# ANALYSIS OF VIDEO LATENCY IN UAV

# WIRED AND WIRELESS NETWORKS

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Gloice Dean Works, III

Certificate of Approval:

| | |
|---|---|
| Richard O. Chapman<br>Associate Professor<br>Computer Science and<br>Software Engineering | David A. Umphress, Chair<br>Associate Professor<br>Computer Science and<br>Software Engineering |
| James H. Cross, II<br>Professor<br>Computer Science and<br>Software Engineering | Joe F. Pittman<br>Interim Dean<br>Graduate School |

# ANALYSIS OF VIDEO LATENCY IN UAV WIRED AND WIRELESS NETWORKS

Gloice Dean Works, III

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
May 10, 2008

ANALYSIS OF VIDEO LATENCY IN UAV

WIRED AND WIRELESS NETWORKS

Gloice Dean Works, III

Signature of Author

Date of Graduation

THESIS ABSTRACT

# ANALYSIS OF VIDEO LATENCY IN UAV WIRED AND WIRELESS NETWORKS

Gloice Dean Works, III

Master of Science, May 10, 2008
(B.W.E., Auburn University, 2006)

73 Typed Pages

Directed by David A. Umphress

The ability to utilize wireless networks to transmit information offers many benefits as compared to wired networks. For example, in Unmanned Aerial Vehicle (UAV) military applications current technology relies on a wired network to transmit real-time video and other information between ground control systems. The reliance of this wired network in the UAV military application has many disadvantages and is plagued by a number of issues. The most significant problem is the actual wired infrastructure. The wired connections that interconnect ground control systems are subject to failure when tanks and other equipment destroy the cabling. Many of these problems could be alleviated by implementing a wireless network to transmit information between the ground control equipment. The feasibility of replacing the UAV wired network with a wireless network must be proven by modeling the UAV systems and subjecting the model to both wired and wireless connection experiments.

The feasibility of the proposed wireless solution was calculated by measuring the time needed to propagate the video signal from the UAV payload camera to the Ground Control Station (GCS). Analysis of these experiments provided a comparison of both network topologies and proved that a wireless implementation would be feasible in delivering a real-time video stream from the UAV payload camera to the GCS. Additionally, experiment data was collected of upper bound and lower bound video latency to provide best case and worst case scenarios for wired and wireless network implementations. In summary, these experiments concluded that the proposed wireless implementation in the UAV application was feasible and provided quantifiable results to show best case and worst case scenarios for the wireless implementation.

## ACKNOWLEDGEMENTS

Above all, I give all the Honor and Glory to my Lord and Savior, Jesus Christ, for the many blessings He has given me.

I am forever grateful to my major professor, Dr. David A. Umphress, for all of his insight, encouragement, and guidance during my graduate studies. I would also like to take this opportunity to thank Dr. Richard O. Chapman and Dr. David A. Umphress for providing the funding which made my research and graduate studies possible. I would also like to express my gratitude to my advisory committee members, Dr. Richard O. Chapman, and Dr. James H. Cross, II, for their participation on my advisory committee and their impact on my education as both an undergraduate and graduate student.

Finally, I would like to thank my entire family for supporting and believing in me throughout my college career.

| | |
|---|---|
| Style manual or journal used | ACM Computing Survey |
| Computer software used | Microsoft Word 2003 |

TABLE OF CONTENTS

LIST OF FIGURES

# CHAPTER 1 PROBLEM DESCRIPTION

The use of Unmanned Aerial Vehicles (UAV) in a battlefield scenario allows for information to be collected and decisions to be made based on a video signal that is transmitted from the vehicle to soldiers on the ground. The information gathered by the UAV provides soldiers with a real-time video stream to carry out reconnaissance and surveillance missions that potentially can save the lives of troops in harms way. Additionally, the UAV excels in providing valuable intelligence to commanders about high-value battlefield targets. Unmanned Aerial Vehicles are an indispensable weapon to the United States Army because of all of the information that can be gathered and acted upon with speed and accuracy.

Every mission performed by the UAV relies on the real-time video stream to have minimal end-to-end delay—referred to as video latency—for the system to operate properly. The larger the delay from the payload camera to the soldiers on the ground, the more difficult it is to operate the camera. In fact, if the delay is too high between the UAV and its Ground Control Station (GCS) then the control of the payload camera will be infeasible because the UAV will have flown past the area of interest before the ground control station operator is able to train the camera on a target. In order to provide accurate and quick information to troops stationed at the GCS the end-to-end delay between the UAV and GCS must be kept to an absolute minimum.

Our research focuses on determining the feasibility of a network architectural change in the way in which the Unmanned Aerial Vehicle communicates with the GCS, while still providing a low latent, time sensitive video stream from the UAV payload camera to the GCS. For our study, the UAV that was used was the Shadow 200 Tactical Unmanned Aerial Vehicle (TUAV).  The Shadow 200 TUAV communicates with soldiers on the ground with the GCS via intermediate control systems.  One of the intermediate control systems is the Ground Data Terminal (GDT). The GDT relays communications between the Shadow 200 TUAV and the GCS.  The second intermediate control system is the TUAV Automated Landing System (TALS).  TALS is used to automatically land the Shadow 200 TUAV without any operator interaction. The following two diagrams show the current implementation of the Shadow 200 as well as the implementation proposed by our solution.

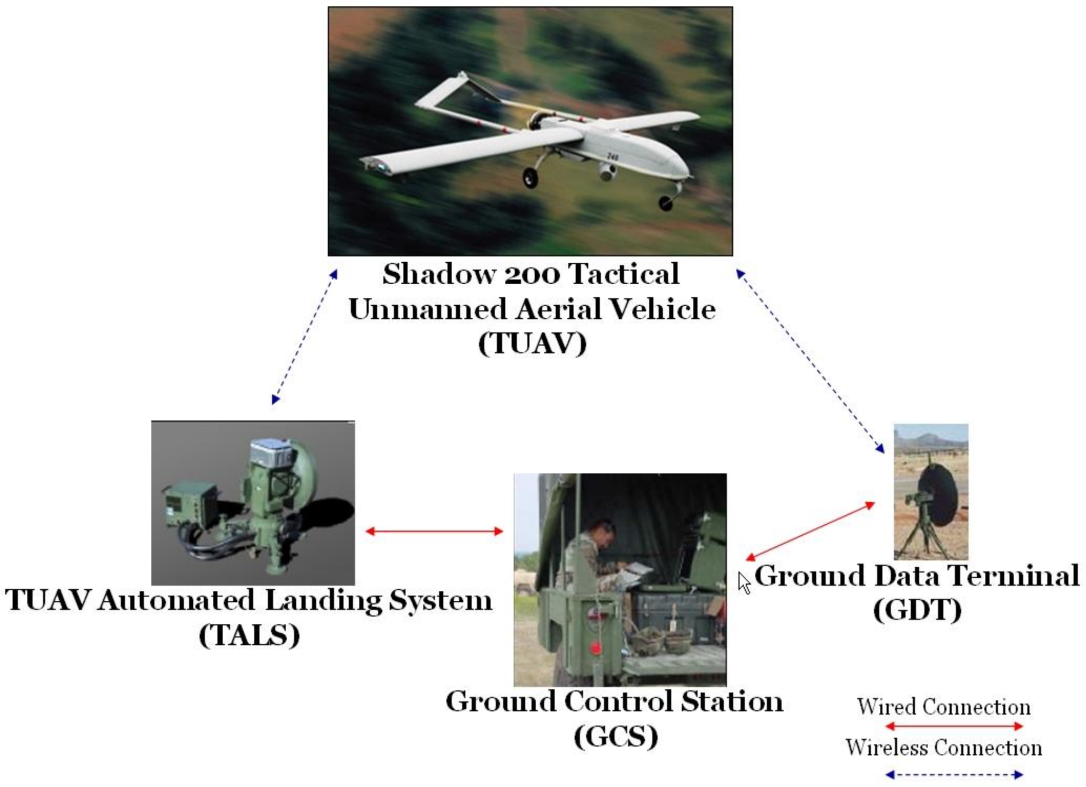

Shadow 200 Tactical Unmanned Aerial Vehicle (TUAV)
TUAV Automated Landing System (TALS)
Ground Control Station (GCS)
Ground Data Terminal (GDT)
Wired Connection
Wireless Connection

