires IEEE 802.3 nodes to detect all signal interference caused by a transmission collision. *Transmission collisions* are simultaneous transmissions by two or more IEEE 802.3 nodes on the common signalling media. Collisions can even occur when IEEE 802.3 nodes wait for an idle media because non-zero time elapses as the signal propagates across the media. Multiple IEEE 802.3 nodes may detect an idle media nearly simultaneously and all immediately begin transmitting, causing a transmission collision. When an IEEE 802.3 node detects a collision it halts its transmission, jams the media so all attached nodes know the transmission is damaged, and delays for a random period before attempting to retransmit.

2.1.4 IEEE 802.3 Transmission and Reception

The IEEE 802.3 specification provides a simple mechanism for data transmission. IEEE 802.3 nodes contend for exclusive access to the media with CSMA/CD media access control protocol. After winning contention, a node broadcasts data serially on the media to all nodes connected to the IEEE 802.3 network. All data transmitted on an IEEE 802.3 network is encapsulated in an IEEE 802.3 frame. The IEEE 1394 frame contains the data payload and IEEE 802.3 details such as addressing.

The process of receiving a frame is straightforward as well. All nodes attached to the IEEE 802.3 network receive all IEEE 802.3 frames. Under normal operation, each node inspects each IEEE 802.3 frame for corruption and forwards frames addressed to them to the operating system for further possessing. An IEEE 802.3 node does not acknowledge any frames it receives; higher layer protocols handle that job.

The IEEE 802.3 has a promiscuous mode, where nodes forward all IEEE 802.3

frames to the operating system for further processing regardless of frame's destination address. Nodes usually ignore IEEE 803.2 frames with destination addresses that do not match their own node address. This research is not concerned with the promiscuous mode of IEEE 802.3 and it is not discussed further.

2.1.5 IEEE 802.3 Network Configuration

The configuration of an IEEE 802.3 network may require very little effort or involve complex maintenance. Adding a node to an IEEE 802.3 network immediately permits the new node to send and receive according to the IEEE 802.3 media access control protocol. The complexity arises from not knowing the address of the target IEEE 802.3 node. The IEEE 802.3 does not manage the addresses of its nodes.

There are two general schemes to manage IEEE 802.3 addresses of a network. The first method is suitable for small networks that don't change often. In this method, each node has a static lookup list of all the IEEE 802.3 addresses on the network. When the IEEE 802.3 network changes with the addition or removal of a node, each node must have its lookup list updated to reflect the new IEEE 802.3 network configuration. The perpetual maintenance required for some IEEE 802.3 networks prompted the development of address resolution automation tools. The second method addresses the problem of manual maintenance of IEEE 802.3 address lookup lists. This method uses a software service to maintain a list of the current node addresses of all nodes of an IEEE 802.3 network. The most common software protocol to maintain a network's IEEE 802.3 addresses is RFC-826 [5].

2.1.6 IEEE 802.3 Summary

The IEEE 802.3 specification is a common local area network physical and data link layer specification. The IEEE 802.3 specification contains no central controller or arbitrator. Each IEEE 802.3 node operates using the CSMA/CD protocol independently of other nodes to access the shared media. The CSMA/CD protocol does not provide bounded access time to the media. The IEEE 802.3 provides probabilistic

media access time suitable for many applications on networks with light to moderate loads. The probable media access time for the IEEE 802.3 increases quickly as media utilization increases.

2.2 IEEE 1394: FireWire

The IEEE 1394 specification [6] defines a high performance serial bus suitable for local area networks and peripheral connection. The IEEE 1394 serial bus is available under the generic label IEEE 1394 and the trademarked labels FireWire and i.Link by Apple Computer and Sony Corporation respectively. The IEEE 1394 began when Texas Instruments and Apple Computer designed a cheap alternative to Small Computer Systems Interface (SCSI). Their original release of FireWire in 1992 supported transmission speeds of 100 megabits per second. The IEEE Microcomputer Standards Committee based their high-performance serial bus specification, called IEEE 1394, on Apple's FireWire in 1995. The current IEEE 1394 specification, IEEE 1394a-2000, supports transmission speeds of 100 Mbps, 200 Mbps, and 400 Mbps simultaneously.

2.2.1 Description of IEEE 1394

The basic components of an IEEE 1394 bus are IEEE 1394 nodes and sections of signalling path to connect the nodes together. Each IEEE 1394 node is a point on the bus that is able to transmit and receive data. Each node provides from one to seven ports to connect to the IEEE 1394 bus. A node may be located within a computer host or a peripheral device such as a video camera. Sections of signalling path connect nodes together forming a tree topology.

There are three restrictions to the IEEE 1394 bus topology. The first restriction forbids loops in the bus topology. Another view of this restriction is that there must be exactly one path between any two nodes connected on the bus. The second restriction allows for a maximum of sixteen hops between any two nodes. This restriction is a requirement to guarantee an upper bound on signal propagation time on any IEEE

1394 bus. The third restriction permits at most sixty-three nodes attached to one bus. The IEEE 1394 detects all these invalid bus configurations and only operates in the presence of a valid bus configuration.

The three previous rules apply only to individual IEEE 1394 buses. An IEEE 1394 bridge can connect multiple IEEE 1394 buses together to form a pooled bus structure. Each individual IEEE 1394 bus must conform to the rules of IEEE 1394 bus configuration, but the pooled bus does not. For example, by connecting multiple IEEE 1394 buses together with an IEEE 1394 bridge, the maximum number of nodes increases from 63 to over 64,000.

Signals propagate from one IEEE 1394 node through a signalling segment to directly connected nodes. A node that receives a signal conditionally re-creates that signal on one or more of its other connected ports. A node may selectively transmit a control signal to some, none, or all of its neighbour nodes. The IEEE 1394 specification determines when nodes absorb control signals and when nodes pass on control signals to their neighbours. A node unconditionally regenerates all data signals it receives on one of its ports and broadcasts those signals to all of its other connected ports.

Don Anderson [7] covers the basic operation of the IEEE 1394 with more detail in "FireWire System Architecture". For complete coverage, the reader should consult the IEEE 1394 specification [6] and recent supplements [8].

2.2.2 IEEE 1394 Addressing

In every IEEE 1394 bus, each node assigns itself a temporary address called a *node id.* After a bus reset, the IEEE 1394 bus enters the self-id phase where each node assigns itself a new node id. The node id associated with an IEEE 1394 node is only valid until the next bus reset.

A bus reset is caused by a physical change to the bus such as adding a new node or removing an existing node. Any node through software control may also initiate a bus reset. A bus reset, among other things, re-assigns node ids to all nodes and

selects a root node and an isochronous resource manager.

Immediately prior to the self-id phase, the IEEE 1394 bus performs the tree identification process. Tree identification determines the IEEE 1394 bus' logical topology, which the self-id phase uses for assigning node ids. The tree identification process begins with every node determining how many directly connected neighbours it has. *Leaf* nodes have only one neighbour and *branch* nodes have two or more neighbours. Then, nodes signal their probable parents with a parent notify signal. Only nodes that receive a parent notify signal from all but one of its neighbour may signal parent notify. Immediately after a bus reset, only leaf nodes meet this condition because they have only one neighbour. When a node receives parent notify signals from all but one of its neighbours, it signals child notify to all its neighbour child nodes and signals parent notify to its last remaining neighbour. This process continues until there is one node remaining without a node to signal parent notify. This node becomes the *root* node.

The IEEE 1394 bus' logical topology is now determined. Every node knows which neighbours are its children and which neighbour is its parent. The tree identification processes defines the logical network topology and defines an explicit bus hierarchy with the root node at the top. The root node becomes the central controller and arbitrator of the bus IEEE 1394 bus.

The self-id phase begins immediately following the tree identification process. The self-id phase begins with every node initializing its self-id count to zero. The root node then sends its lowest child neighbour a grant signal. Nodes propagate the grant signal from parent to child down the bus hierarchy until the grant signal reaches the last lowest child node. This node realizes it does not have any children to forward the grant signal to and identifies itself by broadcasting a self-id message to all nodes on the bus. From this point forward, this node only passively participates in the self-id phase.

All nodes on the IEEE 1394 bus observe every self-id message and increment their own self-id count with each self-id message. The root node repeats the grant

signalling process until it has zero unidentified child neighbours. Finally, the root node sends its self-id message and assigns itself the last node id.

Each self-id message contains information about the maximum speed and the capabilities of the sending node. This allows each node to build a topology map of the bus configuration and node capabilities. The IEEE 1394 bus may have mixed speeds and different capabilities at each node. All nodes monitor the self-id messages and increment their self-id counter for each self-id message they receive. A node's self-id counter contains the node id to assign itself when it is time to send its own self-id message. The self-id process also elects one node to be the isochronous resource manager (IRM) for the bus.

The isochronous resource manager election process is as follows. Each node's self-id message tells if the node is isochronous resource manager capable. The last node to identify itself with the isochronous recourse manager capability becomes the isochronous resource manager. This node is usually the root node but not always.

The self-id phase is now complete with each node assigning itself a sequential node id beginning with zero. The tree identification and self-id phases elect nodes for the important roles of root node and isochronous resource manager. All nodes connected to the IEEE 1394 bus know the node id of the root node and which direction the root node is. The root node has a well-known address and special responsibilities. All nodes connected to the IEEE 1394 bus know the node id of the isochronous resource manager.

This section provides only a brief overview of the IEEE 1394's addressing. I omit several important details required for the correct operation of the IEEE 1394 bus such as forced root and tie resolution. These tasks are not optional for proper IEEE 1394 operation, but are not relevant of this discussion. Interested readers should consult the IEEE 1394 references [6, 7, 8] for more information.

2.2.3 IEEE 1394 Arbitration

The IEEE 1394 bus arbitration facilitates guarantee bus bandwidth and maximum bus access time. The arbitration process may only begin after the tree identification and self-id phases establish the root node and the bus topology.

Arbitration begins when one or more nodes detect that the IEEE 1394 bus is idle for a specified interval and sends an arbitration signal towards the root node. Every node knows that its only parent neighbour is closer to the root node than it is and therefore, every node always knows the direction of the root node. Intermediate nodes forward the arbitration signal to their parent's until the signal reaches the root node.

The root node returns a grant signal back to the first port that sent it an arbitration signal. Intermediate nodes again forward the grant signal from the root node back toward the node first sending an arbitration signal. Once a node asserting an arbitration signal receives the grant signal it has won arbitration and may transmit on the bus without risk of transmission collision. After a node completes its transmission, the bus becomes idle again and the arbitration process begins again.

The arbitration process always has a deterministic outcome because all nodes, except the root node, have exactly one parent node. All nodes connected to the bus know the direction to the root node and collaborate to complete each arbitration request. Nodes closer to the root node have an arbitration advantage because their arbitration signal arrives at the root node faster. To overcome this bias, nodes may only successfully arbitrate on behalf of themselves once until all nodes have had an opportunity to successfully arbitrate for the IEEE 1394 bus.

Again, I omit several important details required for the correct operation of the IEEE 1394 bus such as tie resolution and on-the-fly arbitration. These details are not central to this research and may be safely omitted from the discussion.

2.2.4 IEEE 1384 Transmission Types

The IEEE 1394 specification defines asynchronous and isochronous data transmission. A node must win arbitration from the root node in the manner specified in Section 2.2.3 before it may perform either asynchronous or isochronous transmission.

2.2.4.1 IEEE 1394 Isochronous Transmission

Isochronous transmission provides bounded access time and guaranteed bandwidth on the IEEE 1394 bus. Before a node performs any isochronous transmission, it must reserve a channel number and sufficient bandwidth from the isochronous resource manager. Nodes reserve bandwidth in the form of *bus allocation units* that specify the number of quadlets (32-bit unit) that it may transmit once per isochronous bus cycle. The isochronous resource manager grants resources on a first come first server basis until all reservable resources are reserved. The isochronous resource manager allows bandwidth reservation of up to the maximum amount of data that it can transmit during one isochronous period. The isochronous resource manager denies all reservation requests that it does not have sufficient resources to satisfy.

At the beginning of every bus cycle, the isochronous resource manager sends a cycle start packet informing all nodes they may begin arbitrating for isochronous bus ownership. The isochronous resource manager starts a new bus cycle approximately every 125 microseconds. All nodes that successfully reserve bandwidth from the isochronous resource manager will successfully win isochronous arbitration every bus cycle because the isochronous resource manager prevents over-reservation of isochronous resources.

Nodes may begin arbitration for bus ownership after observing an idle bus for the *isochronous gap time.* Nodes that reserve multiple channel numbers arbitrate multiple times per bus cycle to perform one transmission for each reserved isochronous channel. A node may not transmit more than its reserved bus allocation units per bus cycle, although it may transmit less than or none of its reserved bus allocation units. Unreserved and unused reserved bus allocation units become available for

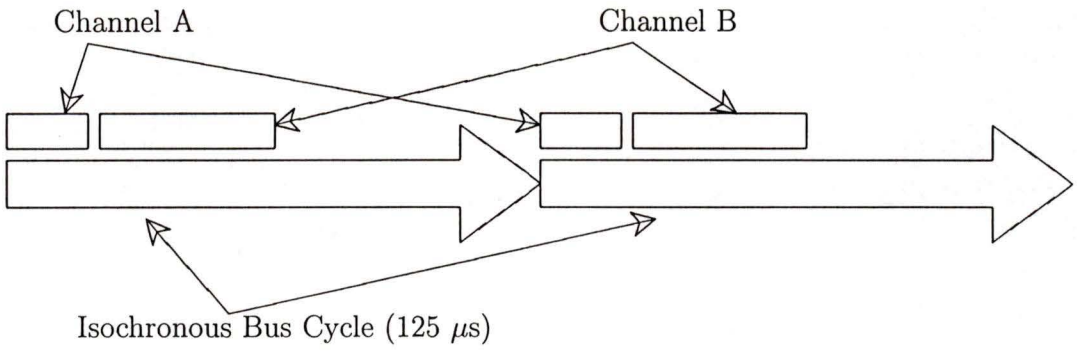

Figure 2.1: IEEE 1394 bus cycle and isochronous channel transmission.

asynchronous transmission.

Figure 2.1 illustrates the isochronous transmission of two channels and their relation to the IEEE 1394 bus cycle. The large horizontal arrow represents the time of a single bus cycle. The transmission of isochronous channels "A" and "B" consume a portion of each bus cycle. Bandwidth unreserved from isochronous resource manager and unused bandwidth from isochronous channels becomes available for asynchronous transmission after the last isochronous transmission.

Isochronous channels are transmitted in the same order each bus cycle provided all channels are ready to transmit at the beginning of the bus cycle. Isochronous channels that become ready after the cycle start may execute out of order.

2.2.4.2 IEEE 1394 Asynchronous Transmission

The asynchronous transmission mode of the IEEE 1394 guarantees maximum bus access time, but does not guarantee bus bandwidth. Nodes performing asynchronous transmissions do not reserve resources from the isochronous resource manager or any other facility of the IEEE 1394 bus.

Asynchronous transmission begins after the last isochronous transmission and completes when the isochronous resource manager sends the next cycle start packet. A node may begin arbitrating for asynchronous bus ownership once it observes the bus idle for an *asynchronous gap interval.* The IEEE 1394 bus does not guarantee a

node will win asynchronous arbitration every bus cycle.

To promote fairness, after a node performs one asynchronous transmission, it disables its own asynchronous arbitration. Once a node has performed an asynchronous transmission, it may not attempt another asynchronous transmission, even if it detects the bus idle for an asynchronous interval gap. It may still arbitrate on behalf of its child nodes. The node may again asynchronously arbitrate on behalf of itself after an arbitration reset. An *arbitration reset* is caused by an idle period on the bus for *a fairness interval gap.* Following an arbitration reset, all nodes re-enable their asynchronous arbitration.

Three possible events, excluding bus reset, occur after the IEEE 1394 bus is idle. The IEEE 1394 bus assigns different priorities to each event to impose an order to the events. The highest priority event, isochronous transmission, has the shortest idle time and occurs first during each bus cycle. Asynchronous transmission requires a longer bus idle time than the isochronous interval gap and always occurs after all isochronous transmission is complete. The lowest priority event, arbitration reset, has the longest idle time and occurs after all other events. Isochronous and asynchronous events always occur every bus cycle, but arbitration reset events do not always occur every bus cycle.

The IEEE 1394 bus supports read, write, and lock asynchronous transactions. These three asynchronous transactions target a specific address in one IEEE 1394 node. The read and write transactions simple read and write memory in a node. The lock transaction performs one of six atomic write operations supported by the IEEE 1394. In the event of multiple simultaneous lock operation attempts to the same address of the same node, only one lock operation will succeed. A failed lock operation is not generally a catastrophic error. Multiple nodes racing for the resource guarded by the lock usually cause a failed lock operation.

2.2.4.3 IEEE 1394 Transmission Type Summary

The IEEE 1394 bus provides asynchronous and isochronous transmission capabilities. The requirements of the data determine whether asynchronous or isochronous transmission is more suitable. There are four basic IEEE 1394 transactions: isochronous write, asynchronous read, asynchronous write, and asynchronous lock. Isochronous read is a passive operation and not considered an IEEE 1394 bus transaction.

Asynchronous transmissions occur at irregular intervals whereas isochronous transmission occur approximately every 125 microseconds. Asynchronous transmissions are addressed to a single node on the bus whereas isochronous transmissions are broadcast on an isochronous channel to all nodes. Asynchronous transmissions are error checked and acknowledged whereas isochronous transmissions are only error checked. Sending node detects failed asynchronous transmissions and, at its option, re-transmits. Since the IEEE 1394 bus is not aware of failed isochronous transactions, it does not re-transmit failed isochronous transmissions.

Asynchronous transmission is suitable for applications that require correct data transmission above guaranteed delivery time and bandwidth. External hard drives are an example of an IEEE 1394 device suitable for asynchronous transmission. Isochronous transmission is suitable for applications that require guaranteed delivery time and bandwidth above correct data transmission. Digital video cameras are an example of an IEEE 1394 device suitable for isochronous transmission. A hard drive relies on uncorrupted and complete data delivery for correct operation. A video camera can tolerate small amounts of data loss, but cannot tolerate untimely data delivery.

2.2.5 IEEE 1394 Node Implementation Options

The IEEE 1394 specification permits several levels of IEEE 1394 implementation. Nodes have the option of not implementing root node capabilities, isochronous resource manager capabilities, or both. There must exist at least one node connected to each IEEE 1394 bus capable of performing the role of root node and isochronous resource manager. Nodes also have the option of not implementing isochronous trans-

mission capabilities entirely. Every node must implement asynchronous transmission and honour the different arbitration gap intervals.

2.2.6 IEEE 1394 Transmission and Reception

The IEEE 1394 specification has similar transmission and reception mechanisms for its two data transmission types. Nodes arbitrate for isochronous or asynchronous bus ownership after they observe the bus idle for the appropriate time. Once a node wins bus ownership, it broadcasts data serially to all nodes on the IEEE 1394 bus.

The IEEE 1394 associates isochronous data with a channel number and broadcasts it to all nodes connected to the bus. Nodes only forward isochronous data from channels they subscribe to to the operating system for further processing. Nodes simply ignore isochronous channels they have not subscribed to. A node may subscribe to as many or as few isochronous channels as required. Nodes that omit isochronous capabilities entirely must still forward isochronous transmissions to their neighbours, but do not process any isochronous data themselves.

The IEEE 1394 broadcasts asynchronous transmission to all nodes connected to the bus. Only the node with a node id matching the transmission destination of the asynchronous packet forwards the data to the operating system for further processing.

2.2.7 IEEE 1394 Bus Configuration

The self-configuration process the IEEE 1394 bus performs is not trivial. Fortunately, the IEEE 1394 manages the bus configuration and does not require outside intervention. The IEEE 1394 bus signals a bus reset when it detects a change in the bus topology. During a bus reset, the IEEE 1394 bus determines its physical and logical topology. Nodes assign themselves node ids during the self-id phase. The last node to assign itself a node id becomes the root node that is responsible for bus arbitration and event signalling. A node's node id may or may not change after each bus reset. Each node maintains a list of valid node ids and explicitly knows which node(s) are filling the role of root node and isochronous resource manager. The IEEE

1394 maintains current bus configuration information, simplifying bus configuration.

Severing a bus into two disjoint groups of nodes causes the two groups to re-form as two separate IEEE 1394 buses. A node in each segment detects the loss of a neighbour node and signals a bus reset that propagates to all connected nodes in each disjoint group. After both bus resets are complete, each bus establishes its new topology and elects a root node and isochronous resource manager. Nodes from the two separate IEEE 1394 buses are, obviously, not able to continue communication outside their new bus configuration.

Conversely, joining two disjoint IEEE 1394 buses together causes the two groups to re-form as one unified IEEE 1394 bus. The point where the nodes are joined detects the presence of a new node and triggers a bus reset in both buses. The bus reset causes all the nodes to organize under the new bus topology. The new bus functions as a unified bus as long as the IEEE 1394 bus configuration meets the requirements set out in Section 2.2.1.

2.2.8 IEEE 1394 Summary

The IEEE 1394 specification is a high-performance, low cost, serial bus. The IEEE 1394 contains a central arbitrator that controls access to the bus and imposes an order of events. The IEEE 1394 bus also contains an isochronous resources manager that prevents the bus from over allocating resources. Each node works collaboratively to transmit and receive data. The IEEE 1394 provides guaranteed bandwidth isochronous broadcast channels and bounded access time asynchronous transmission. The access time to the IEEE 1394 bus does not degrade without bounds as bus utilization increases for either isochronous or asynchronous transmission. Nodes always have bounded access time to the IEEE 1394 bus.

2.3 Comparison of the IEEE 802.3 and the IEEE 1394

After a description of the IEEE 802.3 network and the IEEE 1394 bus, a brief comparison of the two technologies is in order. At the highest level, both provide

high-speed local area data transmission. On a specification level, both the IEEE 802.3 and the IEEE 1394 are vendor neutral and have been adopted by international standards bodies. The differences between the IEEE 802.3 and the IEEE 1394 are their service guarantees, operations, and intended purposes.

A central authority of the IEEE 1394 arbitrates bus resources and bus access. The arbitration layer of the IEEE 1394 guarantees transmission bandwidth and maximum bus access time. The IEEE 802.3 uses the CSMA/CD media access control protocol that does not guarantee bandwidth or deterministic media access. The lack of a central authority of the IEEE 802.3 network simplifies its implementation, but does not provide real-time services or guarantees.

Unlike the IEEE 802.3, there is no possibility of transmission collisions from two nodes simultaneously transmitting on the IEEE 1394 bus. The arbitration process of the IEEE 1394 bus grants exclusive bus access to exactly one node; there is never transmission contention on the IEEE 1394 bus. Contention on an IEEE 802.3 network is resolved through the CSMA/CD media access control protocol that may introduce random delay to transmission.

The IEEE 802.3 network has one transmission mode and the IEEE 1394 bus has two transmission modes. The IEEE 802.3 supports asynchronous access to the network, but it does not enforce fairness or guarantee maximum network access time. The IEEE 1394 supports asynchronous and isochronous access to the bus. The asynchronous mode of the IEEE 1394 enforces fairness and guarantees maximum bus access time. The isochronous mode of the IEEE 1394 guarantees bus bandwidth and maximum bus access time.

The IEEE 802.3 supports a single transaction type and the IEEE 1394 supports four different transaction types. Both the IEEE 802.3 and the IEEE 1394 support a write transaction, but the IEEE 1394 provides write transactions for both isochronous and asynchronous transmission modes. The IEEE 1394 also supports asynchronous read and lock transactions, not supported by the IEEE 802.3.

A service provided directly by the IEEE 1394 bus, valid address listings, is not

supported by the IEEE 802.3 and must be provided by an outside entity. The IEEE 802.3 uses static addresses that are immutable for the lifetime of the node. The IEEE 1394 implements a dynamic node-addressing scheme, where node addresses change after every bus reset. Topology changes of an IEEE 1394 bus cause nodes to re-assign addresses to themselves automatically. Topology changes of an IEEE 802.3 network require either manual reconfiguration of all nodes or an additional service to maintain network addresses.

The intended purpose of the IEEE 802.3 and IEEE 1394 is different, but the two local area network technologies have considerable overlap. Both specifications were designed for a local environment and a small to moderate number of connected nodes. The IEEE 1394 was design as a peripheral interconnect for temporal and non-temporal data. The IEEE 802.3 was designed as a computer interconnection for non-temporal data. The IEEE 802.3 and the IEEE 1394 fill different roles, but they share enough in common that in many situations one may be substituted for the other.

The fundamental difference of the two technologies is that the IEEE 1394 contains a central authority and the IEEE 802.3 does not. The nodes of an IEEE 1394 bus work collectively whereas the nodes of an 802.3 network work independently. The central authority of the IEEE 1394 enables it to provide service guarantees that the IEEE 802.3 cannot.

I will introduce one piece of non-standard terminology and define two other common terms used throughout this thesis. The term *channel* describes a computer interconnect that may use contention or arbitration before transmitting. I will use the term channel as a broader term that covers both an IEEE 1394 bus and an IEEE 802.3 network. The term *frame* refers to a complete message at the application layer. The term *packet* refers to a message at a lower layer such the transport layer found in Figure 2.2 on page 21.

2.4 Software Protocols

Software protocols are pre-defined methods for computer communication. A software protocol provides a set of well-known services that other protocols or applications may use. This research directly uses the services of two software protocols: the Transmission Control Protocol / Internet Protocol (TCP/IP) [9] and the User Datagram Protocol / Internet Protocol (UDP/IP) [10].

The seven layers of the International Organization for Standardization Open System Interconnect (ISO OSI) [11] network model forms the basis of most computer communication protocols. There is also the four layer TPC/IP network model, not illustrated, that combines some adjacent layers and splits other layers of the ISO OSI. Figure 2.2 illustrates the seven layers of the ISO OSI. Communication cascades from the application layer of one node down the seven layers of the ISO OSI, across the physical layer and up to the application layer of another node. In practice, most implementations do not use all seven ISO OSI layers. For example, this research omits the session and presentation layers.

The *Internet Protocol (IP)* [12] provides a foundation layer for both the TCP/IP and the UDP/IP protocols to base their services on. The IP is one of the most basic software protocols used by the Internet. The IP is a network layer responsible for determining how to send packets to their destination. The IP is a stateless protocol meaning that the reception of previous or future packets does not affect the state of current packets. The IP detects and discards corrupted IP packets.

The *User Datagram / Internet Protocol (UDP/IP)* is a simple transport layer protocol with the IP as the network layer. The UDP/IP is a stateless protocol meaning that it doesn't have knowledge of past or future transmissions. The UDP/IP cannot determine if the data it receives is in the correct order or if some of the data packets are missing. The UDP/IP can only determine the integrity of each UDP/IP packet, rejecting those packets that are corrupted. The UDP/IP is widely used for streaming temporal data across the Internet because of its low protocol overhead.

The *Transmission Congestion Protocol / Internet Protocol* is another common

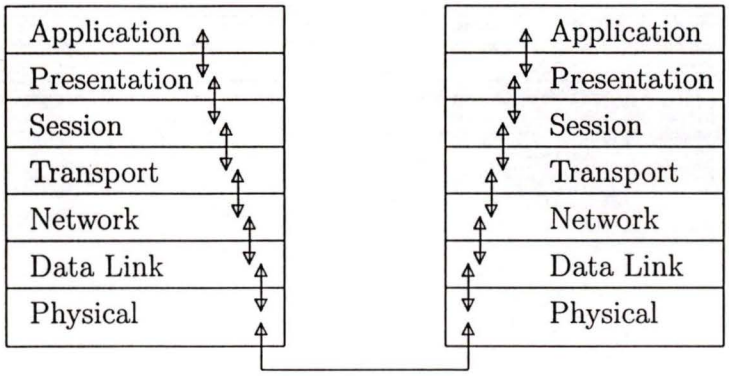

Figure 2.2: ISO OSI network model.

transport layer protocol with the IP as the network layer. The TCP/IP creates a virtual circuit between sender and receiver, guaranteeing the integrity and order of data. When the TCP/IP transmission stream is complete, the TCP/IP tears down the virtual circuit, freeing all resource associated with it.

The TCP/IP is a connection oriented transport layer that relies on the stateless IP network layer to handle the actual packet transmission. The TCP/IP must reassemble the packets it receives from the IP layer into an uncorrupted in-order data stream without missing pieces. The data received by the TCP/IP protocol is possibly out of order because of the multiple routeing paths of IP network layer. The TCP/IP reorders the data packets it receives from the IP layer so that higher layers always receive data from the TCP/IP in the same order it was sent. To ensure a complete transmission stream, the TCP/IP re-transmits corrupted and missing data packets. If all goes well, the application layer receives data exactly as it was sent. Despite the extra overhead of the TCP/IP protocol, it is still widely used for streaming temporal data.

Figure 2.3 illustrates the ISO OSI network model and the layers of the IEEE 802.3, the IEEE 1394, the TCP/IP, the UDP/IP, and the IP. This figure roughly maps the layers of the IEEE 802.3 based channels to the IEEE 1394 asynchronous and isochronous channels. The IEEE 1394 implements its network and transport layers in hardware; the IEEE 802.3 relies on software for the network and transport

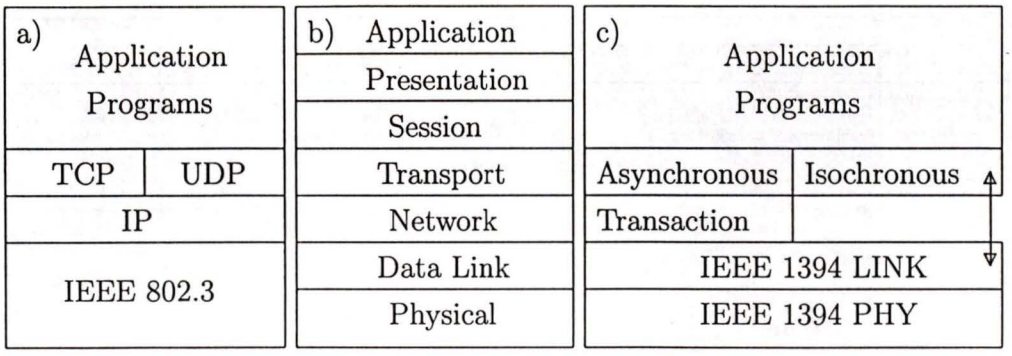

Figure 2.3: a) IEEE 802.3 protocol stack, b) ISO OSI network model, and c) IEEE 1394 protocol stack.

layers.

2.5 Data Types

Different types data have different requirements for correct handling. An application transmitting data through a channel must match the minimum requirements of the data type and the capabilities of the channel. This research examines the special characteristics of streaming digital temporal data.

Data *compression* is a process of transforming input data into a smaller output representation. *Decompression* is the reverse transformation of compression. Decompression creates a suitable representation of the original data from its compressed representation. *Lossless* data compression is a class of compression methods that have an exact reverse transformation. *Lossy* compression is class of compression methods that do not have an exact reverse transformation, but a reverse transformation that is close enough for its intended purpose. Lossy compression is useful when the exact input information is unimportant for its intended purpose, such as data representing picture and sound information. Lossy compression usually compresses data more than lossless compression methods.

Data compression reduces the size of its input data by finding and removing information redundancy. A compression transformation finds information that it can

recreate from a smaller representation. It replaces the original information from the input data with the smaller, compressed, representation in the output data. In most cases, information is not distributed evenly throughout the input data, with areas of low information content and areas of high information content. This causes the degree of compression to fluctuate throughout the input data.

Temporal data is a class of data that has a time axis, also referred to as a time component. The temporal data describes a state that is valid for a finite time interval. The value of temporal data changes as the time component advances. Temporal data may have fixed start and end times or it may be continuous.

Streaming is a technique of flowing data from a producer to a consumer on demand. The consumer contains only a small fraction of the total stream size during any interval of time. A consumer usually, but not always, discards all the streaming data it receives after processing.

Streaming temporal data is a class of data that combines temporal data with a streaming technique. This class of data comprises all audio and video broadcast formats. Although temporal data is often created, transported, and displayed in analog form, this research only explores streaming digital temporal data.

2.5.1 Digital Video

Digital video is a class of formats for creation, transmission, and display of motion video. Like their analog counterparts, digital video contains a sequence of frame information that, when displayed at a sufficiently fast frame rate, creates the illusion of continuous motion. The human eye cannot detect fast, small, discrete changes in the frame sequence, so the image appears smooth and continuous to human viewers.

Digital video is temporal in nature and there are several digital temporal media formats that support digital video. The Moving Picture Experts Group (MPEG) has created several standards for digital video and digital audio. Another industrial research group, the International Telegraph and Telephone Consultative Committee

(CCITT) developed the H.261 standard for video conferencing. A consortium of ten companies created the Digital Video (DV) standard for digital video. For an in-depth discussion of temporal media types, see Gibbs *et al.*[13].

Most digital video standards use compression to manage the massive amount of data needed to encode video. The standard format for television broadcasts in North America is the National Television Standards Committee (NTSC). A NTSC sequence, when digitized, is 640 by 480 pixels and 30 frames per second. If each of the three colour components of each pixel is digitized with eight bits, then one second of uncompressed digitized NTSC video is approximately 27.5 megabytes. Fortunately, video compression can reduce the size of NTSC video streams substantially.

There are currently three ratified MPEG digital video standards. These three MPEG standards support streaming digital video at various rates and picture qualities. The MPEG-1 [14] standard provides approximately one quarter of the resolution of the NTSC. The MPEG-1, at 352 by 240 pixels and 30 frames per second, is ideal for low-bandwidth and low-quality video. The MPEG-2 [15] standard provides superior picture quality to NTSC. The MPEG-2, at 720 by 480 pixels 30 frames per second, is designed for high-bandwidth and high-quality video. The MPEG standards body has not yet ratified the complete MPEG-4 [16] standard, but the visual portions of the MPEG-4 are ratified. MPEG-4 provides a range of picture-qualities and bandwidth ranges in addition to several new features. Table 2.1 contains a summary of the three ratified MPEG standards.

The MPEG standards define the stream format and the compression and decompression algorithms for digital video and audio. All MPEG standards use lossy compression techniques to maximize the compression ratio with a minimum amount of distortion. All MPEG compression algorithms can compress blocks of frames together to find and exploit more information redundancy. The greater the information redundancy found, the greater potential for higher compression ratios. The result of compressing blocks of frames together is that individual compressed MPEG frames may not contain sufficient information for the decoder to reconstruct the original

Table 2.1: MPEG standards for digital video encoding and compression.

	Quality	Bit Rate	Application
MPEG-1	low	1.5 Mbps	CD presentation
MPEG-2	high	4 Mbps and 9 Mbps	HD TV
MPEG-4	low-high	5 kbps to 20 Mbps	many proposed, few exist

frame. The MPEG decoder may require several sequential MPEG frames to completely reconstruct one original video frame.

Motion-JPEG (M-JPEG) is another digital video encoding format. Motion-JPEG encodes each frame as a separate Joint Photographic Experts Group (JPEG) [17] image. Each M-JPEG frame contains the complete information required by the M-JPEG decoder to reconstruct the original video frame.

The MPEG format usually provides better compression than M-JPEG at any given picture quality level. The advantage of M-JPEG over MPEG is reduced complexity of the encoder and the decoder and increased speed of the encoder. Unfortunately, M-JPEG is not an accepted international standard and several incompatible M-JPEG stream formats exist.

2.6 An Introduction to Self-Similarity

A *self-similar* process is a stochastic time series process that exhibits correlation at many different time scales. Self-similarity is sometimes called *fractal* because of its scale-invariant nature. Researchers have observed self-similar structure in natural phenomena [18], computer networks [19], computer communication protocols [20, 21, 22], and temporal data [23, 24, 25, 26].

