11-14-2003

Characterization and Generation of Streaming Video Traces

John N. Shahbazian
University of South Florida

Characterization and Generation of Streaming Video Traces

by

John N. Shahbazian

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Kenneth J. Christensen, Ph.D.
Miguel Labrador, Ph.D.
Dewey Rundus, Ph.D.

Date of Approval:
November 14, 2003

Keywords: networks, multimedia, modeling, simulation, software testing

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Kenneth J. Christensen for his guidance, assistance, and support. Without his patience and dedication, this work would not be possible. Many hours have been spent on this thesis, and it has come a long way. I would also like to thank my committee: Dr. Dewey Rundus and Dr. Miguel Labrador. Finally, I would like to thank Dr. Abraham Kandel for his support and generosity during my time as a research assistant.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHARACTERIZATION AND GENERATION OF STREAMING VIDEO TRACES

John N. Shahbazian

ABSTRACT

This thesis describes two methods collectively called Time Series Generation (TSG) that can be used to generate time series inputs modeling packet loss to test IP-based streaming video software. The TSG methods create packet loss models that recreate the mean, variance, and autocorrelation signatures of an actual trace. The synthetic packet loss traces can have their inherent statistics altered, thus allowing for thorough testing of video software in ways that could not be done on actual networks. The two methods comprising TSG, which are individually called the primary and secondary method, use the principle of iterated uniformity to create a time series that attempts to match mean, variance, and autocorrelation. The two methods differ in their approach to generating autocorrelation. This leads to trade-offs between the two. The TSG methods are embodied in a software program called TSGen. An evaluation of TSGen is conducted, including a comparison with the well-known Autoregressive-To-Anything Generation algorithm (ARTAGEN) method and tool. The details of capturing packets and parsing video frame counts from packet streams are explained and demonstrated. Sixteen video stream traces were collected from a variety of sources and used to evaluate TSGen. Synthetic traces are generated for the sixteen original traces and both their summary

statistics and autocorrelation signatures are compared against the originals. One of the sixteen traces is also compared against a synthetic trace generated using the ARTAGEN tool. Twelve out of the sixteen synthetic traces when compared to the actual traces had Least Square Error (LSE) values under 0.1, three were under 0.4, and the remaining one was under 1.1. Nine synthetic traces had their percent error differences between the mean and variance of the synthetic and actual traces below 5%, one was below 7%, four were under 18%, and the two remaining were at 41%. TSGen is able to effectively model autocorrelation, mean, and variance. Additional intangible benefits of TSG include adjustable run time for the matching process, with longer run time equating to better accuracy, and a simple theoretical model that was easily implemented.

CHAPTER 1

INTRODUCTION

1.1 Background

Streaming video is a method of transferring video over a network that allows a user to watch the video in real-time without saving the video data on the user's computer. As bandwidth and computer processing power increase, the popularity and feasibility of streaming also increase. However, a significant challenge lies in testing the software that encodes, transfers, decodes, and plays streaming video. To fully test this software, the behaviors of the network transferring the video need to be considered. This includes packet loss behaviors typical of small and large networks. Packet loss is a primary determinant of video quality at a user's computer.

Applications of streaming video include watching television over a network, watching video clips stored on a server, video conferencing, and video surveillance. The bandwidth requirement of streaming video can vary from under 0.4 Mbits/s to over 10 Mbits/s, with the higher date rate being equal to higher quality, such as sharper resolution, bigger images, more colors, better audio, etc. The methods described in this thesis work with MPEG-1 video, which encodes at 1.5 Mbits/s. The audio portion of streaming video is not examined. The methods presented can also be applied to MPEG-2 and MPEG-4 streaming video.

The processor and bandwidth requirements of a server for streaming video are greater than those of the client because the server may have to stream to multiple clients. For applications such as video conferencing, the 'server' may actually be a computer that has the processing power and bandwidth restrictions of a 'client,' thereby restricting the frame rate and resolution of the video. The speed of the connection between client and server must also be considered, as well as the load placed on the server by clients. If the server or connection is overloaded, then excessive packet delay or loss will occur.

The Internet has best-effort delivery; this means that while the hardware and software that make up the Internet will try to get packets from a server to a client, there is no guarantee that this will happen unless an application or higher layer protocol is specifically made to do this on its own. Video streaming applications are not made to guarantee such a delivery because doing so would interrupt the stream while the client waited for retransmission of lost packets. Figure 1 shows packet loss in the video streaming process across the Internet that could be caused by router congestion. Lost packets will cause errors in videos, ranging from minor artifacts that

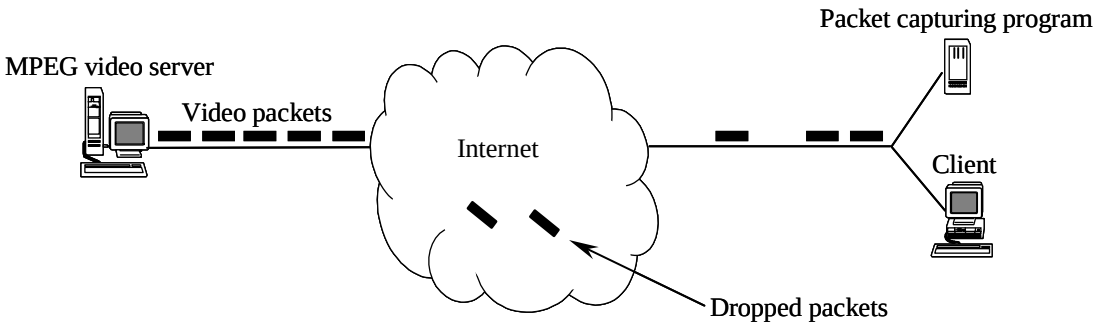

Figure 1. Packet Loss Over the Internet

are not noticeable by the human eye to multiple lost frames that cause the viewer to miss portions of the video. Some applications may have mechanisms to try to lower the effect of errors on the video, but for this thesis, those types of applications will not be examined.

1.2 Motivation

Video streaming software must have error-handling capabilities because of the best-effort nature of the Internet. An application may experience severe packet loss that causes degraded quality of video for several seconds, minutes, or longer. Testing such error-handling capabilities is an important aspect of streaming video software development, especially when the software is designed to be scalable [3]. In order to test such software, being able to control the network is key. If the network cannot be controlled, then controlling the video streams is key. By controlling the video streams, errors can be introduced and controlled allowing for many possible conditions to be emulated. An important part of controlling these streams is modeling their statistics, which itself is the main contribution of this thesis.

1.3 Contributions of this Thesis

This thesis investigates new methods for packet capturing, traffic modeling, and statistical modeling. It develops and evaluates two new methods for generating synthetic time series that model the autocorrelation and summary statistics of an input time series. The main contributions of this work are:

- Developed a set of tools to capture and analyze video packet streams

- Created algorithms to count good and bad video frames for RTP and UDP based streaming video
- Associated packet loss in a video stream with user-perceivable errors in the video
- Created two algorithms called Time Series Generation (TSG) to model the mean, variance, and autocorrelation of a time series
- Developed a tool called TSGen to implement the two modeling algorithms
- Evaluated all algorithms against sixteen video streams and confirmed that they can count good and bad video frames, and model mean, variance, and autocorrelation

1.4 Organization of this Thesis

The remainder of this thesis is organized as follows:

- Chapter Two reviews basic concepts in the area of traffic measurement, MPEG video, and time series modeling. This chapter also reviews literature in traffic modeling and MPEG video modeling.
- Chapter Three describes new methods to determine the number of bad video frames from packet loss. Methods to generate synthetic time series that have the same mean, variance, and autocorrelation of an input time series are also described.
- Chapter Four presents evaluations of the methods described in chapter three showing that bad video frames are correlated to packet loss, and that the time series modeling methods are effective using sixteen MPEG-1 video stream traces.

- Chapter Five summarizes the work and describes possible directions for future research.

CHAPTER 2

LITERATURE REVIEW

The following chapter reviews basic concepts in the area of traffic measurement, MPEG video, and time series modeling. This chapter also reviews literature in traffic modeling and MPEG video modeling. The literature builds a foundation for the work of this thesis. Section 2.1 describes traffic modeling. Section 2.2 describes MPEG video measurement and modeling. Section 2.3 contains the problem statement for this thesis.

2.1 Traffic Measurement

Many software libraries exist for the purpose of capturing packets through a TCP/IP network connection. For this thesis, a library was needed to provide functions that were able to quickly and directly access the network and also work with the Microsoft Windows 2000 Operating System. The WinPcap [26] library was used because of its cost, compatibility, and features. The reason why Windows 2000 was chosen over UNIX as a platform was availability, familiarity and simplicity. However, the software developed in this thesis can easily be ported to UNIX.

Packet monitoring can easily be done on client machines, one way to do this is to use SNMP [6] and RMON [30] counters. A counter is contained inside a router that is directly on the network, and the statistics gathered by that counter are contained inside an SNMP MIB. The advantages of counting packets by this method are that the router is directly connected to the network and is dedicated to routing packets. Unfortunately, the

disadvantages are that the counter would have to be implemented specifically for monitoring video packets, and a MIB would have to be written for it. Detecting received packets in an MPEG video stream is relatively easy. However, detecting lost packets is much more difficult. Deductive analysis of the stream must be used to determine how many and which packets have been lost. In some cases where there are a large number of sequentially lost packets, determining the exact number of lost packets is impossible. Loss detection is accomplished by keeping track of the sequence number if the packets are encapsulated in RTP [27], which itself is encapsulated in UDP [24]. See Figure 2 for a diagram of an Ethernet Frame and an IP Header and see Figure 3 for a diagram of the RTP packet header. If the packets are only encapsulated in UDP, then the detection is of the same complexity. See Figure 4 for a diagram of the UDP packet header.

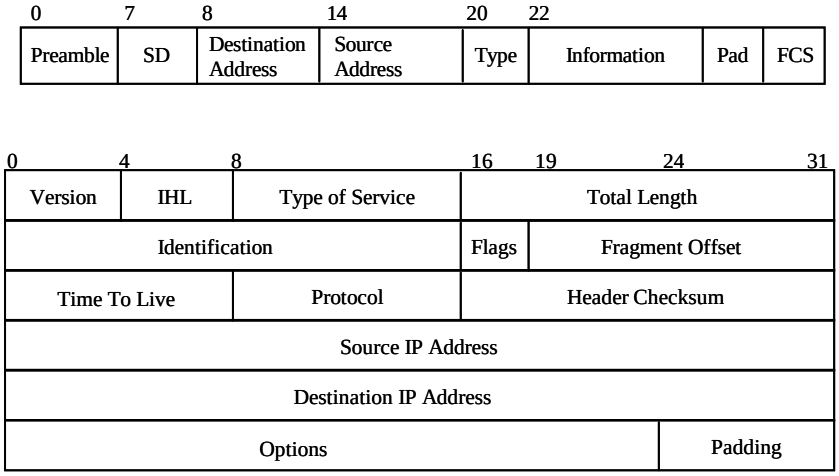

Figure 2. Ethernet Frame and IP Header

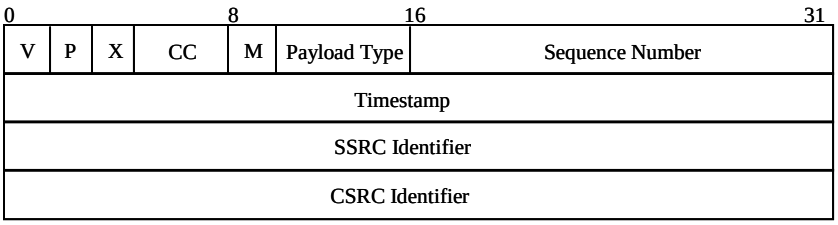

Figure 3. RTP Header

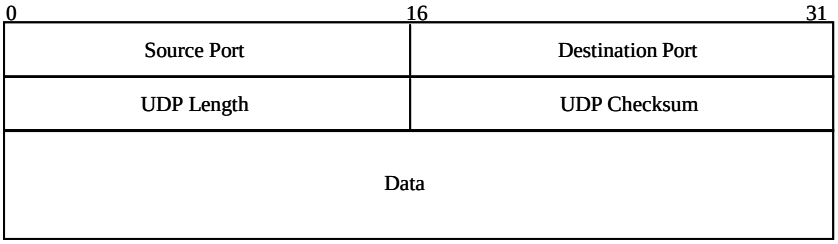

Figure 4. UDP Header

2.2 Traffic Modeling

Jain and Routhier describe modeling packet arrivals on a computer network using what they call packet trains in [16]. They also review previous models such as Poisson arrivals and compound Poisson arrivals and found that the packet arrival process is neither of those. They create a model that places packets into groups called trains. They model the time between packets, and also the time between trains, which is greater than the time between packets. Also discussed is source locality of packet arrivals. In a later traffic modeling paper, Leland et al. [18] show that Ethernet LAN traffic has self-similar properties. They use the Hurst parameter, which can be found using the rescaled adjusted range statistic (known as the R/S statistic), to define the degree of self-similarity, and they visually show modeling traffic as self-similar is much more realistic than as a Poisson process.

In [23], Paxson and Floyd study several wide area network traces and determine that user-initiated TCP session arrivals can be modeled as Poisson processes, but packet interarrivals should not be modeled as Poisson because of their burstiness. They put forth the idea that the wide area traffic is statistically self-similar and would be better modeled as such.

In [32], a method of analyzing multimedia streams and using a threshold to decide when there is too much loss for the stream to be intelligible is developed. The method will cause transmission of packets to stop when there is too much loss so as to allow for the bandwidth to be used by other applications. They authors show that the algorithm can be implemented in Weighted Fair Queue (WFQ) and Core Stateless Fair Queue (CSFQ) fair packet queueing and discarding algorithms. They also show that with the algorithm, the overall intelligibility of the multimedia stream is no different and the data transfer time for other processes is increased significantly with the bandwidth that is freed.