Figure 1-1 Current Shadow 200 TUAV Architecture

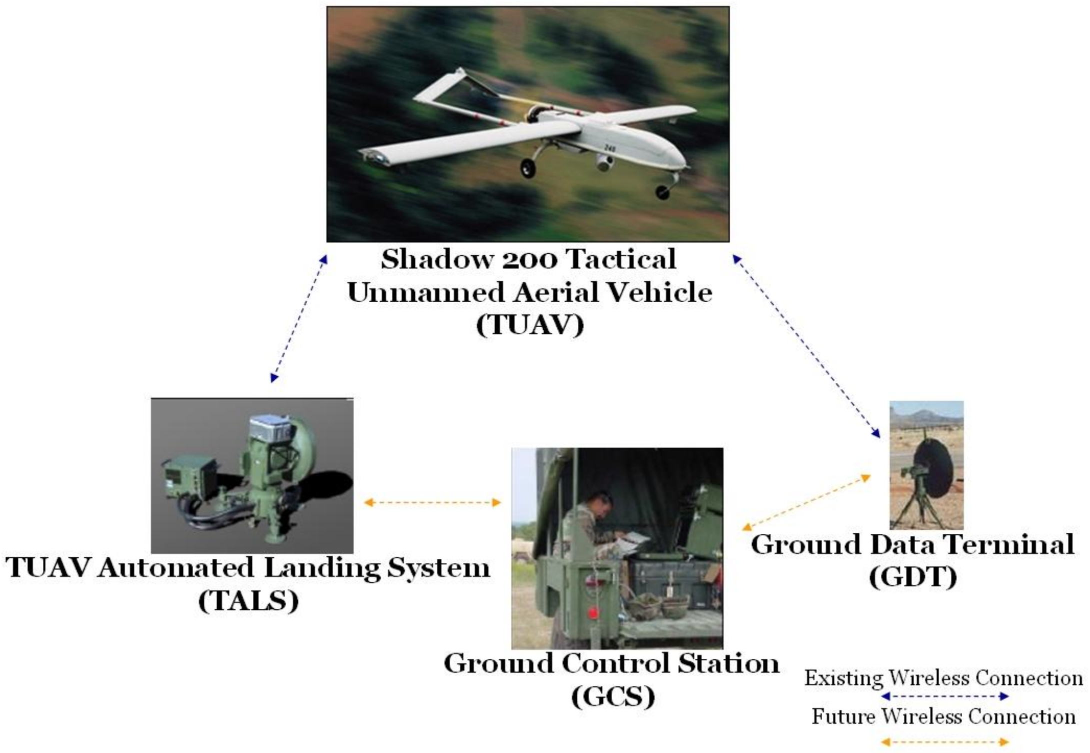

Shadow 200 Tactical Unmanned Aerial Vehicle (TUAV)
TUAV Automated Landing System (TALS)
Ground Control Station (GCS)
Ground Data Terminal (GDT)
Existing Wireless Connection
Future Wireless Connection

Figure 1-2 Proposed Shadow 200 TUAV Architecture

Figure 1-1 depicts the original UAV architecture with the GCS and intermediate control systems linked with wired connections. Figure 1-2 shows the proposed solution with the UAV communicating through the intermediate control systems to the GCS with wireless connections.  The network architectural change was brought about as a result of having to manage thick cables that interconnect the Unmanned Aerial Vehicle and GCS with the TUAV Automated Landing System and the Ground Data Terminal. These thick cables are susceptible to harsh conditions in battlefield environments and act as a failure point when large trucks, tanks, and other vehicles run over them. In addition, if those responsible for the UAV come under attack from an enemy, the thick cabling is one other piece of equipment that must be hurriedly disconnected and abandoned while taking cover. Our research centers on solving these issues with a wireless connection link while still providing seamless integration of components that can provide a low latent real-time video stream.

The focus of our research entails examining video latency as it relates to UAV applications. The main portion of the research involves measuring end-to-end video latency using various encoders and decoders and network media. The goal of the research is to identify upper and lower bounds on video latency for video streams typical of UAV transmissions.

# CHAPTER 2 BACKGROUND

In our bandwidth-constrained systems we must rely on video compression to deliver a video signal from an originating node on a network to the destination node. A video sequence consists of displaying multiple images per second. The sequence of images, when uncompressed and displayed, forms a video sequence. Without video compression, we must use large amounts of bandwidth and network resources to send the video sequence across a network. Fortunately, video sequences have characteristics that can be exploited to allow video sequences to be compressed and sent on a bandwidth-constrained network. Characteristics that can be taken advantage of include redundancy within a single frame, redundancy between consecutive frames, as well as shading nuances that the human eye can't detect. There are additional properties that video compression standards can incorporate, but the key motive behind each technique is the reduction in bandwidth resources.

## 2.1 Video Coding

As stated above, a video sequence is composed of a series of images. Each of these images is referred to as a frame. Every frame is categorized as one of the following three types: an intra-coded frame (I-frame), a predictive-coded frame (P-frame), or a bidirectional-coded frame (B-frame) [Tekalp 1995]. Intra-coded frames are independent of any other frame and are compressed using an algorithm similar to JPEG image compression. I-frames are coded without reference to any other frame in the sequence, and this yields the worst compression when compared to B-frames and P-frames. Predictive-coded and bidirectional-coded frames are more highly compressed because of their reference to previous or future frames. P-frames are more highly compressed than I-frames because they use information from a past I-frame or P-frame. B-frames achieve an even higher compression by referencing a preceding I-frame or P-frame as well as a future I-frame or P-frame.

Each frame is divided into smaller units. These units are referred to as macroblocks [Tekalp 1995]. Macroblocks contain an array of luminance pixels and chrominance pixels that form the cornerstone of video encoding. Splitting frames into smaller parts allows for increased compression based on the inherent redundancies in video sequences.

A series of frames are linked together to form a Group of Pictures (GOP) [Mitchell et al. 2002]. A GOP contains an I-frame that is used as an access point for the video decoding software to begin decoding the video sequence while the rest of the GOP contains P-frames and B-frames.

## 2.2 Spatial and Temporal Redundancy

Groups of Pictures, frames, and macroblocks are the basis for two video coding techniques, spatial and temporal redundancy. Spatial redundancy is a technique associated with video compression when attempting to compress a single frame in a video sequence [Torres, L and Kunt, M 1996]. A single video frame is often composed of similar parts. For example, a frame may be composed of a person standing in front of a green background.  This frame yields similar colors and patterns throughout the entire image, such as the green background or the person’s clothing. Splitting the frame into smaller parts and exploiting the redundant pattern and colors allows for efficient and increased compression of a single video frame.

Another technique employed by video compression algorithms is the use of redundancy for sequential video frames by using prediction algorithms. This technique is known as temporal redundancy. It allows a video frame to be highly compressed based on the knowledge of earlier frames or subsequent frames [Torres, L. and Kunt, M. 1996]. For instance, a television newscast will often show a commentator sitting at a desk and reporting the latest news. This scene has very little motion and most of the frames in this video sequence are an exact copy of previous frames. Since video sequences are often characterized by this type of temporal redundancy, video compression can take advantage of this by not separately encoding each frame. Rather, frames are thought of as copies or slight variations of a previously encoded frame.

## 2.3 YUV Color Space and Chroma Subsampling

The human eye is another area that video compression algorithms can exploit to increase compression. Video compression algorithms increase compression by either reducing or completely eliminating the data that is not detectable by the human eye. They do this by taking advantage of what is called the YUV color space. In the YUV color space, the Y component represents the luminance value while the U and V components represent the chrominance values [Westwater and Furht 1997]. Luminance conveys the grayscale values of the image, whereas chrominance conveys the color information. The human eye is more sensitive to the luminance value than the two chrominance values. Therefore, a technique known as “chroma subsampling” is implemented in video compression to reduce the number of chrominance values that are sampled [Feng 1997].

Since the human eye is not as sensitive to the chrominance components, the video compression algorithm can represent the same image with less information.

## 2.4 Moving Picture Experts Group (MPEG) Overview

The Moving Picture Experts Group (MPEG) is charged with developing “international standards for compression, decompression, processing, and coded representation of moving pictures, audio and their combination” [Moving Picture Experts Group]. Popular standards to date include MPEG-1, MPEG-2, and MPEG-4. The focus of this research is the MPEG-4 standard, but we will examine each of the standards to show the progression of the MPEG standards and the similarities and differences among each of the MPEG standards.