An interesting consequence of self-similarity is a non-degenerate auto-correlation function. As the lag value, k, increases, the auto-correlation function, $r(k)$, decrease only hyperbolically. In contrast, a *Short-Range Dependant (SRD)* time series' auto-correlation function decays exponentially as k increases. *Long-Range Dependent (LRD)* time series have a non-degenerate auto-correlation function that is not

summable, that is $\sum r(k) = \infty$ for all $k \geq 1$.

2.6.1 Self-Similarity in Computer Communication

The first long term study of self-similarity in computer communications was Leland *et al.*[19]. They captured and analyzed hundreds of millions of high-resolution Ethernet packets over a four-year period. Their work showed that Ethernet traffic exhibits self-similarity across time scales ranging from microseconds to years. Willinger *et al.*[27] and Taqqu *et al.*[1] provide an explanation of the self-similarity observed in Ethernet networks.

Most compressed digital video formats produce a stream that is Variable Bit Rate (VBR). The amount of bandwidth a variable bit rate stream requires is not constant and fluctuates over time. A simple example to illustrate this point is compressing a few frames of complete black followed by a few frames of fast moving detail such as a camera panning across an audience. The amount of compression of the black scene is much greater than the high detail scene.

Several authors [24, 25, 28] have shown that compressed variable bit rate digital video, specifically MPEG and M-JPEG, exhibits self-similar structure. A consequence of self-similarity in motion video is that frame size has strong correlations to near and distant frames.

Many researchers have shown a self-similar structure in several implementations of the layers of the ISO OSI network model. Crovella and Bestacros [20, 21] have shown that World Wide Web (WWW) traffic exhibits a self-similar correlation structure. Paxson and Floyd [22] found that Poisson processes, which are only capable of generating short-range dependant time series, couldn't adequately model packet arrival rates of Wide Area Networks (WAN)s. Their research evaluated long traces of Telnet [29] and File Transfer Protocol (FTP) [30] packet arrivals.

Simulation and study of self-similar traffic must be approached apprehensively. The observation of self-similarity at one layer does not always lead to an observation of self-similarity at higher or lower layers. Self-similar traffic created at the application

layer cannot accurately simulate self-similar traffic at the physical layer and *vice-versa*. Willinger *et al.*[27] and references therein conjecture that the interactions between network model layers may compound or diminish the self-similar correlation structure found at other layers.

2.6.2 The Effects of Self-Similarity on Computer Communications

Accommodating self-similar data on a channel presents additional challenges beyond the special requirements of temporal data. Self-similar data affects common techniques designed to decrease data loss and to increase channel utilization. Traditional modeling techniques are inadequate for modeling self-similar traffic.

Buffering self-similar data fails to adequately prevent data loss due to buffer overflow. Buffering is a common technique to accommodate variable bit rate traffic on a fixed capacity channel. Buffering provides sufficient channel capacity to handle bursts of activity by borrowing capacity during lulls of activity. A sufficiently large buffer along with sufficient channel capacity to meet the mean rate of the stream can handle variable bit rate traffic without data loss.

A problem with buffering self-similar data is the reduction in data loss from buffer overflow attained with short-range dependant data is significantly better than with self-similar data. For traffic with a short-range dependant correlation structures, a linear increase of buffer size results in exponential decrease in buffer overflow probability. Increasing buffer size for traffic with a short-range dependant correlation structure, even by a small amount, decreases buffer overflow probability dramatically. In contrast, increasing the buffer size for traffic with a self-similar correlation structure has minimal impact on buffer overflow [31, 32, 33]. Adding extremely large buffers to nodes to accommodate self-similar data obviously increases the cost of the node, but also increases the delay experienced by data traversing the node.

Statistical multiplexing of several self-similar data streams fails to improve channel utilization. *Statistical multiplexing* is the combination of multiple data streams into one unified data stream. Multiplexing variable bit rate data permits multiple data

streams to share less bandwidth than the sum of the maximum bandwidth requirements of each individual data stream. Statistical multiplexing produces a smoothed aggregate data stream by exploiting the zero correlation between individual streams. Ideally, the unified stream from statistical multiplexing requires only the sum of the means of all individual streams, providing substantial bandwidth utilization gains. Each stream receives sufficient bandwidth during bursts of network activity by utilizing the excess bandwidth of a quiet stream. Unfortunately, aggregating multiple self-similar streams produces a self-similar stream with greater variability [25]. Statistically multiplexing self-similar data increases the probability of buffer overflows, rendering statistical multiplexing an ineffective technique to increase channel utilization [19].

Figure 2.4 illustrates the difficulty of estimating self-similar bandwidth requirements. The Starwars [24] and MPEG [23] movie traces are examples of self-similar, variable bit rate video. The box and whisker plots have three components: centre line, box, and whiskers. These three components correspond to the mean, 95% confidence interval and 99% confidence interval respectively. Each box and whiskers point estimates the mean frame size using the first n frames as the sample. The solid horizontal line of each plot is the mean of the complete movie trace.

Estimating resource allocation levels and channel simulation parameters for self-similar data based on the first n observations and a short-range dependant assumption leads to incorrect and misleading results. In these examples, the 99% confidence interval of the mean estimate does not include the actual mean movie trace, except for a few points. The sample mean estimates do converge to the population mean in both cases, but not as quickly as the confidence intervals wrongly indicate.

Many researchers [22, 34, 35] have concluded that traditional short-range dependant Poisson process modeling techniques fail to accurately simulate real channel operation. The goal of traffic modeling is to evaluate how a proposed channel performs under expected traffic conditions. Traditional traffic models based on short-range dependant time series cannot capture the bursty structure of self-similar data [22, 25].

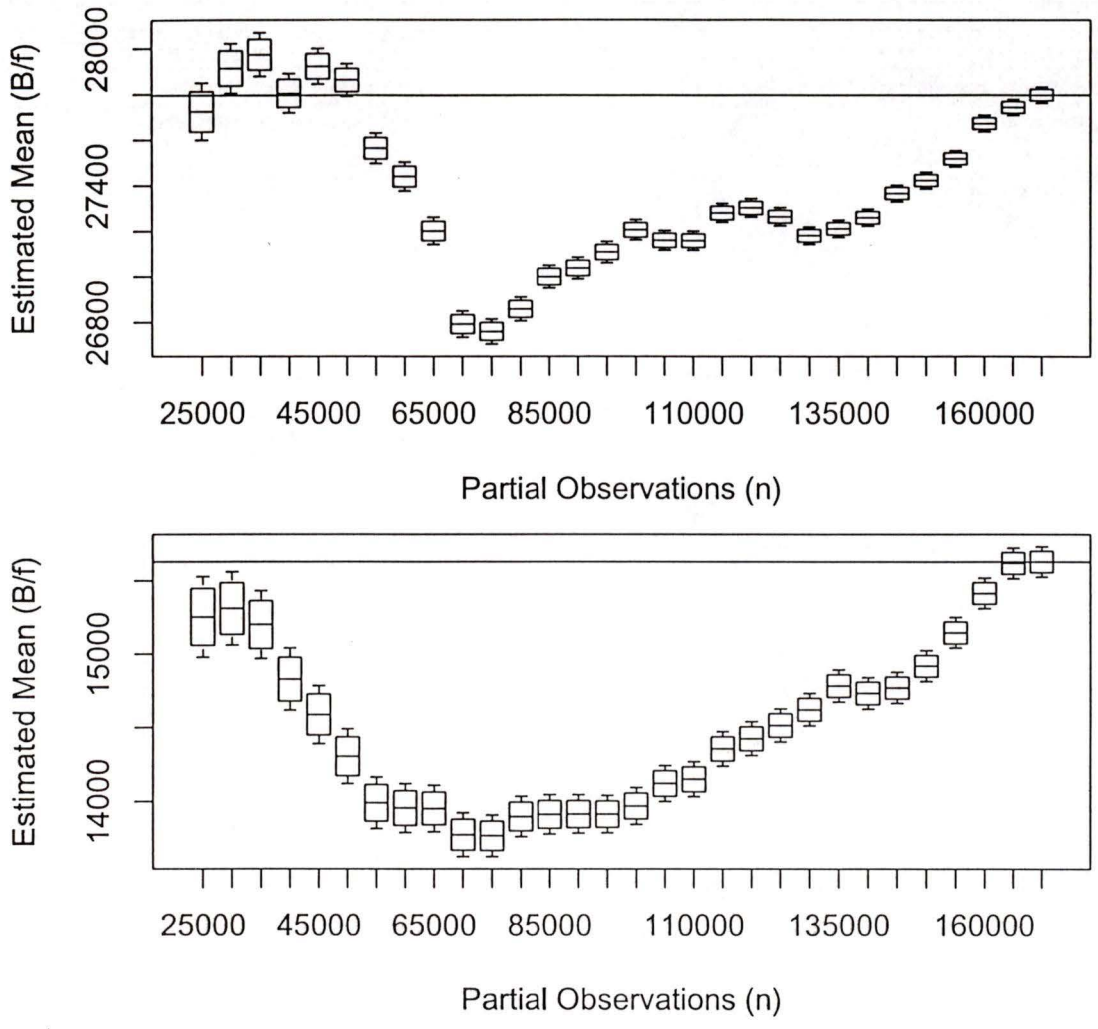

Figure 2.4: Box and whiskers plot of estimated mean with 95% and 99% confidence intervals calculated from the first n frame observations of the MPEG movie trace (top) and Starwars movie trace (bottom).

Generating synthetic self-similar traffic traces has several drawbacks. Exactly self-similar traffic generation techniques are computationally prohibitive for all but short traces [36]. Many researchers [25, 36] have published methods of generating long, approximately self-similar, synthetic traffic traces, but Willinger *et al.*[35] and others point out that the quality of synthetic self-similar traffic models is not well understood. Finally, Huang *et al.*[25] state that selecting parameter values for synthetic self-similar generation models is not trivial because the parameters do not correlate

with attributes of the generated time series.

2.7 Streaming Variable Bit Rate Video

Streaming compressed digital variable bit rate video introduces new challenges that channels must meet. Channels must adequately handle the problems associated with self-similarity, large amounts of data, long connection times, and timely data delivery in order to provide correct handling of variable bit rate video.

Compressed digital video formats, such as MPEG and M-JPEG, have self-similar variable bit rate streams. To correctly handle the self-similarity of the variable bit rate video stream, a channel must accommodate bursts of intense usage and tolerate periods of low usage. Section 2.6.2 discusses several other negative affects of self-similarity relating to computer communication.

A channel transmitting streaming digital video must meet the temporal constraints of the data. To support typical video streams, a channel must transmit a video frame every thirty to forty milliseconds. Channels not capable of transmitting a frame every frame period may use buffers to loosen the time requirements, but not eliminate time requirements entirely. Regardless of a channel employing buffers or not, a channel must provide timely data transmission to correctly handle streaming digital video.

A channel transmitting streaming digital video must support the transmission for the length of the video. To support a typical full-length feature films, a channel must transmit the video stream for approximately two hours. If the channel supports resource reservation, the reservation must last for the length of the complete video stream, which may preclude the channel from servicing future requests. In order to correctly handle streaming digital video, a channel must support sustained transmission for extended periods of time.

A channel carrying digital video must accommodate large amounts of data. Even with compression, a moderate to high quality video requires significant channel bandwidth when compared to other types of temporal media. The high bandwidth

of digital video coupled with long connection-times results in large amounts of data. For example, a two-hour compressed MPEG-2 movie encoded at nine megabits per second generates over eight gigabytes of data. Two hours of uncompressed digitized NTSC video with the same parameters as the example in Section 2.5.1 is a staggering two hundred gigabytes of data. A channel must support sustained high bandwidth for extended periods to correctly support digital video.

A channel must meet the time constraints, the sustained high bandwidth requirements, and the extended connection times of digital video. A channel must also meet the special challenges of self-similar data, which is present in many compressed digital video formats. Failure of a channel to meet all of theses challenges results in incorrect behaviour and an inferior user experience.

2.8 Summary

The IEEE 802.3 is a comprehensive international standard for local area networks employing the CSMA/CD media access protocol. The IEEE 802.3, sometimes called Ethernet, supports only best-effort data transmission of up to 1000 megabits per second.

The IEEE 1394 is an international standard for a local area high-speed serial bus employing arbitration for media access. The IEEE 1394, sometime called FireWire, supports asynchronous and isochronous data transmission of up to 400 megabits per second.

The TCP/IP and the UDP/IP are transport layer networking protocols built on top of the IP network layer. The TCP/IP is connection oriented and provides a virtual channel with error detection and error correction through re-transmission. The UDP/IP is a datagram protocol that does not guarantee in-order or complete delivery of data.

The transmission of variable bit rate compressed video has lulls of low bandwidth requirements and burst of high bandwidth requirements. To correctly transmit compressed digital video requires the channel to properly handle the temporal con-

straints of the data.

A self-similar process is a random process that exhibits correlations at many different time scales. Self-similarity is found in many computer communications such as compressed digital video, IEEE 802.3 at the physical layer, and WWW, FTP, and Telnet at the application layer.

3. Software and Runtime Environment

This chapter describes the runtime environment used and custom software created for this thesis. This chapter provides a general description of the hardware, operating system, application, and Java layers of the runtime environment. This chapter also includes a description of the Java IEEE 1394 interface and client server framework custom software projects. The Java IEEE 1394 interface provides support for the IEEE 1394 that is missing from the core and standard extension Java libraries. The client server framework provides a platform to generate and record streaming data timing information from several channel types in a consistent manner.

3.1 Runtime Environment

A *runtime environment* is a platform for application program hosting and execution. This research uses a non real-time runtime environment that is suitable for general purpose computing. The runtime environment of a general-purpose computer system can be categorized into three broad component classes: hardware, operating system, and application. Together, these components are the basis of a runtime environment.

The runtime environment components form a conceptual hierarchy with the Java layer on the top, application layer on the top, the operating system layer, and finally, the hardware layer on the bottom. The layers of the hierarchy loosely contain their responsibilities and interactions to layers immediately above and below. This loose layer containment is referred to as the *containment guideline*. Users generally interact only with application programs and not with the operating system or hardware directly. The containment guideline limiting layer interaction is not absolute; it is only helpful for building a mental model of the components and their interactions. For example, some peripheral hardware components like keyboards and mice interact directly with the user despite their separation in the runtime environment hierarchy.

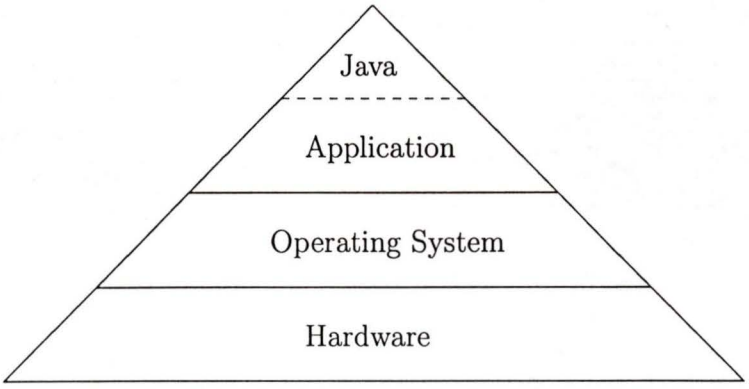

Figure 3.1: Runtime environment conceptual hierarchy.

Figure 3.1 illustrates the conceptual hierarchy of the runtime environment. This model contains a Java application layer, which is not present in the standard three layer runtime hierarchy, on top of the application layer. The Java application layer is an application, but it is an application that deserves special consideration in the conceptual model.

A Java application can be thought of as a regular application that uses the Java Virtual Machine as a middleware. This view of the conceptual hierarchy has only three layers: application, operating system, and hardware. The unified Java application layer / application layer model closely represents the actual implementation of the Java Virtual Machine and Java applications in general purpose runtime environments. For this reason, a dashed line layer in Figure 3.1 separates the Java application layer from the application layer to highlight a conceptual rather than physical separation.

The runtime environment hosting an application has an enormous affect on its performance. A runtime environment with insufficient or inefficient resources may be a poor platform for application execution, causing reduced application performance or failure. For most applications, reduced performance causes longer waiting times, but they still perform correctly. Users may find poor performance applications for many applications to be frustrating, but for some applications, poor performance results in incorrect operation. Applications processing temporal media that don't have their

requirements meet by the runtime environment in a timely manner fail to operate correctly.

3.1.1 Hardware

There are several types of hardware, each providing a service such as computation, storage, communication, or human interface. Hardware components constitute the physical infrastructure supporting its higher layers. Only a few hardware components directly affect the performance evaluated by this research such as processor, memory, and channel adapters. The remainder of the hardware components are important to the operation of the runtime environment, but do not greatly affect the performance metrics under study.

One type of performance critical hardware component is the communication adapter card. *Adapter cards* are hardware implementations of protocols that plug into a computer's peripheral bus providing specialized functionality. This research uses two types of communication adapter cards to support the IEEE 802.3 and the IEEE 1394. These adapter cards provide the runtime environment with physical inter-computer communication capabilities.

Another group of performance critical hardware components is the processor and memory. The speed and size of these components affects every aspect the runtime environment's performance. Sufficient runtime performance is achievable with a 650-MHz Intel Pentium III Central Processor Unit (CPU) and 256 Mbytes of Random Access Memory (RAM).

The performance and capabilities of the research runtime environment are typical of current Personal Computers (PCs) targeted towards consumers. The hardware components of this runtime environment provide inter-computer communication through IEEE 1394 and IEEE 802.3 adapter cards. The interested reader should consult the excellent computer architecture book by Patterson and Hennessy [37] for more information about hardware components and architecture.

3.1.2 Operating System

The operating system is the layer that connects application programs with the underlying hardware. The operating system performs such tasks as process scheduling and resource allocation for all applications. The operating system components for the research runtime environment are the kernel, device drivers, and operating system utilities.

Linux is a modern kernel suitable for general-purpose computing. Among the many uses of today's general-purpose computer is digital audio and video players. These applications are called *soft real-time* because they more or less require deterministic response time from the operating system, but can tolerate occasion delays and still function correctly. The strictness of the deterministic response requirement depends on the application and the type of data. All soft real-time applications can tolerate some amount of missed response times and still operate correctly.

The standard Linux kernel is not suitable for hard real-time applications because it does not provide guaranteed deterministic response time to service requests. *Hard real-time* applications always require deterministic response time from the operating system, which the Linux kernel does not support. There are hard real-time Linux kernel variants such as RTLinux [38], but they are not suitable for this research because they are interrupt driven.

Soft real-time applications can operate correctly despite the lack of guaranteed deterministic response time from the Linux kernel. The Linux kernel supports temporal media applications well when sufficient resources are available. Unfortunately, there is no simple way to prevent the Linux kernel from admitting more applications until insufficient resources remain for soft real-time applications to operate correctly.

Device drivers are modular kernel components that operating system utilities can insert and remove from a running Linux kernel. Device drivers are used to provide a safe software interface for applications to access hardware resources. Device drivers exist to safely and efficiently separate application programs from hardware resources, and abstract disparate hardware interfaces into a common software interface.

The *IEEE 1394 for Linux* [39] kernel module controls the IEEE 1394 adapter card. The IEEE 1394 for Linux kernel module is currently in the development and testing phase, and as such, does not work flawlessly. The IEEE 1394 for Linux kernel module supports nearly all IEEE 1394 bus transactions. The missing functionality from the IEEE for Linux device driver is not required for this research and does not affect IEEE 1394 performance. The IEEE 1394 for Linux device driver has adequate performance for all of its supported IEEE 1394 transactions.

Daemons are operating system utilities that run in the background. They perform tasks at regular intervals or respond to specific events. The research runtime environment uses several daemons. The runtime environment requires several daemons for correct operation, but one is crucial to the research results. The *Network Time Protocol (NTP)* [40] daemon synchronizes a computer's clock with another clock on the Internet or local area network. The NTP daemon maintains the synchronization of the research computers' internal clocks to within several microseconds.

An attractive feature of the Linux kernel, device drivers, and operating system utilities is their open source license [41]. This license allows anyone to freely view and modify the source code. Source code modifications are distributable as long as the modified source code is available under the same license as the original.

The IEEE 1394 device driver required a few changes to make it work properly in the research runtime environment. For example, the IEEE 1394 device driver did not correctly set the contender flag in the IEEE 1394 adapter card. The contender flag instructs an IEEE 1394 node to attempt to become the isochronous resource manager during the next bus reset. Without a node elected as the isochronous resource manager, all attempts to transmit isochronous data fail because no cycle start packet is transmitted. Without the source code of the IEEE 1394 kernel module, fixing problems that prevented smooth operation would have been impossible.

There is much more to a modern operating system than this short introduction. Interested readers should consult Tanenbaum's [42] excellent operating system reference book. For specific information about Linux device drivers, consult the Linux

kernel source [43] or Rubini's [44] Linux device driver book. For general GNU/Linux information, see the book by Welsh *et al.*[45]. The best source of information about the IEEE 1394 for Linux device driver is the freely available source code [39].

3.1.3 Application

The application layer connects the user to the rest of the runtime environment. Applications perform tasks for the user such as word processing and video display. An application program is a list of instructions in a format suitable for the operating system and hardware components to execute. Unlike the operating system layer, where only one operating system exists in a runtime environment, several applications may exist simultaneously in the application layer. The application layer is above the operating system in the conceptual runtime hierarchy because it is incapable of running without the services of the operating system.

3.1.4 Java

The *Java platform* consists of a Java Virtual Machine (JVM) [46] and Java runtime system libraries [47]. The Java platform is a runtime environment in itself and has its own conceptual hierarchy. It has three layers that are roughly parallel to the three bottom layers of conceptual hierarchy in Figure 3.1 on page 34.

The layers of the Java runtime hierarchy are from top to bottom: Java application, Java system packages, and Java Virtual Machine. The Java Virtual Machine restricts Java applications to interacting only with the objects of system and user packages, just as applications in Figure 3.1 interact only with other applications and the operating system. Just as the containment guideline in the rest of the runtime hierarchy is not absolute, the containment guideline applied to Java applications is flexible too.

The Java platform provides a homogeneous runtime environment built over a heterogeneous runtime environment. Unlike traditional compilers, which target a specific CPU and operating system combination, the Java compiler [48] creates Java

byte code targeting the generic Java Virtual Machine. A Java Virtual Machine may execute any Java byte code regardless of the Java Virtual Machine's runtime environment. Sun Microsystems, the creator of Java, officially supports the Java platform for Solaris, Windows, and GNU/Linux runtime environments. Other organizations support the Java platform for their runtime environments. For example, the Java platform for MacOS is available from and supported by Apple Computer.

Java is also a modern, powerful, object-oriented, multi-threaded computer language [49]. The Java language offers increased safety over traditional computer languages, such as C and C++, with strong run-time type checking and careful pointer management.

The streaming temporal media capabilities of Java platform are immature, but may soon be part of the core Java platform. Currently, Java Media Framework [50] that provides the Java platform with streaming media functionality is a standard extension and not a core Java package. The Java platform is network-centric and streaming temporal media is a natural progression for it. The core Java platform supplies a networking package for the TCP/IP and the UDP/IP protocols. The Java platform does not support the IEEE 1394 bus in either core or standard extension packages.

The software designed and created for this research uses two Java technologies. JavaBeans is a reusable software component specification, and the Java Native Interface provides a bridge outside the Java Virtual Machine. The remainder of this section covers the JavaBeans and the Java Native Interface specifications.

3.1.4.1 JavaBeans

JavaBeans [51] is an API specification for creating self-contained reusable software components. A Javabean, also referred to as a bean, is a Java class that adheres to the JavaBeans API specification. Javabeans do not inherit or implement any special Java class or interface[2]. What makes Javabeans different from regular Java classes

[2]Java interface is a construct of the Java language. See The Java Language Specification [49].

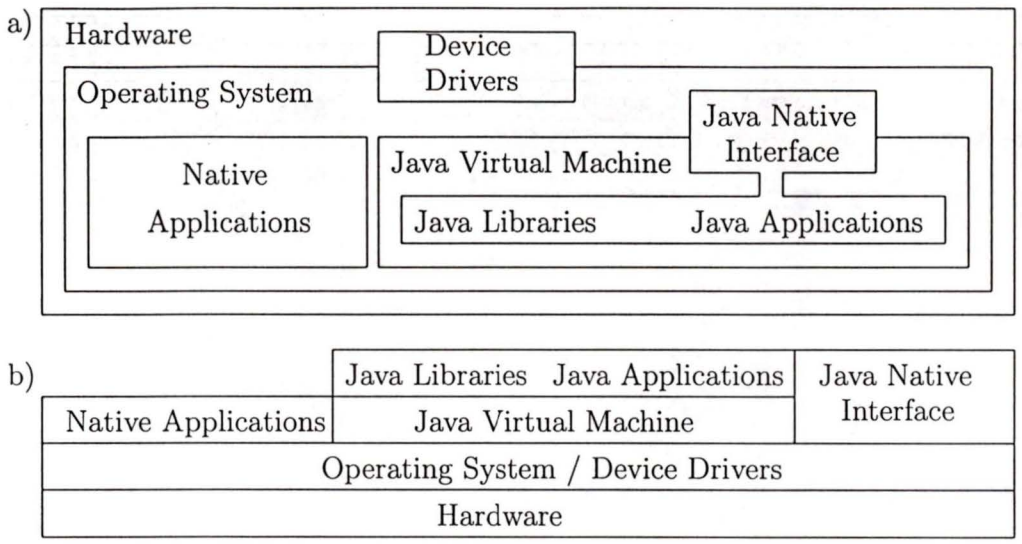

Figure 3.2: Two views of the Java platform and its runtime environment.

is that a Javabean follows the naming conventions of the JavaBeans API specification.

3.1.4.2 Java Native Interface

The *Java Native Interface (JNI)* [52] is an API specification for Java objects to directly access non-Java objects and services. The Java platform normally restricts Java applications to accessing only Java objects in the Java platform. The Java Native Interface allows Java applications to escape the confines of the Java Virtual Machine and call methods in native operating system libraries. Inevitably, loosening the containment guideline has both positive and negative consequences.

Figure 3.2 contains two views of the Java platform's relation with its runtime environment. Both views illustrate the same concept, but from different angles. Figure 3.2 a) shows the hierarchy looking from the top and Figure 3.2 b) shows the same hierarchy from the side. The Java Virtual Machine layer contains Java applications and prevents them from accessing the native runtime environment. The Java Native Interface acts as a bridge to connect Java applications with the underlying runtime environment, bypassing the Java Virtual Machine.

Every Java Native Interface object is composed of two components: a Java class

and a native library. The Java class of a Java Native Interface object is written in the Java language and compiled with standard Java tools. What makes Java Native Interface classes different from regular Java classes is that Java Native Interface classes contain stub (empty) methods with the `native` method modifier. The native library portion of a Java Native Interface contains the implementation of Java class stub methods.

The native library implementation is in a format suitable for the native runtime environment to execute, destroying the portability of the Java object. The native library portion of the Java Native Interface is not restricted by the Java Virtual Machine and is capable of any task the operating system permits. Other Java objects can access a Java Native Interface object's methods, which acts as a proxy for the native library.

The Java Virtual Machine treats native Java methods differently from regular Java methods at runtime. At first invocation of a Java Native Interface method, the Java Virtual Machine loads the native library, links the native library to itself, and maps the Java Native Interface methods to the methods in the native library. A compiled Java Native Interface method is not Java byte code like a regular Java classes. A Java Native Interface method is implemented for one specific runtime environment.

The Java Native Interface performs an analogous role for Java applications as a device driver performs for native applications. Device drivers and the Java Native Interface provide a bridge to a lower layer that is not normally accessible to their respective applications. They provide access to the underlying layers in similar way too. A running kernel loads device driver into itself and a running Java Virtual Machine load Java Native Interfaces into itself.

The Java Native Interface increases the speed and flexibility of the Java platform. By using non-Java methods, the Java Native Interface avoids some performance issues associated with the Java platform. Some functionality, such as array access, is much faster in native languages than the Java language. Java applications that con-

tinuously perform slow Java operations may increase their overall performance when critical portions are implemented directly in the native environment. A pleasant side effect of the Java Native Interface allowing Java applications to access objects outside the Java Virtual Machine is the integration of existing native routines written in non-Java languages. Java applications can access routines in existing native libraries that perform specialized tasks without re-implementing the entire library in the Java language.

The Java Native Interface decreases the safety, portability, and ease of the Java language. The Java language does not have unsafe constructs, such as unchecked array index access and pointer arithmetic. The Java Native Interface circumvents all of the Java language's safety features, allowing Java applications to perform potentially unsafe operations. All Java platforms support all pure Java applications, regardless of the Java Virtual Machine's underlying runtime environment. The native library portion of a Java Native Interface object, which usually contains the bulk of the implementation, is not portable across runtime environments. In the best scenario, the native library must be re-compiled for every runtime environment. In the worst scenario, the native library must be completely re-written for every runtime environment. Creating the native library portion of a Java Native Interface object is not a simple programming task. The design of the Java language encourages simple Java implementations, but the Java Virtual Machine interface to Java Native Interface native methods is not trivial [49, 52]. The native portion of a Java Native Interface object's implementation is not simple.

The negative consequences of the Java Native Interface make it a poor implementation choice for most Java application development. The positive benefits of the Java Native Interface make it a powerful technique for some specialized Java implementations where performance is critical or the functionality is not possible within the Java Virtual Machine. The challenges and restrictions of implementing a Java Native Interface class do not make it useless. Most Java applications use Java Native Interface methods, perhaps without evening knowing it. For example, most of

Table 3.1: Specific components of the thesis runtime environment.

Runtime Component	Details
Java Virtual Machine	J2SE 1.3 HotSpot Client VM [46]
IEEE 1394 Java library	`j1394` package; see Section 3.2.1
TCP/IP Java library	java.net package of J2SE-RE [47]
UDP/IP Java library	java.net package of J2SE-RE [47]
IEEE 1394 device driver	IEEE 1394 for Linux kernel module [39]
IEEE 802.3 device driver	3c59x kernel module of Linux 2.2.16 [43]
IEEE 1394 adapter	ADS Pyro OHCI IEEE 1394 PCI adapter
IEEE 802.3 adapter	3Com Corporation 3c905B 100BaseTX [Cyclone] PCI Ethernet adapter
CPU	650-MHz Intel Pentium III
Memory	256 MB of RAM

the Java Runtime Environment classes distributed from Sun Microsystems have Java Native Interface methods.

3.1.5 Summary

The runtime environment of this research forms a four layer conceptual hierarchy. Each layer abides by the layer containment guideline, which states that layers normally only interact with layers immediately above or below. There are situations where layers interact in ways that violate the containment guideline.

The top layer of the conceptual runtime environment hierarchy is the Java layer. The Java layer provides a virtual homogenous runtime environment for Java applications. Within the Java layer, the Java Native Interface allows Java applications to access the native runtime environment.

The application layer, from a user's point of view, performs all the useful work. The application layer performs task like word processing and video display. Applications may also be implemented in the Java layer.

The operating system layer has executive control of the lower hardware layer and the upper application layer. The operating system contains device drivers that allow applications to access the hardware in a safe and efficient manner.

The runtime environment selected for this research represents a typical modern

desktop personal computer. The hardware components are inexpensive, powerful, and widely available. The operating system is powerful, modern, and widely deployed. The Java Platform Standard Edition is a free virtual runtime environment that supports many runtime environments.

3.2 Custom Software

The research into channel performance analysis in a Java environment requires two custom-built software projects. These software projects provide functionality not provided by core Java libraries or standard extensions. I designed, created, and debugged the Java IEEE 1394 interface and the client server framework software projects.

The Java IEEE 1394 interface is a bridge from the Java platform to the IEEE 1394 bus. The client server framework is a collection of inter-changeable software components that form a tool for generating, transmitting, and observing frame streams.

Together these two software projects generate, transmit, and record frame streams. The client server framework ties together support for the isochronous IEEE 1394, the TCP/IP, and the UDP/IP and support for generation and observation of frame streams. The standard Java `java.net` package provides support for the TCP/IP and the UDP/IP channels, but does not provide support for the IEEE 1394 channel. The Java IEEE 1394 interface provides needed support for the IEEE 1394 channel.

3.2.1 Java IEEE 1394 Interface

The Java IEEE 1394 interface I designed and created is a package of twenty-four Java classes and one Java Native Interface library. The complete Java IEEE 1394 interface is in one Java package called `j1394`. The `j1394` package has a similar structure to the networking classes in the `java.net` package of the Java Platform, Second Edition version.

The `j1394` package transforms Java application requests and data into requests

and data suitable for the IEEE 1394 device driver. The transformations the `j1394` package performs are not always simple data-type conversions. For example, the device driver reports errors encountered on the IEEE 1394 bus with negative integer error codes, but error codes are not the preferred way to communicate exceptional conditions in Java. The `j1394` package converts the numeric error codes received from the IEEE 1394 device driver into Java exceptions with textual descriptions for Java applications.

The `j1394` package only supports the IEEE 1394 for Linux device driver. The `j1394` package is suitable for any Java application running exclusively in a GNU/Linux runtime environment. Supporting other operating systems requires implementations of the native library portion of the `j1394` package targeting those runtime environments. Further development of the `j1394` package could support all platforms supporting Java platform and the IEEE 1394 bus.

The IEEE 1394 for Linux device driver implements a *fill and forward* policy for isochronous input buffers. The device driver accumulates isochronous packets until its input buffer is full, instead of immediately forwarding isochronous packets to the application layer. This policy reduces the number of interrupts the runtime environment must process. For example, if an IEEE 1394 node with a 4096-byte input buffer receives eight thousand 1000-byte isochronous packets per second, the IEEE 1394 device driver *fills* the input buffer with four isochronous packets before *forwarding* the accumulated packets to the application layer. In this scenario, the IEEE 1394 for Linux device driver generates two thousand interrupts per second instead of eight thousand interrupts per second. The IEEE 1394 for Linux device driver increases packet transmission time for reduced overhead and higher throughput.

3.2.1.1 IEEE 1394 Isochronous Resource Management

The `j1394` package contains classes to simplify the isochronous resource management of the IEEE 1394 bus. Actually, the IEEE 1394 isochronous resource manager manages all isochronous resources and the `j1394` package simplifies access to the

isochronous resource manager.

The `j1394` package simplifies access to the isochronous bandwidth and the isochronous channel number resources. A single `j1394` package method performs several IEEE 1394 bus transactions required to reserve bandwidth and channel number resources from the isochronous resource manager. The `j1394` package treats isochronous channel number and bandwidth reservation as an atomic operation, simplifying resource allocation and allocation failures. For example, if the `j1394` package successfully reserves an isochronous channel, but fails to reserve isochronous bandwidth, it releases the reserved isochronous channel before throwing a `ChannelException` exception.

The `j1394` package simplifies releasing IEEE 1394 isochronous resources. The Java programming model allows application programs to generally disregard resources once they are no longer used. The Java Virtual Machine frees Java developers from explicitly releasing unused resources, but also prevents resource leaks. The Java Virtual Machine searches for unused resources and releases them. Java developers should try and manually release external resources like files and network connections, but the Java Virtual Machine will eventually find and release these resources too. The `j1394` package conforms to the Java resource model, forgiving Java developers that do not manually release IEEE 1394 resources. The `j1394` package prevents IEEE 1394 resource leaks by releasing discarded, but unreleased IEEE 1394 resources.

The Java IEEE 1394 interface makes handling IEEE 1394 isochronous resources much easier for Java applications. The `j1394` package functions in a manner consistent with the Java programming model and other Java APIs. The `j1394` package reduces the complexity of reserving and releasing IEEE 1394 isochronous resources, thus reducing programmer effort and the potential for errors.