Raisanen and Lakaniemi [25] introduce a method for modeling and generating synthetic packet loss patterns of an audio stream. The method uses the cumulative probability distributions for inter-loss burst gap lengths and loss burst lengths to model the loss patterns and can generate a synthetic series with the same first-order statistics. The drawback to this method is that it does not capture the correlation between bursts.

In [22], Melamed describes the Transform-Expand-Sample (TES) algorithm, which models both the empirical marginal distribution and the autocorrelation of data through adjustment of parameters. He provides a geometric interpretation of TES as well as the results of observations of various TES processes, such as Uniform, Geometric, and Exponential, and how they perform under a transformation known as stitching. He also

provides a short example of modeling compressed video traffic, and a description of software that implements the algorithm. The TES method operates on the principle of iterated uniformity which involves the creation of a set of random variables that are uniformly distributed using uniform(0,1) variables combined with independent random variables through modulo-1 arithmetic, which is finding the fractional remainder after finding the sum of two values. The resulting variables are then put through a stitching function that smoothes their values while preserving their uniformity. This is done because the modulo-1 arithmetic creates situations where one value may be very close to 1 and the next value close to 0. The stitching function uses a parameter $0 \le \xi \le 1$ to transform the values using this equation:

$$S_\xi(y) = \begin{cases} y/\xi, & 0 \le y \le \xi \\ (1-y)/(1-\xi), & \xi \le y < 1 \end{cases} \quad (1)$$

This gives the values a more continuous visual appearance. The values are then transformed using the inverse of an empirical distribution. This creates the desired distribution in the resulting values. If the desired autocorrelation signature is not obtained, the independent random variables and the parameter of the stitching function can be changed. Changing those values will change the autocorrelation signature. Parameter adjustment is done manually until a desired fit is achieved, with the judging done by eye using a specific tool named TEStool that is no longer available.

In [4], Cario and Nelson describe the autoregressive to anything (ARTA) algorithm. This algorithm can model the summary statistics and autocorrelation signature of a stationary time series. A primitive search is used to adjust values in a process that is attempting to match a given set of autocorrelation values. More specifically, the ARTA algorithm begins by generating input values that are used to solve for modifiers of a

stationary Gaussian autoregressive (AR) process with a specified number of lags. Added to this process is a set of independent Normal random variables with a mean of 0 and a standard deviation that is calculated from the initial input values and the modifiers of the AR process. A set of values is then created by using the AR process as input into an inverse CDF of the desired marginal distribution. The resulting values are uniform(0,1) by the probability-integral transformation, which allows for creation of a random variable with a specific distribution by solving for a variable that has a CDF of that distribution using the inverse of the distribution. The autocorrelations are then checked, and if necessary, the input values are modified and the algorithm is repeated to achieve a better match. The authors believe it is superior to other algorithms because it can simulate more than just lag-1 autocorrelation, and in particular, better than TES because they directly change the autocorrelations of the base process to adjust those of the input process, whereas TES requires input from the user.

Another paper on ARTA, [5], explains the ARTA algorithm in greater detail and provides an efficient method for fitting ARTA processes. They also explain two pieces of software, ARTAFACTS and ARTAGEN, which use the ARTA algorithm. ARTAFACTS is a program developed in FORTRAN that implements the fitting procedure, while ARTAGEN generates values from input generated by the ARTAFACTS program.

Che and Li [7] develop algorithms that create processes that match an existing multimedia traffic processes' rate distribution and autocorrelation. They emphasize on matching complicated autocorrelation functions with a circulant modulated Poisson process (CMPP), which is a special class of the Markov-modulated Poisson process

(MMPP), and identifying parameters for this process to match the given autocorrelation. They also develop a fast search algorithm that finds the parameters for the CMPP.

Liu and Baras propose a new statistical method of modeling traffic loads in different time scales in [19]. They develop a framework for modeling scaling and multi-scale behaviors, such as long-range dependence, self-similarity, and multi-fractality, in traffic. They use only the first two moments to characterize traffic processes and also a general queuing analysis to evaluate performance and behavior.

In [14], Hajek and He study how mean and autocorrelation functions of an arrival process affect the mean queue length of a single deterministic server. They found that with two-state Markov-modulated Poisson processes and periodic-sequence modulated Poisson processes, the mean length of the queue varies greatly, but with randomly filtered white noise, the mean queue length is determined by the mean and autocorrelation functions.

2.3 MPEG Video Measurement and Modeling

The Moving Picture Experts Group (MPEG) was formed in 1988 to develop standards for video and audio compression on Digital Storage Media. Their first standard was MPEG-1 [9] and was issued in 1992. MPEG-1 was developed for a bit rate up to 1.5 Mbits/s and was followed by MPEG-2 [13] in 1994, which was designed for a bitrate between 2 and 10 Mbits/s.

The MPEG compression standards were created because uncompressed video and audio take up enough storage space that storing and transmitting multiple pieces of video

was not practical. Streaming MPEG-1 over a network requires a large amount of bandwidth at 1.5 Mbits/s, while streaming MPEG-2 requires even more at 10 Mbits/s.

MPEG compression takes advantage of temporal and spatial redundancy to reduce the size of the data, but it does not maintain the quality of the original. This is what is known as lossy compression. When the video is uncompressed at the receiver it will not look and sound exactly as it did before it was compressed. MPEG-1 and MPEG-2 compressed videos are composed of pictures (frames) that are separated into three different types: I, B, and P. I frames are intraframes that encode the current picture, while B and P frames interpolate from previous and future frames. When transmitted over an IP network, these frames are segmented into one or more IP packets. Losing a packet (and subsequently having an error in a frame) can have different effects depending on the type of frame. Losing a B or P frame may cause a quick skip in the video that might not be perceivable by the human eye. However, the loss of an I frame will cause a long skip in the video that has noticeable content loss. B and P frames occur much more frequently than I frames, which are usually transmitted every 0.5 seconds. Figure 5 shows an example of how a grouping of frames, known as a Group of Pictures (GOP), might be constructed. The encoder could place I, B, and P frames in different orders or amounts.

In order to show that packet loss is correlated to frame loss, which is then correlated to visibly noticeable errors in a video, a portion of this thesis will examine capturing video frame packets and relating loss of such packets to user-graded errors in an MPEG-1 video that is streamed over the Internet.

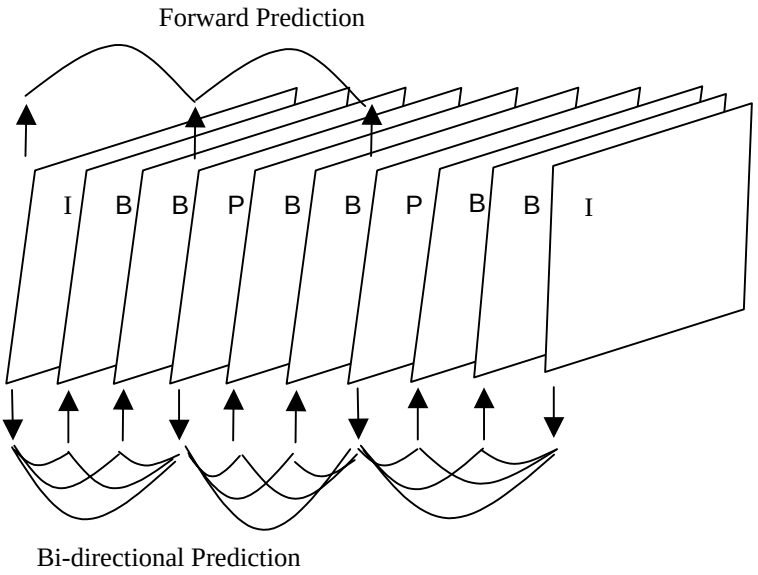

Figure 5. MPEG-1 Group of Pictures

The original MPEG-1 paper [17] by Le Gall describes the fundamentals of the MPEG video compression standard and interframe coding used in the compression algorithm. The paper also compares the quality of MPEG-1 compressed video at 1.5 Mbits/s to a VCR and finds them at about the same level.

In [11], Fitzek and Reisslein study MPEG-4 and H.263 encoded videos and provide a statistical analysis of them. An overview of the compression and encoding techniques are presented for both MPEG-4 and H.263. A detailed analysis of the video traces is provided, including information about first-order statistics and autocorrelation. They examine over ten videos that are 60 minutes long each, and they provide a new metric called rate trace that associates H.263 frame sizes with frame periods.

In [10], Dalgic and Tobagi examine using video glitch statistics as a metric for network performance when video traffic is being carried. Three quantities are presented that can characterize the video performance on a network: glitch duration, the length of

time or number of frames for which a portion of a frame is not displayed, spatial extent, the percentage of undisplayed portions in a frame, and glitch rate, the number of glitches per unit time. Several case studies are presented and it is found that the glitch rate is affected by network type, traffic load, encoder control scheme, video content, and the delay constraint. Spatial extent and duration are affected by network type, video encoding scheme, video content, and delay constraint.

Another paper that describes glitches in multimedia traffic is [3], by Zhang, Zheng, and Ngee. This paper examines network delay and jitter and its effects on several test videos using queuing analysis. In addition, they show the network conditions at the time the experiments were done and that these conditions affect delay and jitter. Their results show that network conditions such as traffic load, burstiness, and burst-length affect the probability distribution function of delay and delay jitter, and also that service quality may be influenced by a large series of clustered or dispersed packets.

In [28], Shahbazian and Christensen develop an SNMP counter that monitors a video stream and counts both good and bad frames within the stream. Two algorithms are designed to monitor RTP-based and UDP-based video packet streams. An example MIB is given showing what would be needed to implement the algorithms in an SNMP counter. The design is simulated and evaluated, using a streaming MPEG-1 video and a human subject monitoring for and grading glitches in the video. It is found that user perceivable anomalies can be measured and correlated to frame loss.

Garret and Willinger [12] analyze a 2-hour long video and model the tail behavior of the marginal distribution using “heavy-tailed” distributions such as a Pareto distribution. They also find that the autocorrelation sequence decays hyperbolically and

can be modeled using self-similar processes. However, their method does not model Short-Range Dependence (SRD) effectively.

Beran, Sherman, and Taqqu demonstrate in [2] that Long-Range Dependence (LRD) exists in Variable-Bit-Rate (VBR) video traffic by examining over 20 large sets of actual video data. Huang et al. also model the marginal distribution and both the Short-Range Dependence and Long-Range Dependence of an MPEG trace in [15]. However, they model each of the frames separately, not accounting for intra-GOP correlation and thereby reducing the effectiveness of the model.

In [31], Wrege and Liebeherr describe a method for characterizing video traffic with a number of leaky buckets. They explore tradeoffs based on the number of leaky buckets and the amount of data from the video trace, and determine how much of a video is need for characterization. They also present an algorithm to select parameters for a number of leaky buckets. Their method is then tested on 30-minute MPEG videos and found to be superior to other characterization methods.

Lombardo et al. [20] describe an algorithm for modeling MPEG video traces. This algorithm applies a method based on Fast Fourier Transform to generate self-similar traces that takes into account intra-GOP correlation. The algorithm imposes SRD and LRD behavior into the trace as well.

Matrawy, Lambdaris, and Huang describe MPEG4 video traffic modeling using the TES algorithm in [21]. They separate the video trace into I, B, and P frames, model each set of frames, and combine them together to form a final model that looks very similar to the original.

In [3], Bolot, Turletti, and Wakeman describe a scalable mechanism that elicits feedback from receivers in a multicast transmission. The mechanism can estimate the number of receivers using probabilistic polling with increasing search scope, and with a randomly delayed response it can control the feedback generated.

Ansari, Liu, and Shi evaluate three methods of modeling MPEG video traffic that they have developed in [1]. The first method involves using a Markov model to transfer between three self-similar processes used to model active, inactive, and very active parts of the video, whose parameters have been determined using the least squares fit. The second method involves grouping the frame types into separate self-similar processes, including separating B frames into several self-similar random processes. The third method simplifies the second method by only having three self-similar processes to model I, P, and B frame types separately. They evaluate using least square errors and find that when compared to M/G/infinity and Markov processes their methods work better with the third method being the most accurate.

2.4 Time Series Modeling

Modeling a time series involves capturing the important statistics of the series and being able to easily create a series with the same statistics. Recreating a time series with the same mean is vital, while other statistics such as variance should be very close if not the same. One way to do this is to fit the time series to a distribution. Many distributions exist, however fitting a limited amount of data to one perfectly may not be possible. A good way to fit data while capturing the mean and variance is to use an empirical distribution. In this distribution, a cumulative distribution function (CDF) is created

using the time series, and numbers are generated from that CDF by indexing a uniform random number against the cumulative frequencies to choose a value. The resulting generated time series will have the same mean and variance at the original time series.

Along with capturing the mean and variance of a time series, the autocorrelation should be captured as well. The autocorrelation is a measure of the strength of the relationship between two variables. Typically, autocorrelation is generated up to a certain number of lags. When modeling the autocorrelation, not only must the first few lags be matched, but also greater ones if there is LRD. LRD has been shown to exist in network traffic, see [18], so it must always be considered.

CHAPTER 3

CHARACTERIZATION AND MODELING OF MPEG VIDEOS

In this chapter the methods used to capture video packets, characterize the inherent statistics of video packet streams, and generate synthetic traces that have the same first-order statistics and autocorrelation as the original traces gathered are described. Section 3.1 describes the method and tools used to capture video stream packet. Section 3.2 shows how packet loss was associated with frame loss. Section 3.3 describes how to generate an empirical distribution. Section 3.4 describes how to generate synthetic traces that match the mean, variance, and autocorrelation of captured data using a new method named Time Series Generation (TSG).