MPEG-1 was standardized in the early 1990s for use with storage media, such as compact discs and other optical drives that had the channel capacity of 1.5 Mbps. This standard was optimized to be encoded at a bit-rate of 1.15 Mbps using a resolution of 352 x 240 at 29.97 frames per second to yield VHS quality video [Morris 1995]. MPEG-1 utilizes I-frames to complete the intra-coded frames and P-frames, and B-frames are used to represent the inter-frame coding. Other features of MPEG-1 include: random access, fast forward and reverse search of the bit stream, flexible frame rate, and variable image size [Tekalp 1995]. This standard was the first in the line of successful multimedia standards. Each subsequent standard developed by MPEG uses MPEG-1 as a building block for the key components of the video compression.

MPEG-2 is the immediate successor to MPEG-1. It was developed as a standard for an increased bit rate over MPEG-1. MPEG-2 supports the interlaced format in addition to the MPEG-1 progressive format. Other features of MPEG-2 include improvements to the quantization and coding options, and alternative sub-sampling to the chroma channels [Tekalp 1995]. Additionally, MPEG-2 offers scalable video coding by encoding a base layer bit stream and enhanced layer bit stream. The base-layer bit stream is an independent bit stream which is capable of producing a lower quality version of the video signal. The lower quality version of the base-layer bit stream is produced by using a reduced set of data to represent the video signal. For example, the temporal property of MPEG-2 relies on a smaller resolution of the video signal while the enhanced layer uses all of the necessary information to produce the higher quality video. MPEG-2 is also well known for its use in DVD media, digital television, and high definition television. MPEG-2 produces bit-rates from 4 Mbps with standard definition television to a higher bandwidth of 25 Mbps for high definition television [Morris 1995].

MPEG-4 was developed as a low bit-rate video compression standard in the late 1990s and became an international standard in 1999. MPEG-4 provides many additional features that extend some of the features present in the previous versions of MPEG; namely, streaming MPEG-4 has become more efficient over the previous MPEG standards. MPEG-4 can deliver the same quality with a smaller file size and reduced bandwidth.

## 2.5 Video Latency Overview

Regardless of the standard or video coding techniques employed, time is required to compress, encode, and transmit the video image. This delay, called video latency, is the amount of time that elapses for a frame of video to be processed by a source until it reaches the video destination and is decoded and displayed. Five main areas contribute to the total amount of latency of a video signal from the video source to the video destination including: video compression, kernel IP transmission stack, kernel IP receiving stack, network architecture, and video decompression [MPEG4IP].

The first source of video latency is the video compression software that uses a NTSC video signal as an input. Video compression standards such as the MPEG standard do not set forth any specific guidelines pertaining to the video encoder. For example, the MPEG standard only requires conformance of a specific syntax of the bit stream and MPEG decoder. This allows for freedom in the compression algorithms that are used by the encoder to increase performance, lower overhead, and any other advances in video encoding technology. Therefore, the video compression software has a pivotal role in the amount of latency that is imposed on the video stream.

Next, the kernel IP transmission stack at the video source and kernel IP receiving stack at the video destination are two other areas where video latency can occur. All network traffic must be taken into account when examining areas where latency occurs. Other applications transmitting packets or incoming network traffic will be handled by either the kernel IP transmission stack or the kernel IP receiving stack. A bottleneck in the transmission or receiving of packets has a direct impact on the latency of the video stream.

The fourth area that can influence video latency is the networking environment. Different factors associated with a network can cause delays in the video signal reaching its destination. These types of factors include: packet errors, network utilization, network infrastructure, and other factors that affect the delivery of packets across the network. One key area that we will examine further in our research is the differences in delivering video across a wireless network link as compared to a wired network.

The final area that can influence the latency of a video stream is the video decoding software at the video destination. Similar to the video encoding software, the video decoding software at the video destination must also be efficient in its implementation of the video decoding standard. If the software is fast and efficient at decoding the video stream and outputting the result to the display buffer, then we can continue to ensure minimal latency if all of the other factors produce very little latency.

## 2.6 Literature Review

Previous work that has focused on quality issues relating to real-time video streaming of MPEG-4 video signals is the work of Shawn Constance. This work focused on evaluating the quality of a streamed video sequence using a No-Reference Blockiness Metric (NRBM) [Constance 2005]. The No-Reference Blockiness Metric quantified defects in a transmitted video signal by detecting errors in the block edges. This work implemented the No-Reference Blockiness Metric in software as a modification to the video decoding software. By using the modified video decoding software the No-Reference Blockiness Metric could be evaluated by running experiments to determine the effects caused by errors in the video stream, such as encoder quantization or packet loss. The unique feature of this research is that the quality of the video stream could be measured “on-line” by the NRBM software functions detecting blurriness within a frame as well as detecting changes in pixel values across adjacent block boundaries. In addition, this work also explored video latency with two different experiments. First, a study was performed by real-time streaming a Pac-Man® game to visually inspect latency.  Next, a quantitative approach was taken by encoding visual timestamps at the video sender and decoding the timestamps at the video destination.  Both the video sender and video destination utilized the Network Time Protocol as a synchronization method to gain an accurate account of latency. Our research expands on the work of Shawn Constance by taking a more in-depth look at video latency with MPEG-4 in an Unmanned Aerial Vehicle environment by utilizing specialized hardware to provide time synchronization and signaling for an expanded view of end-to-end latency.

Another area of research for end-to-end delay includes examining the impact of signal delay in UAV systems that are used for targeting areas of interests on the battlefield or carrying missile payloads. Dougherty, S., Hill R.R., and Moore, J.T. [Dougherty, S , Hill R.R., and Moore, J.T 2002] explored the effects of signal latency with a UAV by establishing a simulation to determine latency of the GCS and UAV signal and control connections. Their paper focused on UAV applications that include UAV weapon payload and target designation capabilities. These applications rely exclusively on the dependability and the robustness of the UAV systems to provide accurate and low latent information for the UAV application to work properly. This work examined a UAV communicating with various satellite communications and the impact on the signal latency with a change in satellite communication, such as geosynchronous orbit satellite communication, medium earth orbit satellite communications, and low earth orbit satellite communications.  A simulation model was constructed to directly measure the miss distance by keeping track of the actual target's location with the perception of the UAV operator. The different in these two values determined the miss distance based on what the UAV was displaying and what the actual location of the target was. This study concluded that larger latencies in the UAV system contribute to large miss distances. Additionally, the velocity at which the ground target is traveling had a direct impact on the miss distance.

Karr, D.A., Rodrigues, C., Loyall, J.P., and Schantz, R.E. [Karr, D.A. et al. 2001] examined a framework for UAV applications that met the needs of a variety of users in the UAV system and adapted to a variety of scenarios specific to a UAV application. This study looked at the need for UAV applications to handle limited resources, deal with a variety of users accessing functions of the UAV, and looking at Quality of Service (QoS) issues associated with delivering a time sensitive video sequence. Video latency of the framework was compared to the latency values without the framework. They concluded that by using their proposed framework, system resources were superior and this allowed for lower video latency because system resources were not occupied with other tasks while delivering the time sensitive video stream.

# CHAPTER 3 SOLUTION

The delay of the video signal in UAV applications was measured with a variety of experiments. The objective of these experiments was to determine end-to-end delay incurred with different network topologies using different video encoders and video decoders. The latency experiments focused on determining an upper bound delay and lower bound delay to determine best case and worst case scenarios for video latency. The overarching goal of the research was to determine how video latency is affected by replacing a wired connection link with a wireless connection link between the GCS and intermediate control systems. By observing these effects we can also determine if the wireless connection link will be a valid solution to the problems of the wired connections in the United States Army Shadow 200 TUAV program.

## 3.1 Latency Experiments

The first series of experiments that were completed examined lower and upper bound end-to-end delay. These two experiments were formed based on the implementation of the MPEG standard. The lower bound experiment was calculated by creating a scenario that imposed a light load on the encoder. In order to force a lighter load on the encoder, the generated video sequence included mostly static sequences with only small changes in the video sequence. Based on the MPEG standard, the encoder is taxed more heavily when there are rapid changes to sequential frames in a video sequence; therefore, by limiting the load placed on the encoder, a lower bound can be established assuming all other variables in the system stay constant. All of the low latency scenarios were captured using both wired and wireless network configurations to capture differences in network infrastructure.