3.2.1.2 IEEE 1394 Transactions

The `j1394` package wraps isochronous read and write operations in the standard Java interfaces `java.io.InputStream` and `java.io.OutputStream`. Conforming to

a standard interface provides many benefits. Programmers already familiar with the standard interface already know how it works and feel comfortable using it. The interface represents the *best practices* for the problem, encouraging an efficient solution. The use of common stream interfaces makes isochronous streams interchangeable with other Java stream implementations. Several existing Java classes can augment objects implementing the Java stream interfaces, including the isochronous streams of the `j1394` package, with additional features such as buffering, compression, and encryption.

3.2.1.3 Summary

The description of the `j1394` package is fairly short because the task it performs is easy to comprehend. However, as in most things that look simple from the outside, "The Devil's in the details". The `j1394` package must handle concurrency issues from the Java Virtual Machine, the IEEE 1394 device driver, and the IEEE 1394 hardware. The `j1394` package must monitor error conditions from several sources and handle asynchronous call-backs from the Java Virtual Machine and the IEEE 1394 device driver. Finally, the `j1394` package must carefully manage memory shared between the hardware, the operating system, the native library, and the Java Virtual Machine.

The Java IEEE 1394 interface is a standalone package for the IEEE 1394 bus on the Java platform. I only implemented the `j1394` package for the GNU/Linux runtime environment, but it could be extended to all runtime environments that support the Java platform and the IEEE 1394 bus. The Java IEEE 1394 interface, `j1394`, contains methods and objects to simplify resource management, normal operations, and error conditions.

3.2.2 Client Server Framework

The *client server framework*, as the name implies, is a software framework supporting client server communications. The framework is a skeleton implementation of the client server network. The client server framework is written entirely in the

Java programming language and runs on all Java platforms. Both the client and the server applications host pluggable software modules that determine the client's and the server's behaviour. The client server framework hosts three types of modules that determine the traffic generation, the traffic transmission, and the traffic observation behaviour of client server communication.

The client server frame's primary role is to provide an environment for evaluation of streaming client server communications. Replaceable modules implement all operations related to communication. With the implementation of additional modules that perform specific behaviour, the client server framework can simulate additional client server streaming communications.

Client server framework's secondary role is to provide a moderate sized Java application for client server streaming. The Java Input / Output (I/O) routines, including all channel routines, are usually thin Java wrappers around the underlying operating system's I/O routines. A trivial Java application evaluating basic channel operations are only evaluating the very fast Java interface to the operating system's channel routines. A meaningful simulation of a Java application performing channel operations should include a moderately sized Java application. Evaluating Java channel performance with a simple Java application will not introduce latencies and interruptions caused by the Java Virtual Machine and Java libraries that a complete Java application will encounter.

3.2.2.1 Client Server Modules

The server supports all three types of modules and the client, which has no need for traffic generation, supports the other two types of modules. Each module is defined by a Java interface, and each module implementation is interchangeable with all other module implementations. A *Java interface* defines what methods the module provides, but does not provide the implementation of the methods. The realization of each module implements the methods of the module's common interface. Each module interface conforms the JavaBeans API specification introduced in section

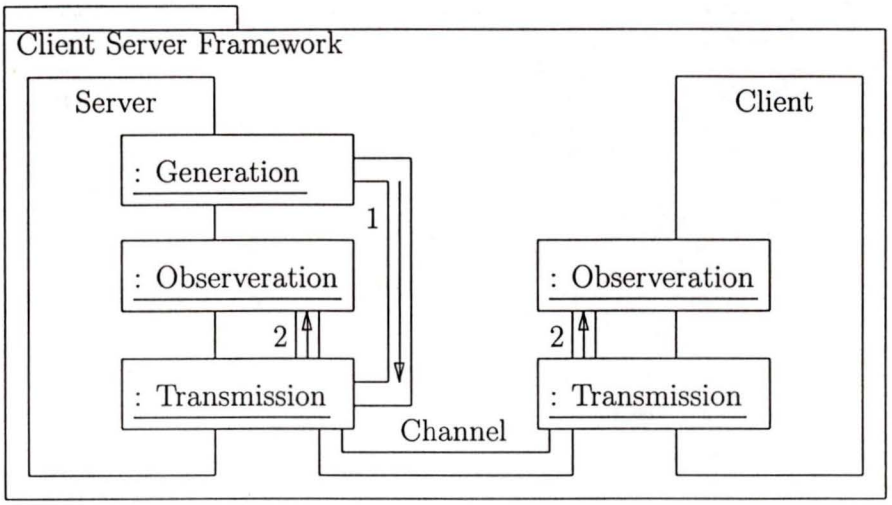

Figure 3.3: Collaboration diagram of the client server framework and modules.

3.1.4.1 on page 39.

Each module realization is a Javabean that implements one of the module interfaces in Figure 3.3. A Javabean, like all Java classes, that implements a Java interface becomes interchangeable with all other Java classes that also implement the same Java interface. The JavaBeans API specification defines a framework that handles inter-bean communication and configuration of the client server framework's modules.

Figure 3.3 illustrates a collaboration diagram of the client server framework and its traffic modules. A collaboration diagram is part of the Unified Modeling Language (UML) [53] specification for object specification and visualization.

3.2.2.2 Generation Module

Generation modules create a frame stream according to their internal traffic generation model. Only the server portion of the framework supports generation modules because only the server is responsible for generating traffic. Label 1 in Figure 3.3 represents the generation module's message sent to the transmission module. This message instructs the transmission module when and how large of a frame to transmit to the client.

The client server framework includes seven realizations of the generation module, implementing a variety of traffic generation models. There are two categories of generation models implemented for this research: pseudo-random models and deterministic models.

Pseudo-random models generate traffic according to a pseudo-random process shaped by the module's parameters. Deterministic models generate traffic based solely on their parameters and deterministic input. Both pseudo-random and deterministic modules can recreate a packet sequence if the generation modules have equivalent parameters.

The pseudo-random process shaping the frame stream of a pseudo-random generation module always generates an identical sequence of numbers when its pseudo-random number generator is initialized with an equivalent seed. A pseudo-random number generator configured with a different seed creates a different pseudo-random number sequence. The seed of a pseudo-random process is initialized with determines the frame sequence a pseudo-random models produces.

The client server framework includes four pseudo-random generation modules. Each of these modules is capable of generating a pseudo-random sequence of frames according to their internal generation model, pseudo-random process, and module parameters. The *inter-arrival time* module generates a frame stream of constant sized frames with a pseudo-random process determining the inter-frame gap. The *pulsed variable bit rate* model generates a frame stream of random sized frames with a fixed inter-frame gap. The *modulate poison process* model generates a frame stream based on a simple two state Markov chain. The *noise* model generates a frame stream of random sized frames and random inter-frame gaps.

The statistical distribution of a random process influences its shape. The client server framework supplies pseudo-random processes for the following five statistical distributions: Normal, Exponential, Extreme, Gaussian, and Pareto. The configuration of each pseudo-random module includes the type of pseudo-random distribution, a pseudo-random number generator seed, and any distribution parameters. This the-

sis does not present the results from pseudo-random models because they are only capable of short-range dependant frame sequences, which are not suitable for simulating real channel operation. Previous research [22, 34, 35] indicates simple pseudo-random traffic models are not sufficient for evaluating streaming channel performance.

The client server framework includes three deterministic generation modules. Each of these modules is capable of generating a deterministic sequence of frames according their internal generation model and module parameters. The *maximum* traffic model generates a frame stream of fixed sized frames without any artificial inter-frame delay. The *pulse constant bit rate* model generates a frame stream of constant sized frames with constant inter-frame gap. The last module, the *playback* model, has an internal generation model driven by an external source. The external source may be a traffic trace created from another traffic module or an externally generated synthetic trace.

All traffic modules discussed in 3.2.2.2 on page 49 generate frames according to an internal model, except the playback module. Instead of using an internal traffic model, the playback module uses an external model created by an external source.

The frame sequence from a deterministic model is never exactly deterministic or repeatable because of the non-deterministic runtime environment. The client server framework runs within a runtime environment that does not guarantee process scheduling. When possible, experiments with the client server framework are repeated several times to overcome the non-determinism of the runtime environment and provide statistically valid results.

3.2.2.3 Observation Module

Observation modules receive packet event notification from transmission modules. Both the client and server portions of the framework support observation modules.

The client server framework includes one realization of the observation module called the *recorder*. The recorder creates an archive the frame stream sent by a

transmission module. The recorder module permanently stores the archive it creates in a disk file for future use. The archive contains summary information about each frame suitable for statistical analysis or stream recreation.

The recorder archives frame size, frame sequence number, start frame timestamp, and stop frame timestamp information. Frame size is the size of the frame at the application layer and does not include information about fragmentation at lower layers. The frame size also does not include the headers wrapped around the packet by lower layers. The sequence number is an increasing integer count of frames observed for each transmission connection. Sequence numbers start from zero and increment with each frame. The start frame timestamp is the time the transmission module begins processing the first packet of a frame. The stop frame timestamp is the time immediately following the transmission module processing the last packet of a frame.

3.2.2.4 Transmission Module

Transmission modules support all aspects of communication required for client server communication. Both the client and server portions of the framework support the transmission modules. Label 2 in figure 3.3 on page 49 represents the transmission module's message sent to the observation module. This message informs the observation module about each frame processed, either sent or received, through the channel.

The client server framework includes three realizations of the transmission implementing isochronous IEEE 1394, the TCP/IP, and the UDP/IP channels. Each traffic module supports the following set of common abstract operations: initiate connection, terminate connection, accept connection, send data, and receive data. Transmission modules encapsulate their channel specific operations for these abstract operations.

The connection lifecycle between transmission modules follows four steps. First, a transmission module listens for and accepts incoming connection requests. Accept-

ing requests means listening, in a channel specific way, for an indication that another transmission module wants to connect. Second, a transmission module sends a connection indication request to an accepting transmission module. Third, the two transmission modules create a connection suitable for streaming data. Fourth, a transmission module terminates the connection, releasing all the resources allocated connection.

All transmission modules contain a maximum packet size that when set, fragments frames larger than the maximum packet size into multiple smaller packets. The receiving transmission module reconstructs frames from the packet pieces. By performing its own packet fragmentation, the client server framework implements a common fragmentation policy for all channels and eliminates special handling of very large packets.

Each packet a generation modules creates contains information the client server framework uses to check the integrity of the packet stream and reconstruct frames. The generation modules insert a header into each packet containing the following information: frame sequence number, packet sequence number, and frame size. The transmission module uses this information to reconstruct fragmented packet into frames, detect lost packets, and detect corrupted packets.

3.2.2.5 Server Application

The server portion of the client server framework in Figure 3.3 on page 49 simulates a Java server application streaming data to one or more clients. The server is capable of hosting multiple instances of all three module types simultaneously. The server relies on the generation, transmission, and observation modules it hosts to perform most of its work.

The server is a powerful Java application with many features and configuration options. The server initially configures itself at start-up with command line arguments and a configuration file. The server's graphical user interface (GUI) embeds the graphical interfaces' of all the modules it hosts. The server's graphical user in-

terface provides a unified interface to the server and the modules that determine its behaviour. The unified interface configures and displays status information regarding the server, its modules, and all current channel connections. The server application and its modules may be re-configured through the server's graphical user interface anytime during operation.

The server also includes support for advanced configuration of modules through Java object serialization and cloning. Java *serialization* converts a Java object into a string of bytes that it can later use to regenerate the Java object. *Cloning* replicates a Java object so there are two active objects with identical state. The server application uses object serialization to store and later retrieve modules with their entire configuration intact. The server uses cloning in a similar manner as object serialization, but it creates two active instances of an object instead of one active instance and one inactive instance. By supporting many features, the server application implementation is suitable to simulate a moderately sized Java application.

3.2.2.6 Client Application

The client portion of the client server framework in Figure 3.3 on page 49 simulates a Java client application requesting and receiving streaming data from a single server. The client is limited to hosting one instance of each of the transmission and observation modules. Not unlike the server, the client application relies on its hosted modules to perform most of its work.

The client is a simple Java application with minimal features. The client application configures itself at start-up with command line arguments and a configuration file. The client application provides no mechanism for re-configuration once it starts executing. The client application is a short-lived process supporting only a single server connection before terminating. The client application does not have a graphical user interface, and does not support the graphical user interfaces' of its hosted modules. By being as simple as possible, the client application implementation is small and uncomplicated.

3.2.2.7 Summary

The client server framework is a collection of inter-changeable software components that form a tool for generation, transmission, and observation of a frame stream. Generation modules generate a stream of frames according to their internal traffic model. Transmission modules support channel communication through the isochronous IEEE 1394, the TCP/IP, and the UDP/IP. The observation module archives summary information of the frame stream. The analysis phases of this research uses the data recorded by the client server framework to evaluate the performance of the Java platform for streaming temporal media.

3.2.3 Software Design Goals

The `j1394` package and the client server framework projects each have their own set of design goals and requirements. The central design goal of both software projects is usability. I defined several smaller sub-goals for the `j1394` package and the client server framework to accomplish the overall usability goal of each project. The sub-goals of the client server framework are modularity, comprehensive configuration, and robustness. The sub-goals of the Java IEEE 1394 interface are simplicity, performance, interoperability, and robustness. Combined, these goals contribute to create two software projects that meet overall goal of usability.

The modularity of the client server framework allows the construction of diverse testing and recording environments. The framework accepts different modules to generate traffic and record channel activity. Adding or removing traffic generation modules to the framework creates a wide-range of possible traffic simulations. Creating new modules increases the scope of possible simulation scenarios. Modularizing the traffic generation, traffic transmission, and traffic observation capabilities of the client server framework simplifies future development.

Comprehensive configuration of the client server framework allows extensive adjustment to traffic generation, traffic transmission, and traffic observation. Changing a traffic module's parameters shapes the traffic it generates. The other module

types support configuration to their operation as well. The construction of a client server environment is configurable with the many implementations of the modules of the client server framework. Configuration of the client server framework occurs through serialized traffic modules, configuration files, graphical user interfaces, and command line arguments. Each configuration method contributes to comprehensive configuration of the client server framework.

Robustness of the client server framework allows long running and repeatable experiments. Robustness in the context of this software project is detecting invalid conditions and either recovering gracefully or failing safely and predictably. Incorrect conditions result from exceptional data, unexpected environmental conditions, or software errors. The client server framework performs integrity checks on all input data to detect invalid conditions.

The simplicity of the IEEE 1394 Java interface encourages programmers to incorporate it into their projects. The helper classes of the `j1394` package encapsulate the difficult task of resource management. Java applications can rely on the IEEE 1394 management and configuration provided by the `j1394` package or manually perform their own management and configuration. The `j1394` package does not provide management and support of some unusual IEEE 1394 configuration options, but these operations can always be performed manually with its basic IEEE 1394 transactions.

The performance of the IEEE 1394 Java interface makes it suitable for applications requiring low-overhead access to the IEEE 1394 bus in a Java environment. The Java application, the Java Virtual Machine, the IEEE 1394 for Linux device driver, and the runtime environment, are the limiting factors to the performance of the `j1394` package. I have profiled the entire `j1394` package for excessive garbage creation and other performance pitfalls. By improving a few methods of the `j1394` package, I was able to substantially decrease its overhead and increase overall performance. The `j1394` package does marginally increase access time to the IEEE 1394 bus, but this should be acceptable for Java applications because any application that

requires absolute maximum performance is not suitable for a Java implementation.

The interoperability of the Java IEEE 1394 interface allows easy interchange with other Java stream classes. The `j1394` package uses the core Java platform interfaces `java.io.inputstream` and `java.io.outputstream`, common to all Java core and standard extension stream libraries. Several classes in the core Java library augment classes conforming to these standard stream interfaces, such as the `j1394` package, with additional characteristics. Java applications can use the `j1394` package with minimal implementation changes to replace the TCP/IP or other stream types.

The robustness of the `j1394` package allows the long running and the high volume data transfers this thesis research requires. The correct operation of the `j1394` package is crucial for the client server framework. When the `j1394` package encounters an error condition that it does not have sufficient information to take corrective action, it raises an exception for a higher level to handle. The exception model of error handling is fundamental to the Java language and very powerful.

The goals of comprehensive configuration, modularity, interoperability, performance, and simplicity facilitate the larger goal of usability. I cannot comment on how others may feel about the usability of these two software projects, but I found them suitable for capturing the test data of this thesis.

3.3 Java Performance

Application performance problems can cause annoying delays or catastrophic failure. Small delays are usually tolerable, although undesirable, for most applications. Large and frequent delays are at best very undesirable and at worst cause catastrophic application failure. Application performance is a compromise of functionality, implementation complexity, and design. For acceptable operation, performance issues must be identified and those issues that cause unacceptable performance decreases must be addressed.

Java performance in the context of the client server framework and the `j1394` package is availability. Availability includes the ability to respond to events quickly

and without interruption.

The start-up phase of the server is an example of an annoying delay, which is an acceptable compromise of application performance for application features. When the server begins, it performs a great deal of work to support its configuration and graphical user interfaces. It takes the server several seconds to finish its start-up phase and enter a usable state. The delay the server incurs is unfortunate, but isolated to the start-up phase. The length of the state-up phase does not impact the critical streaming functions of the server. The features that cause the performance problems of the server start-up outweigh their performance cost.

An early version of the `j1394` package had a performance problem that is an example of catastrophic application failure. A brief description of the problem, its cause, and solution follow. The `j1394` package allocates a new buffer for every packet it receives. The Java Virtual Machine handles this well when the packet allocation is relatively infrequent. The problem arises when the `j1394` package processes 8000 isochronous packets per second. The `j1394` package instructs the Java Virtual Machine to create buffers to hold each of the thousands packet it receives from the IEEE 1394 bus. The Java Virtual Machine quickly exhausts all of its available memory and must begin reclaiming thousands of pieces of discarded memory. The `j1394` package runs for a few seconds then pause for several seconds as the garbage collector cleans up discarded memory to replenish its pool of free memory. An application using the `j1394` package would experience data loss and frequent large interruptions. Applications processing temporal media and using the `j1394` package have catastrophic failures. The current `j1394` package implementation corrects this performance problem, and now creates almost no garbage during normal operation. The solution required a change to the `j1394` package to reuse packet buffers, reducing the amount of garbage collection to almost zero.

A major source of Java application performance problems is the result of excessive garbage collection. Discarded Java objects, called garbage, accumulate until the garbage collector finds and reclaims them. The Java garbage collector runs automat-

ically when certain memory conditions exist. In general, the Java Virtual Machine activates the garbage collector when a predefined memory allocation threshold is crossed. The required memory conditions for the Java Virtual Machine to start the garbage collection thread depend on many factors, and cannot be exactly specified for the general case. When the Java Virtual Machine schedules the garbage collection thread to execute, all other Java thread execution halts [3]. Java applications that create large amounts of garbage execute slowly and erratically as the garbage collector executes longer and more frequently.

Three features of the client server framework that cause performance problems are object serialization, eXtensible Markup Language (XML) [54] configuration, and XML reporting. Object serialization does not produce a significant amount of garbage in itself, but an object serialization implementation detail creates a significant amount of garbage elsewhere. To reduce the amount of garbage created by the observation module, I replaced object serialization with simple binary data output. This does not affect the portability of the data or preclude the representation working with future versions of the Java platform. In my experience simple binary representation of data classes is superior to object serialization. Reading, writing, and processing XML documents with the current Java standard extension XML library [55] parsers creates large amounts of garbage. The time required by the Java Virtual Machine to reclaim the garbage causes unacceptable delays for processing temporal media. To reduce the performance penalty caused by XML processing, I replaced all the XML configuration features with simple text files.

3.4 Supporting Software

I used many software packages and tools during the development, testing, debugging, and analysis phases of this research. The freely available software tools Forte for Java [56] and Source Navigator [57] are invaluable for development and testing

[3] This is generally true even for multi-processor environments and the current Java Virtual Machine.

of Java and C applications. A single integrated tool, the R Statistical Environment [58], performs all the statistical analysis, data plots, and statistical summary tables. Several other tools and utilities also contribute to the final research product.

R is an interpreted language [59] and environment especially capable of, but not limit to, statistical analysis and graphics. The R Environment is capable of performing statistical analysis of millions of data points. The R Environment easily produces graphics summarizing raw data, descriptive statistics, and time series analysis. The output from the R Environment is configurable and exports graphics and data to numerous formats. None of the other numerical tools that I evaluated support the range of statistical and graphic capabilities or the power to process millions of data points that the R Environment supports.

3.5 Summary

The runtime environment consists of hardware, operating system, application, and Java layers. The runtime components form a conceptual hierarchy following the containment guideline.

The Java environment provides a homogeneous runtime environment for applications written with the Java programming language. The Java environment supports several technologies such as JavaBeans for reusable software components and the Java Native Interface for direct interaction with the underlying runtime environment.

The Java IEEE 1394 interface and the client server framework are software projects created for the research of this thesis. The `j1394` package adds support for the IEEE 1394 bus to the Java environment. The client server framework is a skeleton for streaming data experiments. The client server framework includes several modules that implement the behaviour of the streaming traffic generation, transmission, and observation. The recorded traffic from the client server framework is analyzed later.

During the development of the two custom software projects, I encountered several performance problems. I discuss these Java performance problems, their causes, and their solutions.

Finally, I credit several pieces of supporting software that were instrumental to the development and analysis of this research.

4. Maximum Utilization Experiment and Results

This chapter describes the first group of channel performance experiments, the channel performance metrics, and the statistical methods that support the significance of the results. This group of experiments evaluate the performance of maximum transmission rate of each channel type with several frame sizes.

4.1 Experiment Description

This group of experiments compares the maximum utilization performance of the isochronous IEEE 1394, the TCP/IP, and the UDP/IP channels. Chapter 3 describes the runtime environment that affects channel performance and the client server framework that creates the frame stream and gathers the raw frame timing information.

The client server framework collects frame timing information from a frame stream generated with the maximum generation module. The frames created by the maximum generation module all have the same size and have no artificially induced delay between frames.

The hardware layer of the runtime environment consists of two identical computers connected by 100 megabits per second IEEE 802.3 and 400 megabits per second IEEE 1394. During the maximum utilization experiments, the two physical channels only carry client server framework packets. The operating system does not permit packets not related to the client server framework to use the physical channels for the duration of each maximum utilization experiment run.

The maximum utilization experiment configuration isolates both physical channels from external channel communication that would induce unpredictable contention time. Isolation from external traffic is required for the IEEE 802.3 to avoid contention and collisions necessary to produce repeatable frame timing results. Isolation of the IEEE 1394 from external traffic does not significantly change its transmis-

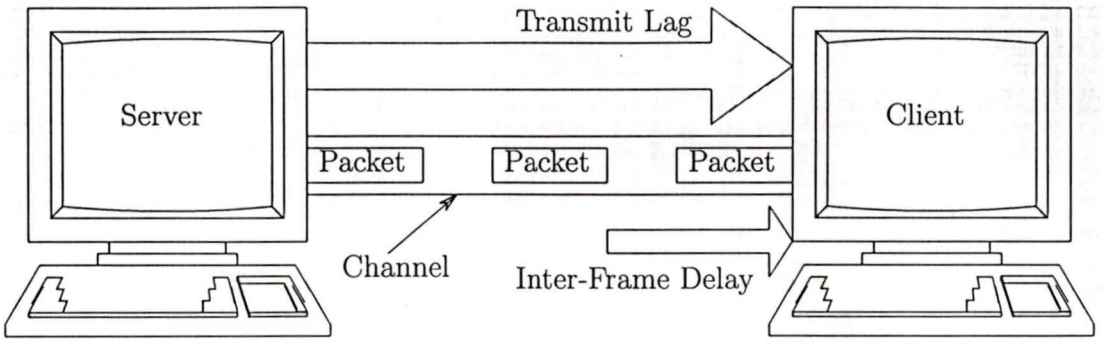

Figure 4.1: Performance metrics of the maximum utilization experiments.

sion timing behaviour because it is not affected by channel contention or collisions. Although the order of isochronous channels transmission may change with external traffic, this change would account for very small frame timing changes. For the highest quality experiment data and simplest experiment configuration, the maximum utilization experiments dedicate physical channel resources to remove the effects of channel contention and transmission collisions.

4.1.1 Channel Performance

The maximum utilization experiments use two metrics derived from frame timing information to evaluate channel performance. *Transmit lag* is the time that elapses from the start of a frame's transmission to the end of the frame's reception. *Inter-frame delay* is the time that elapses between the reception of successive frames. The client server framework takes all frame timing measurements at the Java application layer. The maximum utilization experiments do not evaluate other channel performance metrics, such as bit error rate and bandwidth.

Transmit lag includes the time that elapses in all layers of the network model as the frame moves from the sender's application layer, through the physical channel medium, to the receiver's application layer. Transmit lag combines the effects of channel buffer delays, protocol processing overhead, operating system overhead, process scheduling delays, bus arbitration or network contention, and physical transmission. It is critical to understand and manage transmit lag to meet the time requirements

of streaming temporal media.

This research regards the results from inter-frame delay cautiously because of the averaging affect of protocol buffering. Protocol buffering smoothes inter-frame delay by absorbing frame bursts until adjacent layers are ready to process them. Inter-frame delay values for individual frames actually represent the inter-frame delay average from several sequential frames. This is not to say that inter-frame delay is a completely useless performance metric. Statistical methods that utilize averaging, such as the mean, provide useful performance metric results. Additionally, outlier inter-frame delay values provide important information about channel saturation, process scheduling, and protocol operation.

All performance metrics are affected by deterministic factors such as electrical signal propagation delay through the media and non-deterministic factors such as cache-hit ratios. The non-deterministic experimental runtime environment causes channel performance to vary within experimental blocks, even when all controllable factors remain constant.

4.1.2 Maximum Utilization Experiment with Default Buffers

This experiment gathers frame information with channel input and output buffers set to their default size. The maximum traffic model, described in Section 3.2.2.2, generates 50,000 fixed sized frames for transmission through the three channel types.

This experiment has two factors of three levels each, producing nine experimental blocks. The frame size factor has three levels: 992 bytes, 396 bytes, and 24 bytes. The channel type factor also has three levels: the isochronous IEEE 1394, the TCP/IP, and the UDP/IP. All channels use their default buffer input and output buffer sizes. The TCP/IP and the UDP/IP have 32,767-byte input and output buffers. The isochronous IEEE 1394 has a 4096-byte input buffer and a 0-byte output buffer. This experiment repeats each of the nine experiment blocks five times for a total of forty-five experimental runs.

4.1.3 Maximum Utilization Experiment with Reduced Buffers

This experiment gathers frame information with reduced input and output buffers. This experiment also uses the maximum traffic module to generate 50,000 fixed sized frames through two of the three channel types. This experiment omits the isochronous IEEE 1394 because it does not have configurable buffer sizes.

This experiment has two factors producing six experimental blocks. The frame size factor has the same three levels as the previous experiment: 992 bytes, 396 bytes, and 24 bytes. The channel type factor has only two levels: the TCP/IP and the UDP/IP.

The transmission module of the client server framework configures the TCP/IP and the UDP/IP to use reduced input and output buffer sizes of 2048 bytes. This experiment repeats each of the six experiment blocks five times for a total of thirty experimental runs.

4.1.4 Frame Size

The three levels of the frame size factor for both maximum utilization experiments provide channel performance information about small, medium, and large frame sizes. Two conditions guide the frame size select process. First, all frame sizes must satisfy the constraints of all layers of the three channels. Second, the frame sizes should provide some useful information about streaming data. The frame selection process uses four physical constraints one usefulness criterion to determine the final frame sizes.

Both maximum utilization experiments purposely use frame sizes that are small enough to avoid fragmentation by the client server framework or any layer of the channels. The maximum utilization experiments reflect the performance of application frames that fit completely inside a single packet at the physical layer.

To avoid wasted packet payload, all frames must be divisible by the channel's Minimum Addressable Unit (MAU). The *minimum addressable unit* is the smallest quantity of data that a channel's physical layer can process. Higher layers in a

channel's network module may process data amounts smaller than a physical layer's minimum addressable unit by padding the data out to the next largest minimum addressable unit boundary. This padding, although usually very small, is wasted packet payload. The minimum addressable unit of the all IEEE 1394 transactions is one quadlet, which is 32 bits, and the minimum addressable unit of the IEEE 802.3 is one octet, which is 8 bits. The Least Common Multiple (LCM) of thirty-two bits and 8 bits is 32 bits. All frame sizes must be multiples of 32 bits (4 bytes) to avoid wasting packet payload.

To avoid fragmentation by the channel, all frames must be smaller than the channel's Maximum Transmission Unit (MTU). The *maximum transmission unit* is the largest packet a channel can transmit without fragmentation. The maximum transmission unit of the TCP/IP, the UDP/IP, and the IP is 65,535 bytes including all protocol headers. The theoretical maximum transmission unit of the isochronous IEEE 1394 is 6144 bytes at 400 megabits per second, but the actual maximum transmission unit of the IEEE 1394 for Linux is 4088 bytes. The maximum transmission unit of the IEEE 802.3 100BaseTx adapter cards is 1500 bytes. The smaller maximum transmission unit of two physical channel layers is 1500 bytes. All packets at the physical layer must be 1500 bytes or less in size.

The maximum packet size is 1500 bytes, but this size must include all headers appended to each frame by higher layers. Failure to take protocol overhead to frame size into account may result in unwanted frame fragmentation. The "TCP/IP Tutorial and Technical Overview" [60], states the headers without any options for the TCP is 40 bytes, the UDP is 16 bytes, and the IP is 20 bytes. The largest combination of protocol headers under normal conditions comes from the TCP/IP at 60 bytes. A frame grows a maximum of 60 bytes from the application layer to the physical layer. All frames at the application layer must be 1440 bytes or less in size.

The minimum frame size that can accommodate the client server framework's integrity checks is 12 bytes. The client server framework inserts integrity check information into the first 12 bytes the each frame; the frame size is not increased because

of the integrity check information. The integrity checks contain information about frame sequence, packet sequence, frame size, and packet size. Client server framework uses this information to detect transmission errors and re-assemble fragmented frames. All frame the client server generates must be a 12 bytes or greater in size.

The channel constraints limit the range of frame size to between 12 and 1440 bytes, and limit all frame sizes to multiples of 4 bytes. To provide useful insight and information about streaming temporal data, three frame sizes representing small, medium, and large transmission rates are selected. The small frame size of 24-byte frames at 8000 frames per second corresponds to the MPEG-1 target stream rate of 1.5 Mbps. The medium frame size of 396-byte frames at 8000 frames per second corresponds to the DV compressed video rate of 25 megabits per second. The large frame size of 992-bytes frames at 8000 frames per second corresponds to 64 megabits per second.

4.1.5 Summary

The maximum utilization experiments consist of two separate experiments that collect and analyze raw frame timing information. The maximum generation module of the client server framework transmits 50,000 fixed sized frames for each experimental run. The maximum utilization experiment with default buffer sizes determines the effect of frame size on channel performance. The maximum utilization experiment with reduced buffer sizes determines the effect of buffer size on channel performance. Both experiments have frame size and channel type factors.

The frame size factor has the same three levels for both maximum utilization experiments. The frame size factor levels are 992 bytes, 396 bytes, and 24 bytes. These levels are chosen to avoid fragmentation and provide useful information about streaming temporal data.

The channel type factor has different levels for the two maximum utilization experiments. The maximum utilization experiment with default buffer sizes has three levels: the isochronous IEEE 1394, the TCP/IP, and the UDP/IP. The maximum

utilization experiment with reduced buffer sizes has two levels: the TCP/IP and the UDP/IP.

Together, the two maximum utilization experiments produce a huge amount of data. Below is a quick numerical summary of the frames and data from the maximum utilization experiments.

- Number of frame sizes: 3 (24 bytes, 396 bytes, and 992 bytes)
- Number of channels: 3 (isochronous IEEE 1394, TCP/IP, UDP/IP)
- Number of runs: 75
- Number of frames transmitted: $3,750,000$
- Number of bytes transmitted: $1,765,000,000$

4.2 Runtime Environment Contribution to Response Variables

The runtime environment determines resource performance and resource scheduling and thus affects response variables. The two response variables for the maximum utilization, inter-frame delay and transmit lag, are derived from frame timing information and sensitive to the performance of the runtime environment. This research classifies the runtime environment's contribution to response variables as either systematic or random.

The systematic and random runtime environment contributions to response variables affect the strength of the statistical analysis and the repeatability of the experiments. *Systematic contributions* of the runtime environment affect response variables consistently within factor levels. *Random contributions* of the runtime environment affect response variables unpredictably within factor levels and between factors. The channel performance metrics contain the effects of both systematic and random contributions of the runtime environment.

The contribution of the runtime environment to response variables is unique to each runtime environment and response variable. Modifications to the runtime environment alter its contribution to response variables and alter experiment results. To enable conclusions supported by strong statistical evidence, the runtime environment must remain unaltered for each run of the maximum utilization experiments. The maximum utilization experiments take steps to minimize the effect of random contributions of the runtime environment to response variables.

4.2.1 Systematic Contributions

The runtime environment's systematic contribution to response variables increases the variance between levels and factors. Increasing the *between variance* implies affecting each experimental block's mean unequally, but affecting each variable within an experiment block equally. Systematic contributions to response variables are part of a channel's performance and provide insight to the performance characteristics of the runtime environment.

An example of a systematic contribution to the response variable transmit lag is physical information transmission time. The transmit lag recorded for each frame includes the time the physical layer requires to transmit the frame through the physical medium. Each frame's transmit lag value includes physical information transmission time. The physical information time is approximately equal for each frame size and each physical layer. Changes in either the frame size or the physical layer, including topology changes, change the contribution of physical information transmission.

Systematic contributions may be correlated with response variables, in which case they have a fixed affect within a factor level. Alternatively, systematic contributions may be uncorrelated with any predictor variables and have a fixed affect between factor levels.

4.2.2 Random Contributions

The runtime environment's random contribution to response variables increases the variance within levels and between factors. Increasing the *within variance* implies altering the mean of levels within a factor in an unpredictable manner. Within variance, or unexplained error, reduces the statistical strength of experimental results. Random contributions to response variables increase confidence intervals and weaken the statistical support for conclusions about the runtime environment.

An example of a random contribution to the response variable inter-frame delay is machine activity. The inter-frame delay recorded for each frame includes the time a frame may have arrived at the client application, but not yet processed. The amount of machine activity affects the kernel's scheduling, which affects the client application execution. If the machine activity level is high, it prevents the client application from processing the frame immediately. The amount of time added to each inter-frame delay values varies with machine activity and may be different for each frame.

Random contributions, like systematic contributions, affect response variables, but their contribution is not correlated with a predictor variable or fixed between factor levels. Random contributions to response variables increase variance throughout all factors and levels.

Reducing random contributions to response variables reduces within error, which increases the strength of the statistical analysis. I identify several sources of random contributions to response variables within the runtime environment and take steps to minimize their contributions.

4.2.2.1 Reducing Random Contributions

The Java Virtual Machine's garbage collection mechanism can cause random contributions to inter-frame delay and transmit lag performance metrics. The Java Virtual Machine starts the garbage collector thread when an application allocates sufficient memory to cross allocated-memory threshold conditions. The garbage collector thread works deterministically under the same memory conditions that each

run within an experimental block produces. The random contribution arises when the same Java Virtual Machine processes multiple experimental runs. When the server application completes an experiment run, the conditions for the next garbage collection event are partially met. The first garbage collection event of each experiment run is affected by the memory conditions created by the previous experimental run.