3.1 Packet Capture Method

Capturing packets from a network requires hardware and software that can read from a network card and write to the hard disk quickly. For this task, the WinPcap library [26] was chosen because of several reasons. First and foremost, it is open-source, and it works directly with a device driver instead of using a Windows API to make the calls to the driver. This allowed for a few simple modifications to be made to existing sample code that was included with the WinPcap source. The code is powerful and stable enough to handle many thousands of packets streaming in for several hours. Second, it

did everything that was required, such as execute in the Windows operating system, and capture all packets.

The setup of the system used in this thesis is as follows: a Windows PC running the Cisco IP/TV Viewer [8] acts as the client and connects to a video server that is transmitting a MPEG-1 video stream. The stream is transferred over the Internet and when it arrives at the client PC not only is it played, it is also monitored by the packet capturing software. While the packets are being captured they are not stored. Instead, counts of the number of received packets and the number of lost packets are kept using the RTP and UDP loss algorithms described later in this chapter. Once a file of packet counts is gathered, simple tools built using the C programming language are then used to gather statistics such as mean, variance, and autocorrelation over several lags. These tools are open source and are freely available at [29].

Capturing MPEG video packets is a complex process that involves parsing through every packet reaching the receiver. The packets themselves can vary as to what protocol is used to transmit them. However, for this thesis servers that are monitored used the RTP protocol to send packets. RTP is easy to detect and is preferred because of the easily parsed information carried within the packets. RTP packets carry a temporal reference number and a sequence number that helps in calculating the number of packets that have been lost during an interruption in the stream.

3.2 Associating Packet Loss with Frame Loss

The analysis of errors in video streams over the Internet must first begin with an understanding of packet loss. Packet loss will cause errors in a video stream, the severity

of which depends on the amount and type of packet loss. Errors in a single B or P frame will have little impact on the video as compared to errors in a single I frame. Frame lag, where a video frame is delayed by a noticeable amount of time, is possible due to packet delays over the Internet. This type of error will not be addressed in this thesis because jitter buffers in the video software buffer packets and remove delay jitter. It must first be shown that packet loss does indeed cause frame errors because packet loss will be measured and frame errors will be interpolated from this data. To this end, two similar algorithms have been developed to analyze a packet stream and determine when there have been frame errors and how many frames have been affected. For every frame that has an error in it, that frame will be considered 'lost.' Experiments have been conducted (see Chapter 3) that have a user grade a video while programs that implement these algorithms monitor the video stream. A graph of frame rate, packet loss, and user-graded errors presented in Chapter 4 will show that packet loss correlates with frame errors.

The first algorithm applies to RTP-based streams and the other applies to UDP-based streams. Key fields in the RTP header that are needed to detect frame loss are the packet sequence number (Sequence Number), marker bit (M), and picture type (appended after the RTP header). The picture type indicates the type of frame and the marker bit indicates the end of a frame. The packet sequence number is an integer variable that increases by one for each successive count. Also needed is the picture start code (PSC) that is contained in the MPEG data and is used to indicate the start of a new frame. The PSC is 3-byte code of 001 that is followed by a single byte with the stream ID. Figure 6 shows the RTP loss algorithm.

The *sn* is the current packet sequence number, *sn_last* is the last received packet sequence number, *good_count* is the current count of good frames, *bad_count* is the current count of bad frames (lost packets result in bad frames), and *good_flag* is a Boolean flag. The variable *good_flag* is initialized to false and *good_count* and *bad_count* are initialized to zero. The UDP algorithm, shown in figure 7 requires the

```
for (each received packet) do
  if (sn – sn_last != 1) and (good_flag == TRUE) then
    increment bad_count
    good_flag = FALSE
  if (there is a PSC in the MPEG data) then
    good_flag = TRUE
  sn_last = sn
  if (marker bit in RTP header == 1) and (good_flag == TRUE) then
    increment good_count
  if (one second has passed) then
    output good_count and bad_count
    set good_count and bad_count to 0
```

Figure 6. RTP Loss Algorithm

PSC (*packet.framestart*), packet identification value (*packet.id*), fragment offset (*packet.foffset*), fragment size (*packet.fsize*), more fragment (*packet.moref*), and assembled packet size fields (*packet.psize*). The *id_last* variable holds the last *packet.id* value, and the *fsum* variable holds the sum of the packet fragment sizes. The *good_count* variable holds the count of good frames, while *bad_count* holds the count of bad frames. The algorithms generate a series of frame counts for good and bad frames. These frame counts, and the packet counts described above, are characterized and modeled. While the focus of this paper is on the video frame counts, there is no difference in the process of characterizing and modeling packet counts.

```
for (each received packet) do
   if (packet.foffset == 0 && packet.moref == 1 ||
        packet.foffset == 0 && packet.moref == 0)
     if ((packet.id - id_last) != 1)
       good_frame = FALSE
     fsum = 0
   if(packet.foffset != 0 || packet.moref != 0)
     if (packet.foffset != fsum)
       good_frame = FALSE
     fsum = fsum + packet.fsize
   if (packet.framestart == TRUE)
     if (good_frame == TRUE)
       good_count++
     else
       bad_count++
     good_frame = TRUE
   if (packet.moref == 0 && fsum != packet.psize && foffset != 0)
     good_frame = FALSE
```

Figure 7. UDP Loss Algorithm

Both algorithms suffer if large amounts of packets are lost. The algorithms will not be able to count more than one frame as bad if more than one frame is completely lost because there is no record of past frames. For example, if during transmission the latter half of the packets of a frame are lost, and all of the packets of the next two subsequent frames are lost, then there is no way to know if more than one frame was in error (i.e. lost). However, it is rare to have all of the packets of multiple frames lost.

3.3 Characterization of Captured Stream Data

Once the data from a video stream has been captured, it has to be characterized by a model. At first a Markov model was used to try to model the data, but generating the transition values proved to be too complicated, requiring long equations for a model with just a few states. Next, distribution fits were tried. After going through a number of distributions and finding that the data did not come closely enough for a good fit, an

empirical distribution was used. An empirical distribution uses a uniform random number to pick a value based on a CDF generated from the data. This was explained in more detail in Section 2.3. An empirical distribution will capture all of the first-order statistics, such as mean and variance. However, such a distribution will only model the data that it has been given, making capturing as much data as possible from a stream an important task. If a rare, but regular trend, such as a substantially large amount of packet loss that occurs every 24 hours is not in the captured window of data, then it will not be reproduced by the distribution.

Once it was decided to use an empirical distribution to model the first-order statistics, a method of modeling the autocorrelation had to be chosen. A quick to implement and easy to implement method was desired, so two new algorithms based on the idea of iterated uniformity [22] were developed. The main focus of these algorithms is to model the mean, variance, and autocorrelation of a time series. Two were developed to handle different autocorrelation signatures.

The mean, variance, and autocorrelations are a signature of the good frames (frames that have no errors) or bad frames (frames that contain errors) of a particular video stream. Figures 8 through 10 show an example characterization. The video used was a stream of the C-SPAN television channel from Northwestern University to the University of South Florida and was viewed with a Cisco IP/TV viewer. The stream was viewed for about 60 minutes. The camera shots in the video ranged from views of a podium to panning of the audience. The overall packet loss rate in the video stream was measured as 0.77 percent and the bad frame rate was found to be 0.70 percent. The lower percentage for frame loss is from the packet loss being concentrated in a few frames

instead of being spread among multiple frames. Figure 8 shows the histogram of bad frame counts (i.e. the size of bad frame bursts in the 0.70 percent bad frame rate). Figure 9 shows the histogram for good frame counts, and figure 10 shows the autocorrelation for both good and bad frame counts. The autocorrelation signature for the good frame count indicates weak correlation throughout a majority of the lags except for lags 61 and 62 at which the autocorrelation increases to almost 0.20. The cause for this is unknown. The autocorrelation signature for the bad count is very low throughout the 100 lags except for lag 62 in which it is 0.50. Once again, the cause for this is unknown. These results indicate that something is happening about every minute (a single lag of counts corresponds to a one second block of frames).

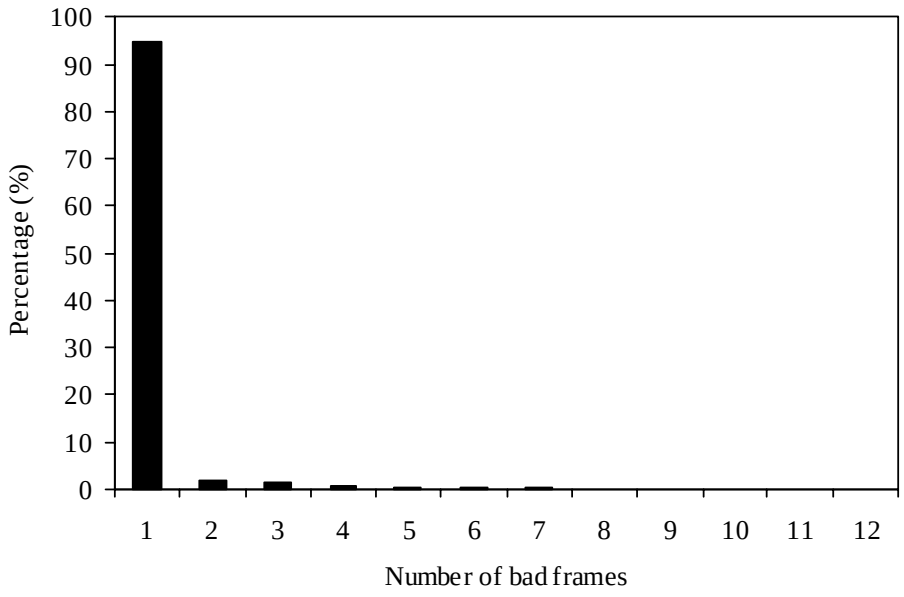

Figure 8. Histogram of Bad Frame Counts for Northwestern University Video

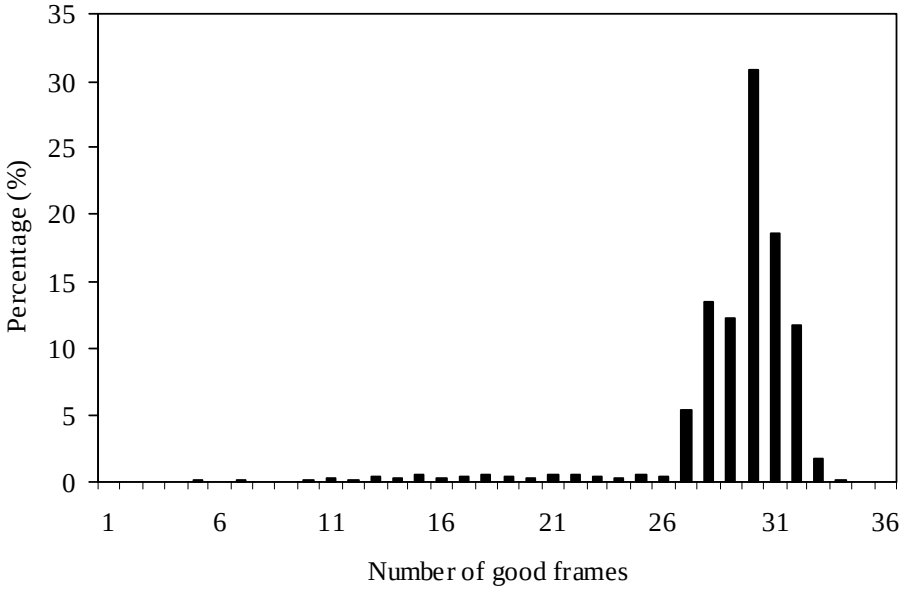

Figure 9. Histogram of Good Frame Counts for Northwestern University Video

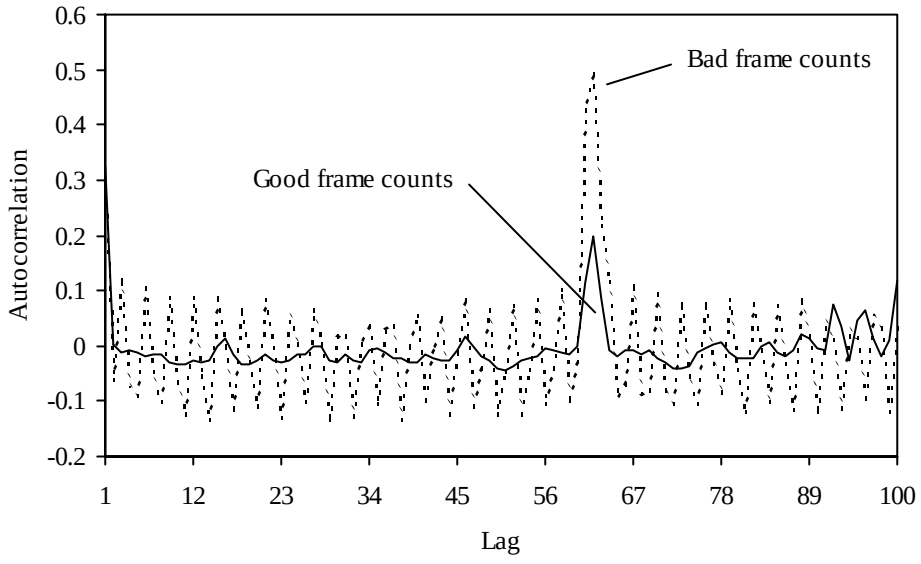

Figure 10. Autocorrelation for Northwestern University Video

3.4 Generating An Empirical Distribution

The goal of this thesis is to develop new methods that can capture the mean, variance, and autocorrelation of a time series and better model that time series than previous methods such as ARTA and TES. Mean, variance, and autocorrelation need to be created in a time series to accurately recreate the original because they are inherent to the structure of it. Before attempting to create synthetic autocorrelation, recreating a time series with the same mean and variance, as the original must be done.