On the other hand, upper bound end-to-end delay was calculated by taxing the video encoder with random noise patterns to increase the time needed to encode the video with the video encoding algorithms. As compared with the lower bound experiments, the MPEG video encoder was forced to handle completely different and dynamic frames that composed the video sequence. The video sequence used random noise patterns as the background to each frame in order to create constantly changing images to stress the MPEG encoder. The random noise pattern was chosen to tax the encoding the most because of its random and unpredictable patterns. The MPEG standard uses previous frames to improve the efficiency of the encoding; however, random noise patterns do not lend themselves to containing information from frame to frame that would allow the encoder to rely on previous frame information to estimate the next frames in the video sequence. The upper bound end-to-end delay and the lower bound end-to-end delay were tested on both a wireless network configuration and wired network configuration. These experiments added an extra layer to compare the lower bound and upper bound across two different network topologies.

Next, baseline experiments were performed to account for typical video sequences in the UAV system. These experiments were representative of a typical video sequence from the payload camera by displaying frames captured from the Shadow 200 TUAV simulator. These video transmissions of the Shadow 200 TUAV were also observed using a wireless and wired network configurations. By comparing the two network configurations we are able to observe one of the main goals of determining feasibility with wireless technology in the UAV infrastructure. Additionally, the research was able to compare the UAV transmission latency calculations with the lower bound and upper bound latency values to characterize every scenario.

Experiments were also conducted with various video coding technologies. Since video coding standards only require conformance of a specific syntax of the bit stream and decoder, the video encoding software is important to examine any differences in video latency. By conducting experiments with a variety of software encoders we were able to analyze different implementations as well as different encoding standards including: MPEG-4 and H.264. All of these scenarios created in conjunction with the two video coding standards created a wealth of data. All of the data is important in characterizing video latency and recognizing trends in different video coding technologies and scenarios.

## 3.2 Network Topology

Another important aspect to each of these experiments was the different network topologies that were tested with both high latency scenarios and low latency scenarios. The primary motivation behind the changes in network topology is to examine the feasibility of replacing wireless connection links instead of wired connections in Unmanned Aerial Vehicle applications. By comparing wired and wireless scenarios, we were able to explore differences in network topology as well as explore the characteristics in both wired and wireless scenarios.

Using the wired and wireless connection scenarios, the experiments used four different network topologies. Both the wired and wireless connection scenarios used a

Cisco security appliance to ensure and certify the security of the UAV application. Additionally, the last two network topologies examined wired and wireless configurations without the Cisco network security appliance. These four network topologies in conjunction with the lower bound latency experiments and higher bound latency experiments create numerous data points that allow the video encoding scenarios to be analyzed against the four network topologies.

# CHAPTER 4 SOLUTION VALIDATION

The delay in the propagation of the video signal from the UAV to the GCS is essential in determining a successful solution of providing a low latent video signal with wireless connection links. The solution to examining the end-to-end delay between the UAV and the GCS is mapped to three components which emulate the functions of the entire UAV architecture. In addition to these components, different pieces of software and hardware are used to assist in capturing an accurate representation of video latency. By utilizing this architecture, we were able to appropriately model and simulate all of the facets of the UAV system to evaluate various scenarios including: worst case latency, best case latency, and a comparison of network infrastructures.

## 4.1 System Overview

Figure 4-1 shows the three key components, the role each component has in the system, as well as the interactions between each of the three components.  The next figure, Figure 4-2, shows the actual configuration in the research laboratory.

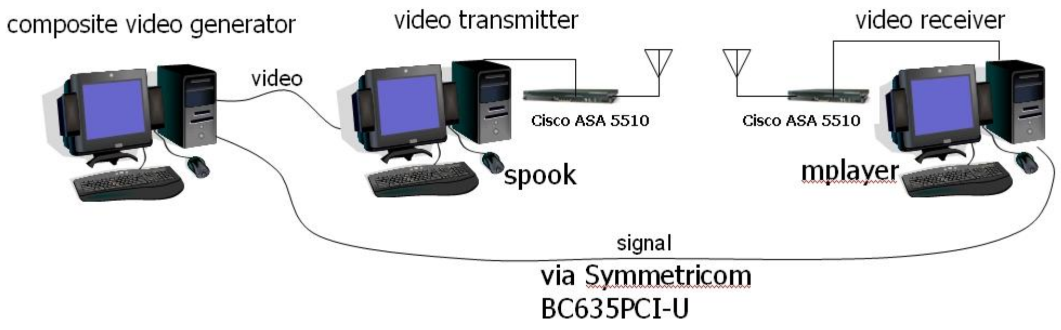


Figure 4-1 System Overview

Figure 4-2 Video Transmitter, Video Receiver, and Video Generator System Setup

The first component in the system is the video generator. The role of the video generator in the system is to emulate the payload camera attached to the UAV.  The video generator outputs a standard NTSC video signal that is transmitted via a composite video cable to the video input at the video transmitter. The video generator contains software that outputs various video sequences including: random noise or captured video sequences from the Shadow 200 TUAV simulator. The video generator can display a video sequence that would be representative of video found in a real scenario with the

Shadow 200 TUAV.  The output of the video generator can also impose a static background to produce low latency conditions or random, dynamic images to force higher latency conditions. One other crucial component of the video generator is the time signaling hardware. The time signaling is handled at the video generator with a Symmetricom PCI card. The PCI card at the video generator interfaces with the second Symmetricom PCI card at the video receiver.  Figure 4-3 depicts the Symmetricom bc635PCI Time and Frequency PCI hardware utilized by both the video generator and video receiver. Figure 4-4 illustrates the interface cable used to interconnect the Symmetricom hardware of both the video generator and video receiver.