The client server framework uses two methods to reduce the random effect of garbage collection events. The server application initiates a full garbage collection before each experimental run. This removes all pre-existing garbage memory conditions and starts each experimental run with the same memory conditions. The client application creates a new Java Virtual Machine for each experimental run. This eliminates all pre-existing affects, including garbage memory, of previous experimental runs and starts each experimental run under the same Java Virtual Machine conditions.

The amount of available computation can cause random contributions to channel performance response variables. The runtime environment does not guarantee the amount computation resource or when the computation resource is available to an application program. The kernel schedules the computation resource to all applications, multiplexing the resource. All processes in the runtime environment, including the kernel, compete for the computation resource, which affects the amount of computation resource available for the client server framework. Unprivileged applications, such as the client server framework, cannot influence the kernel for move favourable scheduling. The activity level of all processes in the runtime environment affects the scheduling and the availability of computation resource to the client server framework.

The runtime environment maintains the process activity as stable as possible in the runtime environment for all experiment runs. The activity level within the runtime environment is minimized to reduce the unpredictability of the available computation resource and it's scheduling. Each experiment should receive the approximately the same scheduling and amount of computation.

Clock drift can cause random contributions to the channel performance response variables. *Clock drift* is the term describing how a computer's clock does not keep

accurate time, drifting away from the correct time. Both channel performance metrics rely on accurate frame timestamp information and rely on the client and server clocks to be synchronized. The amount of clock drift a computer experiences is dependent on runtime components and runtime environment activity levels. Machines with identical runtime components can experience different clock drift due to manufacturing imperfections. Clock drift affects the accuracy of the frame timestamps and introduces error in channel performance metrics.

The runtime environment minimizes clock drift with the *Network Time Protocol Daemon (NTPD)*. The NTPD synchronizes the clocks of both computers of the experiment runtime environment with very accurate timeserver computers. The two computers clocks synchronize their clocks with a maximum estimated error of seven microseconds.

Channel contention can cause random contributions to the channel performance response variables. The methods the IEEE 802.3 uses to resolve channel contention involves a random process, which contributes randomly to frame timing information. The time the IEEE 802.3 requires to successfully contend for exclusive access to the network is not bounded or deterministic. Each packet may incur a different amount of lag from IEEE 802.3 contention.

To minimize the effects of IEEE 802.3 network contention of the maximum utilization experiments, the runtime environment permits only traffic related to the client server framework to enter the private IEEE 802.3 connecting the two computers. Careful control of the experiment runtime environment resolves IEEE 802.3 contention without employing its random back-off scheme.

4.3 Preliminary Statistical Analysis

The objective of the preliminary statistical analysis is to evaluate the channel performance data and validate the multiple runs of experiment blocks. The R statistical language and environment performs the statistical analysis, produces the statistical tables, and produces the response variable plots.

The first step of this preliminary statistical analysis determines if there is sufficient evidence to reject the hypothesis of equal means of the individual runs within each factor level. When there is insufficient evidence to reject the equal mean hypothesis, the data is not heavily affected by random contributions. This data captures repeatable performance characteristics of the runtime environment, suitable for making statistical conclusions. When there is sufficient evidence to reject the equal mean hypothesis, further investigation into the data set is required. This data is tainted by random contributions.

The critical information contained in Analysis Of Variance (AOV) tables is the $Pr(> F)$ column. This column contains the probability that through pure chance the data within a factor level may have equal means. It should be noted that a larger $Pr(> F)$ value does not provide more or greater evidence of equal means. The only potential conclusions are either there is *sufficient* evidence to reject the hypothesis that the means differ or there is *insufficient* evidence to reject the hypothesis that the means differ.

A $Pr(> F)$ value less than the acceptance threshold indicates that there is sufficient evidence to reject the hypothesis of equal means. The analysis of variance statistical technique provides no information about which run caused the rejection of the equal means hypothesis.

The linear regression statistical technique finds information about the relation between multiple predictor variables and a response variable. The linear regression technique for multiple handling predictor factors is to create a dummy variable for each level of each factor. In this case, each run within each experiment becomes a predictor variable. The linear regression calculates the amount of correlation between the individual runs of each experiment block.

The regression coefficients represent the independent contribution of each predictor variable to the response variable. Maindonald [61] gives a thorough explanation of linear regression and analysis of variance tailored for the R statistical language and environment.

Table 4.1: Analysis of variance of inter-frame delay for the maximum experiments.

		Df	Sum Sq	Mean Sq	F value	Pr(>F)
IEEE 1394 default buffers	992-byte	4	718035	179509	1.70	0.1457
	Residuals	249990	26321265965	105289		
	396-byte	4	368216	92054	1.00	0.4055
	Residuals	249990	22989440072	91961		
	24-byte	4	52828	13207	0.38	0.8262
	Residuals	249990	8789096253	35158		
TCP/IP default buffers	992-byte	4	170431	42608	0.33	0.8592
	Residuals	249990	32454658176	129824		
	396-byte	4	10771	2693	0.07	0.9918
	Residuals	249990	10032052302	40130		
	24-byte	4	17899	4475	0.18	0.9506
	Residuals	249990	6341345231	25366		
UDP/IP default buffers	992-byte	4	82188	20547	0.19	0.9415
	Residuals	249990	26437029553	105752		
	396-byte	4	16295	4074	0.04	0.9964
	Residuals	249990	23397272291	93593		
	24-byte	4	270266	67566	0.14	0.9662
	Residuals	249990	118208869019	472854		
TCP/IP reduced buffers	992-byte	4	30368070	7592017	4.63	0.0010
	Residuals	249990	409606873097	1638493.03		
	396-byte	4	1895548	473887	0.15	0.9634
	Residuals	249990	794021638397	3176214		
	24-byte	4	1133708	283427	0.10	0.9823
	Residuals	249990	704803748944	2819328		
UDP/IP reduced buffers	992-byte	4	1398485419	349621355	1675.43	0.0000
	Residuals	249990	52166702770	208675		
	396-byte	4	246783	61695	0.46	0.7669
	Residuals	249990	33707772346	134836		
	24-byte	4	102293	25573	0.21	0.9326
	Residuals	249990	30328495510	121319		

4.3.1 Inter-Frame Delay

Table 4.1 contains the analysis of variance results of inter-frame delay observed from seventy-five experimental runs of 50,000 frames each from the maximum traffic model. Two blocks of inter-frame delay have sufficient evidence to reject the hypothesis of equal means. The 992-byte level of the TCP/IP with reduced buffers and the 992-byte level of the UDP/IP with reduced buffers of Table 4.1 have a probability

less than the 0.10 significance level.

Table 4.2: Linear regression of TCP/IP inter-frame delay of 992-byte frames, and with reduced buffers.

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	315.0447	5.7246	55.03	0.0000
run 2	5.9984	8.0957	0.74	0.4587
run 3	3.1940	8.0957	0.39	0.6932
run 4	31.0961	8.0957	3.84	0.0001
run 5	10.2079	8.0957	1.26	0.2073

Table 4.3: Revised analysis of variance of TCP/IP inter-frame delay of 992-byte frames, with reduced buffers, and with run 4 excluded.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
run	3	2814469	938156	0.61	0.6088
Residuals	199992	307873477163	1539429		

Table 4.2 contains the linear regression of the five runs of the TCP/IP inter-frame delay, with reduced buffers, and within the 992-byte frame size factor level. The independent contributions of indicator variables $run2$, $run3$, and $run5$ provide little additional information, indicated by their high $Pr(> |t|)$ value. A high $Pr(> |t|)$ value from an indicator variable shows it has little independent contribution to the baseline established. In Table 4.2, the baseline is established by $run1$, labelled intercept. However, the independent contribution of $run4$ is significant, indicated by its low $Pr(> |t|)$ value. A $Pr(> |t|)$ value less than the significance threshold of 0.10 indicates $run4$ contributes independently to the mean of its experiment block.

The analysis of variance in Table 4.3 summarizes the inter-frame delay of the TCP/IP, with reduced buffers, within the 992-byte frame size factor level, and with $run4$ excluded. This revised analysis of variance shows insufficient evidence to reject the hypothesis of equal means. The data contained in $run4$ contributes significant independent information to the average of the other experimental runs.

Table 4.4: Linear regression of UDP/IP inter-frame delay of 992-byte frames, and with reduced buffers.

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	204.4716	2.0429	100.09	0.0000
run 2	−0.3688	2.8891	−0.13	0.8984
run 3	−1.5506	2.8891	−0.54	0.5915
run 4	186.4725	2.8891	64.54	0.0000
run 5	−0.1043	2.8891	−0.04	0.9712

Table 4.5: Revised analysis of variance of UDP/IP inter-frame delay of 992-byte frames, with reduced buffers, and with run 4 excluded.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
run	3	76362	25454	0.15	0.9325
Residuals	199992	34959133140	174803		

Similarly, Table 4.4 contains the linear regression of the five runs of the UDP/IP inter-frame delay with reduced buffers, and within the 992-byte frame size factor level. There is sufficient evidence that $run4$ contributes independently to the baseline established by $run1$. The revised analysis of variance Table 4.5 contains the UDP/IP with reduced buffers, within the 992-byte frame factor level, and with $run4$ excluded. There is insufficient evidence to reject the hypothesis of equal means for the revised data set.

4.3.2 Transmit Lag

Table 4.6 contains the analysis of variance results of the transmit lag channel performance metric. Each entry in column $Pr(> F)$ is below the significance threshold of 0.10. There is sufficient evidence to reject the hypothesis of equal transmit lag means for every block of the default and reduced buffer maximum utilization experiments.

Table 4.12, Table 4.14, Table 4.16, Table 4.18, and Table 4.20 (pages 96 through 100) contain the linear regression results of all the transmit lag factor levels that have

Table 4.6: Analysis of variance of transmit lag for the maximum experiments.

		Df	Sum Sq	Mean Sq	F value	Pr(>F)
IEEE 1394 default buffers	992-byte	4	241736193	60434048	32.96	0.0000
	Residuals	249995	458437996406	1833789		
	396-byte	4	31205231	7801308	3.71	0.0051
	Residuals	249995	525965794291	2103905		
	24-byte	4	826707430	206676856	9.33	0.0000
	Residuals	249995	5538164775701	22153102		
TCP/IP default buffers	992-byte	4	2363689182	590922295	212.40	0.0000
	Residuals	249995	695528377774	2782169		
	396-byte	4	204561901	51140475	6.52	0.0000
	Residuals	249995	1961263538616	7845211		
	24-byte	4	237047430422	59261857606	631.33	0.0000
	Residuals	249995	23466595494193	93868259		
UDP/IP default buffers	992-byte	4	83771084	20942771	49.96	0.0000
	Residuals	249995	104803608110	419223		
	396-byte	4	156336261	39084065	30.15	0.0000
	Residuals	249995	324126082597	1296530		
	24-byte	4	11100913880	2775228470	40.20	0.0000
	Residuals	249995	17256742533416	69028351		
TCP/IP reduced buffers	992-byte	4	1007208214	251802054	50.70	0.0000
	Residuals	249995	1241615726136	4966562		
	396-byte	4	1098992089	274748022	10.84	0.0000
	Residuals	249995	6336913443467	25348161		
	24-byte	4	15087754944	3771938736	19.64	0.0000
	Residuals	249995	48006167404802	192028510		
UDP/IP reduced buffers	992-byte	4	1368296431	342074108	1492.92	0.0000
	Residuals	249995	57281710600	229131		
	396-byte	4	9115308	2278827	14.79	0.0000
	Residuals	249995	38510229803	154044		
	24-byte	4	1739206118	434801530	323.22	0.0000
	Residuals	249995	336301572134	1345233		

sufficient evidence to reject the hypothesis of equal means. The process described in Section 4.3.1 for finding and excluding variables that contribute independent information is also used for transmit lag. Linear regression finds the indicator variables that contribute independent information to the baseline. The revised analysis of variance excludes variables that contribute independently to the baseline. The $Pr(> F)$ value of the revised analysis of variance in Table 4.13, Table 4.15, Table 4.17, Table

4.19, and Table 4.21 (pages 96 through 100) show insufficient evidence to reject the hypothesis of equal means.

4.4 Descriptive Statistics

Descriptive statistics provide a summary of data, which is usually the first step in any quantitative statistical analysis. Three descriptive statistics summarize the inter-frame delay and transmit lag channel performance metrics. The three descriptive statistics are the central tendency measures mean and median, and the dispersion measure standard deviation.

The mean and the median measures of central tendency describe the location of the data sample's centre of density. The mean is the most common statistical measure of central tendency. The mean is the arithmetic average of all values of the group. The median is the value of the middle element of the sorted data. The median is less sensitive to outlier values than the mean. The standard deviation describes the dispersion of values around the data sample's centre of density.

The characteristics of data transmitted through the isochronous IEEE 1394 and the IEEE 802.3 differ in several ways. These differences affect response variables and the interpretation of statistical results. The remainder of this section describes the differences.

The first difference is the minimum time between two frames sent from the same host. An IEEE 1394 host must wait approximately 125 microseconds after an isochronous packet transmission before transmitting another packet on the same isochronous channel. An IEEE 802.3 host must wait a minimum of an Inter-Packet Gap (IPG) of network silence before transmitting another packet, but may wait indefinitely. For the maximum utilization experiments, the runtime environment excludes all packets from the private IEEE 802.3 network not related to the client server framework, which means an IEEE 802.3 node must wait for one inter-packet gap. An inter-packet gap of the IEEE 802.3 100baseTX is approximately 0.96 microseconds.

The second difference is that the IEEE 802.3 usually uses higher-level protocols

and the IEEE 1394 does not. The client server framework transmits data using channels that add buffering and protocol overhead to channel transmission. The fill and forward buffering policy of the IEEE 1394 for Linux device driver affects inter-frame delay and transmit lag values of the isochronous IEEE 1394. The inter-frame delay observed from the TCP/IP and the UDP/IP channels record buffer copying operations, not channel access operations.

The third difference is the factors that determine maximum transmission rate of the IEEE 1394 and the isochronous IEEE 802.3. The maximum isochronous IEEE 1394 transmission rate is a function of frame size. The upper bound of the maximum transmission rate of the isochronous IEEE 1394 is 80% of the 400 megabits per second bus rate, which is 320 megabits per second. The IEEE 1394 bus allows one isochronous packet per channel per bus cycle. The maximum transmission rate of an isochronous IEEE 1394 channel is the number of its reserved bus allocation units multiplied by 8000 bus cycles per second. The maximum transmission rate of the IEEE 802.3 is a function of higher-level protocol overhead. The upper bound of the maximum transmission rate of the IEEE 802.3 is 100 megabits per second for IEEE 802.3 100baseTX. Some layers of the network module wrap a small amount of information around every packet. The amount of extra information added to each message is usually fixed for each layer and is not dependent of the original message size. Large packets have less protocol overhead relative to small packets.

4.4.1 Inter-Frame Delay

Table 4.8 contains the mean, median, and standard deviation for each frame size and channel type for inter-frame delay for both maximum utilization experiments. The three columns contain descriptive statistics with the following conditions: no values removed, the largest 1% of values removed, and the largest 5% of values removed. The median value changes are very small for all rows, showing the median's resilience to outliers. Excluding the highest values from the mean and standard deviation calculations, however, reduces their values significantly.

Table 4.7: Quantile breakdown of inter-frame delay for the maximum experiments.

	Packet Size (bytes)	Q 0% (μs)	Q 5% (μs)	Q 10% (μs)	Q 80% (μs)	Q 90% (μs)	Q 95% (μs)	Q 99% (μs)	Q 99.9% (μs)	Q 99.99% (μs)	Q 100% (μs)
IEEE 1394	992	54	115	123	126	137	168	893	1904	7844	130259
default	396	43	115	123	126	136	166	357	2031	8270	23830
buffers	24	39	114	123	126	134	166	835	1913	7640	16661
TCP/IP	992	27	28	28	52	61	156	1851	3059	8713	51231
default	396	25	26	26	45	55	69	226	1840	7928	20671
buffers	24	24	24	24	28	33	55	169	1644	7273	11940
UDP/IP	992	15	16	16	31	400	585	748	2372	9720	38853
default	396	14	19	19	34	51	180	859	2390	9599	31038
buffers	24	17	26	27	32	33	46	168	10419	21017	146438
TCP/IP	992	27	28	29	376	395	422	1889	18640	27981	199283
reduced	396	25	26	26	332	370	391	899	3240	24796	399608
buffers	24	24	24	24	29	35	66	175	6932	22816	197572
UDP/IP	992	29	89	95	342	345	369	1310	3752	23740	42723
reduced	396	36	69	82	147	171	215	912	3013	22855	29501
buffers	24	17	27	28	34	76	384	580	2529	12807	65680

Table 4.8: Decriptive statistics of inter-frame delay.

	Packet Size (bytes)	Q 100% > Removed Mean (μs)	Median (μs)	SD (μs)	Q 99% > Removed Mean (μs)	Median (μs)	SD (μs)	Q 95% > Removed Mean (μs)	Median (μs)	SD (μs)
IEEE 1394	992	146	125	324	132	125	48.3	125	125	9.27
default	396	145	125	303	129	125	25.5	125	125	9.35
buffers	24	143	125	188	130	125	39.8	124	125	11.1
TCP/IP	992	86.7	30	360	62.8	30	183	36.9	30	16.9
default	396	48.5	27	200	38.0	27	25.1	33.8	27	10.6
buffers	24	37.1	25	159	29.0	25	15.8	26.1	25	3.79
UDP/IP	992	94.8	20	325	78.1	20	160	55.3	20	116
default	396	75.1	28	306	58.1	28	130	33.0	28	19.9
buffers	24	67.9	28	688	31.6	28	14.2	28.9	28	2.98
TCP/IP	992	328	367	1310	262	367	208	241	366	162
reduced	396	146	27	1780	111	27	143	96.4	27	127
buffers	24	62.1	25	1680	30.1	25	17.6	26.8	25	5.68
UDP/IP	992	251	226	474	225	225	126	211	220	94.0
reduced	396	145	119	367	124	118	51.6	116	114	30.8
buffers	24	72.8	29	348	56.4	29	92.4	39.2	29	39.0

Excluding higher quantile values from descriptive statistic calculations serves two purposes. First, separating the normal case from the extreme case demonstrates the importance or unimportance of outlier values. If the difference between the Q 95% and the Q 100% quantiles is small, then the effect of outliers to the behaviour of the system in small. On the other hand, if the difference is large, then by addressing the cause of the outliers instead of concentrating on normal operating conditions, large performance increases can be made. Second, if the magnitude of the outlier is of little importance to the application, but the descriptive statistics of typical values is important, then separating the Q 99% and Q 95% quantiles is important. An example to illustrate this point is compression with information redundancy. If compression is incorporated with information redundancy, then the complete data stream is not required for decompression. If the application can tolerate a small percentage of data loss, then the outlier values may be safely be regarded as lost and should not be included in descriptive statistic calculations. The decompression of the data stream

does not require the outlier values and outliers should not be included in performance calculations.

The reduction of mean, median, and standard deviation is greatest for the TCP/IP and the UDP/IP channels with reduced buffers. Inter-frame delay values in higher quantiles contribute significantly to the mean and standard deviation and insignificantly to the median.

The ideal isochronous IEEE 1394 transmission rate is 125 per frame, regardless of frame size. The observed mean and median inter-frame delay values are 145 microseconds per frame and 125 microseconds per frame respectively. Table 4.7 shows the cause of the difference between the mean and the median of inter-frame delay. The extremely large inter-frame delay values in higher quantiles have a far greater affect on the mean than the median and cause the difference in central tendency measures.

Inter-frame delay values of the TCP/IP and the UDP/IP normally measure buffer copying operations because both of these protocols buffer their output. Even with buffering, there are a few valuable pieces of information contained in Table 4.7 and Table 4.8.

Table 4.7 shows the magnitude of outlier values in upper quantiles of inter-frame delay. These relatively rare values substantially affect the mean, but not the median. As with the isochronous IEEE 1394, a few large values in higher quantile heavily affect the mean, but not the median.

The TCP/IP inter-frame delay median values for all factor levels are very close for both maximum utilization experiments, with one exception. Most median inter-frame delay is within the range of twenty-five microseconds to thirty microseconds. The one exception is the 992-bytes packet factor level of the reduced buffer size maximum utilization experiment. At this factor level, the median inter-frame delay value is more than three hundred microsecond. This is not surprising as the input and output buffers are only large enough for a single 992-byte frame, which affects all frames with increased inter-frame delay, not just a few outliers.

The ratio of buffer size to frame size required to minimize inter-frame delay of the TCP/IP is less than five. The 396-byte frame size factor level of both maximum utilization experiments has the same inter-frame delay value of twenty-seven microseconds. Increasing buffer size beyond five frame sizes does not significantly reduce the median amount of inter-frame delay for this runtime environment.

Both maximum utilization experiments indicate that the UDP/IP inter-frame delay mean values decrease as frame size increases. The median inter-frame delay value for all frame size levels with default buffers are similar, ranging from twenty microseconds for 992-byte frames to twenty-eight microseconds for 396-byte and 24-byte frames. Reducing the output buffer to 2048 bytes affects the median inter-frame delay of both 992-byte and 396-byte frames. The median inter-frame delay value for 24-byte frames remain almost constant at twenty-five microseconds for 32,767-byte buffers and twenty-nine microseconds for 2048-byte buffers.

4.4.2 Transmit Lag

Transmit lag of the isochronous IEEE 1394 should be independent of frame size, but input buffering by the receiver increases transmit lag inversely to frame size. The IEEE 1394 for Linux device driver uses a fill and forward input buffer policy that buffers incoming isochronous frames until 4096 bytes accumulate before forwarding isochronous frames to user applications.

Figure 4.2 clearly shows the effects of input buffering on the isochronous IEEE 1394. The nearly infinite positive slope demarks the end of one input buffer and the beginning of the next. The sharply negative slope shows the client processes frames very quickly once the device driver the forwards them to the application layer. The IEEE 1394 input buffer delays each isochronous frame slightly less than previous frames, except on input buffer boundaries where it dramatically increases the delay of the first frame.

Input buffering by the IEEE 1394 for Linux device driver also significantly contributes to transmit lag variability. An IEEE 1394 host receiving data delays

Table 4.9: Quantile breakdown of transmit lag for the maximum utilization experiments.

	Packet Size (bytes)	Q 0% (μs)	Q 5% (μs)	Q 80% (μs)	Q 90% (μs)	Q 95% (μs)	Q 99% (μs)	Q 99.9% (μs)	Q 99.99% (μs)	Q 100% (μs)
IEEE 1394	992	479	550	860	907	1105	6857	23060	26899	142230
default	396	643	760	1592	1763	2412	7090	19894	24587	25531
buffers	24	3965	4894	16443	17975	18908	24062	33091	37001	38221
TCP/IP	992	321	2664	5035	5291	5473	10068	23385	28415	28450
default	396	154	388	480	1241	4694	15813	25261	26934	27206
buffers	24	101	1561	3500	10334	28438	40502	44137	60202	60487
UDP/IP	992	235	508	994	1072	1124	1494	9638	29284	36727
default	396	151	790	1944	2034	2146	3710	13680	34812	35017
buffers	24	8922	21538	30826	33570	40220	65646	72355	74074	74307
TCP/IP	992	274	997	1480	1496	1522	3021	19600	32389	199314
small	396	161	714	1470	1540	1577	1761	17153	400053	598248
buffers	24	109	826	3156	11840	16566	20011	197556	199229	199496
UDP/IP	992	349	383	614	618	627	1514	4092	24834	42952
small	396	212	228	298	306	322	430	3214	23010	33825
buffers	24	106	698	2005	2081	2236	3809	17875	34805	34985

Table 4.10: Decriptive statistics of transmit lag.

	Packet Size (bytes)	Q 100% > Removed Mean (μs)	Median (μs)	SD (μs)	Q 99% > Removed Mean (μs)	Median (μs)	SD (μs)	Q 95% > Removed Mean (μs)	Median (μs)	SD (μs)
IEEE 1394	992	912	752	1468	789	752	424	727	749	125
default	396	1439	1254	1289	1331	1250	557	1245	1244	324
buffers	24	11979	11878	4744	11826	11801	4502	11465	11493	4221
TCP/IP	992	4406	4382	1472	4303	4372	953	4226	4335	865
default	396	1114	438	2724	918	438	1877	565	435	568
buffers	24	4710	1908	8060	4325	1905	7119	3047	1893	3444
UDP/IP	992	849	811	647	813	809	191	797	802	177
default	396	1704	1671	1123	1629	1666	420	1591	1645	378
buffers	24	28551	27937	6986	28138	27882	5672	27334	27648	3961
TCP/IP	992	1470	1420	2352	1311	1420	240	1295	1417	223
reduced	396	1386	1199	6228	1213	1198	246	1196	1195	236
buffers	24	4244	1137	14498	3156	1121	4465	2513	1084	3240
UDP/IP	992	569	576	521	542	570	117	531	548	90.5
reduced	396	291	273	393	271	272	32.1	268	268	27.0
buffers	24	1789	1785	1201	1707	1781	454	1667	1766	414

isochronous frames until its input buffer is full. Any frame in the input buffer may be an extreme value contained in the 99% quantile. Since the IEEE 1394 host delays the entire buffer until the last frame arrives, a 99% quantile frame significantly contributes to the transmit lag of each frame in the receiver's input buffer that arrives before the 99% quantile frame.

The transmit lag experienced by 992-byte TCP/IP frames with default buffers is surprisingly high at more than four milliseconds. Figure 4.3 provides the information to explain the unusually large transmit lag. The lower (dashed line) inter-frame delay plot has spikes that coincide in time and magnitude with the drops in the upper (solid line) transmit lag plot. The steep positive slope of transmit lag indicates that a buffer is filling faster than it can be emptied. The bottleneck may be between the sender's output buffer and the physical network, or between the receiver's input buffer and the TCP/IP protocol processing. The former scenario appears more likely because the spikes comprise 45% of the inter-frame delay in Figure 4.3 and the IEEE 802.3

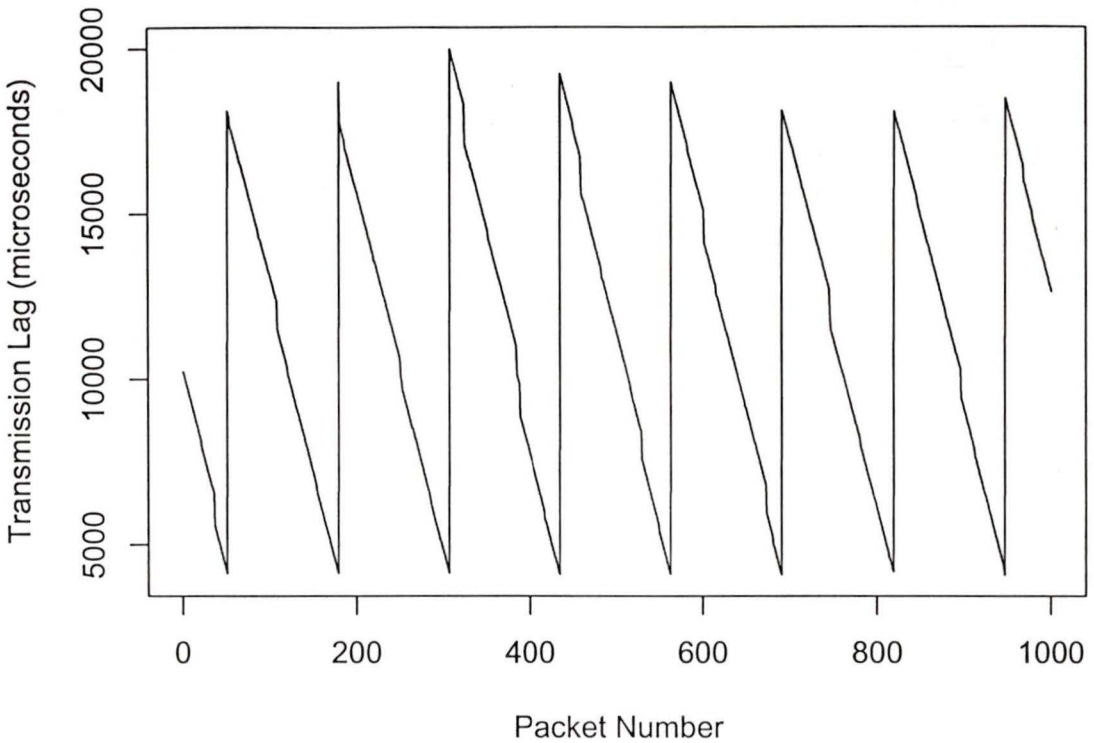

Figure 4.2: Transmit lag of the isochronous IEEE 1394 for a sequence of 1000 24-byte frames.

physical layer utilization is approximately 99%. If the bottleneck were the receiver's input buffer, the network would be idle during sender inter-frame delay spikes.

The sender's output buffer delays frames several milliseconds before transmission, significantly contributing to the TCP/IP transmit lag. The transport layer delays frames because the physical layer is saturated and cannot accept packets any faster. Transmission of smaller frame sizes does not saturate the IEEE 802.3 media because the increased protocol overhead per byte transmitted is sufficiently high that the runtime environment becomes the bottleneck.

Figure 4.4 shows that the large output buffering induced transmit lag present at the 992-byte factor level in the TCP/IP with default buffers is not present with reduced buffers. The reduced TCP/IP output buffer is only large enough to buffer one 992-byte frame, destroying its ability to buffer substantial number of frames.

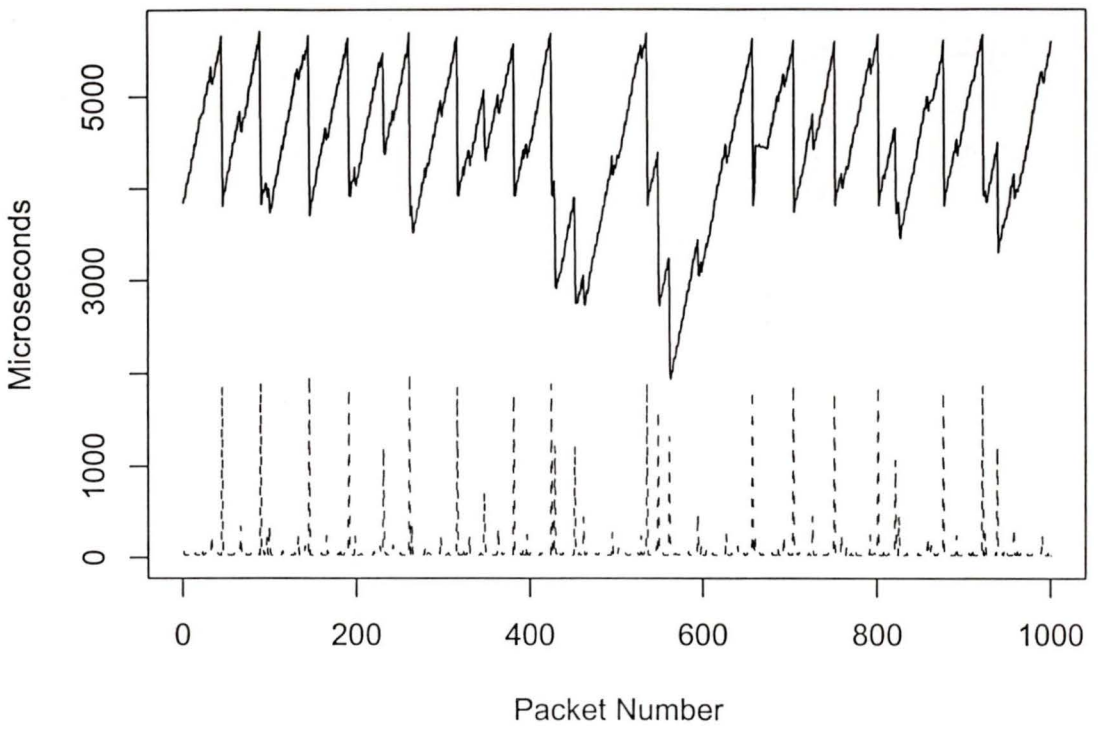

Figure 4.3: Transmit lag (upper - solid line) and inter-frame delay (lower - dashed line) plot of the TCP/IP with default buffers for a sequence of 1000 992-byte frames.

When the TCP/IP uses reduced input and output buffers, the effect on transmit lag is different at each factor level of frame size. Table 4.10 on page 85 provides a summary of descriptive statistics of transmit lag for the TCP/IP.

The transmit lag for 992-byte frames of the 100% quantile drops by approximately $\frac{2}{3}$ to 1470 microseconds when the TPC/IP buffers are reduced. This is the largest improvement between the default buffer size and reduced buffer size of the maximum utilization experiments.

The standard deviation from the 99% and 95% quantiles of the 396-byte frame size factor level are lower for the reduced buffer than the default buffer TCP/IP channel. The other descriptive statistics are greater for the TCP/IP with default buffers at the 396-byte frame size level.

The reduced buffer TCP/IP channel has lower mean and median values for the

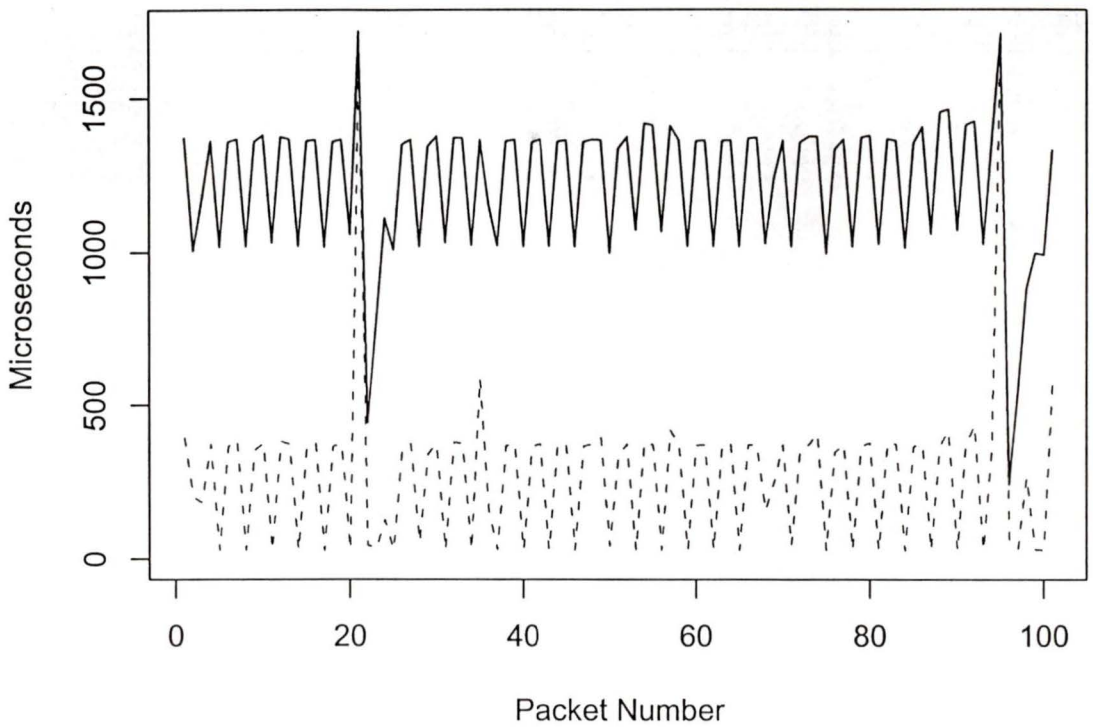

Figure 4.4: Transmit lag (upper - solid line) and inter-frame delay (lower - dashed line) plot of the TCP/IP with reduced buffers for a sequence of 100 992-byte frames.