The mean and variance of a time series can be accurately generated using an empirical distribution of the time series. Figures 8 and 9 showed an empirical distribution in histogram form. Generation of an empirical distribution is accomplished by generating random uniform(0,1) values and mapping them to the CDF of a time series. The resulting mapped values will have the same moments as the original time series, but they will also be completely independent. Any correlation between the values of the time series will be lost. This loss of correlation will have a noticeable impact on experiments whenever the synthetic time series is used in place of the original.

3.5 Generating Synthetic Autocorrelation

Recreating autocorrelation in a time series synthetically is the main problem covered in this thesis. To solve this problem in a simple way, TSG has been developed. TSG consists of two methods that rely on the principle of iterated uniformity. The following summarizes this principle: Given that U is distributed uniform(0,1), then $W = \langle U + V \rangle$ will also be uniform(0,1) where V are independent of U and can be from any distribution. The $\langle\ \rangle$ operator is the modulo-1 operator and is defined as

$\langle x \rangle = x - \max\{\text{integer}(n) : n \le x\}$. For a series of *U*, *V*, and resulting *W*, the values of *V* are called the innovation sequence. This principle is important because the underlying distribution of the values is not altered. Therefore, it is possible to adjust these values to achieve a desired autocorrelation while retaining an unchanged empirical distribution. The problem is then how to adjust the values to get the desired autocorrelation. Autocorrelation ρ_k for a series *x* with index *i* and lag *k* is,

$$\rho_k = \frac{E[(x_i - \mu)(x_{i-k} - \mu)]}{\sigma^2} \tag{2}$$

where *E*[] is expected value and σ^2 is the variance of the series. The value of ρ_k ranges from –1 to +1. Values at lag *k* are independent when $\rho_k = 0$.

For positive autocorrelation, the value of ρ_k corresponds to the probability that two values are the same. This is exploited in the *primary method* to be able to generate innovations, *V*, to modify a uniform(0,1) random number generator to generate correlated uniform(0,1) random variates. These correlated random variates are mapped to the CDF of an empirical distribution to generate empirical random variates that are correlated.

For a lag of 1 and $j = 1,3,5,...\infty$, define $x_j =$ uniform(0,1) and $x_{j+1} = x_j$ with a probability π and $x_{j+1} = \text{uniform}(0,1)$ with a probability $1 - \pi$. The resulting series *x* will have $\rho_k = (1/2)\pi$ for $k = 1$ and $\rho_k = 0$ for $k > 1$. Let *k* be the current lag and *N* the final and highest lag that will be calculated. When $N = 2$ and $k = 1$, the resulting series will have $\rho_1 = (1/3)\pi_1 + (1/3)\pi_1\pi_2$. For $N = 3$ and $k = 1$ the series will have $\rho_1 = (1/4)\pi_1 + (1/4)\pi_1\pi_2 + (1/4)\pi_2\pi_3$. For $N = 3$ and $k = 2$ the series will have $\rho_2 = (1/4)\pi_2 + (1/4)\pi_1\pi_3$.

For an arbitrary *N* and *k*,

$$\rho_k = \frac{1}{N+1}\left(\pi_k + \sum_{j=1}^{N-k} \pi_j \pi_{j+k}\right) \tag{3}$$

This formula calculates the autocorrelation at a specific lag *k* based on the input values π_m. This is a new and novel way to quickly calculate what the autocorrelation will be when the algorithm in figure 12 is applied. This allows the algorithm to calculate the autocorrelation without having to generate the samples, thus saving computational time.

The difficult part is to be able to determine the π_m values, where $m = 1,2,...N$, that will result in the desired (input) autocorrelation values (stored in array *real_ac*). The algorithm to do this is shown in figure 11. In this algorithm,

loop for user-determined amount of time
  for (*i* = 1 to *N*) do
    π_i = *uniform*(0,1)
  for (*k* = 1 to *N*) do
    $\rho_k = \frac{1}{N+1}\left(\pi_k + \sum_{j=1}^{N-k} \pi_j \pi_{j+k}\right)$
  store the ρ_k values are in *synthetic_ac*
  calculate LSE between *synthetic_ac* and *real_ac*
  if (LSE is smallest so far) then
    copy π to *bestp*

Figure 11. Primary Method

random uniform(0,1) numbers are generated for each π_m value and equation 2 is used to determine the autocorrelations that would result from using those values to generate a synthetic series. The autocorrelation values from equation (2) for a given π_m are stored

in the array *synthetic_ac*. A least square error (LSE) method is used to determine the quality of the match between the autocorrelation values stored in *synthetic_ac* and the desired autocorrelation values stored in *real_ac*. The equation for the LSE between two sets of autocorrelations, x_1 and x_2 where N is the total number of lags, is as follows:

$$LSE = \sum_{i=1}^{N} [x_2(i) - x_1(i)]^2 \tag{4}$$

The best π_m values are stored in the floating point *bestp* array. The LSE method is only used for measuring the accuracy of an autocorrelation match. Figure 12 shows the algorithm for generating correlated empirically distributed random variates for N lags given probability values π_m.

```
loop until num_samples have been output
  z = emp(cdf, x)
  output z
  for (k = 1 to N) do
    if (uniform(0,1) < π_k ) then
      output z
    else
      output emp(cdf, x)
```

Figure 12. Algorithm for Generating Correlated Random Variates

Each execution of the loop in the algorithm of figure 12 generates $N + 1$ values and checks to see if *num_samples* values have been generated. The value *num_samples* is the number of samples that the user wishes to generate. Inside the loop in figure 12, the algorithm generates a value *z* using the CDF of the empirical distribution (represented in figure 12 by $emp(cdf, x)$ where x is a uniform(0,1) value and *cdf* is a the

CDF of the empirical distribution. Then for *N* lags there is a probability equal to ρ_k to repeat this number. Otherwise, it generates a new value based on the CDF. The primary method does have shortcomings. For some autocorrelation signatures, such as ones that are very cyclic or have negative values, the primary method is unable to match them (however, in all cases the summary statistics will match). For these cases, a secondary method was developed.

For the *secondary method*, a search method is used to find the π_m values that result in a best match with input autocorrelation values. In this method, shown in figure 13, random uniform(-1,1) values are generated for π_m. Then an initial uniform(0,1) value is generated for z_0, and values for z_i are generated using modulo-1 addition of the previous z_{i-1} value and the corresponding π_i value. These z_i values are then transformed using the CDF of the empirical distribution in the function $emp(cdf, z_i)$ and copied into an array of floating point values called *testarray*, which is of size *Batchsize*. *Batchsize* is a value that is calculated from the number of lags (*N*) in the input autocorrelation file multiplied by an integer value named BSM (which is an acronym for batchsize multiplier). Once *Batchsize* values have been generated, with the *count* variable used to keep track of how many values have been generated, the autocorrelation is calculated for all of the values in *testarray*. If the LSE between the autocorrelation values generated from *testarray* and the *real_ac* values is the lowest LSE value up to that point, then π_i is copied to the *bestp* array and the *testarray* is copied to the floating point *bestarray* array. The algorithm then generates new values for π_i and repeats the above steps for as much time as specified by the user. The longer this method (or the

primary method) is run, the more likely the algorithms will find a closer match. Because the user may have time constraints, they are given control over how long the algorithms

loop for user-determined amount of time
  for (*i* = 1 to *N*) do
    π_i = *uniform*(-1,1)
  count = 0
  while (*count* < *Batchsize*)
    z_i = *uniform*(0,1)
    for (*i* = 1 to *N*) do
      $z_i = \langle z_{i-1} + \pi_i \rangle$
      testarray(*count* + *i*) = *emp*(*cdf*, z_i)
    count = *count* + *N*
  calculate autocorrelation for *N* lags for *testarray*
  the autocorrelation values are stored in *synthetic_ac*
  calculate LSE between *synthetic_ac* and *real_ac*
    if (LSE is smallest so far) then copy *testarray* to *bestarray*

Figure 13. Secondary Method

will run. To generate the synthetic series, *bestarray* is repeatedly outputted as many times as necessary to get the desired number of samples.

The secondary method relies on a series held in *testarray* of size *Batchsize* to calculate autocorrelation values from. The size of this series has an effect on the resulting autocorrelation values and summary statistics. If the series was very large, then calculating the autocorrelation in every iteration would slow down the algorithm significantly and finding a good match could take days instead of minutes. The summary statistics, however, would match because of the large number of values. Having too small of a series affects the summary statistics. The autocorrelation signature may be a good match because the algorithm would be able to process the smaller number of values in a shorter time, but the summary statistics, especially the higher order statistics such as variance, may not match. Finding the right BSM is key to this method.

A tool by the name of TSGen has been developed as an implementation of TSG and of mapping uniform(0,1) values to an empirical distribution. This tool is evaluated in Chapter 4. The input to the tool consists of a histogram (the measured empirical distribution) and a series of autocorrelation values. For a packet count trace, figure 14 shows a flowchart of the process for capturing, analyzing, and generating a synthetic packet loss series or trace. A frame count trace uses the output of the RTP or UDP loss algorithm as input into the TSGen tool. For step 1 in figure 14, the *vpgrab*

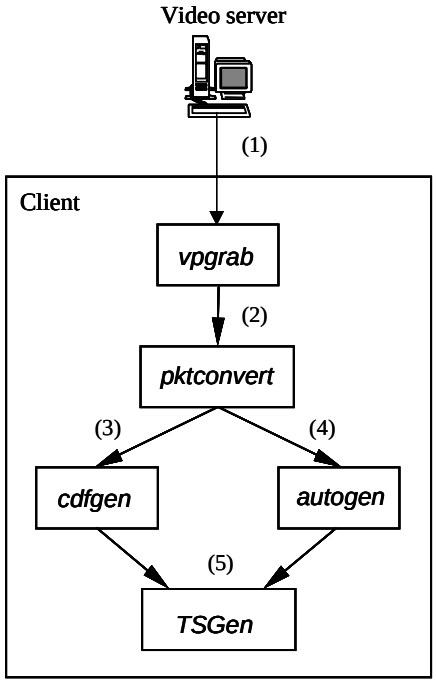

Figure 14. Flowchart of Packet Count Characterization

tool captures video packets from the network. In step 2, the *pktconvert* tool converts the output of *vpgrab* (0 for a received packet, and 1 for a lost packet) to counts of received and lost packets. The *cdfgen* tool in step 3 generates an empirical CDF based on the counts from the *pktconvert* tool, while in step 4 the *autogen* tool generates an

autocorrelation signature based on the output of *pktconvert*. In step 5, the CDF and the autocorrelation signature is used as input to the TSGen tool. The TSGen tool has been implemented in the C programming language as a console-mode program readily portable between Windows and Unix. The inputs to TSGen come from *cdfgen* and *autogen* and are two files. The first file, from *cdfgen*, is a series of tuplets <value, cumulative percentage> and the second file, from *autogen*, is a series of tuplets <lag, autocorrelation value>. Example inputs are shown in Figure 15. The output from the tool is a time series that matches the empirical distribution and has an autocorrelation close to that of the input autocorrelation signature. TSGen is readily available from the author in source code form [29].

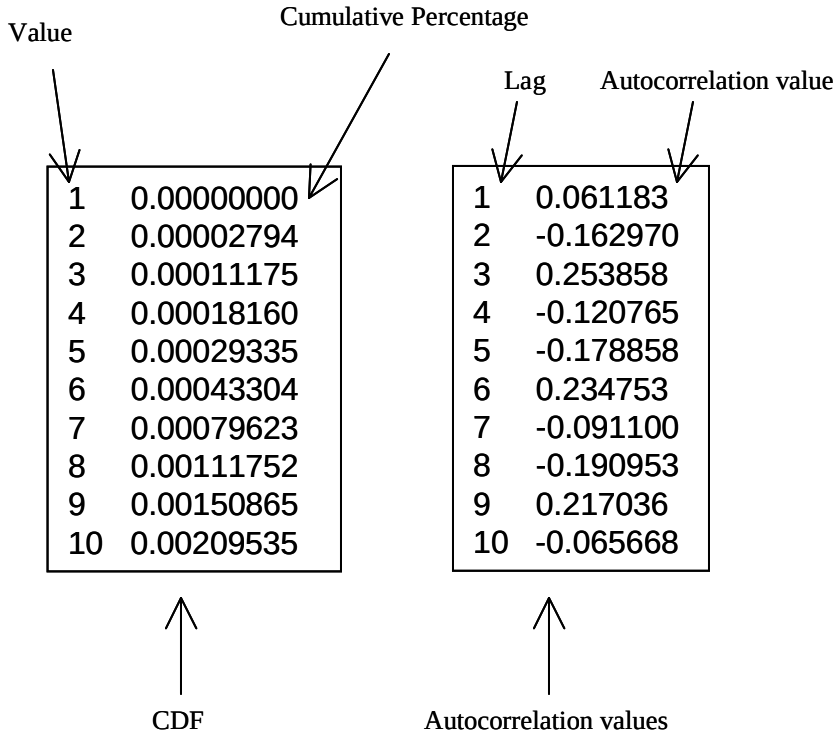

Figure 15. Example Inputs for TSGen

CHAPTER 4

EVALUATION OF TRACE GENERATION METHODS

This chapter evaluates the methods presented in Chapter 3, including the algorithms for capturing video packets and generating traces with matching first-order statistics and autocorrelations of captured video streams. Section 4.1 presents the evaluation methods and describes the evaluations conducted. Section 4.2 characterizes the data used for evaluation, and section 4.3 presents the evaluation results. Section 4.4 describes the overall results, and the methods are compared with existing methods.