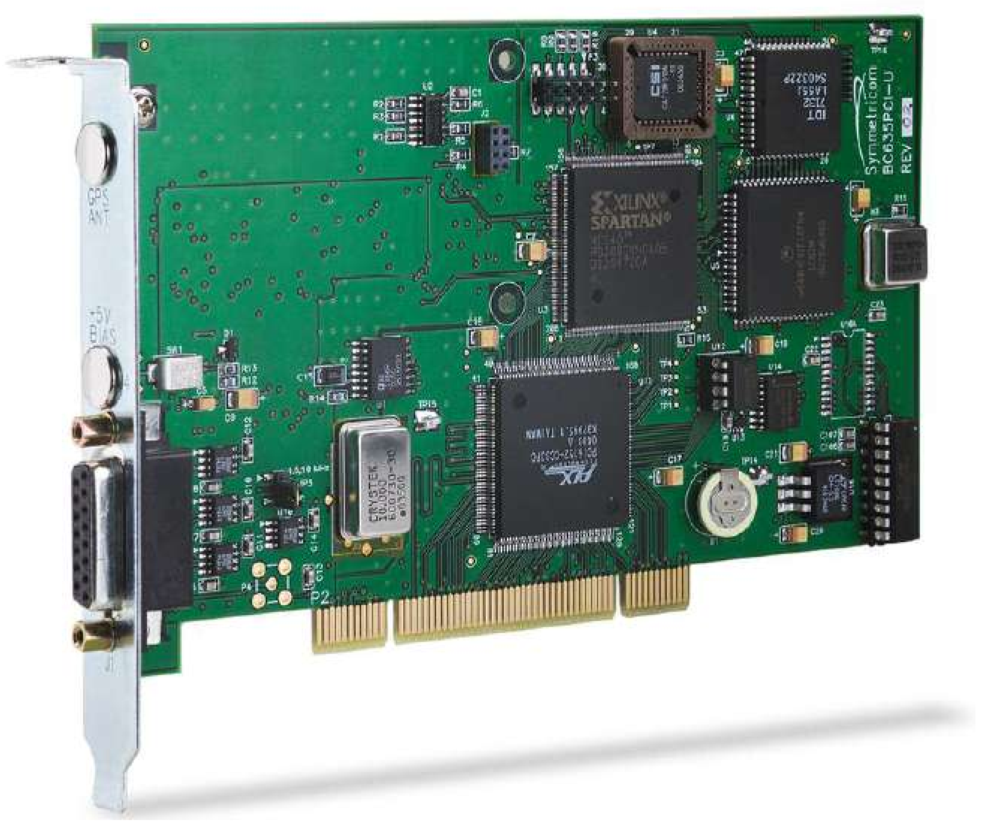


Figure 4-3 Symmetricom bc635PCI Time and Frequency Processor

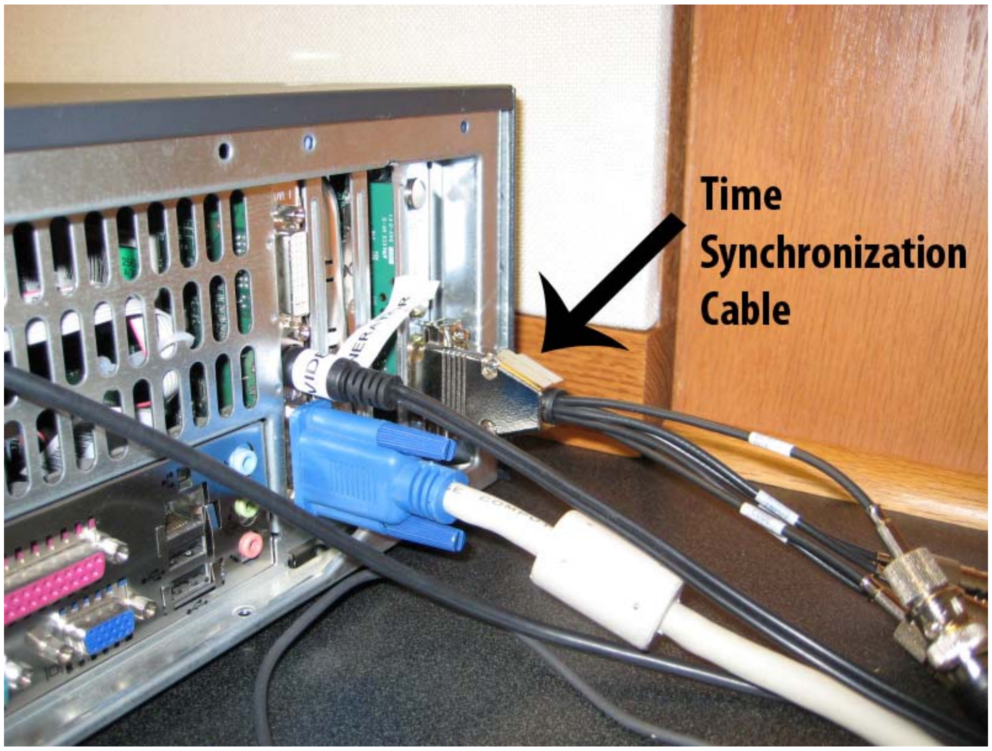


Figure 4-4 Symmetricom Time Synchronization Cable

Time signaling is an important part of the latency experiments with the video generator because it ensures accurate latency data by signaling an exact time value of the encoded video frame. This signal is captured by the second Symmetricom PCI card at the video receiver and stored to compute the latency. Without the time signaling, the video latency experiments could not ensure an accurate measurement of the end-to-end delay.

The video transmitter is the next component in the system. The video transmitter accepts the analog video input from the video generator, and the video signal is digitally encoded with MPEG-4 video compression. The video compression is handled with software using a program called Spook. Spook interfaces with an Osprey video capture card and streams the video signal across the network in real-time. Spook is a real-time MPEG-4 streaming solution that can be customized to accept multiple bitrates and different encoding standards. In addition to Spook, mp4live was also used at the video generator to observe any changes to latency with a different video encoder.

The final component of the system is the video receiver. The role of the video receiver is to establish a connection with the real-time video steam and emulate the Ground Control Station of the UAV architecture. This component also includes synchronization and signaling hardware to coordinate the clock of the video generator and video receiver. The video receiver uses a MPEG-4 video player, MPlayer, to capture and decode the real-time stream from the video transmitter. In addition, MPlayer was modified to include a custom filter that handled the calculation of end-to-end delay. This filter interfaced with the Symmetricom time synchronization PCI card to store the time the video was encoded at the video generator by receiving the propagated signal from the Symmetricom PCI card at the video generator. After the filter stored the signaled time value of the created frame, it continuously decoded frames until it recognized the frame that was originally signaled. Once it was decoded, the filter requested the current time. The latency value was then computed by the filter using a subtraction of the current time with the stored time it received from the Symmetricom time synchronization PCI card at the video generator. These steps were repeated until a stop frame was detected at the video receiver.

## 4.2 Time Signaling

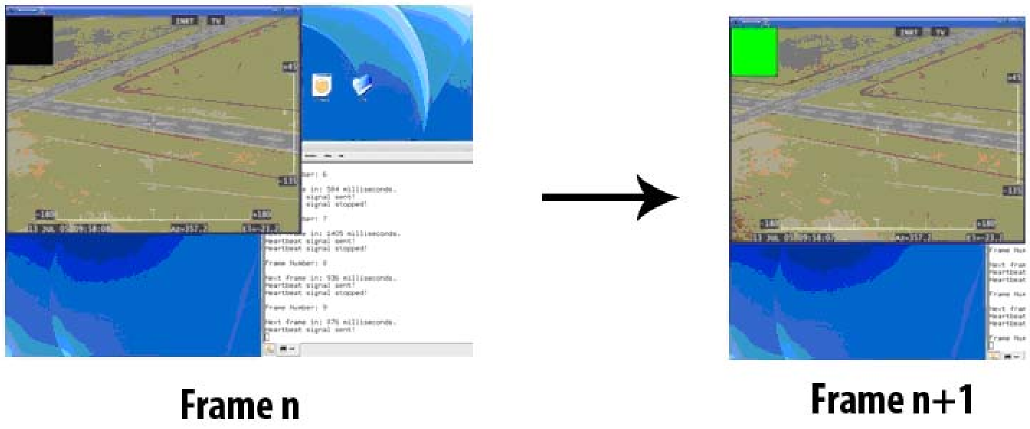
Time signaling is the most integral component in determining the latency in the system. Time signaling was accomplished by using two Symmetricom bc635 Time and Frequency PCI cards, the first card was installed in the video generator and the second card was installed in the video receiver. These cards are used in applications that require the synchronization between two computers that rely on a stable and synchronized clock. Our initial design included utilizing the Symmetricom PCI cards to provide synchronization between the video generator and video receiver; however, the design of the system evolved into using the Symmetricom PCI cards as an interface for the video generator to signal the video receiver when a frame was being created. In order to provide accurate end-to-end latency data, the Symmetricom cards must not incur changes in the clock or large delays while performing a read operation on the current time of the card. Since the Symmetricom time synchronization PCI cards do not incur large delays while reading the current time or detecting the time signal from the video receiver, this is an acceptable use for calculating end-to-end video latency. As a frame was created, the Symmetricom card at the video generator signaled the Symmetricom card at the video receiver to signal a newly created frame.  Once the frame was received, the video receiver could perform a subtraction of the current time and the signaled time to determine the end-to-end delay. In order to provide for the signaling, the video generator used the heart beat signal output of the card as a signal. The heart beat signal could be signaled at any point in time and acted as the signal for a generated video frame. The time signal was captured by using the event input at the video receiver. Once a heart beat signal was transmitted and recognized on the event input, an interrupt was triggered and this signified a newly created frame. In the interrupt service routine, the time value was stored to be used to calculate the delay from when the frame was created until it was decoded.

## 4.3 Overview of Software Components

Various software programs were used in our experiments of video latency. First, the video generator was composed of a custom application based on the Qt GUI toolkit. The Qt application was used to emulate the UAV payload camera by displaying images captured from the Shadow TUAV simulator. Additionally, random noise was generated to create a scenario for increased video latency on the system. Other experiments were slight variations of the random noise experiments to determine an upper bound on the amount of latency incurred by the system. The Qt application provided flexibility in creating low latency, high latency, and simulated Shadow 200 TUAV video sequences. The video transmitter relied on an application called Spook. This application provided a RTSP streaming solution that encoded the video input from the video generator using MPEG-4. The final piece of software used in the system was a video decoding application. The video application was based on MPlayer with modifications to interact with the Symmetricom Time and Frequency library. A custom filter was added to the MPlayer software to detect the incoming frames and determine the latency in the system.