24-byte frame size factor level than the default buffer TCP/IP channel. Overall, the size of the TCP/IP buffers affects the 396-byte and 24-byte frame size factor levels much less than the 992-byte factor level.

Two general trends exist between the TCP/IP with default, 32, 767-byte, buffers and reduced, 2048-byte, buffers. The mean, median, and standard deviation of small frames is always much larger than the other frame sizes, regardless of buffer size. Excluding higher quantiles from the descriptive statistic calculations reduces the mean and median slightly, but reduces the standard deviation significantly. This indicates the largest contributors to dispersion are located in the upper quantiles.

The mean, median and standard deviation of the UDP/IP transmit lag at the 992-byte frame factor level is consistently lower than the TCP/IP, for both buffer sizes. The UDP/IP with default buffers has similar descriptive statistics as the isochronous

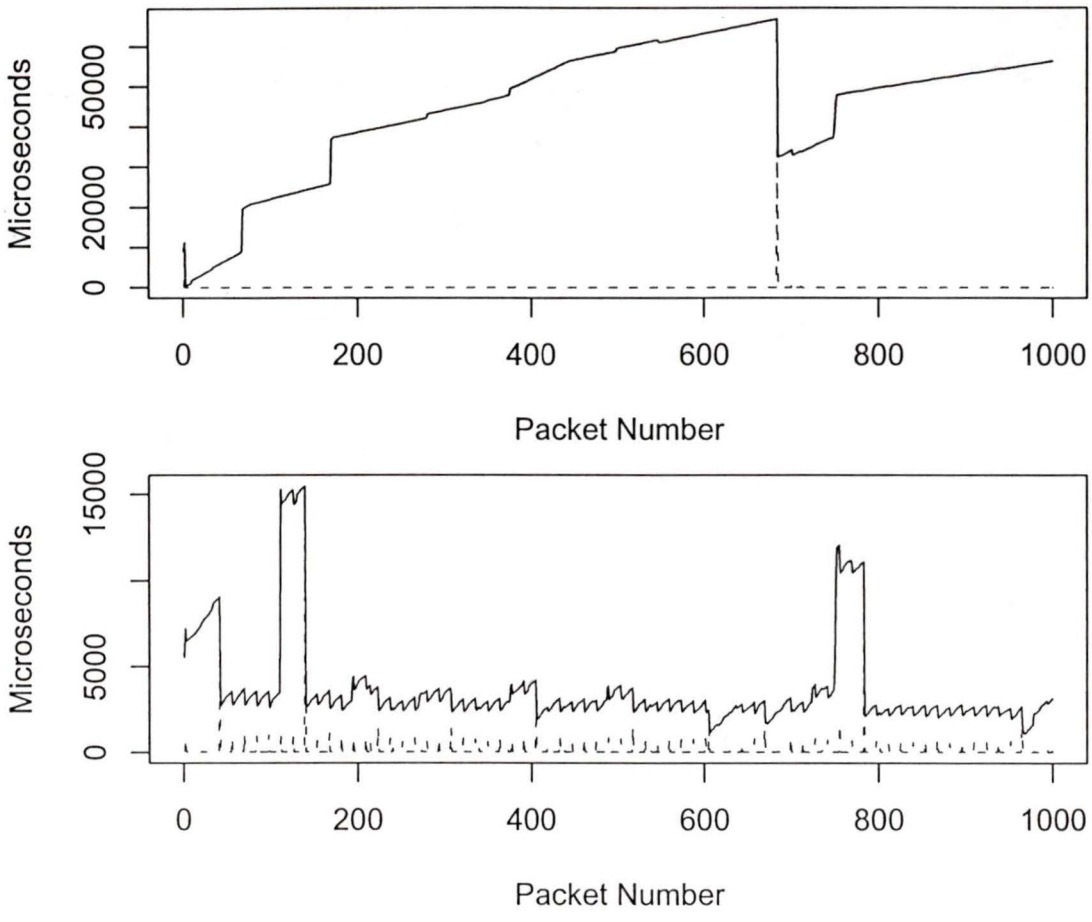

Figure 4.5: Transmit lag (upper - solid line) and inter-frame delay (lower - dashed line) plots of the UDP/IP with default (upper plot) and reduced (lower plot) buffers for a sequence 1000 24-byte frames.

IEEE 1394, but lower values with reduced buffers.

For the UDP/IP with default buffers, 396-byte frames have a higher mean and standard deviation than 992-byte frames. The opposite is true for the UDP/IP with reduced buffers, where the mean and standard deviation of 396-byte frames is lower than 992-byte frames. This provides support to the findings of the TCP/IP; where a buffer size large enough to hold approximately five frames provides better transmit lag than substantially smaller or large buffers.

The UDP/IP's mean transmit lag for 24-byte frames with default buffers is

abysmally poor at approximately twenty-nine milliseconds. With the reduction of buffer size to 2048 bytes, the mean, median, and standard deviation of transmit lag drop significantly. Figure 4.5 shows how the reduction of buffer size reduces transmit lag and improves the stability of the UDP/IP protocol transmitting 24-byte frames.

Large outlier transmit lag values have the least affect on the UDP/IP channel when compared to the isochronous IEEE 1394 and the TCP/IP channels. Two observations from Table 4.10 on page 85 support this statement. First, the mean and median are closer for the UDP/IP channel than the other channels. Second, with higher quantiles removed, the change in transmit lag descriptive statistics is smallest for the UDP/IP channel. The UDP/IP provides the most predictable and stable transmit lag.

With the exception of 24-byte frames with default buffers, the mean, median, and standard deviation of the UDP/IP's transmit lag are the lowest or very close to the lowest for all channels. Only the isochronous IEEE 1394 provides better transmit lag in some factor levels of the maximum utilization experiments.

4.4.3 General Trends of Inter-Packet Delay and Transmit Lag

Table 4.8 on page 81 and Table 4.10 on page 85 contain the descriptive statistics for inter-frame delay and transmit lag. Some performance characteristics are common to all channel types.

Small frames, 24 bytes, generally have large mean and median transmit lag values. There are two reasons that contribute to the poor transmit lag results of small frames. First, each frame contains the same overhead in protocol processing time and appended protocol headers. Protocols add fixed size headers to each packet, regardless of frame size. The amount of overhead is proportionally greater for smaller frames. Second, the fixed sized buffers used by all the channels can hold more small frames than large frames. Transmit lag contains the time each packet waits in input and output buffers, so with more packets waiting in buffers the potential for buffer induced lag is greater.

The 24-byte frames of the isochronous IEEE 1394, the 992-byte frames of the TCP/IP, and the 24-byte frames of the UDP/IP underline the importance of buffer size. These three levels have the worst transmit lag with default buffers, but among the best with reduced buffers. Increasing the frame size affectively decrease the buffer size of the isochronous IEEE 1394, which has unconfigurable buffer sizes. The transmit lag of isochronous IEEE 1394 with 396-byte frames is substantially lower than 24-bytes frames. This corresponds to a buffer size of approximately 10 frames having much lower transmit lag than a buffer size of approximately 170 frames. The transit lag of the TCP/IP and the UDP/IP with default buffers is unacceptably high because of buffer induced lag. With reduced buffer sizes, the TCP/IP and the UDP/IP provide much better transmit lag.

Reducing protocol buffers increases inter-frame delay performance, but decreases transmit lag performance. In most cases, reducing buffer size has a positive affect on the mean, median, and standard deviation of transmit lag. Decreasing buffer size has the opposite affect on inter-frame delay.

4.5 Garbage Collection

Garbage collection is an automated heap memory management technique used by the Java platform. Garbage collection releases the programmer from the task of freeing unused dynamic memory. The Java Virtual Machine employs several techniques to reduce the performance overhead of garbage collection. The remainder of this section covers garbage collection performance enhancement techniques of Sun's HotSpot Java Virtual Machine [46] and discusses the impact of garbage collection in the maximum utilization channel performance experiments.

The Java 2 platform uses generational copying to limit holes in heap memory caused by short-lived Java objects. Every explicit call to `new` in the Java language creates a new Java object in heap memory. Java applications typically create many small, short-lived objects. The heap memory becomes riddled with many active Java objects interspersed with small gaps of free heap memory. The Java Virtual

Machine may have a large number of total available free heap memory, but only a very small amount of contiguous heap memory. The heap memory is fragmented into small pieces unsuitable for large objects. Eventually, the Java Virtual Machine must begin a process called *compaction* where heap memory is de-fragmented. Compaction rearranges heap memory so all the many free memory fragments are together and coalesces them into a single large free piece of heap memory. Unfortunately, the compaction process is time consuming and interrupts the Java application during operation.

To minimize the processing interruptions associated with compacting heap memory, the Java Virtual Machine creates new objects in a special heap memory area, called the *memory bucket.* During garbage collection, the Java Virtual Machine copies non-garbage objects in the memory bucket into heap memory. When the garbage collection event is over, the memory buck is contains only garbage objects. The Java Virtual Machine quickly empties the memory bucket of garbage objects, without causing heap memory fragmentation. Short-lived Java objects do not fragment heap memory because they only exist in the memory bucket.

The Java 2 platform uses incremental garbage collection to limit the length of application pauses interruptions due to garbage collection. The Java Virtual Machine breaks garbage collection events into several pieces to reduce the length of individual garbage collection events. The garbage collection cost is spread over several interruptions causing only small interruptions to applications.

The Java 2 platform HotSpot Java Virtual Machine [46] used for this research uses several garbage collection techniques to improve Java performance. Sun Microsystems' Java Virtual Machine documentation describes several other Java Virtual Machine performance improvements [46]. Despite the best efforts of the Java Virtual Machine, the ultimate job of Java application performance falls with the application developer. Section 3.3 discusses the Java application implementation steps taken to reduce the impact of garbage collection to application performance.

Table 4.11: Mean and total of garbage collection time for each frame size factor level.

	Frame Size					
	24 bytes		396 bytes		992 bytes	
	Mean	Total	Mean	Total	Mean	Total
IEEE 1394						
Server (7 events)	127 μs	892 μs	112 μs	785 μs	107 μs	748 μs
Client (3 events)	301 μs	902 μs	320 μs	960 μs	334 μs	1001 μs
TCP/IP						
Server (8 events)	122 μs	972 μs	116 μs	931 μs	126 μs	1009 μs
Client (3 events)	290 μs	869 μs	325 μs	976 μs	320 μs	959 μs
UDP/IP						
Server (3 events)	146 μs	438 μs	136 μs	407 μs	166 μs	499 μs
Client (4 events)	197 μs	788 μs	204 μs	817 μs	202 μs	806 μs

4.5.1 Results

The results of garbage collection timing from the client and server applications show that garbage collection does not cause intolerable application processing interruptions. Table 4.11 contains the mean time, total time, and count of garbage collection events from the maximum utilization experiment. The garbage collection results are from a separate set of client server framework runs of the maximum utilization experiment with default channel buffer sizes. The garbage collection information was collected separately from the other performance metrics because garbage collection monitoring of the Java Virtual Machine affects the Java Virtual Machine's performance.

An interesting observation from Table 4.11 is that garbage collection and frame size are uncorrelated, but garbage collection and frame count are correlated. The number of garbage collection events the client and server applications experience for each frame size factor level is the same. The mean and total garbage collection time are also very close for each frame size (table row). Further experiments and statistical analysis are required to fully investigate the relationship frame size and channel type have with garbage collection.

The average garbage collection interruptions experienced by the server applica-

tion for all channel types and frame sizes is approximately 130 microseconds. The client application average is approximately 280 microseconds. A garbage collection event with a length of 300 microseconds, which is close to the largest garbage collection event observed, is less than 1% of the time between individual frames displayed at thirty frames per second. Garbage collection does not appear to cause intolerable interruptions for applications processing streaming temporal data.

4.6 Conclusion

This chapter describes the maximum utilization experiments that evaluate channel performance at maximum channel utilization levels of the Java platform. The experiments evaluate the effect of protocol buffering, frame size, and channel type has on maximum utilization channel performance. The following are the main findings of the maximum utilization experiments.

Decreasing protocol buffering generally increases inter-frame delay and decreases transmit lag. A decrease of transmit lag can increase performance significantly as illustrated in Table 4.10 on page 85 and Figure 4.5 on page 89.

Small frames generally have poor transmit lag when compared to large frames. Not all large frame experiment runs have good transmit lag performance. Figure 4.3 on page 87 illustrates a circumstances where large frames have very poor transmit lag.

Garbage collection does not appear to be a significant factor for handling streaming temporal data on the Java platform. The average garbage collection event is less the 1% of the time between two frames at thirty frames per second.

Large processing interruptions exist in all channel types and frame sizes. The length of the interruptions is larger than the combined interruption caused by garbage collection. The interrupts are most likely caused by the runtime environment preempting the client server framework to schedule other processes.

4.6.1 Recommendations

Based on the experiment procedures and experimental results, I can make two recommendations for increased performance of the isochronous IEEE 1394, the TCP/IP, and the UDP/IP channels. These recommendations are specific to Java applications, but general enough, I believe, for all applications.

The default size of protocol buffers is too large for most applications. A buffer large enough to contain several (five to ten) frames is sufficient to keep inter-frame delay low and significantly reduce transmit lag.

Tuning channel parameters and application protocols should not be neglected. Despite the best intentions of a developer to produce a high performance application, an application cannot overcome the poor performance caused incorrect buffer size. This is not to say application design and implementation do not affect application performance. Correctly sizing channel buffers can bring an order of magnitude improvement to performance metrics.

Table 4.12: Linear regression of isochronous IEEE 1394 transmit lag.

	Estimate	Std. Error	t value	Pr(>|t|)
992-byte packets				
(Intercept)	910.1793	6.0561	150.29	0.0000
run 1	−74.8996	8.5646	−8.75	0.0000
run 3	−2.7647	8.5646	−0.32	0.7468
run 4	12.0297	8.5646	1.40	0.1601
run 5	−2.7432	8.5646	−0.32	0.7487
396-byte packets				
(Intercept)	1431.4920	6.4868	220.68	0.0000
run 2	7.2784	9.1737	0.79	0.4275
run 3	32.8116	9.1737	3.58	0.0003
run 4	13.5643	9.1737	1.48	0.1392
run 5	7.2659	9.1737	0.79	0.4283
24-byte packets				
(Intercept)	11957.7469	21.0490	568.09	0.0000
run 2	45.2628	29.7678	1.52	0.1284
run 3	−114.1917	29.7678	−3.84	0.0001
run 4	40.3407	29.7678	1.36	0.1754
run 5	−1.8296	29.7678	−0.06	0.9510

Table 4.13: Revised analysis of variance of isochronous IEEE 1394 transmit lag.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
992-byte packets					
run	3	7462434	2487479	1.15	0.3257
Residuals	199996	431091838930	2155502		
396-byte packets					
run	3	4611737	1537246	0.92	0.4277
Residuals	199996	332397821028	1662022		
24-byte packets					
run	3	96246323	32082108	1.43	0.2331
Residuals	199996	4500583486499	22503368		

Table 4.14: Linear regression of TCP/IP transmit lag with default buffers.

	Estimate	Std. Error	t value	Pr(>|t|)
992-byte packets				
(Intercept)	4416.1743	7.4594	592.02	0.0000
run 2	29.5955	10.5493	2.81	0.0050
run 3	124.5294	10.5493	11.80	0.0000
run 4	242.9164	10.5493	23.03	0.0000
run 5	−19.5484	10.5493	−1.85	0.0639
396-byte packets				
(Intercept)	1133.3226	12.5261	90.48	0.0000
run 2	23.5386	17.7146	1.33	0.1839
run 3	−19.8219	17.7146	−1.12	0.2632
run 4	42.0851	17.7146	2.38	0.0175
run 5	−37.2522	17.7146	−2.10	0.0355
24-byte packets				
(Intercept)	4691.5112	43.3286	108.28	0.0000
run 1	−349.1968	61.2759	−5.70	0.0000
run 3	2383.1363	61.2759	38.89	0.0000
run 4	404.7164	61.2759	6.60	0.0000
run 5	36.6926	61.2759	0.60	0.5493

Table 4.15: Revised analysis of variance of TCP/IP transmit lag with default buffers.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
992-byte packets					
run	1	9553479	9553479	4.41	0.0357
Residuals	99998	216605302513	2166096		
396-byte packets					
run	2	34740785	17370393	2.34	0.0963
Residuals	149997	1113294668243	7422113		
24-byte packets					
run	1	33658709	33658709	0.52	0.4717
Residuals	99998	6496830615860	64969606		

Table 4.16: Linear regression of UDP/IP transmit lag with default buffers.

	Estimate	Std. Error	t value	Pr(>|t|)
992-byte packets				
(Intercept)	849.4871	2.8956	293.37	0.0000
run 1	20.2155	4.0950	4.94	0.0000
run 2	41.9509	4.0950	10.24	0.0000
run 3	38.2770	4.0950	9.35	0.0000
run 5	−1.4403	4.0950	−0.35	0.7251
396-byte packets				
(Intercept)	1711.0067	5.0922	336.00	0.0000
run 1	53.9605	7.2015	7.49	0.0000
run 3	−4.2587	7.2015	−0.59	0.5543
run 4	−12.9077	7.2015	−1.79	0.0731
run 5	−12.2693	7.2015	−1.70	0.0884
24-byte packets				
(Intercept)	28547.3829	37.1560	768.31	0.0000
run 1	541.1961	52.5465	10.30	0.0000
run 2	379.0167	52.5465	7.21	0.0000
run 4	7.4017	52.5465	0.14	0.8880
run 5	257.7343	52.5465	4.90	0.0000

Table 4.17: Revised analysis of variance of UDP/IP transmit lag with default buffers.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
992-byte packets					
run	1	51859	51859	0.12	0.7249
Residuals	99998	41874260361	418751		
396-byte packets					
run	3	5933239	1977746	1.57	0.1948
Residuals	199996	252251921499	1261285		
24-byte packets					
run	1	1369622	1369622	0.03	0.8670
Residuals	99998	4880331111444	48804287		

Table 4.18: Linear regression of TCP/IP transmit lag with reduced buffers.

	Estimate	Std. Error	t value	Pr(>|t|)
992-byte packets				
(Intercept)	1471.9072	9.9665	147.69	0.0000
run 1	−107.3821	14.0948	−7.62	0.0000
run 2	−121.0630	14.0948	−8.59	0.0000
run 3	−154.2139	14.0948	−10.94	0.0000
run 5	−3.5911	14.0948	−0.25	0.7989
396-byte packets				
(Intercept)	1364.7881	22.5158	60.61	0.0000
run 1	90.9885	31.8422	2.86	0.0043
run 2	127.7889	31.8422	4.01	0.0001
run 4	42.4194	31.8422	1.33	0.1828
run 5	−60.3365	31.8422	−1.89	0.0581
	Estimate	Std. Error	t value	Pr(>|t|)
24-byte packets				
(Intercept)	4207.8559	61.9723	67.90	0.0000
run 1	−623.6028	87.6421	−7.12	0.0000
run 2	−133.7373	87.6421	−1.53	0.1270
run 4	−265.3087	87.6421	−3.03	0.0025
run 5	72.3106	87.6421	0.83	0.4093

Table 4.19: Revised analysis of variance of TCP/IP transmit lag with reduced buffers.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
992-byte packets					
run	1	322393	322393	0.06	0.8092
Residuals	99998	553068989734	5530800.51		
396-byte packets					
run	1	44985137	44985137	1.16	0.2815
Residuals	99998	3878805177938	38788828		
24-byte packets					
run	1	130720499	130720499	0.62	0.4303
Residuals	99998	21017477431771	210178978		

Table 4.20: Linear regression of UDP/IP transmit lag with reduced buffers.

	Estimate	Std. Error	t value	Pr(>|t|)
992-byte packets				
(Intercept)	468.8634	2.1407	219.02	0.0000
run 2	−5.0655	3.0274	−1.67	0.0943
run 3	−18.1161	3.0274	−5.98	0.0000
run 4	181.4431	3.0274	59.93	0.0000
run 5	18.5844	3.0274	6.14	0.0000
396-byte packets				
(Intercept)	295.1382	1.7552	168.15	0.0000
run 1	9.2271	2.4823	3.72	0.0002
run 3	−3.2196	2.4823	−1.30	0.1946
run 4	−4.2776	2.4823	−1.72	0.0848
run 5	−8.8716	2.4823	−3.57	0.0004
24-byte packets				
(Intercept)	1817.2880	5.1870	350.36	0.0000
run 1	−224.8650	7.3355	−30.65	0.0000
run 2	−42.1164	7.3355	−5.74	0.0000
run 3	−56.8645	7.3355	−7.75	0.0000
run 5	1.4363	7.3355	0.20	0.8448

Table 4.21: Revised analysis of variance of UDP/IP transmit lag with reduced buffers.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
992-byte packets					
run	1	641472	641472	3.17	0.0749
Residuals	99998	20215660025	202161		
396-byte packets					
run	2	496386	248193	1.55	0.2120
Residuals	149997	23997741542	159988		
24-byte packets					
run	1	51577	51577	0.04	0.8479
Residuals	99998	140140839960	1401436		

5. Movie Experiment and Results

This chapter describes experiments that evaluate the performance of the isochronous IEEE 1394, the TCP/IP, and the UDP/IP channels transmitting long running, self-similar, variable bit rate video. This chapter also provides a valuable survey of the Hurst effect, several Hurst parameter estimation techniques, and discusses some problems encountered while trying to apply these Hurst parameter estimation techniques.

5.1 Hurst Parameter

The *Hurst parameter* [18], H, is the degree of self-similarity and falls in the range $(0.5 < H \leq 1]$ for self-similar time series. A larger Hurst parameter indicates stronger self-similarity, which manifests itself as a bustier time series. The remainder of this section develops a conceptual understanding and mathematical definition of self-similarity.

Let $X = (X_t : t = 1, 2, 3, \ldots, N)$ be a covariance stationary time series. A covariance stationary time series has constant mean, $\mu = E[X_t]$, finite variance, $\sigma^2 = E[(X_t - \mu)^2]$, and an auto-correlation function of the form Equation 5.1 [19, 62].

$$r(k) = \frac{E[(X_t - \mu)(X_{t+k} - \mu)]}{E[(X_t - \mu)^2]}, k = 0, 1, 2, \ldots \tag{5.1}$$

The aggregation of the time series X, expressed as $X^{(m)}$, is defined by Equation 5.2, where $X_t^{(m)}$ is defined by Equation 5.3. The aggregated time series, $X^{(m)}$, is the average of successive non-overlapping blocks of size m of the original time series X. The aggregated time series $X^{(m)}$ is also a covariance stationary time series.

$$X^{(m)} = (X_t^{(m)} : t = 1, 2, 3, \ldots, \frac{N}{m}) \tag{5.2}$$

$$X_t^{(m)} = \frac{(X_{tm} + X_{tm-1} + \ldots + X_{tm-(m-1)})}{m} \tag{5.3}$$

Figure 5.1 illustrates two different time series at five different time scales. Both time series have equal mean packet sizes and mean packet arrival rates. The left column is a sample of self-similar Ethernet traffic from Leland *et al.*[19] at five time scales. The right column, from Taqqu *et al.*[1], is an appropriately configured Poisson model simulating the Ethernet time series at the same five time scales.

The top plot of both columns from Figure 5.1 shows the number of packets arriving per time unit of 100 seconds for each time series. Subsequent plots within each column are created by selecting a random 10% region within the previous time series, highlighted region in Figure 5.1, and increasing the time unit resolution by a multiple of ten. Figure 5.1 represents $mX^{(m)}$ for $m = 10$, 100, 1000, 10,000, and 100,000. If both time series were collected at millisecond time intervals, then plots from top to bottom of both columns represent $100,000 \cdot X^{(100,000)}$, $10,000 \cdot X^{(10,000)}$, $1000 \cdot X^{(1000)}$, $100 \cdot X^{(100)}$, and $10 \cdot X^{(10)}$.

The self-similar time series in Figure 5.1 looks *similar* to itself at time scales of milliseconds to hours. There is no natural length of bursts in the self-similar time series. Periods of activity followed by periods of relative quiet are visible at all time scales. The self-similar structure manifests itself as scale invariant, or non-degenerate, structure of the aggregated time series. Some literature refers to self-similar time series as fractal because of their scale invariant correlation structure.

In contrast, the short-range dependant time series in Figure 5.1 exhibits a degenerate structure across the five time scale plots. A characteristic of all short-range dependant time series that can clearly be seen in Figure 5.1 is $X^{(m)}$ tends to second order pure noise as $m \to \infty$ for all $t \geq 1$ [63].

Figures and discussions in references [1, 19, 63] provide additional information about the *self-similar picture proof* in Figure 5.1.

The Hurst parameter is the degree of self-similarity a time series possesses. A larger Hurst parameter indicates a greater degree of self-similarity and as a conse-

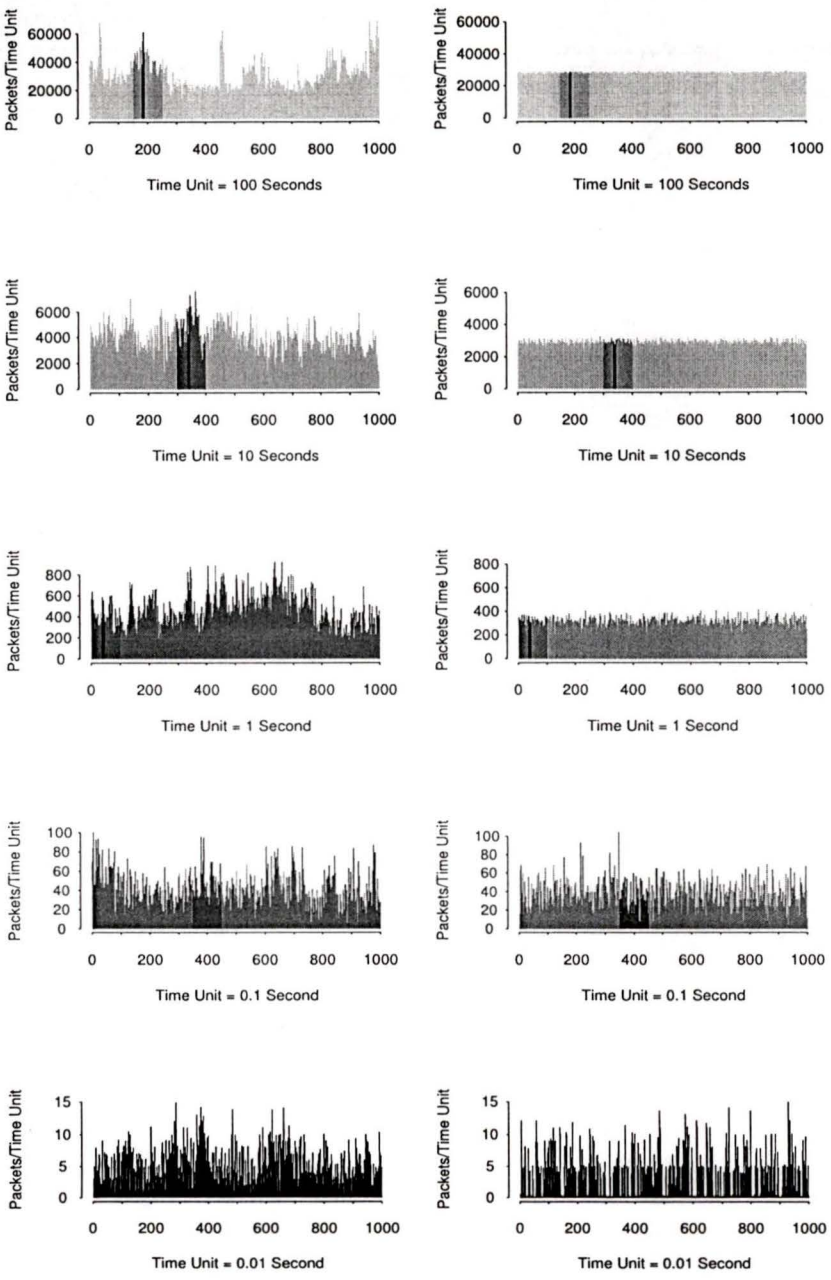

Figure 5.1: Pictorial view of a self-similar process, left column, and a short-range dependant Poisson process, right column, at five time scales. Adapted from Taqqu *et al.*[1].

quence, greater burstiness. The Hurst parameter of a self-similar time series has a range of $(0.5 < H \leq 1]$. A self-similar time series has a non-degenerates correlation structure at all time scales. Hurst first observed this unusual phenomenon and called it the *Hurst effect* [19].

Short-range dependant time series have a Hurst parameter of $H \approx 0.5$. Short-range dependant time series are not self-similar and their correlation structure degenerates into second order pure noise at higher time scales.

5.2 Methods of Estimating the Hurst Parameter

This section describes four methods of estimating the Hurst parameter of self-similar time series. Self-similarity is a property that manifests itself as a slowly decaying variance, long-range dependence, and a spectral density obeying a power law near the origin. Self-similarity can be studied with an analysis of variance examination of the aggregated time series, a time domain examination of the auto-correlation function, and a frequency domain examination of the periodogram. In addition to these three methods, a fourth method, the rescaled adjusted range statistic, is a popular heuristic for estimating the Hurst parameter in the time domain.

5.2.1 Auto-Correlation Function

The *Auto-Correlation Function (ACF)* is the correlation of a time series with itself. The auto-correlation function at lag k measures the linear relationship between a time series and itself k units of time ago. More specifically, the auto-correlation function, $r(k)$, is the correlation between X_t and X_{t-k} for all t satisfying $(k < t \leq N]$. The value of the auto-correlation function at lag zero, $r(0)$, is always one because it is the correlation of a time series, X_t, with itself.

The exploration of self-similarity in time series is usually limited to second-order. Analysis of n^{th} order self-similarity requires an equal or an asymptotic relation for all statistical moments up to and including n. For example, third-order self-similarity requires the distributions of X and $X^{(m)}$ to have equivalent means (first-

order), variance (second-order), and skewness (third-order). The analysis of higher orders is possible, but many researchers find second order self-similarity is sufficient. Tsybakov and Georganas [62] provide several equivalent mathematical definitions for exactly and asymptotically second-order self-similar for discrete time series.

A time series X is called *exactly second-order self-similar* if the auto-correlation function of $X^{(m)}$, $r^{(m)}(k)$, is equal to the auto-correlation function of X, $r(k)$, for all $m = 1, 2, 3, \ldots$. Equation 5.4 specifies the strict condition for exactly second-order self-similarity. Tsybakov and Georganas [62] claim that the class of exactly self-similar processes is not practical for modeling network traffic because its narrow range. Currently, researchers generally limit the use of exactly self-similar time series to theoretical purposes.

$$r^{(m)}(k) = r(k) \tag{5.4}$$

A time series, X, is called *asymptotically second-order self-similar* if $r^{(m)}(k)$, for sufficiently large k, is asymptotically equal to $r(k)$. Equation 5.5 specifies the conditions required for asymptotically second-order self-similarity. This thesis uses the general term *self-similar* in place of the term *second-order asymptotically self-similar* when the difference is not important to the discussion. The full self-similar descriptive term is used when the difference between exactly and asymptotically second-order self-similarity is important and to avoid ambiguity where necessary.

$$r^{(m)}(k) \to r(k), \text{ as } m \to \infty \tag{5.5}$$

A time series, X, is second-order self-similar, either exactly or asymptotically, if the auto-correlation function of X is indistinguishable from the auto-correlation function of the aggregated time series $X^{(m)}$ for sufficiently large values of k.

5.2.2 Short-Range and Long-Range Dependence

Short-range dependant (SRD) time series have an exponential or faster decaying auto-correlation function, Equation 5.6, and a summable auto-correlation function, Equation 5.7. The aggregation of short-range dependant time series has a degenerate auto-correlation function as $m \rightarrow \infty$ and $k \geq 1$, Equation 5.8. Traditional traffic generation models using Poisson or Markov processes are short-range dependant.

$$r(k) \approx p^k, 0 < p < 1 \tag{5.6}$$

$$\sum_k r(k) < \infty \tag{5.7}$$

$$r^{(m)}(k) \rightarrow 0 \text{ as } m \rightarrow \infty \tag{5.8}$$

In contrast to short-range dependant time series, *Long-Range Dependent (LRD)* time series have a hyperbolic or slower decaying auto-correlation function, Equation 5.9, and a non-summable auto-correlation function, Equation 5.10. The aggregation of long-range dependant time series has a non-degenerate auto-correlation function structure as $m \rightarrow \infty$ and $k \geq 1$, Equation 5.5.

$$r(k) \approx k^{\beta_{acf}} \tag{5.9}$$

$$\sum_k r(k) = \infty \tag{5.10}$$

If $X^{(m)}$ has a non-degenerate auto-correlation structure as $m \rightarrow \infty$ (long range dependant) then X has a self-similar structure [19, 24]. Furthermore, Equation 5.9 and Equation 5.11 approximate the Hurst parameter of a self-similar time series [26, 64].

$$\hat{H} = \frac{\beta_{acf} + 2}{2} \tag{5.11}$$

5.2.3 Rescaled Adjusted Range Statistic

The *rescaled adjusted range* statistic, or R/S statistic, is a time domain heuristic developed by Hurst [18] to estimate the Hurst parameter of a time series. The rescaled adjusted range statistic is $R(n)$, Equation 5.12, divided by $S(n)$, Equation 5.13. $S(n)$ is estimated with the sample standard deviation of $X_1, X_2, \ldots, X_n$, given in Equation 5.13. Hurst found that for many empirical time series, the R/S statistic satisfies the power-law relation given in Equation 5.14. This phenomenon is known as the *Hurst effect* or *Hurst phenomenon.*

$$R(X,n) = \max(\sum_{t=1}^{k}(X_t - \mu), 1 \leq k \leq n) - \min(\sum_{t=1}^{k}(X_t - \mu), 1 \leq k \leq n) \tag{5.12}$$

$$S(X,n) = \sqrt{E[(X_t - \mu)^2]} \tag{5.13}$$

$$E[R(X,n)/S(X,n)] \sim n^{\beta_{rs}} \qquad \text{as } n \to \infty \tag{5.14}$$

A self-similar structure appears as a straight line on a log-log R/S pox plot, where β_{rs} is the slope of the least squares line through the R/S statistic points. Equation 5.15 relates the approximated Hurst parameter, $\hat{H}$, and the slope of the least squares line through the R/S points, β_{rs}.

$$\hat{H} = \beta_{rs} \tag{5.15}$$

The method used to generate a rescaled adjusted range plot, described by Garrett and Willinger [24], is as follows: Subdivide the time series X into K sequential non-overlapping blocks. Compute the $R(t_i,n)/S(t_i,n)$ statistic for each of $t_1 = 1$,

$t_2 = N/K+1$, $t_3 = 2N/K+1$, ..., $t_K = (K-1)N+1$, which satisfy $(t_i - 1)+n \leq N$. A single lag value, n, can generate as many as K R/S statistic values when n equals zero and as few as one when $n \geq N - N/K$. Finally, the R/S pox plot is plotted as $\log(R/S)$ against $\log(n)$.

If X has strong short-range dependency, then the R/S pox plot values near the origin are scattered. Time series with long-range dependencies cluster points around a straight line with slope β_{rs} outside the short-range dependency correlation reach.

5.2.4 Variance-Time Statistic

The *variance-time* statistic is a graphical method for estimating the Hurst parameter of a time series. For a self-similar time series, the variance of $X^{(m)}$ decreases more slowly than the reciprocal of m. This is the basis of the variance-time statistic that shows the slowly decaying variance property of all self-similar time series [19].