4.1 Evaluation Method

The methods presented in Chapter 3 were evaluated using MPEG video stream traces taken using the WinPcap library described in Chapter 2. For the first experiment, the packet capture method was used to validate the frame loss detection algorithms by associating packet loss with frame loss using the RTP loss algorithm (shown in figure 6). For the experiments thereafter, the two methods described in Chapter 3 will be tested and evaluated.

The system setup for the first experiment involved capturing the video packets from a stream sent through the Internet to the University of South Florida. The stream was viewed with the Cisco IP/TV viewer. While watching a streaming video, human-determined subjective grades for observed anomalies were recorded with a time-stamp by a single user. To improve upon this subjective grading, more than one human observer

should be used. A grade of "1" indicated a visual artifact in the picture that did not result in loss of video content. A grade of "2" indicated a delay in the video that resulted in the lagged portion of the video being shown at a higher frame rate to "catch up" to the proper position with no perceivable content loss. Anomalies resulting in content loss were given a grade of "3."

TSG was tested by gathering time-series data that included packet sizes from traces, received packet counts, and lost packet counts. After each trace was gathered, separate software tools described in Chapter 3, *cdfgen* and *empgen*, were used to characterize them. The CDF and autocorrelations up to lag 50 were calculated. TSGen then took the CDF and autocorrelations, and using the methods described in Chapter 3, generated a synthetic time series using both the primary and secondary method that attempted to match the empirical distribution and the autocorrelations up to that lag. For the primary method, TSGen was run for 10 minutes. For the secondary method, BSM values of 1, 2, 10, 20, 30, and 40 were used and each was run for 10 minutes. The best of those was chosen to represent the results of the secondary method. The machine that the evaluations were run on was a Pentium III 866 Mhz with 128 MB of RAM.

The results of the evaluation of TSG will be presented as follows: first, a table showing the results of using the primary method to generate a synthetic trace for each of the traces mentioned in the previous section will be given. Second, a table showing the results of using the secondary method will be given. Because several different experiments were conducted using the secondary method with different BSM values, only the best overall fit will be shown. The best fit is determined for the primary method by the lowest LSE value. For the secondary method, it is found by finding a group of the

lowest LSE value and all other LSE values that are within 10% of the lowest. From that group, the trace with the lowest percent error for variance is determined to be the best fit. After the table for secondary method is given, a table with the overall best fit for each trace will be given. The overall best fit will be determined from the tables for the primary and secondary results. Finally, graphs for the autocorrelation values of each trace and its overall best synthetic trace will be given.

For the comparison between TSG and ARTA, TSGen was run for 10 minutes for the primary method, and 10 minutes for each BSM value for the secondary method. For the secondary method, BSM values of 1, 2, 10, 20, 30, and 40 were used, and the best of those was chosen to represent the results of the secondary method. A table summarizing the statistical results is given, as well a graph comparing the autocorrelation values.

4.2 Data Used for Evaluation

For the validation of the frame loss algorithms, a stream of the C-SPAN television channel from Northwestern University was viewed for about 60 minutes. The content was a single speaker and a sitting audience. The camera shots ranged from views of the podium to panning of the audience, all of which yielded motion. The results of the experiment will be discussed in section 4.3.

For the evaluation of TSG, thirty traces were gathered using the Cisco IP/TV viewer at the University of South Florida. The traces were chosen to provide for experiments that examined as many as possible scenarios for video streams. Their source locations varied across the United States, including the University of Oregon and Columbia University. Different types of streams, such as frame rate counts, lost packet

counts, received packet counts, and packet sizes, were chosen to show that TSG can model any type of time series. Inter-arrival times were gathered to show that TSGen could model those as well. Some of the traces were chosen because they had a small mean and variance while others were chosen because they had a large mean and variance. The autocorrelation signatures varied from cyclic to non-cyclic, and from high to low. Of the thirty traces taken, nine were counts of lost packets from video streams, nine were counts of received packets from video streams, three were frame rates of video streams, three were inter-arrival times from a web server, and six were packet size traces from video streams and from the Internet at USF. Only fifteen of these traces had autocorrelation signatures that were greater than 0, and therefore, only these fifteen of the thirty traces were used for evaluation. Table 1 summarizes all of the traces used for the evaluation of TSGen.

Table 1. Trace Information

Trace	Type	Length of Time	Date Taken	Mean	Variance
10L	Lost packets	10 hours	10/2002	16.70	103871.57
10R	Received packets	10 hours	10/2002	1894.54	15791572.29
19L	Lost packets	19 hours	12/2002	4.73	17.86
19R	Received packets	19 hours	12/2002	3286.75	101204838.70
50L	Lost packets	50 hours	9/2002	10.72	985.30
50R	Received packets	50 hours	9/2002	11374.59	215812122.77
1R [11]	Received packets	1 hour	12/2001	80.87	9330.42
2FR1	Frame rates	2 hours	9/2002	29.26	10.66
2FR2	Frame rates	2 hours	9/2002	29.57	7.20
2FR3	Frame rates	2 hours	9/2002	29.81	2.91
7IA1	Inter-arrival times	7 hours	5/2003	9.45	410.51
7IA2	Inter-arrival times	7 hours	5/2003	15.94	979.73
7IA3	Inter-arrival times	7 hours	5/2003	58.69	1579.81
12R	Received packets	12 hours	1/2003	1820.15	24360044.27
1PS	Packet sizes	1 hour	10/2002	725.29	440843.38
48R	Received packets	48 hours	1/2003	9100.28	736439991.20

An additional evaluation will be run to compare TSG with the ARTA methods. The trace, called 1R, will be from a one hour-long clip of the movie Aladdin [11], and it will consist of good packet counts. The trace naming convention is as follows: the length of time of the trace followed by an acronym for the type of trace, and if there is more than one of those, a number is added to the end indicating the order in which it was taken. The acronym 'R' is used for received packet counts, 'L' for lost packet counts, 'FR' for frame rates, 'IA' for inter-arrival times, and 'PS' for packet sizes. Received packet counts are the number of packets received until a packet is lost, and lost packet counts are the number of packets lost until a packet is received. The frame rates are the number of video frames displayed in one second, and the packet sizes are in bytes. The inter-arrival times are in milliseconds.

Some of the traces, such as 2FR1 and 2FR2 are very cyclic, whereas others, such as 10L and 7IA1, are not. Some have a small mean and variance, such as 19L, while others have a large mean and variance, such as 50R. These choices provide for many different situations in streaming video.

4.3 Evaluation Results

The frame loss detection algorithms described in Chapter 3 were validated by associating packet loss with frame loss. For a separate (from Table 1) graded video stream, figure 16 shows a plot of frame rate, a cumulative count of frame loss, and a cumulative count of the graded errors of the video stream. The average frame rate for the capture was 29.19 frames per second. The expected frame rate for NTSC television is 29.97 frames per second, so there was a loss of about 2.6%.

Table 2 shows the results of the primary method being used on each trace from table 1, with LSE value, synthetic mean and variance, mean and variance for the actual trace, and percent error between the actual and synthetic mean and variance being shown. Table 3 shows the results of the secondary method being used on each trace, with LSE value, BSM value, synthetic mean and variance, mean and variance for the actual trace, and percent error between the actual and synthetic mean and variance being shown.

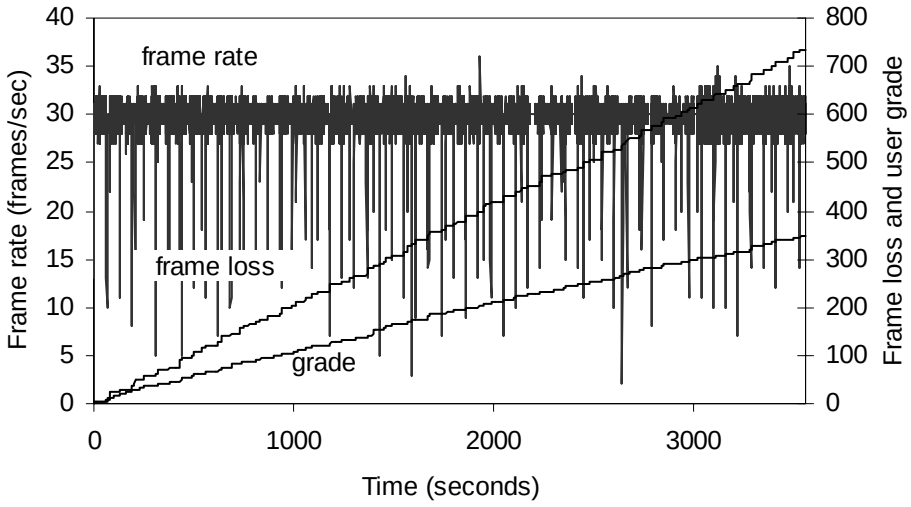

Figure 16. Frame Rate, Loss Counts, and Grades for Measured Video Stream

Table 4 was chosen from the best results of the two methods for each trace. The table contains the best overall fit for each trace, including the method used, LSE value, BSM value, synthetic mean and variance, mean and variance for the actual trace, and percent error between the actual and synthetic mean and variance being shown.

Figures 17 through 31 compare the autocorrelation values of the best fit for each trace against the autocorrelation values of the actual trace. Figure 32 compares the autocorrelation values of the best fit for the 1R trace using TSG against a fit done by the

ARTA method and the actual trace. Table 4 compares the summary statistics for the 1R trace. Discussion of these results will be done in the next section.

Table 2. Trace Generation Results for Primary Method

Trace	LSE	% Error Mean	% Error Variance	Synth. Mean	Actual Mean	Synth. Variance	Actual Variance
10L	0.123	8.16	17.05	15.34	16.70	86165.72	103871.57
10R	0.145	0.34	2.68	1888.13	1894.54	15368223.18	15791572.29
19L	0.059	0.15	1.35	4.72	4.73	18.10	17.86
19R	0.394	3.42	12.60	3174.37	3286.75	88451812.69	101204838.70
50L	0.147	5.19	9.37	10.16	10.72	892.97	985.30
50R	0.042	1.10	4.29	11499.79	11374.59	225065384.78	215812122.77
1R [11]	0.036	1.41	3.83	82.01	80.87	8972.91	9330.42
7IA1	0.026	2.08	3.17	9.25	9.45	397.48	410.51
2FR1	1.638	0.02	0.95	29.25	29.26	10.56	10.66
2FR2	1.475	0.05	2.84	29.58	29.57	7.00	7.20
2FR3	0.005	0.11	18.00	29.84	29.81	2.39	2.91
7IA2	0.012	0.60	0.38	16.04	15.94	983.49	979.73
12R	0.122	0.00	3.50	1820.14	1820.15	23506244.69	24360044.27
1PS	0.067	1.29	0.45	715.96	725.29	438874.85	440843.38
7IA3	0.006	0.02	0.40	58.70	58.69	1573.57	1579.81
48R	1.021	3.27	12.50	8802.96	9100.28	644420921.39	736439991.20

Table 3. Trace Generation Results for Secondary Method

Trace	BSM	LSE	% Error Mean	% Error Variance	Synth. Mean	Actual Mean	Synth. Variance	Actual Variance
10L	20	0.000	13.79	11.81	19.01	16.70	91605.42	103871.57
10R	20	0.309	0.20	6.23	1898.31	1894.54	16775815.42	15791572.29
19L	40	0.037	1.71	4.92	4.81	4.73	18.74	17.86
19R	40	2.979	2.27	6.25	3212.23	3286.75	94882227.61	101204838.70
50L	40	0.021	1.11	0.67	10.84	10.72	991.89	985.30
50R	10	0.159	2.62	4.41	11672.35	11374.59	206290159.64	215812122.77
1R [11]	20	0.073	3.55	18.45	78.00	80.87	7609.23	9330.42
7IA1	40	0.029	5.20	5.28	8.96	9.45	388.83	410.51
2FR1	2	0.368	1.40	40.60	28.85	29.26	14.99	10.66
2FR2	10	0.315	1.12	40.59	29.24	29.57	10.12	7.20
2FR3	30	0.622	0.16	3.09	29.76	29.81	2.82	2.91
7IA2	10	0.090	9.35	13.75	17.43	15.94	1114.46	979.73
12R	20	0.088	0.72	3.36	1833.33	1820.15	25179074.86	24360044.27
1PS	20	0.067	2.12	0.60	740.64	725.29	438208.84	440843.38
7IA3	20	0.058	1.20	6.47	59.39	58.69	1477.56	1579.81
48R	40	5.956	3.52	8.08	8779.89	9100.28	676929866.54	736439991.20

Table 4. Best Overall Fit for Each Trace

Trace	Method	BSM	LSE	% Error Mean	% Error Variance	Synth. Mean	Actual Mean	Synth. Variance	Actual Variance
10L	Secondary	20	0.000	13.79	11.81	19.01	16.70	91605.42	103871.57
10R	Primary	N/A	0.145	0.34	2.68	1888.13	1894.54	15368223.18	15791572.29
19L	Secondary	40	0.037	1.71	4.92	4.81	4.73	18.74	17.86
19R	Primary	N/A	0.394	3.42	12.60	3174.37	3286.75	88451812.69	101204838.70
50L	Secondary	40	0.021	1.11	0.67	10.84	10.72	991.89	985.30
50R	Primary	N/A	0.042	1.10	4.29	11499.79	11374.59	225065384.78	215812122.77
1R [11]	Primary	N/A	0.036	1.41	3.83	82.01	80.87	8972.91	9330.42
7IA1	Primary	N/A	0.026	2.08	3.17	9.25	9.45	397.48	410.51
2FR1	Secondary	2	0.368	1.40	40.60	28.85	29.26	14.99	10.66
2FR2	Secondary	10	0.315	1.12	40.59	29.24	29.57	10.12	7.20
2FR3	Primary	N/A	0.005	0.11	18.00	29.84	29.81	2.39	2.91
7IA2	Primary	N/A	0.012	0.60	0.38	16.04	15.94	983.49	979.73
12R	Secondary	20	0.088	0.72	3.36	1833.33	1820.15	25179074.86	24360044.27
1PS	Primary	N/A	0.067	1.29	0.45	715.96	725.29	438874.85	440843.38
7IA3	Primary	N/A	0.006	0.02	0.40	58.70	58.69	1573.57	1579.81
48R	Primary	N/A	1.021	3.27	12.50	8802.96	9100.28	644420921.39	736439991.20
Average of Results			0.162	0.48	1.84	1762.70	1780.46	62439303.92	68385413.04