## 4.4 Implementation of Experiments

Each experiment followed the same sequence of events and each system component communicated in the same way. The only modification to the system with each experiment was a newly created video sequence at the video generator or a change in video encoding software at the video transmitter. The newly created video sequence was modified with each experiment to create high latency conditions, low latency conditions, or baseline conditions typical of UAV transmissions. The video encoding software was modified throughout the experiments to measure differences between video encoders. The video sequence was composed of a series of images with a color box that was overlaid and changes colors to represent a newly created frame. This was used as a visual marker for the video decoder to recognize a new frame and compute latency based on the time the frame was created and when the frame was decoded. Figure 4-5 shows a typical UAV transmission with the overlaid color boxes. The first frame is represented by the black color box and then the second frame is rotated to a green color box. This pattern is repeated until the end of the simulation. Figure 4-6 and Figure 4-7 displays the use of the random noise patterns with the overlaid color boxes.



```
ber: 6

e in: 584 milliseconds.
 signal sent!
 signal stopped!

ber: 7

e in: 1405 milliseconds.
Heartbeat signal sent!
Heartbeat signal stopped!

Frame Number: 8

Next frame in: 936 milliseconds.
Heartbeat signal sent!
Heartbeat signal stopped!

Frame Number: 9

Next frame in: 376 milliseconds.
Heartbeat signal sent!
```

Figure 4-5 Video Generator Sequence

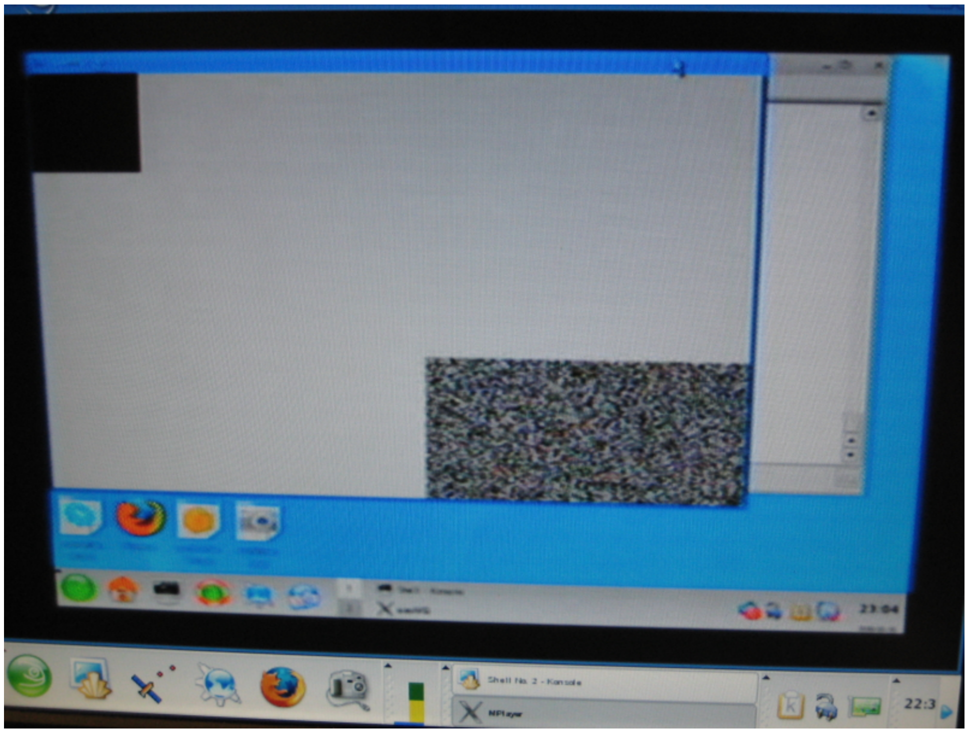


Figure 4-6 Random Noise Pattern with Black Color Box

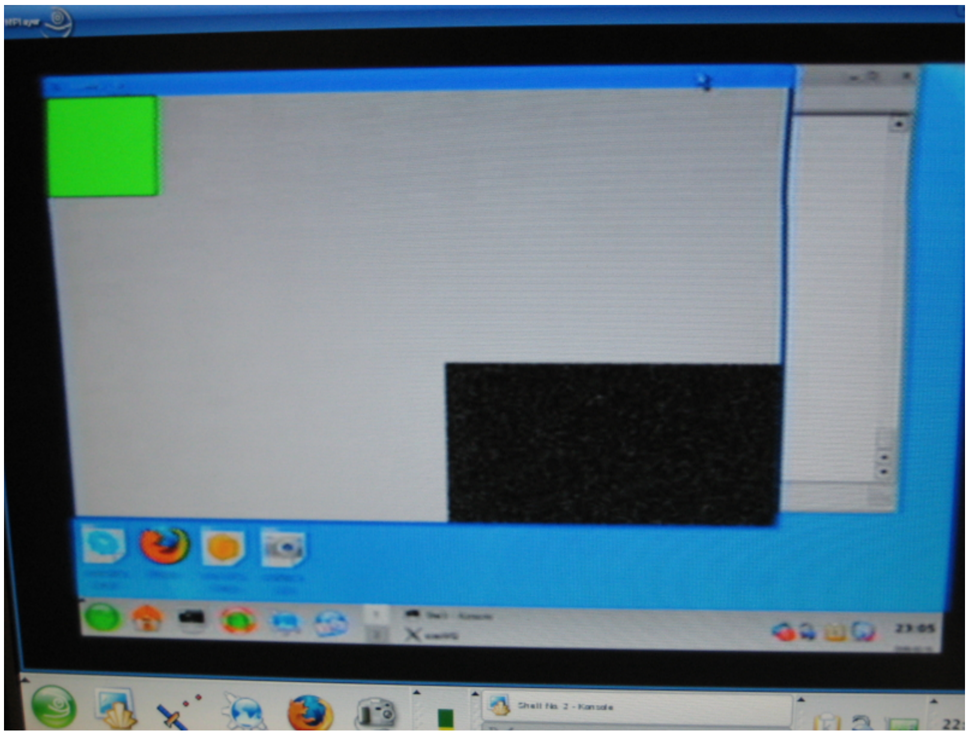


Figure 4-7 Random Noise Pattern with Green Color Box

The following steps occurred in each experiment to determine the video latency:

1. The video sequence was started based on the conditions of the experiment (low latency, high latency, or typical UAV transmissions).
2. Once the initial experiment conditions were created, the video sequence was generated based on the use of low latency, high latency, or typical UAV transmissions. The generated video frame was created based on these conditions and then a green box and a black box were alternately superimposed on each frame. The color box was constant in terms of its location on the frame across all experiments and the only variable of each frame in the video sequence was the generated background. The

generated background was based on the experiment conditions and includes dynamic, random noise patterns, or static backgrounds. The complete random noise patterns were used for generating high latency experiments while the static backgrounds were used in low latency experiments. Additionally, we measured typical UAV video sequences by incorporating simulated UAV payload camera imagery as the background of the video sequence.

3. Once each frame was created it was displayed and sent to the video transmitter via a composite video cable. As each new frame was displayed, the video generator signaled the video receiver with the Symmetricom PCI card. Once the video receiver received the signal it recorded the time.

4. The video transmitter received an analog video signal and digitally encoded and transmitted the video signal with a MPEG-4 stream across the network.

5. The video receiver used video decoding software to connect to the real-time stream and decode the frames. Once the frame was decoded and a new color box was detected the time value was captured from the Symmetricom PCI card and compared to the value it stored when the video generator signaled a newly created frame. This time difference accounts for the video latency.

6. After the simulation was complete, a histogram of latency values is displayed.

The first experiment, no noise, was the lower bound latency experiment. The goal of this experiment was to create the least amount of work for the video encoding software. Based on the details of MPEG encoding we created an experiment that would cause the least amount of work for the video encoder. This scenario was completed by displaying a static background that never changed. The only change in the scene was the color box that signified a change in frame. By using a static background with only a small portion of the screen changing we can ensure that the MPEG encoding is not burdened by motion estimation and intensive prediction algorithms.