The variance of an aggregated time series, $\mathrm{Var}(X^{(m)})$, for large m behaves as Equation 5.16.

$$Var(X^{(m)}) \sim m^{-\beta_{vt}} \qquad \text{as } m \to \infty \tag{5.16}$$

The following method determines the Hurst parameter using the variance-time statistic: Plot $\log(\mathrm{Var}(X^{(m)}))$, aggregated variance, against $\log(m)$, time. The asymptotic slope of the least squares line drawn through the variance-time points, omitting small values, gives an estimate of $-\hat{\beta_{vt}}$. Equation 5.17 shows the relation of β_{vt} to H.

$$\hat{H} = \frac{-\hat{\beta}_{vt} + 2}{2} \tag{5.17}$$

Short-range dependant time series have a Hurst parameter of $H \approx 0.5$, corresponding to a $\hat{\beta}_{vt} = 1$ and long-range dependant time series have a Hurst parameter of $(0.5 < H \leq 1]$ corresponding to slope estimates of $(0 < -\hat{\beta}_{vt} \leq 1]$. The variance-time plots of this thesis include a line with a slope of -1, corresponding to $-\hat{\beta}_{vt} = -1$. A $\hat{\beta}_{vt} = 1$ line represents the Hurst parameter of a short-range time series, $H = 0.5$.

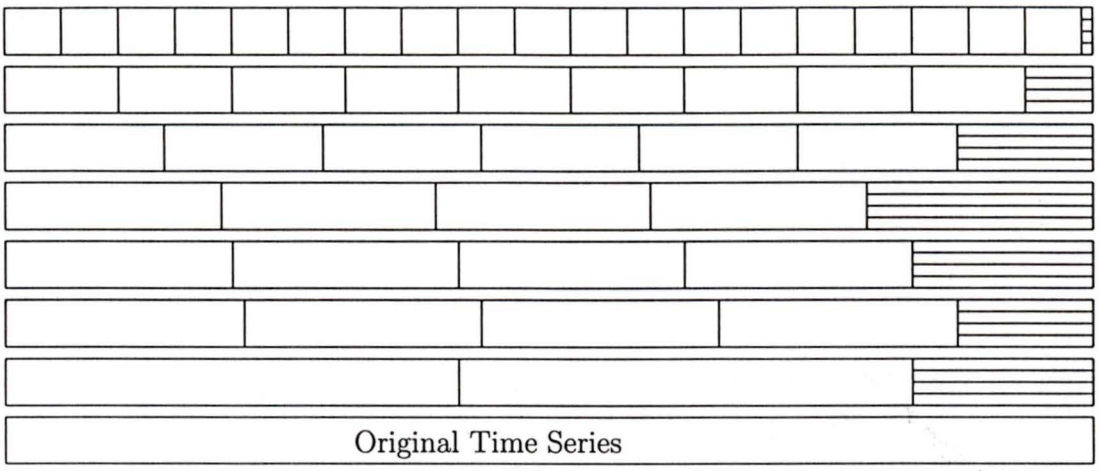

Figure 5.2: Original time series unevenly partitioned into seven aggregated time series. The aggregated time series excludes the horizontally stripped regions because they are smaller than the block size.

5.2.4.1 Unsuitability of Variance-Time Plots for Some Self-Similar Time Series

The variance-time plot provides poor Hurst parameter estimates for time series with a slowly converging variance-time statistic. This problem, common to all asymptotic relations, is particularly difficult to solve for the variance-time statistic.

When a time series is broken into large fixed-sized blocks that don't evenly partition the entire time series, the variance-time calculation omits the final portion of the time series. This causes the variance-time statistic estimate to oscillate when particularly bursty or quite regions of the time series are included and omitted from the aggregated time series as the block size changes. Figure 5.2 illustrates how the portion of the original time series omitted for the variance-time calculation changes as the block size of aggregated time series changes.

When the block size is small in relation to the time series, the maximum portion of the original time series discarded is small. As the block size increases, so does the potential to discard large portions of the original time series from the variance-time calculation.

The variance-time statistic implementation I developed avoids omitting large

portions of the time series from the variance calculation by selecting large block sizes that evenly partition the original time series. Block sizes less than 10% are selected so the density of points is uniform along the log time axis. I select block sizes greater than 10% of the original time series to evenly partition the original time series. This partitioning scheme guarantees the variance-time statistic calculation omits most 10% of the original time series, but reduces the number of estimates from large block sizes.

Making accurate estimates of a slowly converging variance-time statistic is difficult because the most important values, the values from the largest blocks, are scarce. It is only feasible to generate variance-time statistics values for a few large sized blocks that include the complete time series, or generate many large sized blocks that omit significant amounts of the time series.

5.2.5 Periodogram

A *periodogram* is a graphical method that estimates the Hurst parameter of long-range dependant time series [36]. A periodogram examines the slowly decaying property of all long-range dependent time series in the frequency domain.

A periodogram plots power spectral density against frequency on a log-log plot. *Power spectral density* illustrates how the power, or equivalently variance, of a time series is distributed within the frequency domain. The periodogram shows the contributions of each frequency to the overall variance. Spikes in a periodogram are frequencies with high variance in the time series.

Short-range dependant time series, time series with summable auto-correlation functions, have points scattered around a constant level near the origin of a periodogram. Long-range dependant time series, time series with non-summable auto-correlation functions, have points scattered along a negative slope near the origin of a periodogram. Equation 5.18 provides an estimate of the Hurst parameter, where $\hat{\beta_{psd}}$ is the slope of the least squares line of the periodogram points near the origin.

$$\hat{H} = \frac{(1 - \hat{\beta_{psd}})}{2} \tag{5.18}$$

5.2.6 Summary

This section describes four methods of estimating the degree of self-similarity of a time series. Self-similar time series have a Hurst parameter in the range $(0.5 < H \leq 1]$, and short-range dependant time series have a Hurst parameter of $H \approx 0.5$. The auto-correlation function, the variance-time plot, and the periodogram, Hurst parameter estimation methods examine the property of a slowly decaying auto-correlation function common to all self-similar time series. The R/S pox plot is a graphical heuristic to estimate the Hurst parameter.

The variance-time plot and the R/S pox plot are very robust to asymptotically second-order self-similar time series with strong short-range dependencies and a strong periodic correlation structure. This thesis uses these two Hurst parameter estimation methods.

This thesis does not present the auto-correlation function plot and the periodogram plot methods for estimating the Hurst parameter. These two methods are susceptible to the strong periodic correlations caused by inter-frame video compression. The variance-time plot and the R/S pox plot also breakdown for strong periodic correlations, but their disruptions are limited to the lower tail of the plot and not dispersed throughout the plot.

5.3 Experiment Description

This experiment explores the effect of long-running, self-similar, variable bit rate video traffic on the isochronous IEEE 1394, the TCP/IP, and the UDP/IP channels. The movie experiments consist of the client server framework streaming data with a variable bit rate video correlation structure across the three channels. Two movie traces derived from real compressed variable bit rate digital video provide the input for the simulation.

The movie experiments are setup similar to previous experiments described in Chapter 4. The client server framework generates, transmits, and records information about each frame for the three channels. The movie experiments uses the same three

types of client server modules described in Section 3.2.2.1 on page 48. The runtime environment, specified in Table 3.1 on page 43, is identical to the one used by the maximum utilization experiments. The same steps taken in the maximum utilization experiments, described in Section 4.2.2.1 on page 70, to reduce the effects of random contributions to response variables are also applied to movie experiments.

The only difference in the configuration of movie experiments compared to the maximum utilization experiments is the traffic generation module. The maximum utilization experiments use the maximum traffic general module described in Section 3.2.2.2 on page 49. The movie experiments use the playback traffic generation module, also described in Section 3.2.2.2. The movie experiments configure the playback module to generation traffic from two traffic traces of full-length, compressed, variable bit rate, digital video.

The transmission module segments large frames into several smaller packets to prevent the channel from performing its own fragmentation scheme. The frame sizes transmitted during the maximum utilization experiments were purposely chosen to be small enough to avoid fragmentation. The frames sent during the movie experiment are often much larger than the channel maximum transmission unit and are subject to channel fragmentation. The transmission modules fragment large frames into several smaller packets before transmission and reassemble multiple packets into complete frames after reception.

The observation module the movie experiments use is identical to the observation module the maximum utilization experiments use. The observation module is unaware of frame fragmentation, from either the channel or the transmission module, and only records information about complete frames.

The movie experiment uses two movie traces derived from real full-length, compressed, variable bit rate, digital video rather than a synthetic self-similar time series. Real variable bit rate video is preferable for simulations because of the challenges of creating quality self-similar synthetic traffic. Section 2.6.2 on page 30 describes the problems and challenges of synthetically generating quality long-running self-similar

traces.

The only real difference between configuration of the client server framework for the maximum utilization and movie experiments is the server configuration file. This file determines what modules are loaded and their configuration for each experiment. The client application configuration is identical for both the movie and maximum utilization experiments. The ease at which both groups of experiments were performed with the same software framework demonstrates the framework's power and configurability introduced in section 3.2.3 on page 55.

5.3.0.1 Buffering Client Input

Buffering client input is common technique to improve the quality of streaming media. Accumulating incoming frames in a time window reduces the effect of variable transmit lag time on the display mechanism. Starving a consumer of periodic data, in this case a display mechanism, is called buffer *underflowing*. The client application's buffer supplies the display mechanism with an uninterrupted stream of periodic frames as long as the client's input buffer is sufficiently large to overcome transmit lag. Client input buffering reduces the incidence of buffer underflowing.

5.3.1 Statistical Measures of Burstiness

This research uses several statistical techniques to quantify the burstiness of a time series. These techniques fall into two categories with several techniques in each category. The first category, which is the more traditional approach, treats the time series as an unordered set of observations. The second approach investigates the correlation structure of the time series at many time scales.

Dispersion is the tendency for data points to stray from the mean. A large dispersion value of a sample indicates points within the sample have a large average distance from their mean. Dispersion treats the elements of the time series as an unordered set. Calculating the dispersion of a time series ignores the serial structure of the time series.

The *standard deviation* is the most common statistical measure of dispersion. It estimates the relation data values have with their mean from a sample of data points. A large standard deviation, relative to the mean, indicates a large dispersion of values, or less central tendency, within a sample. The standard deviation of a data set is dependant of the mean of the data set and cannot be used in isolation.

The *coefficient of variation* is a measure of relative dispersion within a sample. The use of coefficient of variation is common when the mean and standard deviation change together at different levels of an experiment. The coefficient of variation is the ratio of standard deviation to mean. The use of the coefficient of variation is the preferred method of dispersion estimation for the statistical analysis of the movie experiments because the mean changes at each level of the experiments. The coefficient of variation is independent of the mean and can be used separately from the mean.

The *peak to mean ratio* is another technique to quantify burstiness that does not examine the correlation structure of a time series and treats the time series as an unordered set. The peak to mean ratio is not statistically rigorous, but is common in informal statistical analysis. The major advantage of the peak to mean ratio is people with limited statistical backgrounds can easily interpret and visualize the results.

The Hurst parameter is the degree self-similarity a time series possesses and is based on the slowly decaying auto-correlation function of a self-similar time series. The Hurst parameter examines the correlation structure of a time series at many time scales. Section 5.1 on page 101 explains what the Hurst effect is and section 5.2 on page 104 describes several techniques to estimate the Hurst parameter.

These statistical burstiness quantification estimation methods view burstiness in two different ways. The non-correlation methods examine the statistical properties of the time series as an unordered distribution of values. The Hurst parameter examines the correlation structure of a time series at several time scales. The order of the time series observations is integral to the Hurst parameter estimate, while the order of observations is of no consequence to the other measures of burstiness.

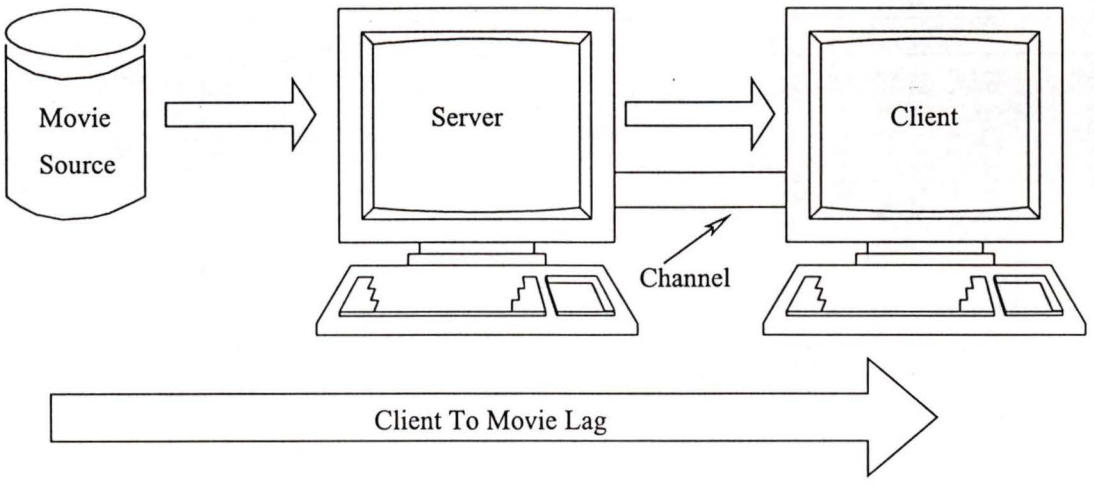

Figure 5.3: Movie to client lag performance measure used in the movie experiments.

5.3.2 Performance Metrics

Instead of investigating inter-frame delay and transmit lag as the primary performance metrics of channel performance, these experiments investigate client movie lag and how it affects the burstiness of client buffer occupancy. This section provides a general description and a formal mathematical definition of movie to client lag and client buffer occupancy.

Movie to client lag is the time between the client receiving the last packet of a frame and the movie trace's time-stamp of that frame. Movie to client lag is the cumulative effect of channel transmission, operating system, Java Virtual Machine, Java application, and other sources of lag. Figure 5.3 illustrates movie to client lag as a movie frame travels from the movie source through the server application to the client application. This illustration is similar to Figure 4.1 containing the performance measures of the maximum utilization experiments.

Client buffer occupancy is the number of bytes accumulated in the client's input buffer immediately prior to removing the oldest frame for display. In this experimental configuration, the client application buffers a fixed number of milliseconds of frames ahead of displaying them. The client application is always *running behind* the movie source by this fixed number of milliseconds. The client buffer occupancy is a trans-

formation from the continuous client movie lag time series to the discrete client buffer occupancy time series.

Client buffer occupancy is a time series that can be expressed as $X_t^{(\phi - L(t))}$. The first parameter, ϕ, is the client buffer size divided by the frame period, and the seconds parameter, $L(t)$, is the movie to client lag time series. The value of ϕ is the ratio of buffer size and frame period, which is constant for each experiment run. On the other hand, the client to movie lag time series, $L(t)$, is affected by the runtime environment and not constant. Movie to client lag may have different values for different values of t, *e.i* $L(t) \neq c$ for all $t \geq 1$.

If $\mathrm{Var}(L(t))$ is approximately zero, then $\phi - L(t)$ is approximately constant for all values of $t \geq 1$. In this case, $X_t^{(\phi - L(t))}$ can be approximated by simplifying the movie to client lag, $L(t)$, from a time series to a constant. The client buffer occupancy can now be expressed as $X_t^{(c)}$, where c is the difference of the two constants ϕ and $L(t)$. The time series $X_t^{(c)}$ is one aggregation of the time series X_t, defined in equation 5.3 on page 102. As a consequence, the aggregated time series $X_t^{(\phi - L(t))}$ has the same correlation structure, therefore the same Hurst parameter, as the original time series X_t.

Section 5.2.1 contains one definition of self-similarity, which stipulates that if X_t is self-similar then so are all elements of $X_t^{(m)}$. The correlation structure, and therefore the Hurst parameter of X_t and $X_t^{(m)}$ are equivalent.

A special case of the aggregated time series $X_t^{(\phi - L(t))}$ is when $(\phi - L(t)) = 1$ for all $t \geq 1$. In this case, the time series X_t is aggregated over blocks of size one and equivalent to the original time series. Formally, the time series X_t is equivalent to $X_t^{(1)}$ for all $t \geq 1$. This situation occurs when the client buffer contains exactly one movie frame. In this case, the client buffer occupancy, $X_t^{(1)}$, is equivalent to the original frame size time series, X_t, used to generate the experiment traffic.

If $\mathrm{Var}(L(t))$ is not zero, the correlation structure, and therefore the Hurst parameter, of $X_t^{(\phi + L(t))}$ is affected by client to movie lag, $L(t)$. The Hurst parameter of the client buffer occupancy time series may be different from the frame size time

series.

The R statistical environment was used to generate the client buffer occupancy time series from the channel activity recorded by the client server framework. The generated client buffer occupancy time series simulates a client application's buffer filling from the client activity trace and emptied one frame per frame period. The simulation immediately discards frames arriving at the client after their scheduled display time; these frames are called *dropped frames*. The simulation assumes the client has an effectively infinite size input buffer that does not overflow.

The client buffer occupancy time series is generated from simulation results instead of captured directly from the client application's input buffer to aide statistical analysis. Capturing the client buffer occupancy results directly from the client application's input buffer because this requires separate runs for each buffer size. Instead, the R statistical environment generates multiple client buffer occupancy time series from the same channel activity log. This permits burstiness statistics from the same movie trace and channel type with several different sized client input buffers, but without having to statistically isolate the individual contributions of multiple experiment runs.

5.3.2.1 Unsuitability of Inter-Frame Delay and Transmit Lag Channel Performance Metrics

The inter-frame delay and transmit lag performance metrics used by the maximum utilization experiments are not suitable for the movie experiments for the following three reasons: they do not capture the burstiness of self-similar variable bit rate video, their values calculated for variable bit rate video are difficult to interpret, and the frame times they are derived from do not represent true channel performance. For these reasons, inter-frame delay and transmit lag are not used to evaluate the performance of the movie experiments.

Inter-frame delay and transmit lag do not capture the burstiness of self-similar variable bit rate video. The self-similar structure of variable bit rate video presents

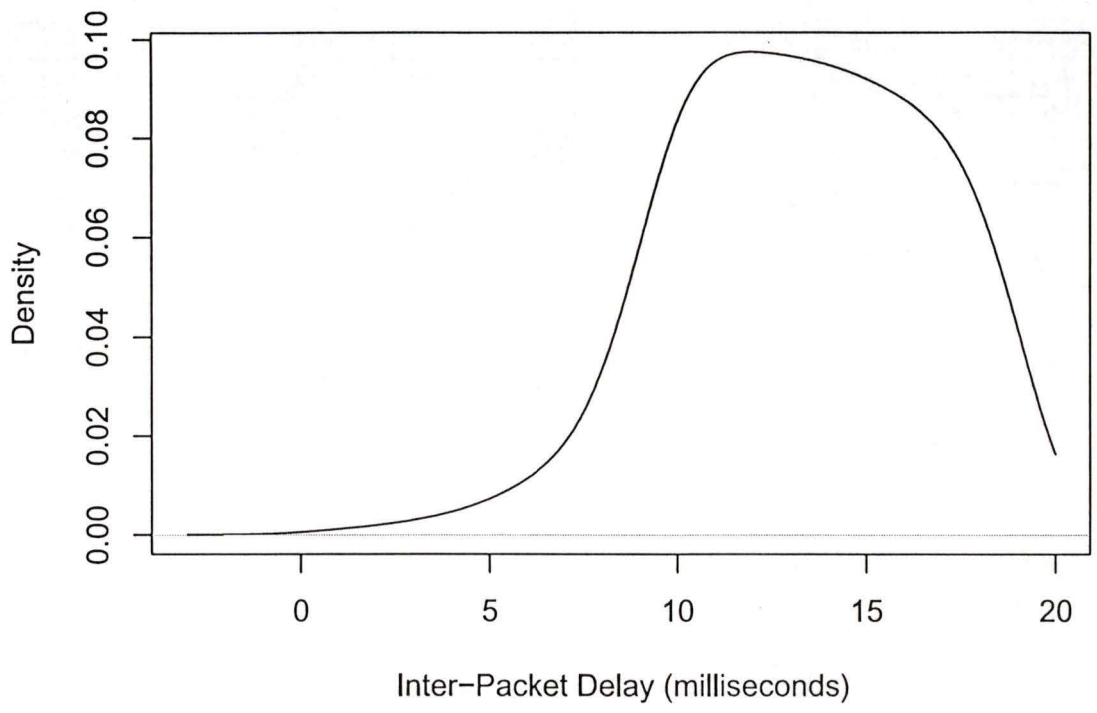

Figure 5.4: Density plot of lag from the MPEG movie trace to server frame timestamp. The visible portion of the plot represents 99.6% of the MPEG movie trace frames.

additional challenges that applications must satisfy to provide correct handling of streaming temporal data. Section 2.6.2 on page 27 contains further details about the difficulties self-similar correlation structure cause computer communications. These performance measures look at only at frame timing and do not incorporate the self-similar frame size correlation structure.

Inter-frame delay and transmit lag performance measures are difficult to interpret for channels transmitting variable bit rate video. Transmission of variable bit rate video, as the name implies, uses variable sized frames. The server application transmits each frame at regular intervals determined by the frame rate, but each frame requires different transmission time. The ideal values for inter-frame delay and transmit lag vary with each frame size. To further complicate the matter, two non-ideal transmission times, one too small and one too large, create a mean transmission time deceptively closer to the ideal transmission time.

Inter-frame delay and transmit lag performance measures do not represent true channel performance for channels transmitting self-similar variable bit rate video. The client and server applications are idle between frames because of excess channel and processing capacity of the client server framework. The operating system schedules other processes, including itself, during idle time to maximize its utilization. Ideally, the kernel would reschedule the client and server applications immediately at the start time of each video frame. In reality however, the kernel reschedules applications on scheduling quanta boundaries. Furthermore, the kernel contains critical routines that cannot be preempted without leaving the kernel in an inconsistent and unpredictable state. The resulting scheduling induced delay ranges from zero to $(T + P)$. The parameter T is the operating system scheduling quanta and the parameter P is the length of longest non-preemptable path. The operating system's process scheduling policy dominates packet time values in general purpose operating systems.

Figure 5.4 on page 118 supports the following interpretation of server application behaviour. The server application processes the first packet of a movie frame within the range zero to twenty milliseconds for 99.6% of all frames. This indicates that the kernel's scheduling activation time is approximately twenty milliseconds. The kernel services a processes scheduling request usually within twenty milliseconds. The kernel's process scheduling policy completely dominates all frame timing statistics.

Inter-frame delay and transmit lag are unsuitable channel performance measures for the movie experiments. The performance measures from previous experiments do not capture critical attributes of self-similar variable bit rate data, are difficult to interpret, and are not comprehensive.

5.3.3 Interpreting Variance-Time and Rescaled Adjusted Range Plots

The variance-time statistic and the rescaled adjusted range statistic, R/S , are both graphical methods for estimating the Hurst parameter of a time series. Both of these methods fit a least squares line through a group of points and extract the estimated Hurst parameter from the equation of the fitted line. Each Hurst parameter

estimate of requires manual configuration for both estimation methods to produce meaningful estimation results.

The least squares calculation used by both methods exclude points from the upper and lower tails. The Hurst parameter calculations include points plotted with an "O" and exclude points plotted with an "X". Excluding the lower tails removes short-range correlations, which are not of primary interest. Excluding the upper tails removes statistic points with low reliability. *Low reliability* refers to the lack of statistical strength due to calculations based on only a few data points. The size of the excluded upper and lower tails is dependant on the time series and the Hurst parameter estimation method.

The R/S pox plot has two dashed boundary lines, $y = x$ and $y = \frac{x}{2}$. The area within the boundary corresponds to Hurst parameters in the range $(0.5 < H < 1)$, which is the class of self-similar time series. A R/S pox plot with the majority of its included points within the boundary region indicates a self-similar time series. The slope of the least squares line, β_{rs}, and equation 5.15 on page 107 estimate the Hurst parameter.

The variance-time plot has one dashed boundary line, $y = -x$. The boundary line corresponds to a Hurst parameter of $H = 0.5$, which is the class of short-range dependant time series. Variance-time plots with the majority of included points above the boundary line indicate a self-similar time series. The slope of the least squares line, β_{vt}, and equation 5.17 on page 108 estimate the Hurst parameter.

5.3.4 Movie Trace Description

Each of the two movie experiments uses a different movie trace as the external source of the playback traffic generation module. The two movie traces were derived from two full-length, variable bit rate compressed digital videos with two different types of compression. The first movie trace, referred to in this thesis as *MPEG*, uses the MPEG-1 [14] compression method. The second trace, referred to as *Starwars*, uses a M-JPEG compression method. Section 2.5.1 on page 23 provides additional

information about both digital video compression methods.

The MPEG-1 and M-JPEG digital video compression methods share some the same compression techniques, but they differ at what level they compress video. The MPEG-1 compression method used by the MPEG movie trace, compresses groups of frames together, exploiting within frame and between frame information redundancies. The M-JPEG compression method, used by the Starwars video trace, compresses each frame individually.

The *Group Of Pictures (GOP)* is a regular repeating pattern of frame types compressed together. An individual frame within a group of pictures may not contain sufficient information decompress the original frame. A complete group of pictures contains all the information required decompress every frame within the group of pictures. There are three types of MPEG-1 frames called I, B, and P frames. The *intra-frame*, or I frame, contains complete information about one frame of the original source. The two other frame types use inter-frame compression techniques that span more than one frame and cannot be decompressed alone. The *bi-directional* frame, or B frame, contains information about the previous or next I or P frame. The *predictive* frame, or P frame, predicts the location of blocks from a previous I or P frame.

The group of pictures sequence of the MPEG movie trace is "IBBPBBPBBPB". Each frame type uses a different compression technique and compresses frame information at different rates. The periodic structure of the group of pictures and the different compression ratios typically attained with each MPEG-1 frame type creates strong periodic correlation in the MPEG trace. The strong periodic correlation caused by the group of pictures of MPEG-1 compression influences the overall correlation of the movie trace.

The M-JPEG compression method, used by the Starwars video trace, compresses each frame individually. This compression method exploits within frame information redundancies only. The M-JPEG compression method is conceptually the same as the MPEG-1 compression method with a group of pictures of "I". The correlation structure of the movie and not an artifact of M-JPEG compression cause

Table 5.1: Descriptive statistics of original and transformed movie traces.

	MPEG		Starwars	
	Original	Transformed	Original	Transformed
Encoded Frame Rate (f/s)	24	24	24	24
Frame Count	174136	170000	171000	170000
Mean Frame Size (bytes)	15600	15630	27790	27800
Minimum Frame Size (bytes)	476	480	8622	8628
Maximum Frame Size (bytes)	185267	185268	78459	78468
Total Trace Size (Mbytes)	2716	2656	4752	4725
Length (seconds)	7256	7083	7125	7083
Encoded Frame Period (μs/f)	41.67	41.67	41.67	41.67
Mean Bandwidth (Mbps)	2.99	3.00	5.34	5.34
Minimum Bandwidth (Mbps)	0.91	0.92	1.66	1.66
Maximum Bandwidth (Mbps)	35.6	35.6	15.1	15.1

any periodic correlations in a M-JPEG trace.

Both the Starwars and MPEG movie traces are publicly available. Garrett [23] published the MPEG trace in 1993. Garrett and Willinger [24] published the Starwars trace in 1994. Several authors [22, 36] have verified the self-similar results of the movie traces.

5.3.4.1 Movie Trace Statistical Properties

Each movie trace consists of approximately 170,000 video frames. Each frame is encoded at a constant frame rate of twenty-four frames per second. The running time of each movie is approximately two hours. The movie traces only contain the size of each frame after compression, which is sufficient for the movie experiments. The movie traces do not contain enough information to reconstruct the original video picture they were derived from.

The original movie traces underwent three transformations before conducting the movie experiments. The size of each frame was rounded up to a multiple of twelve bytes to accommodate client-server framework data consistency checks. Both movie traces were truncated to 170,000 frames to simplify the visualization of graphical

Table 5.2: Burstiness statistics of the MPEG and Starwars movie traces.

	MPEG		Starwars	
	Original	Transformed	Original	Transformed
Standard Deviation (bytes)	18160	18130	6254	6254
Coefficient of Variation	1.165	1.160	0.2250	0.2250
Peak to Mean Ratio	11.88	11.85	2.823	2.823
Variance-Time Statistic	0.8765	0.8807	0.7704	0.7717
R/S Statistic	0.8806	0.8800	0.8476	0.8570

statistical results. Finally, the movie trace was converted from a text format to a Java binary format to reduce garbage collection of the server application.

Table 5.1 summarizes the general properties and descriptive statistics of the two original movie traces and the two transformed movie traces. The descriptive statistics of the transformed movie traces do not significantly differ from their original form with the obvious exception of total length in seconds and bytes.

Table 5.2 contains estimates of several burstiness quantification methods of the original and transformed movie traces. The top portion contains non-correlation estimates, and the bottom portion contains Hurst parameter estimates.

Table 5.2 contains information to support three conclusions about the movie traces. First, the burstiness statistics of the transformed movie traces do not significantly differ from their original form. Second, the Hurst parameter estimates of the variance-time statistic and the R/S statistic indicate all movie traces are self-similar. Third, the non-correlation statistical properties of the MPEG and Starwars movies traces are very different, while the Hurst parameters of the two movies, especially the R/S statistic, are similar.

The Hurst parameter estimates published by Garrett and Willinger [24] match those results of the original Starwars trace in Table 5.2. This supports the conjectures that I implemented the somewhat complicated Hurst parameter calculations correctly, I trimmed the upper and lower tails of the plots correctly, and I interpreted the plots correctly. This lends creditability to the other Hurst parameter estimates throughout

this research.

Figure 5.5 contains the auto-correlation function of the MPEG movie trace frame size, the MPEG movie trace group of pictures size, and the Starwars movie trace frame size. The top plot shows the auto-correlation function of the MPEG movie trace's frame size, which uses inter-frame compression. This plot shows strong correlations at multiples of twelve lag values. The length of the MPEG movie trace's group of pictures, not surprisingly, is twelve. The middle plot shows the auto-correlation function of the MPEG movie trace's group of pictures size, which removes the inter-frame compression correlation structure. The bottom plot shows the auto-correlation function of the Starwars movie trace's frame size. The Starwars movie trace does not use inter-frame compression and clearly does not possesses strong periodic correlations present in the MPEG movie trace.

All the auto-correlation function plots of Figure 5.5 suggest the movie traces have a self-similar structure. The following three points indicate the movie traces are self-similar. The plots do not converge to zero, indicating a non-summable auto-correlation function, which indicates of self-similarity. The auto-correlation function of the movie traces at lag values of up to $10,000$ also do not converge to zero. The MPEG group of pictures plot and the Starwars plot show a hyperbolically decaying auto-correlation function.

Figure 5.6 and Figure 5.7 show the R/S pox plot and the variance-time plot of the MPEG movie trace. The strong periodic correlation structure, visible in the auto-correlation function plot in Figure 5.5, is caused by the MPEG-1 inter-frame compression used to encode the MPEG movie trace. The periodic correlation structure distorts the R/S and variance-time statistics over small block sizes. To remove the contribution of the MPEG movie trace's group of pictures correlation structure, the lower tail of the R/S poxplot is trimmed at $\log_{10} 1.5$ and the lower tail of the variance-time plot is trimmed at $\log_{10} 1$. The lower tail trim point demarks the periodic trend, short-range dependant, and the self-similar trend, long-range dependant, of the MPEG movie trace.

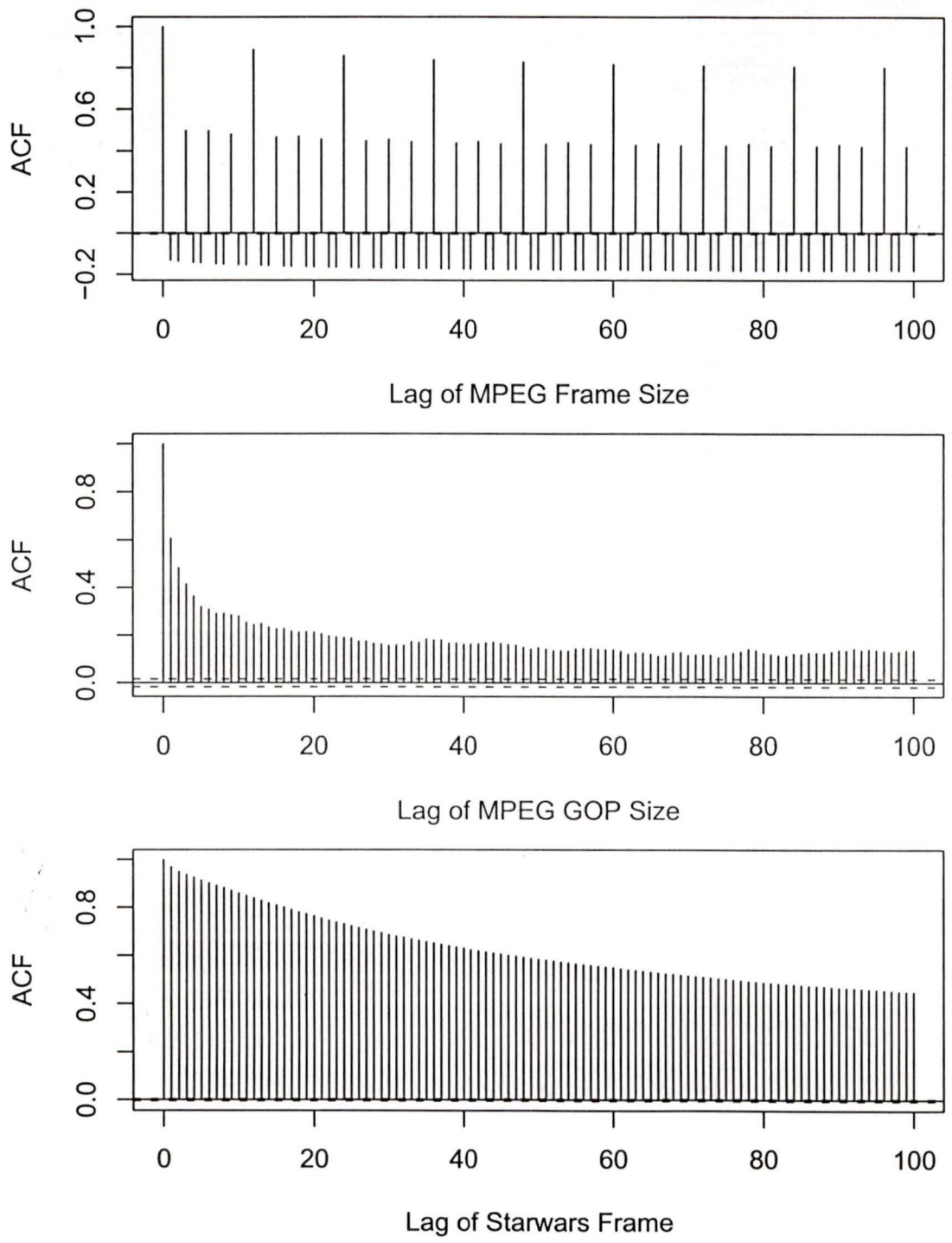

Figure 5.5: Auto-correlation function: top) MPEG movie trace's frame size, middle) MPEG movie trace's group of pictures size, bottom) Starwars movie trace's frame size.