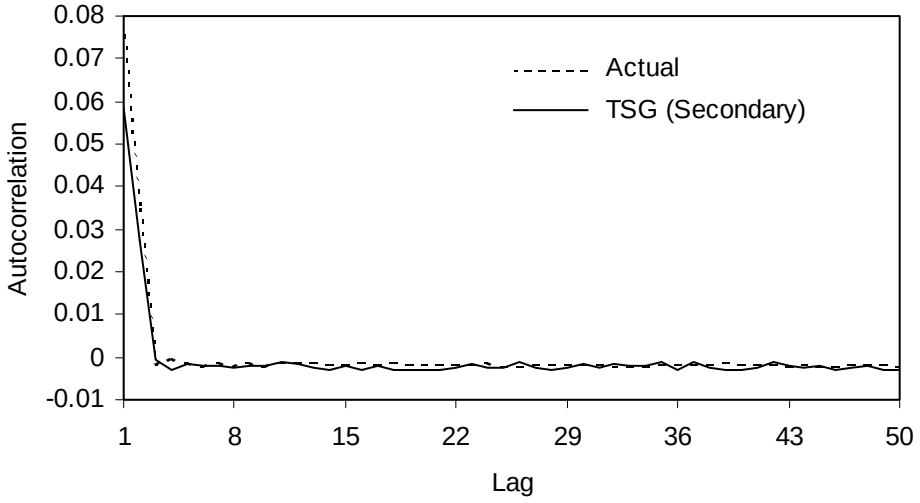

Figure 17. Autocorrelation Values for the 10L Trace

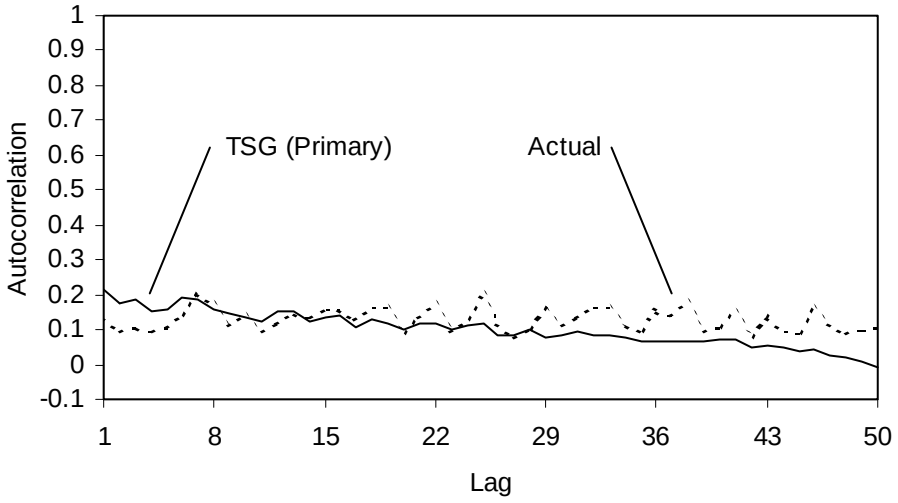

Figure 18. Autocorrelation Values for the 10R Trace

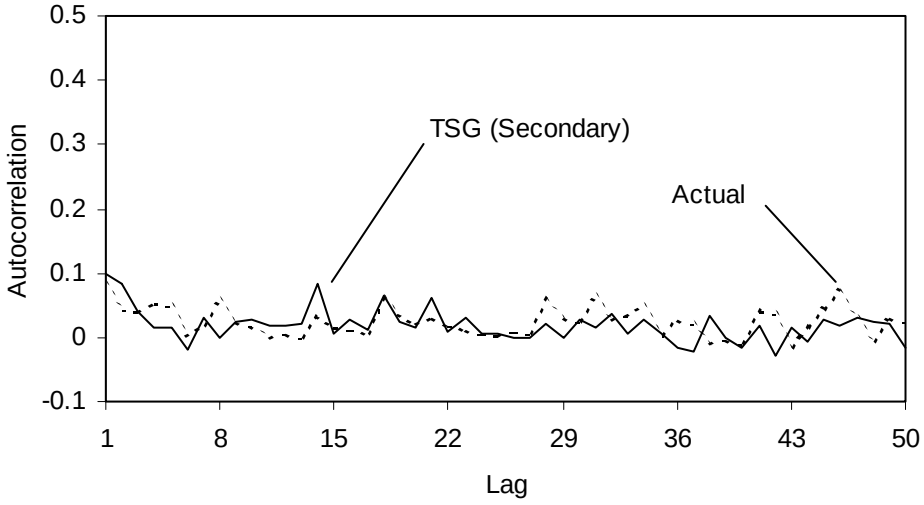

Figure 19. Autocorrelation Values for the 19L Trace

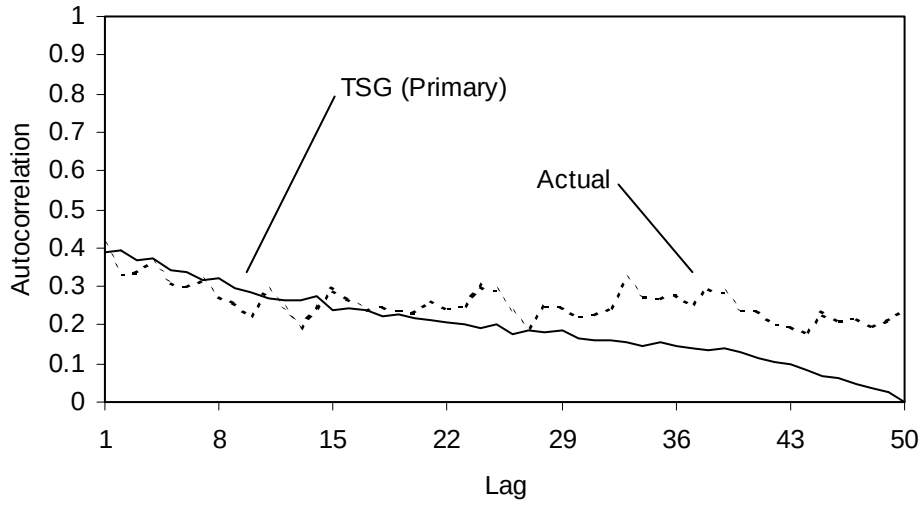

Figure 20. Autocorrelation Values for the 19R Trace

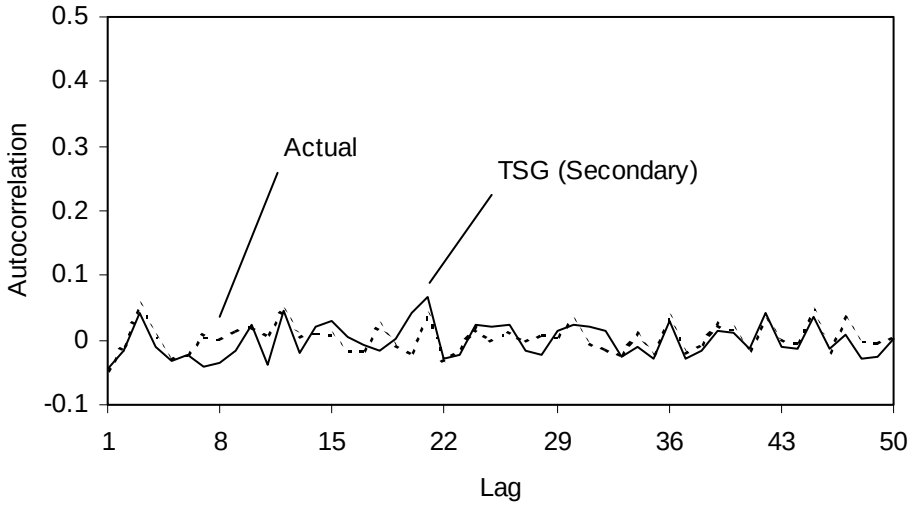

Figure 21. Autocorrelation Values for the 50L Trace

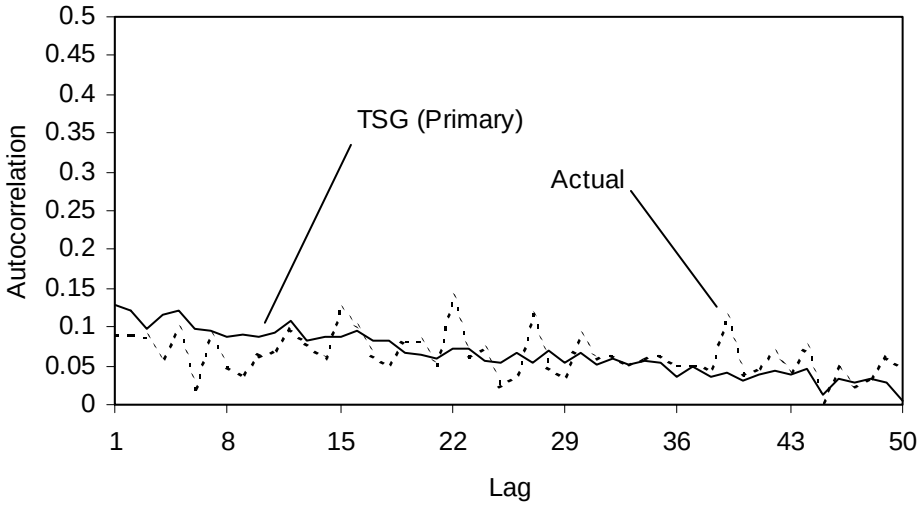

Figure 22. Autocorrelation Values for the 50R Trace

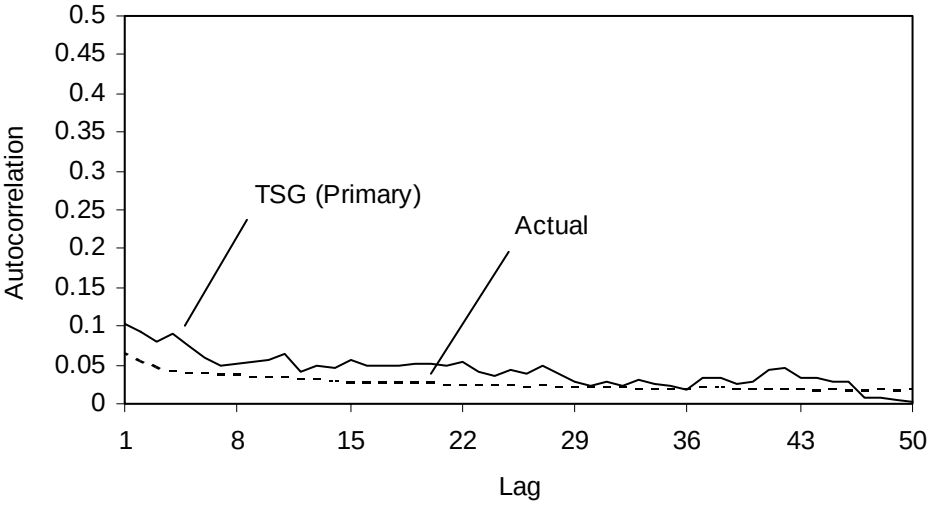

Figure 23. Autocorrelation Values for the 7IA1 Trace

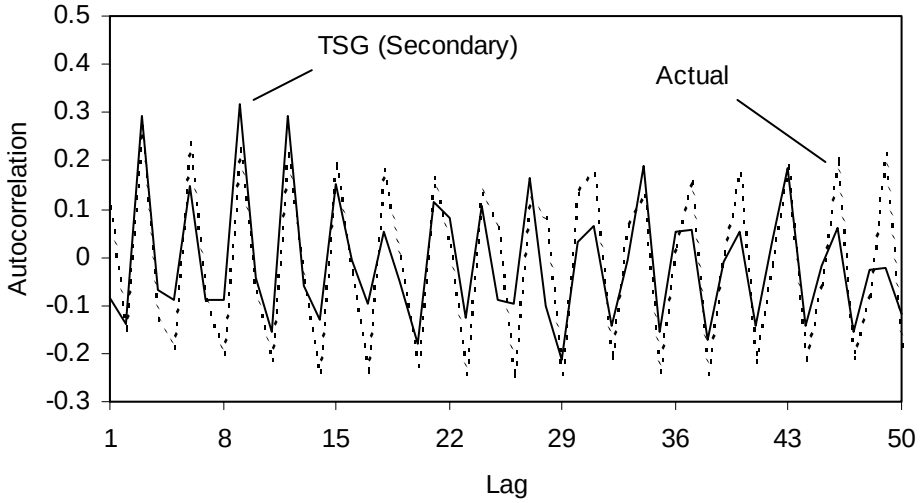

Figure 24. Autocorrelation Values for the 2FR1 Trace

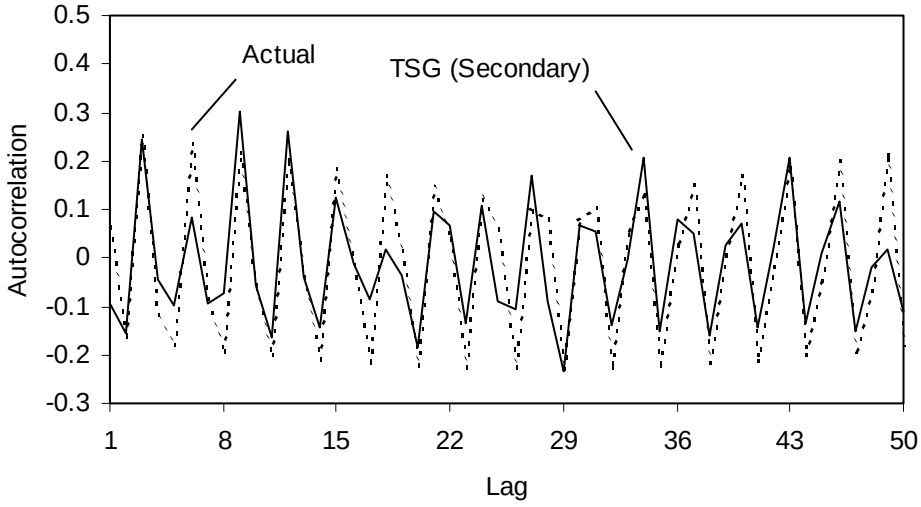

Figure 25. Autocorrelation Values for the 2FR2 Trace

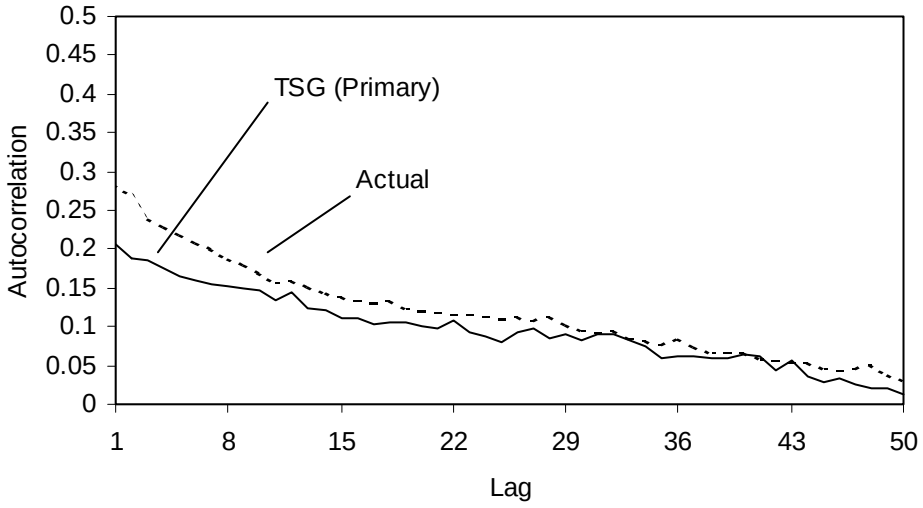

Figure 26. Autocorrelation Values for the 2FR3 Trace

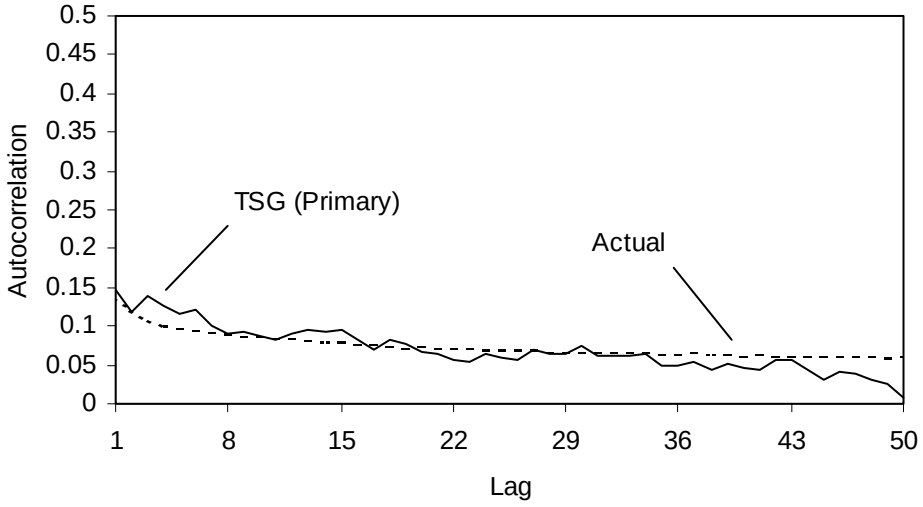

Figure 27. Autocorrelation Values for the 7IA2 Trace

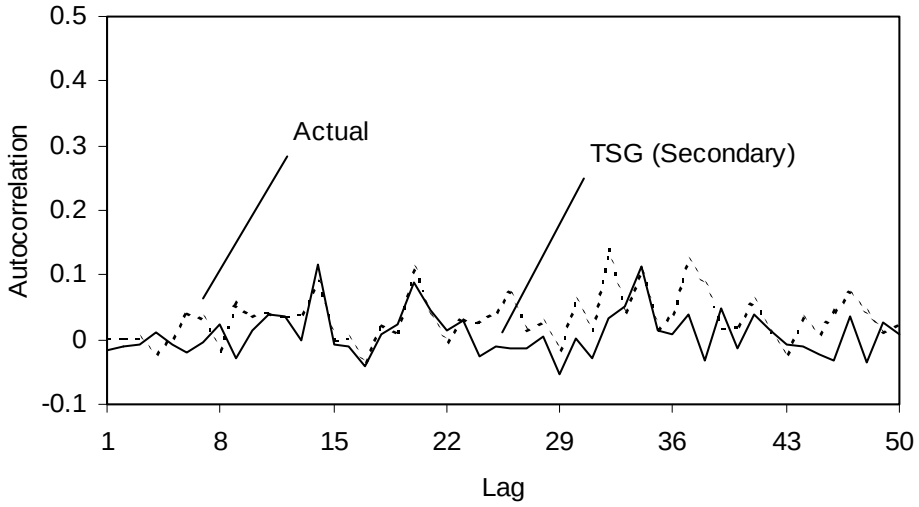

Figure 28. Autocorrelation Values for the 12R Trace

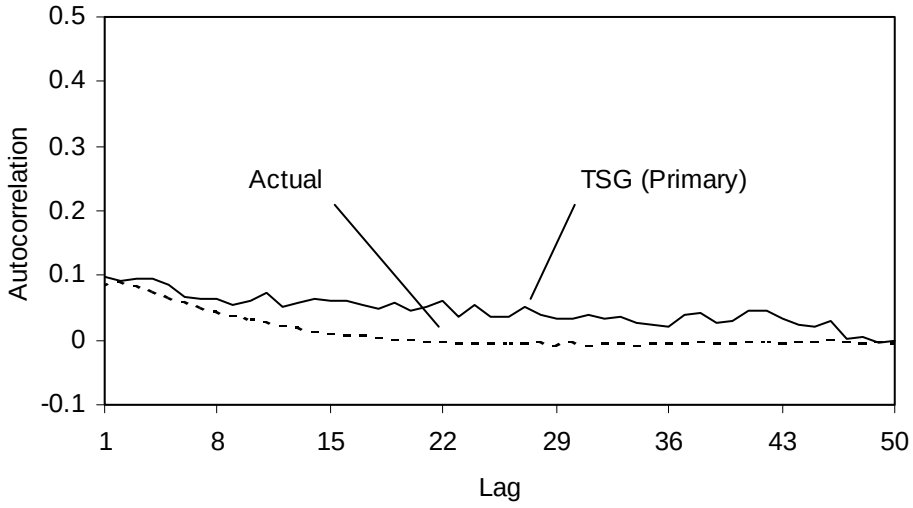