The next experiments focused on creating an upper bound on the latency. One attempt to determine the upper bound was completed using a random noise background. This background was rotated with different random noise patterns to cause the MPEG encoding algorithms increased load. By rotating a background image with different random noise patterns, the experiment attempted to increase the amount of encoded time by causing the algorithms increased work with the estimation and motion algorithms. A variation on this experiment, block noise, involved rotating a random noise background with a random floating position. The goal of this experiment was to include a large amount of random noise while also varying the positions on the screen to force more load on the video encoding. A final variation on this experiment, macroblock noise, included a noise pattern based at the macroblock layer.

The upper bound and lower bound experiments were conducted with both a wired network configuration and wireless network configuration. This allowed us to examine any differences associated with the real-time stream with different network media. Also, these experiments were completed using two separate video encoders, Spook and mp4live, to examine different implementations of video encoding standards.

Each experiment consisted of 1,400 samples being displayed at the video generator and broadcast from the video transmitter to the video receiver. This allowed for over a twenty minute experiment since the interval of new frames was generated based on a pseudorandom number generator to determine the newly generated frame samples. The video receiver typically successfully received and recorded over 98% of the transmitted samples. The successful recording rate dropped significantly with large end-to-end latency values since larger values were ignored.

## 4.5 Experimental Results

The latency experiments provided insight into differences in network media, video encoding software, and video encoding standards. Based on the results of these experiments some conclusions can be drawn based on the experiment results. Figure 4-8 displays the results of the wired and wireless experiments using the Spook video encoding software. Figure 4-9 shows the results of the wired and wireless experiments using the mp4live video encoding software. Furthermore, Figure 4-10 records the results of the H.264 FastVDO video encoding software. The latency experiments recorded in Figure 4-10 were conducted only with reduced load on the video encoder due to the extremely high latency with the FastVDO encoder. Therefore, we will refer to this set of experiments in Figure 4-10 as Reduced Load Experiments. Figure 4-11 illustrates the end-to-end latency when the Cisco ASA network security appliance is factored into the overall system architecture.

| **Video Latency Average & Standard Deviation (Spook)** | | | | |
|---|---|---|---|---|
| **Wired** | | **Average (ms)** | | **Standard Deviation** |
| No Noise | | 127 | | 25.902 |
| Macroblock Noise | | 127 | | 23.096 |
| Block Noise | | 165 | | 28.261 |
| Full Frame Noise | | 142 | | 30.035 |
| | | | | |
| | | | | |
| **Wireless** | | **Average (ms)** | | **Standard Deviation** |
| No Noise | | 131 | | 25.844 |
| Macroblock Noise | | 133 | | 25.668 |
| Block Noise | | 171 | | 30.356 |
| Full Frame Noise | | 151 | | 30.339 |
| | | | | |
| | | | | |
| **Difference** | | **Average (ms)** | | **Standard Deviation** |
| No Noise | | -4 | | 0.059 |
| Macroblock Noise | | -5 | | -2.571 |
| Block Noise | | -6 | | -2.095 |
| Full Frame Noise | | -9 | | -0.304 |

Figure 4-8 Video Latency Experiments with Spook Video Encoder

| **Video Latency Average & Standard Deviation (mp4live)** | | | | |
|---|---|---|---|---|
| **Wired** | | **Average (ms)** | | **Standard Deviation** |
| No Noise | | 116 | | 21.172 |
| Macroblock Noise | | 115 | | 22.012 |
| Block Noise | | 148 | | 23.755 |
| Full Frame Noise | | 121 | | 23.659 |
| | | | | |
| | | | | |
| **Wireless** | | **Average (ms)** | | **Standard Deviation** |
| No Noise | | 119 | | 18.744 |
| Macroblock Noise | | 118 | | 18.093 |
| Block Noise | | 145 | | 20.691 |
| Full Frame Noise | | 132 | | 29.255 |
| | | | | |
| | | | | |
| **Difference** | | **Average (ms)** | | **Standard Deviation** |
| No Noise | | -3 | | 2.427 |
| Macroblock Noise | | -3 | | 3.919 |
| Block Noise | | 3 | | 3.065 |
| Full Frame Noise | | -11 | | -5.596 |

Figure 4-9 Video Latency Experiments with mp4live Video Encoder

| Video Latency Average & Standard Deviation (FastVDO) | | | | |
|---|---|---|---|---|
| **Wired** | | **Average (ms)** | | **Standard Deviation** |
| Reduced Load Experiment 1 | | 344 | | 51.907 |
| Reduced Load Experiment 2 | | 342 | | 43.842 |
| Reduced Load Experiment 3 | | 365 | | 52.398 |
| **Average of Experiments** | | **350** | | **49.382** |
| | | | | |
| | | | | |
| **Wireless** | | **Average (ms)** | | **Standard Deviation** |
| Reduced Load Experiment 1 | | 353 | | 45.429 |
| Reduced Load Experiment 2 | | 386 | | 46.624 |
| Reduced Load Experiment 3 | | 357 | | 48.796 |
| **Average of Experiments** | | **365** | | **46.950** |
| | | | | |
| | | | | |
| **Difference** | | **Average (ms)** | | **Standard Deviation** |
| Wired v. Wireless | | -15 | | 2.433 |

Figure 4-10 Video Latency Experiments with FastVDO (H.264) Video Encoder

| **Video Latency Average & Standard Deviation** | | | | |
|---|---|---|---|---|
| **(Cisco ASA Security Appliance)** | | | | |
| **Wired** | | **Average** | | **Standard Deviation** |
| Experiment 1 | | 137 | | 19.350 |
| Experiment 2 | | 135 | | 19.094 |
| Experiment 3 | | 135 | | 21.356 |
| **Average of Experiments** | | **136** | | **19.933** |
| | | | | |
| | | | | |
| **Wireless** | | **Average** | | **Standard Deviation** |
| Experiment 1 | | 144 | | 18.435 |
| Experiment 2 | | 141 | | 21.361 |
| Experiment 3 | | 141 | | 22.074 |
| **Average of Experiments** | | **142** | | **20.623** |
| | | | | |
| | | | | |
| **Difference** | | **Average** | | **Standard Deviation** |
| Wired v. Wireless | | -6 | | -0.690 |

Figure 4-11 Video Latency Experiments with Cisco Network Security Appliance

## 4.6 Experiment Conclusions

There are some important conclusions that can be drawn from the previous three figures. First, a lower bound of 127 milliseconds and an upper bound of 171 milliseconds were calculated for the Spook video encoder. The mp4live video encoder resulted in a 115 millisecond lower bound and 148 millisecond upper bound delay. Based on this data, it is clear that there are some differences in the implementations of video encoding software. The mp4live video encoder had lower latency values and lower standard deviation values across all experiments; however, the differences between the two encoders are not drastically different and follow the same trends across each experiment. Another important conclusion is the difference in wired and wireless network media. Using both Spook and mp4live, the wired experiments experienced lower latency values than the wireless experiments. However, the difference is small compared to the overall latency. This allows us to conclude that there is little difference in the video latency across our wired and wireless network configurations. This is an important conclusion as it forms the basis for accepting the feasibility of a wireless scenario for the Shadow 200 TUAV program. The wireless scenarios features increases latency, but are still within the bounds of operating the Shadow 200 TUAV from the Ground Control Station.

Figure 4-10 records the data collected using the FastVDO H.264 encoder during three low latency scenarios for both wired and wireless connection scenarios. The table shows an extreme shift in latencies as compared to all other latency values collected throughout this research. The reduced load experiments in the FastVDO scenarios were comparable to the macroblock noise experiments of the mp4live and Spook encoder experiments; however, the latency average was nearly three times larger. This figure demonstrates the differences in encoder implementations. Since the FastVDO implementation experienced such large video latency values, this encoder implementation is infeasible for our real-time video stream requirements.

The Cisco ASA network security appliance was included in the latency experiments to determine the feasibility of providing a low latent, secure network communications. Figure 4-11 illustrates the results of both a wireless and wired network configuration using the Cisco ASA network security appliance. These experiments were conducted similar to the macroblock scenarios of the Spook wired and wireless configurations. The Cisco ASA latency experiments show similar patterns in the wired and wireless latency data and a marginal increase in overall latency when the Cisco ASA network security appliance is implemented in system architecture.