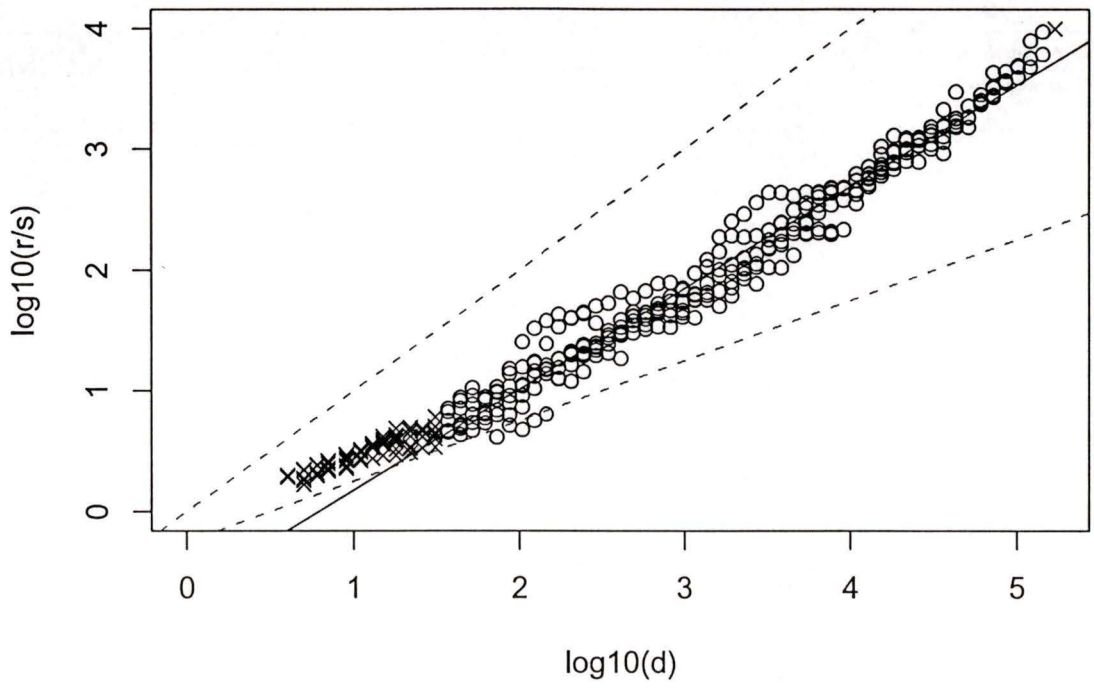

Figure 5.6: R/S pox plot of the MPEG movie trace.

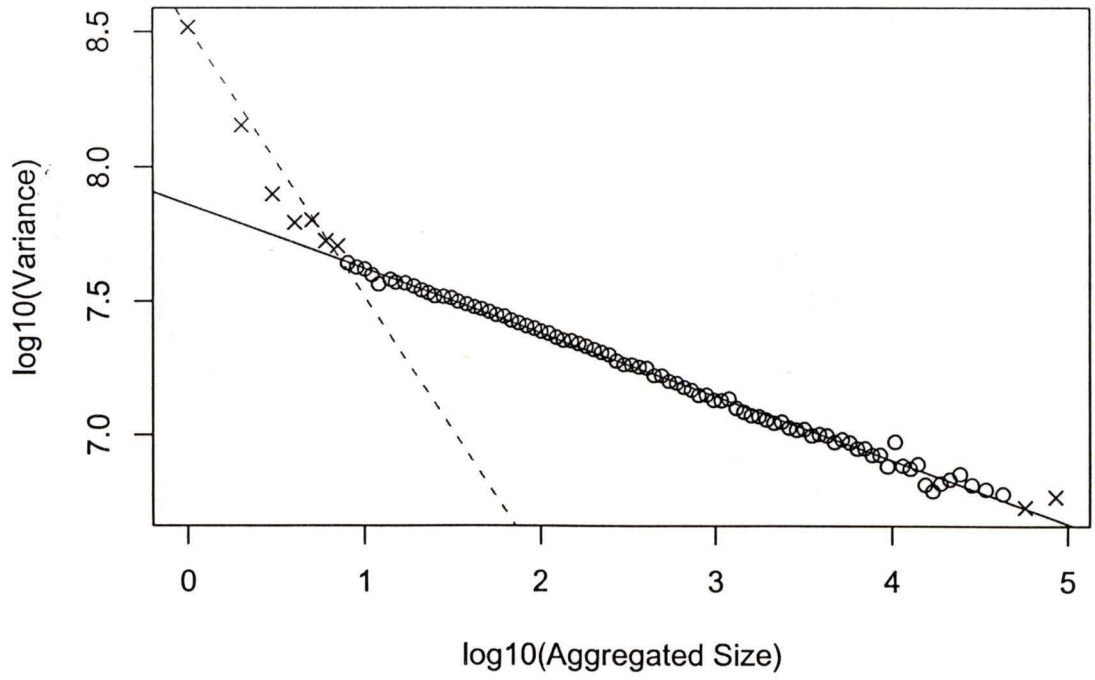

Figure 5.7: Variance-time plot of the MPEG movie trace.

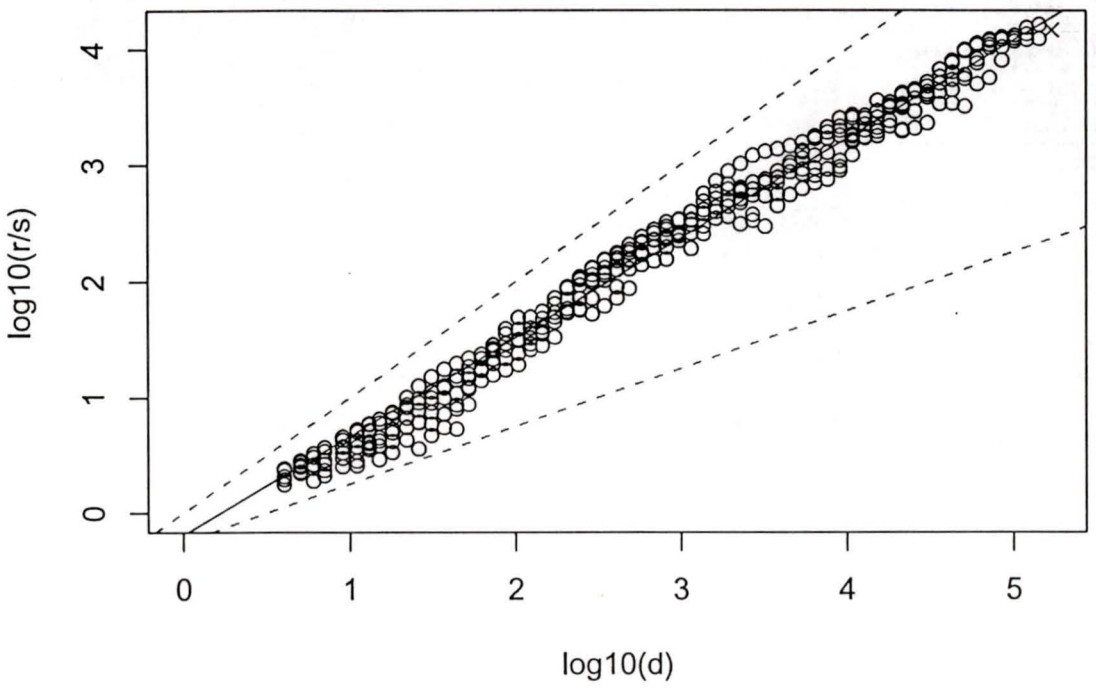

Figure 5.8: R/S pox plot of the Starwars movie trace.

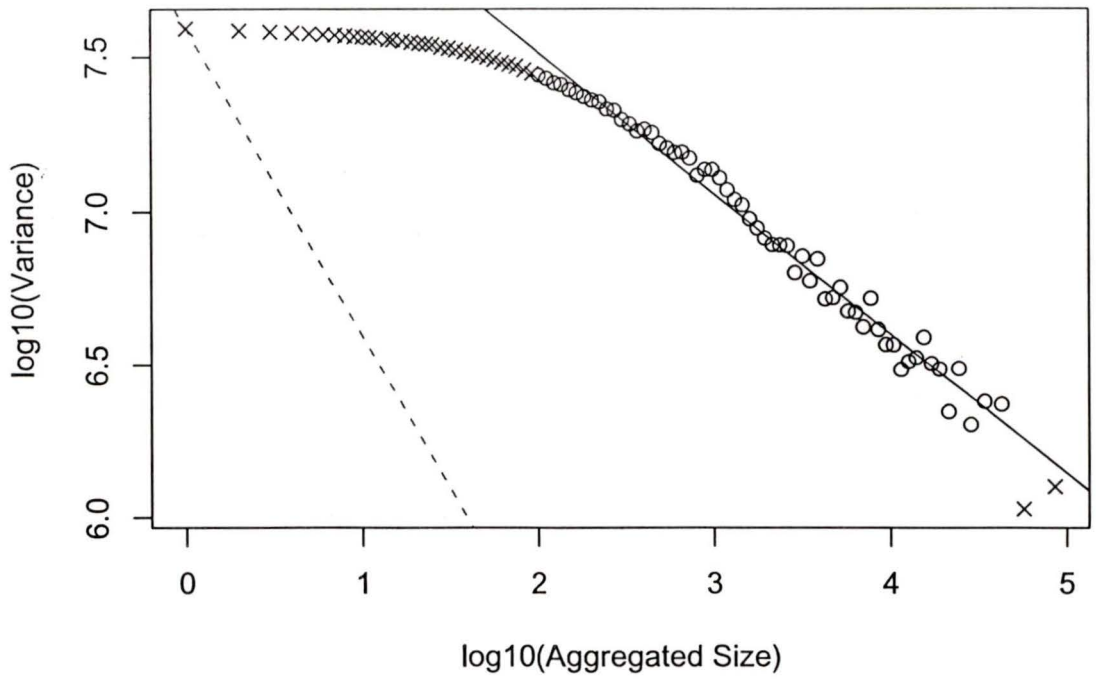

Figure 5.9: Variance-time plot of the Starwars movie trace.

Figure 5.8 and Figure 5.9 show the R/S pox plot and the variance-time plot of the M-JPEG encoded Starwars movie trace. The lower tail of R/S pox plot of the Starwars movie trace is not trimmed because the Starwars time series does not have a periodic correlation structure similar to the MPEG movie trace. The lower tail of the Starwars variance-time plot is trimmed considerably more than the R/S pox plot. The lower tail of the R/S pox plot is trimmed at $\log_{10} 0.5$ and the lower tail of the variance-time plot is trimmed at $\log_{10} 2$.

Table 5.2 on page 123 contains the Hurst parameters estimates for the transformed MPEG and Starwars movie traces from Figure 5.6, Figure 5.7, Figure 5.8, and Figure 5.9. Table 5.2 also includes the estimates of the Hurst parameter from the R/S pox plots and variance-time plots for the original movie traces.

Garrett and Willinger [23, 24], the authors of the video traces, provide a detailed description of the statistical properties and the generation techniques of the Starwars and MPEG movie traces.

5.3.5 Summary

The focus of this experiment shifts away from channel performance measures based solely on frame timing to a performance measure that incorporates frame timing and frame size. The frame size information is from two self-similar movie traces. The frame timing information is from the runtime environment's affect on client to movie lag. The performance measure of the movie experiments is the burstiness of client buffer occupancy and the effect client buffer size has on it.

The movie experiments use the client server framework, which was also used by the previous experiments. Instead of generating synthetic self-similar variable bit rate video traffic, the client server framework generates traffic derived from real self-similar variable bit rate video.

This research treats each movie trace as a separate experiment. The movie traces are encoded in fundamentally different ways and have different statistical properties. The approach taken in this research is to examine the changes within a movie

trace and compare trends between the two traces. Direct comparison of burstiness estimates from the two traces is not meaningful.

This section concludes with a quick numerical summary of the movie experiments.

- Number of movie traces: 2 (MPEG and Starwars)
- Number of channels: 3 (isochronous IEEE 1394, TCP/IP, and UDP/IP)
- Number of runs: 6
- Number of frames transmitted: $1,020,000$
- Number of packets transmitted: $\approx 16,000,000$
- Number of bytes transmitted: $\approx 22,000,000,000$
- Total transmission time: $\approx 42,500$ seconds (11 hours 48 minutes)

5.4 Preliminary Investigation of Client Movie Lag

Movie to client lag is a time series where each element is the time elapsed from a movie frame's timestamp to the client's stop timestamp. Movie to client lag, first illustrated in Figure 5.3 on page 115 and further described in Section 5.3.2, consists of two lag components: the movie to server lag and the server to client lag.

The maximum utilization experiments in Chapter 4 examine the server to client lag for various channel types, buffer sizes, and frame sizes. Table 4.10 on page 85 contains the summary of the server to client lag observed in the maximum utilization experiments. The server to client lag variance contribution to movie to client lag is, in most cases, small relative to the movie frame period of approximately forty-two milliseconds.

The density plot in Figure 5.4 on page 118 shows the movie to server lag for the MPEG movie trace. The peak density range is approximately nine to fifteen milliseconds. The density distribution drops off to almost zero for lag values greater

than twenty milliseconds. A movie to server lag value of twenty milliseconds is almost one half a movie frame period. Lag values of this magnitude are relatively large and significant in the context of streaming media.

5.4.1 Movie to Client Lag Density

Figure 5.10 contains the movie to client lag density from the isochronous IEEE 1394, the TCP/IP and the UDP/IP channels. All three channels have a density peak at approximately five to twenty-five milliseconds. This density peak coincides with the movie to server lag, illustrated in Figure 5.4 on page 118, shifted approximately five milliseconds left.

The isochronous IEEE 1394 contains the initial density peak at five to twenty-five milliseconds and a second, larger, density peak at approximately forty-five to sixty-five milliseconds. This peak is the result the IEEE 1394 device driver forwarding isochronous packets when its internal buffer fills instead of immediately after receiving them. The fill and forward technique employed by the IEEE 1394 for Linux device driver adds approximately one frame period to the lag most frames experience. The isochronous IEEE 1394 plot in Figure 5.10 illustrates this property when the isochronous IEEE 1394 density is shifted one frame period to the left. With the isochronous IEEE 1394 density shifted approximately forty-two milliseconds, the second density peak coincides very closely with the first density peak.

The TCP/IP also contains an initial density peak, followed by a second, smaller, density peak. I cannot definitively explain the cause of the secondary density peak found in the TCP/IP plot without further experimentation. One possible explanation of the secondary density peak is the use of Nagle's algorithm in the TCP/IP implementation. *Nagle's algorithm* combines several smaller packets into a single larger packet. The efficiency of the channel increases when many small packets are replaced with a single larger packet. The channel transmits same amount of information, but in fewer packets and less overhead. The timeliness of the channel decreases because Nagle's algorithm delays packets for combination instead of scheduling them

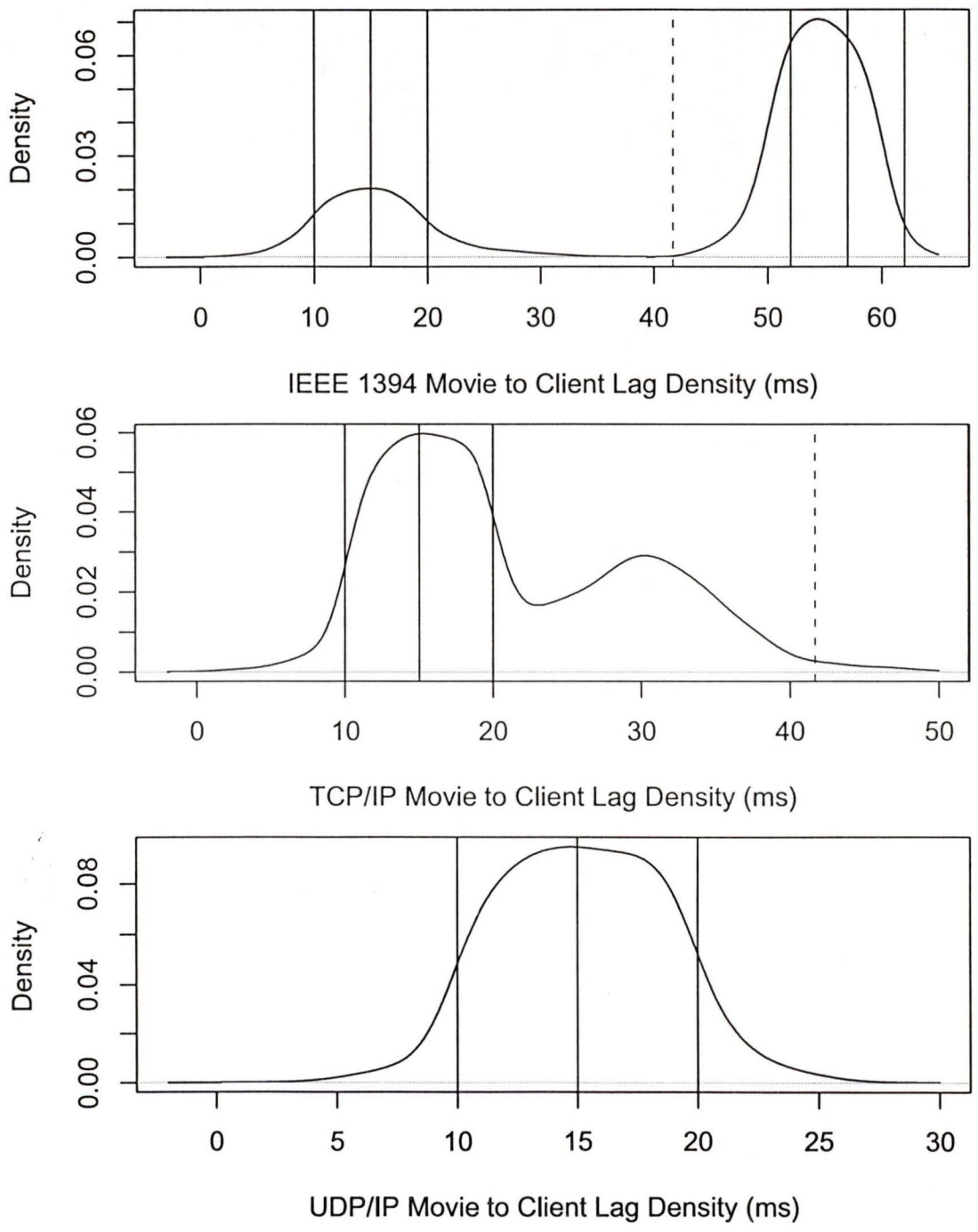

Figure 5.10: Movie to client lag of the MPEG movie trace.

for immediate transmission. Nagle's algorithm reduces responsiveness for increased throughput.

The UDP/IP client to movie lag density plot has a single peak, whereas the isochronous IEEE 1394 and the TCP/IP have secondary density peaks. The UDP/IP does not use the fill and forward technique used by the isochronous IEEE 1394 device driver. Moreover, the UDP/IP also does not use Nagle's algorithm, which may improve channel efficiency at the cost of timeliness. The density plot of the UDP/IP shows the tightest density distribution of the three channel types.

The movie to client lag density plots from Starwars and MPEG movie traces are similar when transmitted with the same channel type. For brevity, this section only presents the MPEG movie to client lag density plots, but analysis does include the results from both movie traces.

The investigation of movie to client lag supports three conclusions. First, all three channels have variable client to movie lag. Second, the size of the lag for the three channels is large relative to the period of a movie frame. Third, each channel type has a different client to movie lag distribution. These three properties of the movie to client lag may change the statistical properties of the client buffer occupancy.

5.4.2 Client Buffer Size

The size of the client's input buffer has an enormous affect on client buffer occupancy and potentially on the burstiness of the client buffer occupancy. The movie to client lag density plots, Figure 5.10, suggests several interesting client buffer sizes to simulate client buffer occupancy.

The solid vertical lines in the movie to client lag density plot of the MPEG movie trace represent the client buffer sizes used for all the client buffer occupancy experiments. The dashed vertical line demarks frame periods in each plot. The client buffer sizes used to calculate client buffer occupancy are ten, fifteen, and twenty milliseconds. These three buffer sizes are repeated for three frame periods. Table 5.3 contains the nine levels of client buffer sizes. These points partition the density of

Table 5.3: Client buffer size levels of the movie experiments.

Frame Periods	Buffer Size		
1	10 μs	15 μs	20 μs
2	52 μs	57 μs	62 μs
3	93 μs	98 μs	103 μs

the client to movie lag in nine ways, spanning three frame periods.

The isochronous IEEE 1394 movie to client lag in Figure 5.10 shows the client buffer size partitioning the density across two frame periods. The TCP/IP also has a small portion of its density in the adjacent frame period. The client buffer of the isochronous IEEE 1394, and to a lesser extent the TCP/IP, underflows with client buffer sizes less than one frame period.

5.5 Estimates of Burstiness

This thesis uses four methods to quantify the burstiness of the client buffer occupancy time series. The movie experiments use two methods from each of the two categories of burstiness quantification described in section 5.3.1 on page 113. The R/S statistic and the variance-time statistic estimate the Hurst parameter, which examines the correlation structure of the time series at many time scales. The peak to mean ratio and the coefficient of variation are non-correlation methods to quantify burstiness.

I chose the R/S statistic and the variance-time statistic Hurst parameter estimation methods because of their containment of strong, short-range, periodic correlations present in the MPEG movie trace. Filtering short-range periodic correlations from these two methods Hurst parameter estimation methods is easy because short-range correlations are present only in the lower tail of the plot.

I chose the coefficient of variation and the peak to mean ratio non-correlation burstiness quantification methods because of their resilience to changing mean. Hurst parameter estimation methods are not susceptible to changing means, and non-

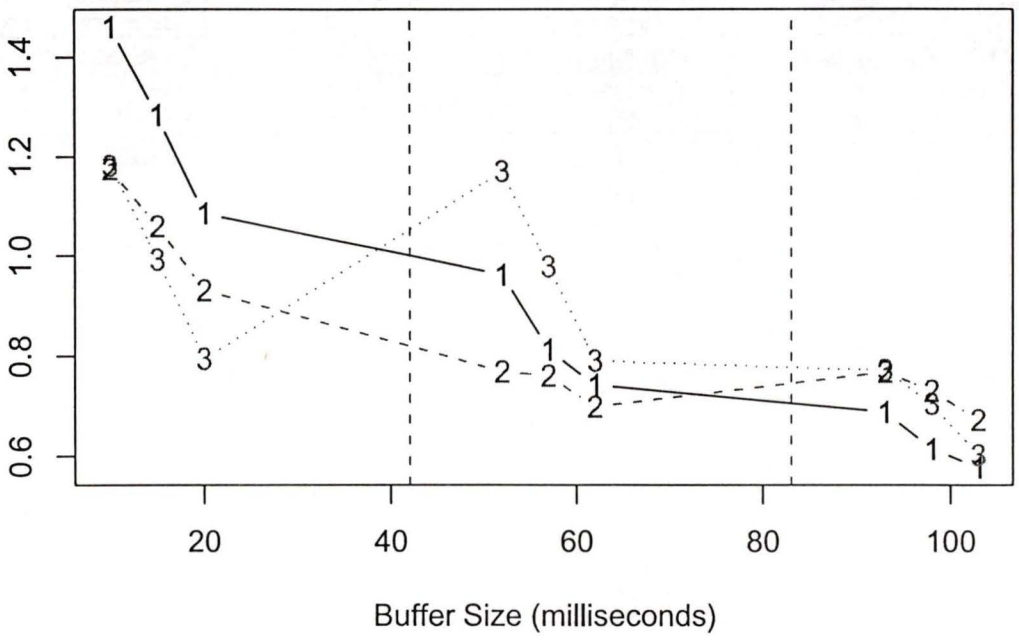

Figure 5.11: Coefficient of variation of the MPEG movie trace at different buffer sizes.

correlation methods are not susceptible to strong periodic correlations. These four burstiness methods provide a well-rounded picture of burstiness.

The standard deviation is not a good method for comparison of burstiness for the movie experiments. Section 5.3.1 describes the problem of using the standard deviation at different levels of an experiment where the mean changes. The standard deviation is directly related to the mean and heavily influenced by any changes in mean. The mean of client buffer occupancy increases as the client buffer size increases and more video frames are buffered. An increase in the standard deviation accompanied with an increase in mean provides little insight, as both the increasing mean and any increase in dispersion cause the increase standard deviation. For this reason, the standard deviation is not included in further movie experiment results.

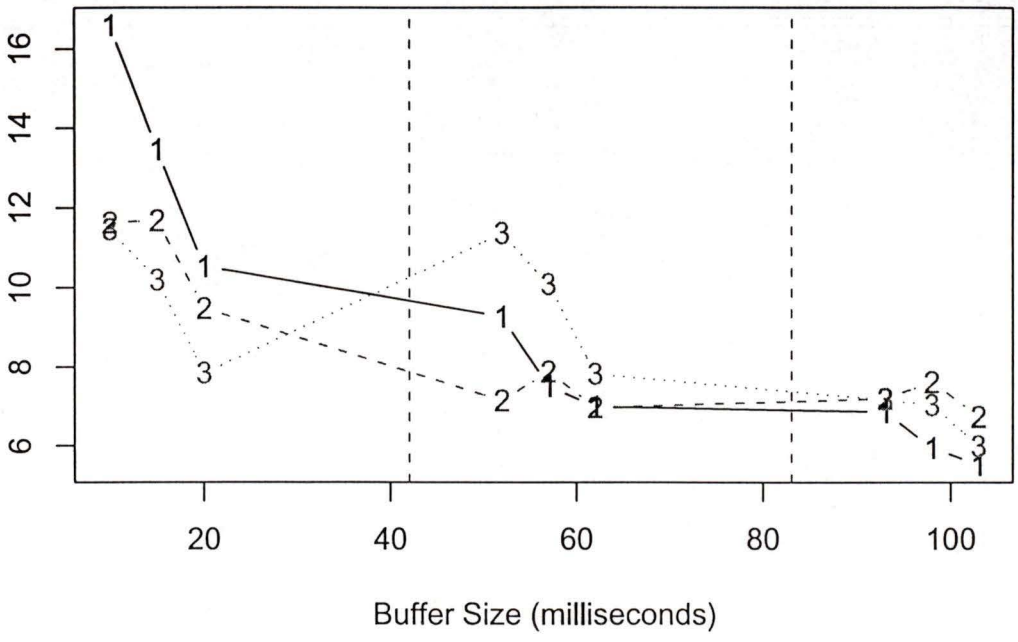

Figure 5.12: Peak to mean ratio of the MPEG movie trace at different client buffer sizes.

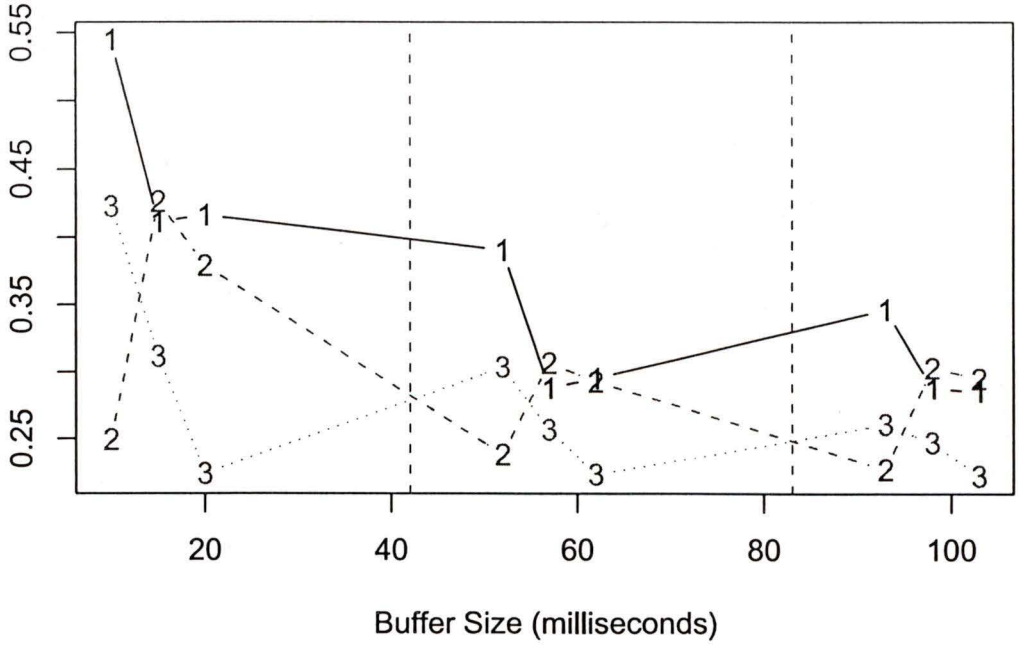

Figure 5.13: Coefficient of variation of the Starwars movie trace at different client buffer sizes.

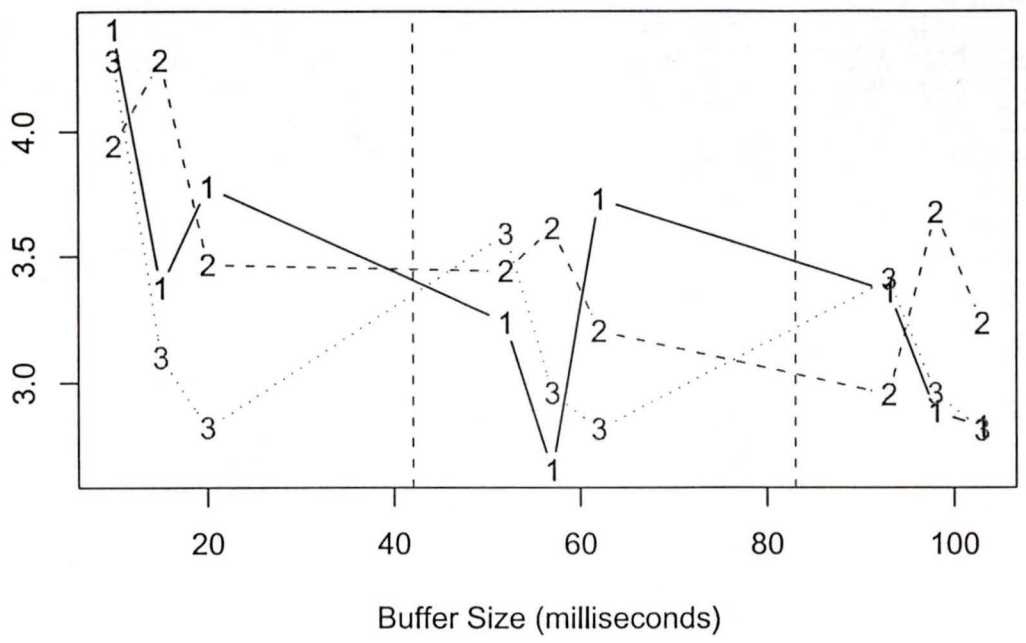

Figure 5.14: Peak to mean ratio of the Starwars movie trace at different client buffer sizes.

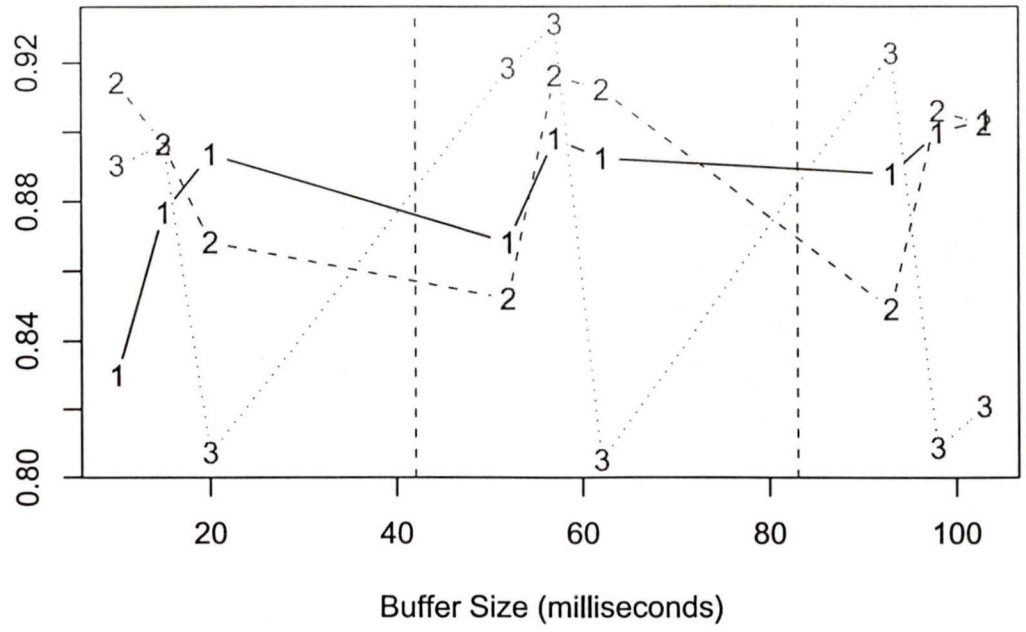

Figure 5.15: R/S Hurst parameter estimate of the MPEG movie trace at different client buffer sizes.

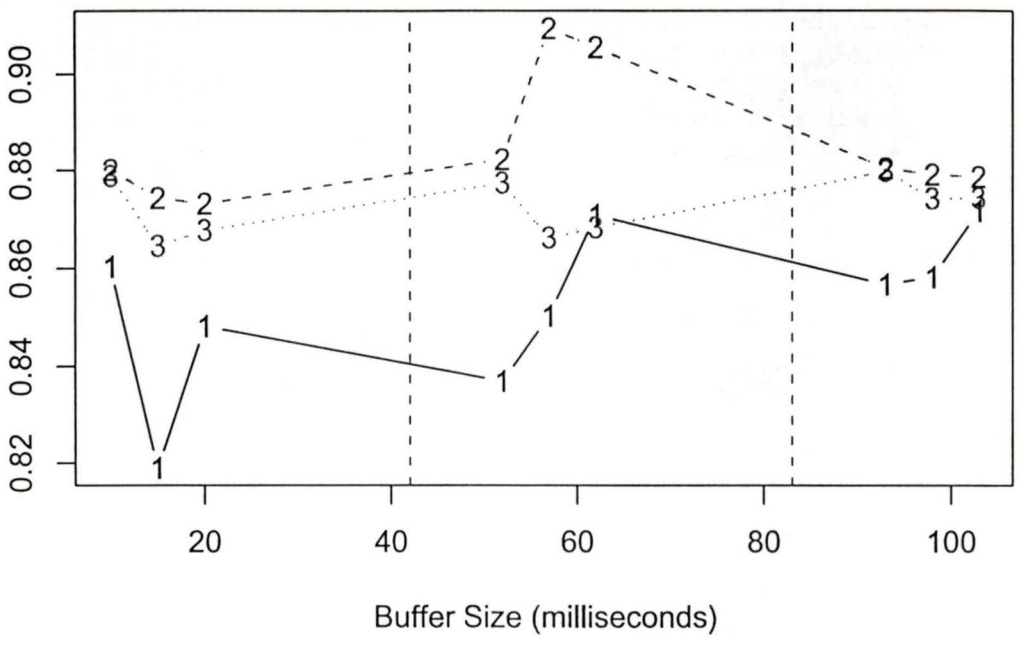

Figure 5.16: Variance-time Hurst parameter estimate of the MPEG movie trace at different client buffer sizes.