Figure 29. Autocorrelation Values for the 1PS Trace

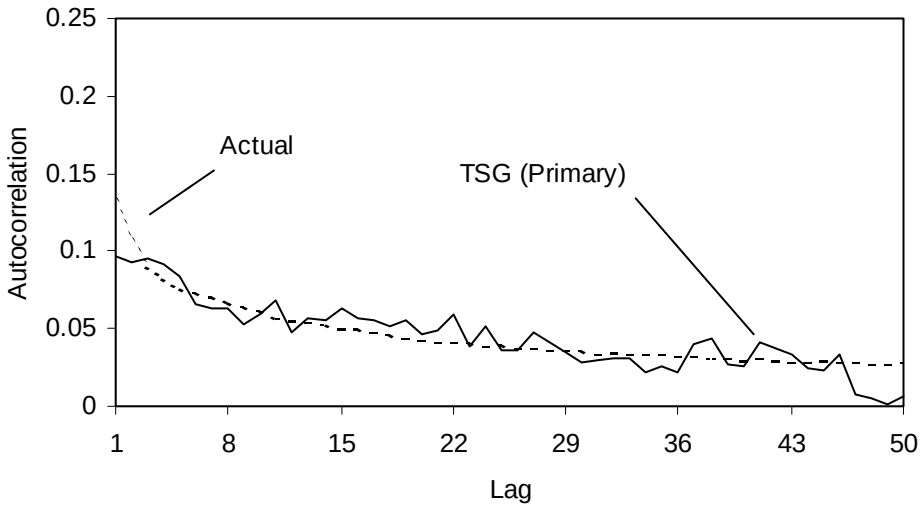

Figure 30. Autocorrelation Values for the 7IA3 Trace

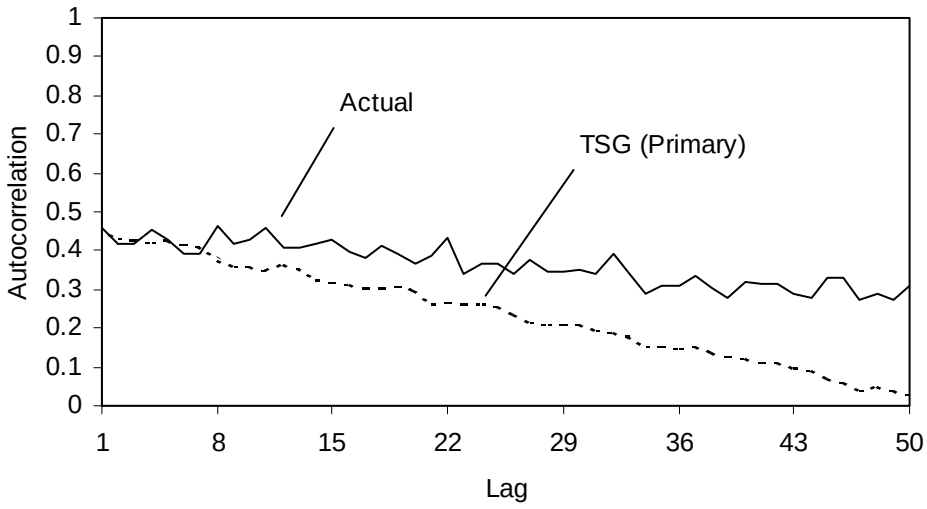

Figure 31. Autocorrelation Values for the 48R Trace

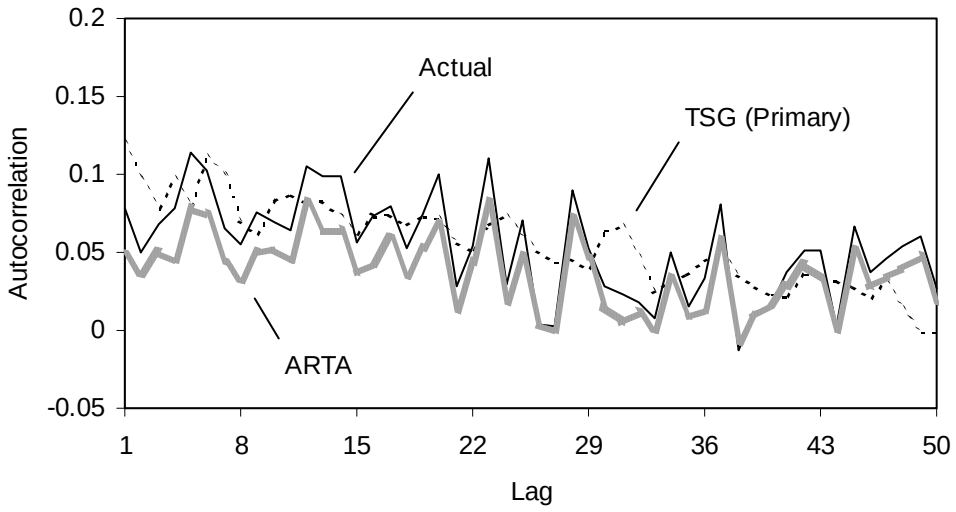

Figure 32. Autocorrelation Values for the 1R Trace

Table 5. Statistics for 1R Trace

Trace	LSE	BSM	% Error Mean	% Error Variance	Mean	Variance
Actual	0.000	N/A	0.00	0.00	80.87	9330.42
ARTA	0.019	N/A	0.26	2.74	81.08	9075.10
TSG (Primary)	0.037	N/A	1.41	3.83	82.01	8972.91
TSG (Secondary)	0.036	40	3.55	18.45	78.00	7609.23

4.4 Discussion of Evaluation Results

For the evaluation of the frame loss detection algorithms, the graph shown in figure 16 shows several sharp drops in frame rate followed by increases in the frame loss and graded errors counts. The loss of a single I-frame or multiple B or P frames will cause a grade 3 error (an error that causes content loss). By positively relating the graded errors with a drop in frame rate, it is verified that TSG can correctly identify anomalies caused by lost MPEG video frames. The matching trend of frame loss and grade shows that frame losses positively relate to anomalies.

The results presented in the previous section show that TSG is able to model both the summary statistics, such as mean and variance, and the autocorrelation of a time series. Table 2 shows that the primary method is better than the secondary method at modeling the mean and variance. The percent error for mean and variance in table 1 are no worse than 18%, and twelve out of the sixteen traces are under 10%. Table 3, which shows the results of the evaluations of the secondary method, is similar in that fourteen out of the sixteen traces have percent error values are under 20%, except for two traces in particular: 2FR1 and 2FR2. This is due to the method for choosing the best fit. As described earlier in this chapter, out of the synthetic traces with different BSM values that were taken for each trace, the ones with the lowest LSE values were placed into a group. Of those traces in that group, the lowest percent error variance determined the trace that had the best fit. There were synthetic traces with lower percent error values (out of all the traces created for 2FR1 and 2FR2), but the LSE values were much higher and were not placed into the low LSE grouping during the best-fit determination.

The graphs show that TSG is able to model the autocorrelation values of the actual traces. For three of the traces, shown in figures 18, 20, and 31, the synthetic trace begins to approach zero halfway through the lags. This is a weakness of the primary method; its ability to match autocorrelation values decreases as the lag increases. The secondary method would not have such a drop, but it also may not (and for these experiments, did not) match overall for the same trace.

The comparison between ARTA and TSG, as shown in figure 32 and table 5, shows that ARTA is better able to match the mean and variance of the actual trace. ARTA also produces a trace that is visually similar in shape to the actual trace but it does

not match exactly. The LSE values produced by the TSG traces are low, but not as low as ARTA. The primary method was chosen to represent TSG because it had better matching mean and variance values than did the secondary method.

ARTA and TES (as described in Chapter 2) are similar in some aspects to TSG, such as both TSG and TES use the idea of iterated uniformity, and all three methods use the inverse of the CDF of an empirical distribution to match the summary statistics. They differ in that TSG does not use a stitching function or an AR process. TSG also uses its own method (see equation 3) of calculating autocorrelation values during the search process based on the chosen π_m values instead of creating the values and then calculating the autocorrelation signature. TSG requires less tuning of parameters by a user than does TES. The only input needed from the user is specifying the run time of the methods. The BSM values were manually entered for these experiments, however TSGen could be made to automatically increase the BSM until a specified value is reached. For TES, parameter adjustment is done until a desired fit is achieved, with the judging done by eye using a specific tool named TEStool that is no longer available. TSG determines goodness of fit within the algorithm using a least squares estimator. This requires less input from the user and allows for more automation, making it better in that aspect than TES. This is similar to the problem with ARTA; if data with an unknown distribution is used, then it is difficult to estimate the parameters. TSG has an advantage over ARTA because it was developed to work with the *cdfgen* and *autogen* tools, which were described in chapter 3, and use its own primitive search to find the best set of π_m values. The disadvantage of TSG is its accuracy with cyclic autocorrelation signatures; the more

cyclic an autocorrelation signature is, the longer it will take TSG to find an accurate match.

The primary method of TSG is best used with non-cyclical, positive autocorrelation signatures. The secondary method, which requires a longer runtime than the primary method because of the larger amount of calculations done, is able to obtain a synthetic autocorrelation that is reasonably accurate. However, an adjustment to a parameter must be made with the secondary method to achieve a good match of summary statistics. While the primary method models cyclical autocorrelation signatures with little accuracy, it is better than the secondary method with regards to modeling summary statistics. The primary method was also quicker in execution because calculation of the autocorrelation of the synthetic trace could be done using an equation, whereas with the secondary method, the entire synthetic trace had to be used to calculate the autocorrelation. These strengths and weaknesses are the reasons why both methods were used; the primary method provided a closer match for 10 out of the 16 traces (10R, 19R, 50R, 1R, 7IA1, 2FR3, 7IA2, 1PS, 7IA3, and 48R), while the secondary method provided a closer match for six out of the sixteen traces (10L, 19L, 50L, 2FR1, 2FR2, and 12R). As shown by the graphs of the autocorrelation signature comparisons, the traces that the secondary method was better at modeling were more cyclical in shape than those that the primary method was better at modeling.

Another aspect of the synthetic trace generation that must be mentioned happens only for the primary method. In some of the traces, after about 20 or 25 lags, the synthetic autocorrelation signature begins to diverge from the actual autocorrelation signature and approach zero. This is a weakness of the primary method; it had difficulty

with matching autocorrelation at high lags. This has an effect on the rest of the trace as well: the more lags there are, the worse the overall match is. This is because the method generalizes the match into a single value to judge the overall fit. The reason for the strength of the match weakening as the lags increase is because of the way it creates synthetic values. The equation it uses to predict the autocorrelation values has to use more values from previous lags as the lag number increases. This makes it difficult to find a good match because all previous π_m values are affecting the current ρ_k value.

The weakness of the secondary method is that it tends to produce traces that are cyclical. This is because of the random nature of the values, which are not dependent on all previous π_m values as in the primary method. This gives it an advantage when trying to match cyclical traces, but this also increases the difficulty in matching non-cyclical traces.

CHAPTER 5

SUMMARY AND FUTURE WORK

A pair of autocorrelation matching methods, collectively called Time Series Generation (TSG) has been developed to generate inputs to test streaming video software over networks. TSG gathers video stream statistics and creates packet loss models of video traces that recreate mean, variance, and autocorrelation signatures. These synthetic traces can have their inherent statistics altered, allowing for thorough testing of video software.

The two methods developed, known as primary and secondary, rely on the principle of iterated uniformity. The primary method is suited for positive autocorrelation values that are not cyclic in shape, whereas the secondary method is suited for any type of autocorrelation signature: positive or negative, cyclic or non-cyclic. The secondary method does require more calculations and is not as accurate with the mean and variance as the primary methods. These trade-offs must be taken into consideration when using TSG.

A tool named TSGen was developed to implement TSG, as well as supporting tools that generated the input that TSGen required, such as the CDF and autocorrelation values of traces. Packet capturing software was also developed to capture the video traces from the Internet.

The packet capturing software was able to capture all the video packets without any error. The correlation of packet loss to frame errors was successful as well. A user-

graded trace clearly showed errors where there were noticeable drops in the frame rate (i.e. packet loss occurred).

TSG was successful in its goal of generating synthetic traces that could match autocorrelation, mean, and variance. It can be seen by examining figures 17 through 31 that several traces with very low Least Square Error (LSE) values were created. LSE is used as a measure of the goodness of fit between the autocorrelation signature of a synthetic trace and the original trace. The mean and variance of the synthetic traces were within 10% of the actual traces for 11 out of the 16 traces, with only two of the traces having a percent error difference of above 20%. The mean of the traces was within 4% for all but one, but the variance was what produced the high percent error values. In those two exceptional cases, traces 2FR1 and 2FR2, it was the variance that produced a high percent error. The reason for this is that the secondary method was used to produce the synthetic trace, and it takes longer than the primary method to get a good match for mean and variance. However, the exact reason why in those two cases the percent error was so high is unknown. The actual variance values were 10.66 and 7.20, so any difference in the synthetic values (such as 14.99 and 10.12, respectively) would produce high percent error differences. A better match of the mean and variance could have chosen at the expense of the autocorrelation match for these particular traces. This was a trade-off that had to be considered for these particular traces.

TSG can be improved in several areas. The accuracy of the primary method when dealing with cyclic and negative autocorrelation signatures can be addressed, and if a satisfactory improvement can be made, then there is no need for the secondary method. The secondary method requires many more operations than the primary method and is

less accurate with the summary statistics. If the primary method cannot be improved, then the secondary method can be improved by increasing its mean and variance matching. Using the BSM to find the best match increases the complexity and reliance on the user, and for the best possible outcome those should be kept to the bare minimum.

REFERENCES

[1] N. Ansari, H. Liu, and Y. Q. Shi, "On Modeling MPEG Video Traffics," *IEEE Transactions on Broadcasting*,Vol. 48, No. 4, pp. 337-347, 2002.

[2] J. Beran, R. Sherman, and M. S. Taqqu, "Long-Range Dependence in Variable-Bit-Rate Video Traffic," *IEEE/ACM Transactions on Communications*, Vol. 43, No. 2, pp. 1566-1579, 1994.

[3] J. Bolot, T. Turletti, and I. Wakeman, "Scalable Feedback Control for Multicast Video Distribution in the Internet," *Proceedings of ACM SIGCOMM*, pp. 58-67, 1994.

[4] M. C. Cario and B. L. Nelson, "Autoregressive to anything: Time-series input processes for simulation," *Operations Research Letters*, 19, pp. 51-58, 1996.

[5] M. C. Cario and B. L. Nelson, "Numerical Methods for Fitting and Simulating Autoregressive-To-Anything Processes," *INFORMS Journal on Computing*, 10, pp. 72-81, 2001.

[6] J. Case, J., M. Fedor, M. Schoffstall, J. Davin, "A Simple Network Management Protocol," *RFC 1067*, 1988.

[7] H. Che and S. Li, "Fast Algorithms for Measurement-Based Traffic Modeling," *Proceedings of IEEE INFOCOM*, pp. 177-186, 1997.

[8] Cisco IP/TV Viewer, URL: http://www.cisco.com/en/US/products/sw/conntsw/ps869/index.html, August 2003.

[9] Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s, ISO/IEC 11172.

[10] I. Dalgic and F. A. Tobagi, "Glitches as a Measure of Video Quality Degradation Caused by Packet Loss," *7th International Workshop on Packet Video*, 1996.

[11] F. H. P. Fitzek and M. Reisslein, "MPEG-4 and H.263 Video Traces for Network Performance Evaluation," *IEEE Network*, Vol. 15, Issue 6, pp. 40-54, 2001.

[12] M. W. Garret and W. Willinger, “Analysis, Modeling and Generation of Self-Similar VBR Video Traffic,” *Proceedings of ACM SIGCOMM*, pp. 269-280, August 1994.

[13] Generic coding of moving pictures and associated audio information, ISO/IEC 13818.

[14] B. Hajek and L. He, “On Variations of Queue Response for Inputs with the Same Mean and Autocorrelation Function,” *IEEE/ACM Transactions on Networking*, 1996. Vol. 6, Issue 5, pp. 588-598, 1998.

[15] C. Huang, M. Devetsikiotis, I. Lambadaris, and A. R. Kaye, “Modeling and Simulation of Self-Similar Variable Bit Rate Compressed Video: A Unified Approach,” *Proceedings of ACM SIGCOMM*, pp. 114-125, January 1995.

[16] R. Jain and S. A. Routhier, “Packet Trains-Measurements and a New Model for Computer Network Traffic,” *IEEE Journal of Selected Areas in Communications*, Vol. 4, No, 6, pp. 986-995, 1986.

[17] D. J. Le Gall, “The MPEG Video Compression Standard,” *Compcon Spring '91 Digest of Papers*, pp. 334-335, 1991.

[18] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, “On the Self-Similar Nature of Ethernet Traffic (Extended Version),” *IEEE/ACM Transactions on Networking*, Vol. 2, No. 1, pp. 1-15, 1994.

[19] N. X. Liu and J. S. Baras, “Statistical Modeling and Performance Analysis of Multi-Scale Traffic,” *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 3, pp. 1837-1847, April 2003.

[20] A. Lombardo, G. Morabito, S. Palazzo, G. Schembra, “MPEG Traffic Generation Matching Intra- and Inter-GoP Correlation,” *SIMULATION*, Vol. 74, No. 2, February 2000.

[21] A. Matrawy, I. Lambadaris, and C. Huang, “MPEG4 Traffic Modeling Using The Transform Expand Sample Methodology,” *Proceedings of the IEEE 4th International Workshop on Networked Appliances*, pp. 249-256, 2001.

[22] B. Melamed, “TES: A Class of Methods for Generating Autocorrelated Uniform Variates,” *ORSA Journal on Computing*, Vol. 3, No. 4, pp. 317-329, 1991.

[23] V. Paxson and S. Floyd, “Wide Area Traffic: The Failure of Poisson Modeling,” *IEEE/ACM Transactions on Networking*, Vol. 3, No. 3, pp. 226-244, 1995.

[24] J. Postel, "User Datagram Protocol," *RFC 768*, 1980.

[25] V. Raisanen and A. Lakaniemi, "A General Method for Analyzing and Synthesizing Loss Patterns," *Proceedings of the IEEE International Conference on Communications*, pp. 2558-2562, 2002.

[26] F. Risso and L. Degioanni, "An Architecture for High Performance Network Analysis," *Proceedings of the Sixth IEEE Symposium on Computers and Communications*, pp. 686-693, 2001.

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

[28] J. Shahbazian and K. Christensen, "SNMP counter for distributed monitoring of MPEG video quality," *Electronics Letters*, Vol. 39, No. 1, pp. 166-168, 2003.

[29] Video Packet Capture Tools, URL: http://www.csee.usf.edu/~jshahbaz/rsoftware.html, October 2003.

[30] S. Waldbusser, "Remote Network Monitoring Management Information Base," *RFC 2819*, 2000.

[31] D. E. Wrege and J. Liebeherr, "Video Traffic Characterization for Multimedia Networks with a Deterministic Service," *Proceedings of the Fifteenth Annual Joint Conference of the IEEE Computer Societies*, Vol. 2, pp. 537-544, 1996.

[32] J. Wu and M. Hassan, "The Issue of Useless Packet Transmission for Multimedia Over the Internet," *Computer Communications*, Vol. 26, No. 12, pp. 1240-1254, 2003.

[33] L. Zhang, L. Zheng, and K. S. Ngee, "Effect of Delay and Delay Jitter on Voice/Video Over IP," *Computer Communications*, Vol. 25, No. 9, pp. 863-873, 2002.