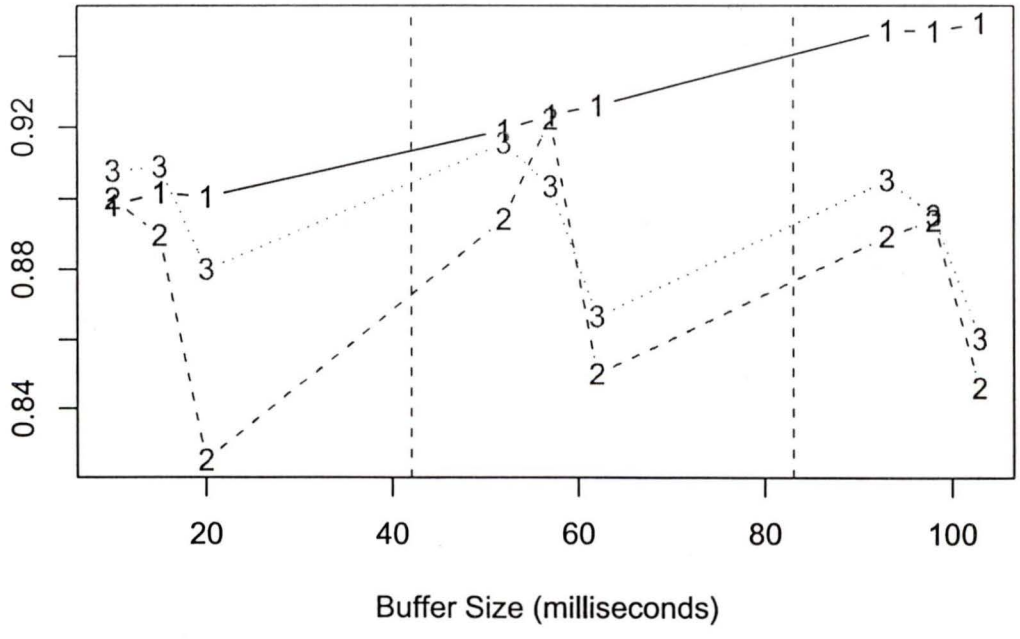

Figure 5.17: R/S Hurst parameter estimate of the Starwars movie trace at different client buffer sizes.

Table 5.4: Burstiness quantification estimates of the client buffer occupancy of the MPEG movie trace.

Channel Type	Buffer Size	Variance-Time	$\frac{R(n)}{S(n)}$	$\frac{\text{Peak}}{\text{Mean}}$	Coefficient of Variation	Dropped Frames
IEEE 1394	10 μs	0.8604	0.8299	16.60	1.4600	41970
	15 μs	0.8189	0.8767	13.47	1.2840	26540
	20 μs	0.8482	0.8935	10.53	1.0850	3148
	52 μs	0.8369	0.8682	9.261	0.9612	1226
	57 μs	0.8503	0.8981	7.509	0.8124	560
	62 μs	0.8709	0.8924	6.986	0.7422	86
	93 μs	0.8567	0.8878	6.854	0.6881	20
	98 μs	0.8582	0.8999	5.951	0.6144	9
	103 μs	0.8717	0.9034	5.514	0.5769	3
TCP/IP	10 μs	0.8801	0.9142	11.63	1.175	441
	15 μs	0.8743	0.8964	11.68	1.059	128
	20 μs	0.8732	0.8682	9.474	0.9304	1
	52 μs	0.8823	0.8521	7.146	0.7687	0
	57 μs	0.9094	0.9162	7.876	0.7608	0
	62 μs	0.9055	0.9122	6.970	0.6990	0
	93 μs	0.8805	0.8490	7.189	0.7680	0
	98 μs	0.8792	0.9061	7.611	0.7301	0
	103 μs	0.8787	0.9021	6.733	0.6722	0
UDP/IP	10 μs	0.8788	0.8902	11.46	1.181	0
	15 μs	0.8645	0.8965	10.19	0.9916	0
	20 μs	0.8677	0.8071	7.842	0.7940	0
	52 μs	0.8774	0.9184	11.36	1.169	0
	57 μs	0.8662	0.9312	10.07	0.9777	0
	62 μs	0.8683	0.8050	7.816	0.7902	0
	93 μs	0.8799	0.9227	7.162	0.7714	0
	98 μs	0.8742	0.8086	7.026	0.7040	0
	103 μs	0.8743	0.8206	5.985	0.6015	0

5.6 Trend Analysis

This section attempts to answer the following two questions. What meaningful trends are present in the client buffer occupancy time series at different levels of buffer size? And, do traditional, non-correlation, burstiness quantification methods exhibit the same trends as correlation based burstiness quantification methods?

Figure 5.11 through Figure 5.18 graphically illustrate the change in burstiness

Table 5.5: Burstiness quantification estimates of the client buffer occupancy of the Starwars movie trace.

Channel Type	Buffer Size	Variance-Time	$\frac{R(n)}{S(n)}$	Peak / Mean	Coefficient of Variation	Dropped Frames
IEEE 1394	10 μs	0.7551	0.8982	4.404	0.5449	29780
	15 μs	0.7526	0.9017	3.374	0.4115	3855
	20 μs	0.7561	0.9006	3.777	0.4163	175
	52 μs	0.7570	0.9193	3.235	0.3906	5
	57 μs	0.7554	0.9234	2.654	0.2877	5
	62 μs	0.7563	0.9262	3.728	0.2954	5
	93 μs	0.9095	0.9473	3.359	0.3465	4
	98 μs	0.9089	0.9471	2.885	0.2879	4
	103 μs	0.9089	0.9494	2.826	0.2853	4
TCP/IP	10 μs	0.7594	0.9001	3.942	0.2493	1
	15 μs	0.7400	0.8896	4.281	0.4263	1
	20 μs	0.7247	0.8253	3.466	0.3789	1
	52 μs	0.7502	0.8942	3.440	0.2382	0
	57 μs	0.7514	0.9224	3.612	0.3067	0
	62 μs	0.7433	0.8500	3.203	0.2927	0
	93 μs	0.7595	0.8894	2.953	0.2272	0
	98 μs	0.7525	0.8939	3.675	0.3032	0
	103 μs	0.7430	0.8460	3.233	0.2951	0
UDP/IP	10 μs	0.7347	0.9078	4.278	0.4226	3
	15 μs	0.7380	0.9089	3.098	0.3114	3
	20 μs	0.7549	0.8800	2.821	0.2241	3
	52 μs	0.7418	0.9156	3.588	0.3036	2
	57 μs	0.7431	0.9034	2.958	0.2565	2
	62 μs	0.7542	0.8665	2.816	0.2234	2
	93 μs	0.7468	0.9052	3.407	0.2606	1
	98 μs	0.7474	0.8952	2.956	0.2470	1
	103 μs	0.7546	0.8603	2.812	0.2223	1

as the client buffer size increases from 10 milliseconds to just over 100 milliseconds for the four burstiness quantification methods. Each plot shows one measure of burstiness for the three channel types. The labels 1, 2, and 3 represent the isochronous IEEE 1394, the TCP/IP, and the UDP/IP channels respectively. The dashed vertical lines demark frame period boundaries. Table 5.4 and Table 5.5 contain the numerical values of client buffer occupancy burstiness presented in the burstiness trend figures.

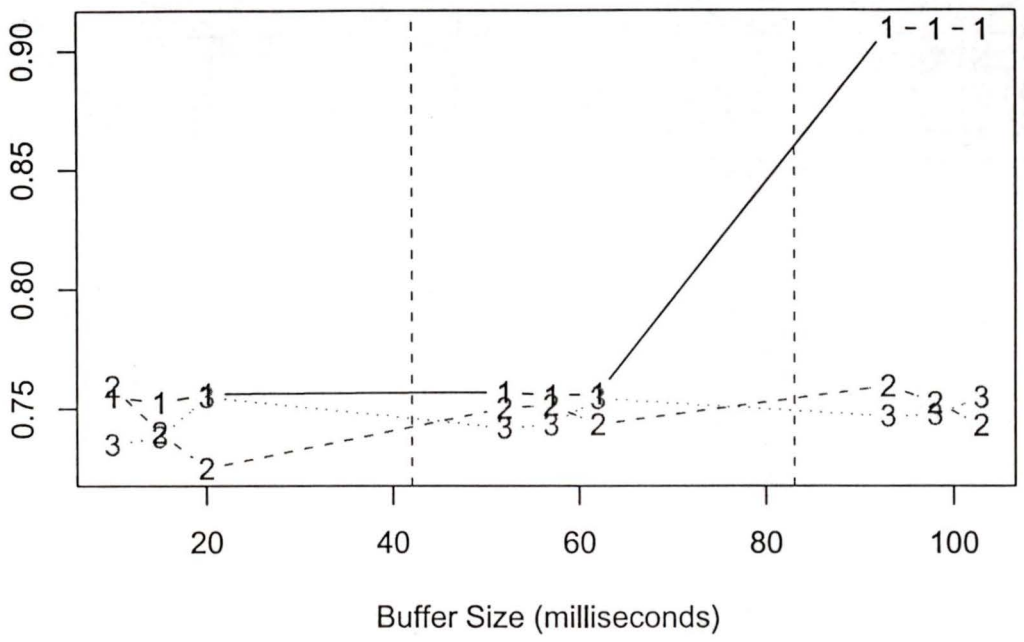

Figure 5.18: Variance-time Hurst parameter estimate of the Starwars movie trace at different client buffer sizes.

The variance-time statistic from the Starwars movie trace does not produce reliable estimates of the Hurst parameter. Figure 5.18 shows the variance-time statistic not changing as the buffer size increases. A slowly converging variance-time statistic and problems generating more than a few variance-time statistic estimates for large blocks causes the unreliable Hurst parameter estimates. Section 5.2.4.1 on page 109 describes the problem of estimating the Hurst parameter from a slowly converging time series with the variance-time statistic. The results of the variance-time statistic for the Starwars movie are unreliable and are not included in further discussion.

The Hurst parameter estimates from the R/S pox plot of the isochronous IEEE 1394 channel for the Starwars movie in Figure 5.17 do not exhibit the trends present in the other plots and channels. I consider this data series an exception and do not include it in the trend analysis. There is no clear indication why the isochronous IEEE 1394 channel responds so much differently than the other channels for the Starwars movie. The lack of reliable information from the variance-time statistic for

the Starwars movie further compounds the difficulty of investigation.

Direct numerical comparison of the estimates from the four burstiness quantification methods is inappropriate. These four methods produce numerical values, but they do not produce accurate confidence intervals. It is possible to produce confidence intervals for the coefficient of variation, but the method to calculate confidence interval is based on the assumption of normality. Calculating confidence intervals for either movie traces or any of the client buffer occupancy time series, none of which have normal statistical distributions, would produce inaccurate and misleading results. Figure 2.4 on page 29 illustrates how violating the assumption of normality results in extremely inaccurate confidence intervals. The Hurst parameter estimates from the R/S and variance-time methods do not produce confidence intervals.

5.6.1 Trends

All trend figures show periodic patterns repeating within frame boundaries. All burstiness quantification methods exhibit more variance within frame periods than between frame periods. An *intra-frame pattern* is a periodic pattern repeating within every frame period, visible in Figure 5.11 through Figure 5.17.

The UDP/IP has the densest movie to client lag, Figure 5.10 on page 131, and the most consistent intra-frame patterns. Conversely, the isochronous IEEE 1394's movie to client lag density is spread over the greater range, and its intra-frame patterns are often the least consistent.

The Hurst parameter estimates do not significantly change as the buffer size increases by multiples of a frame period. Figure 5.15, Figure 5.16, and Figure 5.17 illustrate the relatively small changes in Hurst parameter estimates across adjacent frame periods. The strong periodic intra-frame patterns in Hurst parameter estimates are a consequence of the scale invariant nature of all self-similar data.

The aggregation of a time series is a scale multiplication of the time series, which the correlation structure of a self-similar time series is resilient to. The exception to Hurst parameter stability is the R/S Hurst parameter estimates of the isochronous

IEEE 1394 for the Starwars movie trace. Generally, the correlation based burstiness quantification methods are stable between frame periods.

The Hurst parameter estimates of client buffer occupancy oscillate around the Hurst parameter estimates of their transformed movie traces. The inter-frame patterns of the coefficient of variation and the peak to mean ratio differ between the two movie traces.

The coefficient of variation and the peak to mean ratio burstiness quantification methods significantly decrease as the buffer size increases by multiples of a frame period. These methods have periodic intra-frame patterns, but the overall trend is to decrease as buffer size increases. The non-correlation burstiness quantification methods for the MPEG movie exhibit a stronger decreasing trend than the Starwars movie.

The non-correlation burstiness quantification estimates of client buffer occupancy of the MPEG movie are consistently lower than the burstiness quantification estimates of the transformed MPEG movie trace. The opposite is true for the Starwars movie, where the client buffer occupancy burstiness quantification estimates are consistently higher than the estimates of the transformed Starwars movie trace.

5.6.2 Comparing Trends

The correlation and non-correlation burstiness quantification methods have different trends overall and when compared between frame periods. All methods show strong periodic intra-frame patterns, but the correlation method trends are generally stable as the buffer size increases in frame period increments and the non-correlation method estimates generally lower as the buffer size increases.

The intra-frame patterns of channel burstiness differ for the four burstiness quantification methods. Some burstiness quantification trends decrease, some increase, and some oscillate within frame boundaries.

The coefficient of variation and the peak to mean ratios for each channel type and each movie, Figure 5.11 through Figure 5.14, have similar periodic intra-frame

patterns, but their patterns differ between the two movies. The intra-frame patterns of the MPEG movie are similar for both non-correlation methods and the intra-frame patterns for the Starwars movie are also similar for both non-correlation methods. However, the intra-frame patterns for the two movie traces are not the same. This suggests that for this data set, despite their different views of burstiness, the coefficient of variation and the peak to mean ratio measure the same characteristic of the time series.

The unreliable results of the Starwars movie trace variance-time plot and the Starwars movie trace R/S pox plot of the isochronous IEEE 1394 makes comparison of the two Hurst parameter estimation methods impossible from these time series.

When the burstiness quantification values from the transformed movie traces in Table 5.2 on page 123 and the client buffer occupancy burstiness quantification values are compared, clear trends appear in individual movie traces, but there are no trends common to all both movie traces. Both Hurst parameter estimates of client buffer occupancy oscillate around the Hurst parameter estimates their transformed movies trace. On the other hand, the trends of coefficient of variation and the peak to mean ratio differ for the two movie traces.

Client input buffer sizes that are multiples of a frame period do little to change Hurst parameter estimates. If the goal of selecting a buffer size is to reduce burstiness, the results from these experiments suggest there is little benefit from increasing buffer size by frame period multiples. The non-correlation burstiness quantification methods suggest that an increase in buffer size is almost always accompanied by a decrease in burstiness.

One final observation not directly related to burstiness, but of importance to streaming temporal data. The frame loss rate for client buffer sizes smaller than one frame period, especially for isochronous IEEE 1394, is unacceptably high for many applications. This requirement is not excessively restrictive because the inter-frame compression technique of the MPEG standards requires several future frames buffered in the decoder anyway.

The four burstiness quantification methods view burstiness in very different ways. The experiment results indicate that a single burstiness quantification method cannot capture the full property of burstiness exhibited in the experimental data. Combining the results from several methods creates the best burstiness quantification information. The individual methods should be weighted to suite the investigation and the underlying time series.

5.7 Conclusion

This chapter explores the effect of client buffer size has on the burstiness of client buffer occupancy. The client server framework transmits simulated, long-running, self-similar, variable bit rate video and records timestamps for each frame. The client server framework records the movie to client lag time series from the MPEG and Starwars movie traces streaming across the isochronous IEEE 1394, the TCP/IP, and the UDP/IP channels. The R statistical environment simulates client buffer occupancy at nine different buffer sizes.

This chapter includes a survey of the Hurst effect and four Hurst parameter estimation methods. The Hurst effect is a hyperbolically or slower decaying auto-correlation function exhibited by all self-similar time series. The auto-correlation function, the variance-time statistic, and the periodogram Hurst parameter estimation methods examine the slowing decaying variance of a self-similar time series in the time and frequency domains. The R/S statistic is a popular heuristic Hurst parameter estimation method.

Two Hurst parameter methods and two non-correlation methods quantify the burstiness of the client buffer occupancy time. The variance-time statistic and the R/S pox plot Hurst parameter estimation methods are resilient to short-range dependant, strong, periodic correlations. The peak to mean ratio and the coefficient of variation burstiness quantification methods are resilient to changing mean at different levels of the experiments.

Movie to server lag is the dominant component of movie to client lag. Movie to

sever lag is not channel induced, but an artifact of the runtime environment. Chapter 4 examines server to client lag.

The movie experiments demonstrate that buffer size is crucial affects the burstiness of the client's input buffer occupancy. The Hurst parameter estimation results suggest that increasing buffer size by multiples of a frame period does not reduce the Hurst parameter estimates. The non-correlation burstiness quantification results, however, suggest that all increases in client buffer size result in a decrease in client buffer burstiness.

6. Conclusion

This thesis examines the performance of client server architecture streaming temporal media. Several performance criteria evaluate the isochronous IEEE 1394, the TCP/IP, and the UDP/IP channels performance under several buffer sizes and frame sizes. The key finding of this research is that even the most performance oriented application development cannot overcome improper choices of application frame size and protocol buffer size. Poor buffer size and frame size choices can decrease channel performance by more than an order of magnitude.

The performance of the runtime environment and the two software projects I created to support the generation, transmission, and collection of client server traffic, is critical to channel performance. Great effort went into creating robust and channel performance oriented software to perform the research experiments.

The two software projects I created for this research, the client server framework and the IEEE 1394 Java interface, are reusable software components suitable for future research. The client server framework has proven to be an effective tool for the analysis of channel performance by myself the PANDA research group. The IEEE 1394 Java interface project, due to the evolution of the underlying IEEE 1394 for Linux device driver, is incompatible with the current version of IEEE 1394 for Linux. The evolution of the IEEE 1394 for Linux device driver, while breaking the current implementation of `j1394` does provide more features, higher quality, and higher performance. The amount of work to bring `j1394` into compliance with the current IEEE 1394 for Linux version is very small and worth the effort.

Chapter 4 examines the performance of the three channels under maximum utilization conditions with several frame sizes and protocol buffer sizes. Key findings relating frame size and buffer size to channel performance follow. Decreasing buffering generally increases inter-frame delay and decreases transmit lag. Small frames generally have poor transmit lag when compared to large frames. The default size of

protocol buffers is too large for most applications for optimum channel performance.

Chapter 5 examines the burstiness of the client's input buffer using two self-similar movie traces and several buffer sizes. Key findings relating buffer size to client buffer burstiness follow. The Hurst parameter estimation results suggest that increasing buffer size by multiples of a frame period does not reduce burstiness. The non-correlation burstiness quantification results, however, suggest that all increases in client buffer size result in a decrease in client buffer burstiness.

The main concern of the Java platform's suitability as a streaming temporal media platform is the perceived overhead of garbage collection. The findings of this research indicate that time dedicated to garbage collection is small and distributed throughout application execution time to minimize each garbage collection event. Garbage collection does not appear to be a significant factor for handling streaming temporal media on the Java platform.

Large processing interruptions exist in all channel types and frame sizes. The interrupts are most likely caused by the runtime environment preempting the client server framework to schedule other processes. These interruptions are the largest contributor to transmit lag, but are an artifact of the runtime environment, not the channel or application software.

Tune channel parameters and application protocols should not be neglected. Despite the best intentions of a developer to produce a high performance application, an application cannot overcome the poor performance caused improper buffer sizes. A buffer large enough to contain several (five to ten) frames is sufficient to keep inter-frame delay low and significantly reduce transmit lag.

The most pervasive finding of this research is buffer size is absolutely critical to channel performance. Buffer sizes that are either too large or too small buffers can cause performance to degrade by an order of magnitude or more. The maximum utilization experiments demonstrate that the choice of buffer size is crucial to minimizing transmit lag and inter-frame delay. The movie experiments demonstrate that the choice of buffer size is crucial to the burstiness of the client's input buffer

occupancy.

This research is the beginning point for understanding the performance characteristics of the Java platform for streaming media. The remainder of this chapter presents several possible avenues of future research.

6.1 Improved Runtime Environment

The Java platform and the Linux kernel are quickly evolving computing platforms. The latest Java platform, Java 2 version 1.4, includes improvements to the Java Virtual Machine and Java I/O routines that may affect the experiment results. The development tree of the Linux kernel now includes a low-latency patch that substantially improves the responsiveness of the kernel. I believe the maximum utilization experiments would benefit the most from the improved Java platform, and the movie experiments would benefit the most from improved responsiveness from the kernel. Investigation of the improved Java Virtual Machine and I/O libraries combined with the increased responsiveness of the kernel would provide performance results achievable in widely available runtime environment in the near term future.

6.2 Improved IEEE 1394 Device Driver

The IEEE 1394 for Linux device driver is constantly evolving and improving. The version of device driver used for this research is considerably out of date, even during the early stages of this research. Considerable improvements to the performance of the device driver had to be ignored in order to retain an unaltered runtime environment. Replacing the IEEE for Linux device driver with a current version and repeating all the experiments would certainly improve the observed performance of the isochronous IEEE 1394. Additionally, implementing *packet-per-buffer* [65] functionality in the device driver would reduce the transmit lag significantly.

6.3 Heterogeneous Computing Performance Evaluation

The Java platform provides a homogenous computing platform in a heterogeneous environment. All the components of the experiments could be conducted in any of a number of different runtime environments, except for the `j1394` package. Extending the `j1394` package to support other runtime environments would provide an analysis of the Java platform for streaming media in a heterogeneous environment.

6.4 Increased Resolution of Factor Levels

The maximum utilization and the movie experiments use course-grained factor levels. The advantage of this is a large parameter space can be covered in a reasonable number of tests. A different approach to take is to narrow the parameter space to one or two factors and increase the resolution of the factor levels.

Reducing the number of factors, but increasing the levels of the remaining factors would provide a different perspective of channel performance. The experiments of this research provide a broad view of channel performance, suitable only for general conclusions. To make specific recommendations about buffer sizes and frame sizes for optimum channel performance, finer grained factors are required. For example, to determine the best frame and buffer size of a UDP/IP application for minimum transmit lag, an experiment with 64 factor levels for both frame size and buffer size factors could provide specific channel configuration results.

The maximum experiments only create buffer size two data points for each level of packet size and channel type. Although altering buffer size has dramatic performance changes, the data does not provide enough information to make strong conclusions. Providing more information about performance beyond the buffer size two points would support stronger recommendations and conclusions about ideal buffer size for maximum utilization channel performance.

6.5 Garbage Collection Evaluation

The maximum channel utilization experiments investigate the effect garbage collection has on the client server framework. These results indicate that garbage collection is a relatively minor contributor to application interruption and should not be an impediment to streaming temporal media. What is left to investigate is the effect Java Virtual Machine parameters have on garbage collection for streaming temporal media applications.

6.6 Examiniation of Nagle's Algorithm

Nagle's algorithm may have interesting affects on the TCP/IP's performance. The possibility that Nagle's algorithm affects the performance of the TCP/IP was in introduced in Section 5.4.1, but not explored. Curiously, a technique designed to improve the performance of the TCP/IP may reduce its performance in some cases. A future examination at channel performance should look at when the use of Nagle's algorithm is inappropiate and suggest guidelines for its proper use.

6.7 Validation of Results

The experiments capture performance information from a Java client server application framework simulating streaming temporal media. I believe the performance information was collected in such a way that it accurately represents Java streaming temporal media performance, but this is only a conjecture. To validate the results, performance information from a real Java client server application framework streaming real temporal media is needed for comparison.

6.8 Simulation of Results

The experiments capture the performance information from a real set of Java applications and runtime environments. While using real applications and hardware provides high quality results, the results are difficult to scale to more than a few hosts.

The results from this research could be used to build in a simulation environment capable of simulating a wider range of environments. The simulation of the results could produce a performance analysis for much larger and complex communications networks.

Bibliography

[1] Murad S. Taqqu, Walter Willinger, and Robert Sherman. Proof of a fundamental result in self-similar traffic modeling. *ACMCCR: Computer Communication Review*, 27, 1997.

[2] IEEE, Piscataway, New Jersey. *Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*, 2000 edition, 2000.

[3] Charles E. Spurgeon. *Ethernet: The Definitive Guide.* O'Reilly and Associates, Cambridge, Massachusetts, 2000.

[4] Robert M. Metcalfe and David R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the Association of Computing Machinery*, 19(7):395–404, 1976.

[5] David Plummer. *RFC 826: An Ethernet Address Resolution Protocol.* Massachusetts Institute of Technology, 1982.

[6] IEEE. *IEEE Std. 1394-1995*, 1995.

[7] Don Anderson. *FireWire System Architecture.* Addison Wesley, Reading, Massachusetts, 2^{nd} edition, 1999.

[8] IEEE. *IEEE Std. 1394a-2000*, 2000.

[9] University of Southern California. *RFC 793: Transmission Control Protocol*, 1981.

[10] J. Postel. *RFC 768: User Datagram Protocol.* University of Southern California, 1980.

[11] ISO. *ISO Basic Reference Model for Open Systems Interconnection (ISO/OSI)*, 2000.

[12] University of Southern California. *RFC 791: Internet Protocol*, 1981.

[13] Simon Gibbs and Dionysios Tsichritzis. *Multimedia Programming: Objects, Environments and Frameworks.* Addison Wesley, Reading, Massachusetts, 1995.

[14] Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s, 1999.

[15] Generic coding of moving pictures and associated audio information, 2000.

[16] ISO. *Coding of Moving Pictures and Audio: MPEG-4 Video*, 2nd edition, 2000.

[17] Michael D. Adams. The JPEG-2000 Still Image Compression Standard. Technical report, University of British Columbia, June 2000.

[18] H. E. Hurst. Long-term storage capacity of reservoirs. *Transactions of the American Society of Civil Engineers*, 116:770–808, 1951.

[19] Will E. Leland, Murad S. Taqq, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of Ethernet traffic. In Deepinder P. Sidhu, editor, *ACM SIGCOMM*, volume 23, pages 183–193, San Francisco, California, 1993.

[20] Mark E. Crovella and Azer Bestavros. Explaining world wide web traffic self-similarity. Technical Report 1995-015, Boston University, October 1995.

[21] Mark E. Crovella and Azer Bestavros. Self-similarity in World Wide Web traffic: evidence and possible causes. *IEEE /ACM Transactions on Networking*, 5(6):835–846, 1997.

[22] Vern Paxson and Sally Floyd. Wide-area traffic: The failure of Poisson modeling. *IEEE /ACM Transactions on Networking*, 3(3):226–244, 1995.

[23] Mark Garrett. *Contributions Toward Real-Time Services on Packet Switched Networks*. PhD thesis, Columbia University, New York, New York, May 1993.

[24] Mark Garrett and William Willinger. Analysis, modeling and generation of self-similar VBR video traffic. In *ACM SIGCOMM*, pages 269–280, London, UK, 1994.

[25] Changcheng Huang, Michael Devetsikiotis, Ioannis Lambadaris, and A. Roger Kaye. Modeling and simulation of self-similar variable bit rate compressed video: A unified approach. In *ACM SIGCOMM*, volume 25, pages 114–125, August 1995.

[26] Bong K. Ryu and Anwar Elwalid. The importance of long-range dependence of VBR video traffic in ATM traffic engineering: Myths and realities. In *ACM SIGCOMM*, pages 3–14, 1996.

[27] Walter Willinger, Murad S. Taqqu, Robert Sherman, and Daniel V. Wilson. Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level. *IEEE /ACM Transactions on Networking*, 5(1):71–86, April 1997.

[28] J. Beran, R. Sherman, M. Taqqu, and W. Willinger. Long-range dependence in variable-bit-rate video traffic. *IEEE Transactions on Communications*, 43:1566–1579, 1995.

[29] J. Postel and J. Reynolds. *RFC 854: Telnet Protocol Specification.* Network Working Group, 1983.

[30] J. Postel and J. Reynolds. *RFC 959: File Transfer Protocol (FTP).* Network Working Group, 1985.

[31] A. Karasaridis and D. Hatzinakos. Broadband heavy-traffic modeling using stable self-similar processes. In *Canadian Conference on Broadband Research*, pages 157–168, Ottawa, Ontario, Canada, 1998.

[32] Ilkka Norros. A storage model with self-similar input. *Queueing Systems and Their Applications*, 16:387–396, 1994.

[33] Boris Tsybakov and Nicolas D. Georganas. Overflow probability in an ATM queue with self-similar traffic. In *ICC'97*, 1997.

[34] Craig Partridge. The end of simple protocols. *IEEE Network*, September 1993.

[35] Walter Willinger, Murad S. Taqqu, and Ashok Erramilli. *A Bibliographical Guide to Self-Similar Traffic and Performance Modeling for Modern High-Speed Networks.* Oxford University Press, Oxford, 1996.

[36] Hae-Duck J. Jeong, Don McNickle, and Krzysztof Pawlikowski. Fast self-similar teletraffic generation based on FGN and wavelets. In *International Conference on Networks.* IEEE, 1999.

[37] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface.* Morgan Kaufman Publishers, San Francisco, California, 2^{nd} edition, 1997.

[38] Finite State Machine Labs Inc. RTLinux version 3.0, 2001. http://www.rtlinux.org.

[39] IEEE 1394 for Linux, 2001. http://sourceforge.net/projects/linux1394/.

[40] D. L. Mills. *RFC 2030 Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI*, 1996.

[41] GNU general public license (GPL) - version 2, June 1991. Free Software Foundation, Cambridge, Massachusetts.

[42] Andrew S. Tanenbaum. *Modern Operating Systems.* Prentice Hall, Upper Saddle River, New Jersey, 2^{nd} edition, 2001.

[43] Linus Torvalds. Linux kernel version 2.2.16, 2000. http://kernel.org/.

[44] Alessandro Rubini. *Linux Device Drivers.* O'Reilly and Associates, Cambridge, Massachusetts, 1998.

[45] Matt Welsh, Matthias Kalle Daleimer, and Lar Kaufman. *Running Linux.* O'Reilly and Associates, Cambridge, Massachusetts, 3^{rd} edition, 1999.

[46] HotSpot Technical Team. The Java HotSpot Virtual Machine. Technical white paper, Sun Microsystems, May 2001.

[47] Java 2 Standard Edition (J2SE) Runtime Environment, 2001. Sun Microsystems, Mountain View, California.

[48] Java 2 Standard Edition (J2SE) build 1.3.0_01 mixed mode, 2001. Sun Microsystems, Mountain View, California.

[49] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification.* Addison Wesley, Reading, Massachusetts, 2^{nd} edition, 2000.

[50] Sun Microsystems, Mountain View, California. *Java Media Framework API Guide*, November 1999.

[51] Graham Hamilton. *JavaBeans API specification.* Sun Microsytems, Mountain View, California, 1997.

[52] Sun Microsystems, Mountain View, California. *Java Native Interface Specification*, 1997.

[53] Martin Flower and Kendall Scott. *UML Distilled: a brief guide to the standard object modeling language.* Addison Wesley Longman, Reading, Massachusetts, 2^{nd} edition, 2000.

[54] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0. Recommendation, W3C, October 2000.

[55] Rajiv Mordani, James Duncan Davidson, and Scott Boag. *Java API for XML Processing 1.1.* Sun Microsystems, Mountain View, California, February 2001.

[56] Forte Community Edition, 2001. Sun Microsystems, Mountain View, California.

[57] Source Navigator Team. Source Navigator IDE version 5.0, 2001. Red Hat, Durham, North Carolina.

[58] R Core Team. *R Statistical Language and Environment version 1.3.1*, 2001. http://www.r-project.org.

[59] R Development Core Team. *R Language Definition version 1.3.1*, 2001. http://cran.r-project.org.

[60] Martin W. Murhammer, Orcun Atakan, Stefan Bretz, Larry R. Pugh, Kazunari Suzuki, and David H. Wood. *TCP/IP Tutorial and Technical Overview*, 6^{th} edition, 1998.

[61] John H Maindonald. *Using R for Data Analysis and Graphics: An Introduction.* Statistical Consulting Unit of the Graduate School, Australian National University, 2001.

[62] Boris Tsybakov and Nicolas D. Georganas. Self-similar processes in communications networks. *IEEETIT: IEEE Transactions on Information Theory*, 44:1713–1725, 1998.

[63] Will E. Leland, Murad S. Taqqu, Walter W. Willinger, and Daniel V. Wilson. On the self-similar nature of Ethernet traffic (extended version). In *IEEE / ACM Transactions on Networking*, volume 2, pages 1–15, February 1994.

[64] Stephen Bates and Steve McLaughlin. Testing the Gaussian assumption for self-similar teletraffic models. In *Signal Processing Workshop on Higher Order Statistics*, pages 444–447. IEEE, 1997.

[65] Promoters of the 1394 Open HCI. *1394 Open Host Controller Interface Specification*, October 1997.

APPENDIX A
Acronyms

I have avoided the use of acronyms in this thesis wherever it is sensible. I only use acronyms in situations where the acronym is more comprehensible than the full text it represents. These cases include acronyms that are much better known than their full text representation, acronyms that aren't really acronyms, full text representations that are easily confused with generic terms, and acronyms not based on English phrases.

ACF Auto-Correlation function

AOV Analysis Of Varience

API Application Program Interface

ARP Address Resolution Protocol

BPS Bytes Per Second

bps Bits Per Second

CBR Constant Bit Rate

CCITT International Telegraph and Telephone Consultative Committee

CPU Central Processing Unit

CSMA/CD Carrier Sense Multiple Access / Collision Detect

DV Digital Video

FTP File Transfer Protocol

GOP Group Of Pictures

GUI Graphical User Interface

HAVi Home Audio Video Interoperability

HDTV High Definition TeleVision

HTTP Hypertext Transfer Protocol

I/O Input / Output

IEEE Institute of Electrical and Electronics Engineers

IPG Inter-Packet Gap

IP Internet Protocol

IRM Isochronous Resource Manager

ISO International Organization for Standardization

ITU International Telecommunication Union

J2SE Java 2 Platform, Standard Edition

JNI Java Native Interface

JPEG Joint Photographic Experts Group

JVM Java Virtual Machine

LAN Local Area Network

LCM Least Common Multiple

LRD Long-Range Dependent

M-JPEG Motion-JPEG (Joint Photographic Experts Group)

MAC Media Access Control

MAU Minimum Addressable Unit

MPEG Moving Picture Expert Group

MTU Minimum Transfer Unit

NTP Network Time Protocol

NTSC National Television Standards Committee

OSI Open System Interconnect

PCI Peripheral Component Interconnect

PC Personal Computer

PHY Physical Layer

PSD Power Spectrum Density

RAM Random Access Memory

RE Runtime Environment

RFC Request For Comment

SCSI Small Computer Systems Interface

SRD Short-Range Dependent

TCP/IP Transmission Control Protocol / Internet Protocol

UDP/IP User Datagram Protocol / Internet Protocol

UML Unified Modeling Language

VBR Variable Bit Rate

VM Virtual Machine

W3C World Wide Web Consortium

WAN Wide Area Network

WWW World Wide Web

XML eXtensible Markup Language

Vita

Surname: Norris Given Names: Robert Christopher

Place of Birth: Maria, Québec, Canada

Educational Institutions Attended:

University of Victoria 1995-2002

Educational Institutions Attended:

Malaspina University College 1992-1995

Degrees Awarded:

B.Sc. University of Victoria 1999

Publications:

R. C. Norris and D. M. Miller. Comparing the Performance of IP over Ethernet and IEEE 1394 on a Java Platform, In *2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM 2001)*, Victoria, British Columbia, Canada, August 2001.

R. Chris Norris and D. Michael Miller. Partitioning Method for Estimating Self-Similarity with the Variance-Time Statistic, Accepted for publication in *2002 IEEE International Conference on Communications in Computing (CIC'02)*, Las Vegas, Nevada, U.S.A., June 2002.

Partial Copyright License

I hereby grand the right to lend my thesis to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis:

Performance Analysis of Java as a Streaming Digital Media Platform

Author: 

Robert Christopher Norris

April 16, 2002