# CHAPTER 5 CONCLUSION AND FUTURE EFFORTS

The primary motivation behind this research work was to explore the feasibility of replacing the current wired technology in the Shadow 200 TUAV application with wireless data capability between ground control equipment. Although this research specific focus was on determining the feasibility of the new wireless technology by exploring video latency there are a host of other issues that must be investigated to successfully implement a new solution of the United States Army Shadow 200 TUAV program. These issues include exploring the wireless transmission range with various terrains and implementing and certifying the security of the data transmitting across the wireless network nodes.

Based on this research we can conclude that replacing wired connection links with wireless connection links between the ground control equipment in the Shadow 200 TUAV is feasible. This conclusion is based on the data collected using various wireless connection scenarios and then comparing the same scenarios using wired connections. The wired connections scenarios are designed to be representative of the systems currently implemented in the Shadow 200 TUAV and provide a baseline to compare changes using the wireless connection scenarios. Since all of the scenarios follow the same pattern and there is no evidence of increased latencies in the wireless connection scenarios we are able to demonstrate a successful and feasible wireless solution.

Additionally, by exploring the different network security configurations this research analyzed the overhead that is needed when using the network security appliances. In our research solution, network security was imperative to secure the data and control information being relayed between ground control stations and the Shadow 200 TUAV. Analysis showed that the Cisco network security appliances were able to provide a secure network environment without sacrificing the latency of the video signal.

An additional area of interest to further this research includes creating a system to allow for latency experiments to be measured using long range distances. Currently, the system is limited by the interconnection of Symmetricom time synchronization PCI cards using BNC cables. An important step would be to run similar experiments to include larger distances that are representative of real life scenarios that the Shadow 200 TUAV operates. Other areas of interest include exploring better techniques to timestamp video as well as exploring any additional video compression technologies to gain improvement over the current systems as well as the proposed wireless system.

# BIBLIOGRAPHY

Constance, S. Evaluating the Quality of Real-Time Video Over 802.11G Wireless. 2005. Computer Science and Software Engineering, Auburn University.

Dougherty, S., Hill, R.R., and Moore, J.T. Modeling Signal Latency Effects Using Arena. 2002. *Proceedings of the 2002 Winter Simulation Conference*. Dec. 8 – 11 San Diego California. pp 887 – 892

Feng, W. *Buffering Techniques for Delivery of Compressed Video in Video-On-Demand Systems*, Kluwer Academic Publishers, Norwell. 1997.

Karr, D.A., Rodrigues, C., Loyall, J.P., and Schantz, R.E. Controlling Quality-of-Service in a Distributed Video Application by an Adaptive Middleware Framework. 2001. *Proceedings of the 2001 International Workshop on Multimedia Middleware*. Ottawa, Canada.

Mitchell, J.L., Pennebaker, W.B., Fogg, C.E., and LeGall, D.J. *MPEG Video Compression Standards*, Kluwer Academic Publishers, New York. 2002.

Morris, O.J. (1995): MPEG-2: Where Did It Come From And What Is It? *IEEE Colloquium*, 24 Jan 1995, pp. 1-5.

"MPEG4IP  - Open Streaming Video and Audio," MPEG4IP. http://www.mpeg4ip.net

"MPEG-4 Description," Moving Picture Experts Group (MPEG). http://www.chiariglione.org/mpeg/

Tekalp, A. M. *Digital Video Processing*, Prentice Hall, Upper Saddle River. 1995.

Torres, L. and Kunt M. *Video Coding The Second Generation Approach*, Kluwer Academic Publishers, Boston. 1996.

Westwater, R. and Furht, B. *Real-time Video Compression: Techniques and Algorithms*, Kluwer Academic Publishers, Norwell. 1997.

# APPENDIX

Source Code

Screen.cpp

```
#include "screen.h"

#include "bcpci.h"


   bcpci b;


    screen::screen(int numFrameArgument){

      setRedFrame(false);

      setPaintCount(1);

      setNumberOfFrames(numFrameArgument);

      setFrameCount(0);

      setGeometry(0,0,640,440);

      QPixmap imagebg10("n1.png");

      setBackgroundPixmap(imagebg10);

      srand(time(0));

      int r = (rand()%3000)+1500;

      printf("Next frame in: %d milliseconds.\n",r);

      setTimerId(startTimer(r));
      setConstantTimerId(startTimer(7));
   }

    void screen::setRedFrame(bool flag)
   {
      redFrameFlag = flag;
   }

    bool screen::getRedFrame()
   {
      return redFrameFlag;
   }

    void screen::setConstantTimerId(int id)
   {
      constantId = id;
   }
    int screen::getConstantTimerId()
   {
      return constantId;
   }
    void screen::setTimerId(int id)
   {
     timerId = id;
   }
    int screen::getTimerId()
   {
     return timerId;
   }
    void screen::setPaintCount(int counter)
   {
     paintCount = counter;
   }
    int screen::getPaintCount()
   {
     return paintCount;
   }
    void screen::setAllowStrobeFlag(bool flag)
   {
     allowStrobeFlag = flag;
   }
    bool screen::getAllowStrobeFlag()
   {
     return allowStrobeFlag;
   }
    void screen::setNumberOfFrames(int frames)
```

```
{
   numberOfFrames = frames;
}
 int screen::getNumberOfFrames()
{
   return numberOfFrames;
}

 void screen::setFrameCount(int fCount)
{
   frameCount = fCount;
}

 int screen::getFrameCount()
{
   return frameCount;
}


 void screen::paintEvent( QPaintEvent * ){
   QPainter paint;
   paint.begin(this);
   if(getRedFrame() == true){
      paint.setBrush( QBrush(Qt::red, Qt::SolidPattern) );
      paint.drawRect(0,0, 300,300);
    }
    else{
      if(getPaintCount() == 0){
        paint.setBrush( QBrush(Qt::black, Qt::SolidPattern) );
        paint.drawRect(0,0, 300, 300);
      }
      else if(getPaintCount() == 1){
        paint.setBrush( QBrush(Qt::green, Qt::SolidPattern) );
        paint.drawRect(0,0, 300, 300);
      }
    }
    paint.end();
  }



 void screen::timerEvent( QTimerEvent* t){
    if(getNumberOfFrames() <= getFrameCount()){
      if(getRedFrame() == true){
        b.setStrobe();
        printf("\nTesting complete with %d samples sent.\n", getFrameCount());
        sleep(5);
        exit(0);
      }
      killTimer(getTimerId());
      setRedFrame(true);
      update();
    }
    else if(t->timerId() == getConstantTimerId()){
      setAllowStrobeFlag(false);
    }
    else if(t->timerId() == getTimerId()){
      killTimer(getTimerId());
      int r = (rand()%3000)+1500;
      printf("\nNext frame in: %d milliseconds.\n",r);
      timerId = startTimer(r);
      setTimerId(timerId);
      if(getPaintCount() == 0){
        setPaintCount(1);
      }
      else{
        setPaintCount(0);
      }
      nextBackground();
      update();
```

```
        b.setStrobe();

        int tempFrameCount = getFrameCount();

        tempFrameCount++;

        printf("\nFrame Number: %d\n", tempFrameCount);

        setFrameCount(tempFrameCount);

    }

}



 void screen::nextBackground(){


    QPixmap imagebg0("n1.png");

    QPixmap imagebg1("n2.png");

    QPixmap imagebg2("n3.png");

    QPixmap imagebg3("n4.png");

    QPixmap imagebg4("n5.png");


    if(getFrameCount() % 5 == 0){

        setBackgroundPixmap(imagebg0);

        printf("0");

    }

    else if(getFrameCount() % 5 == 1){

        setBackgroundPixmap(imagebg1);
    printf("1");
  }
  else if(getFrameCount() % 5 == 2){
    setBackgroundPixmap(imagebg2);
    printf("2");
  }
  else if(getFrameCount() % 5 == 3){
    setBackgroundPixmap(imagebg3);
    printf("3");
  }
  else if(getFrameCount() % 5 == 4){
    setBackgroundPixmap(imagebg4);
    printf("4");
  }
}
```

```
/* vo_xv.c, X11 Xv interface (excerpt) */

#define NUM_BUFFERS 3

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//bcuser.h - Symmetricom BC635PCI card
#include "bcuser.h"


//latency variables for added functions
  BC_PCI_HANDLE hBC_PCI;

  BYTE tmfmt;

  DWORD evtmaj=0, evtmin=0;

  DWORD evtmajtmp=0, evtmintmp=0;

  struct tm *evtmajtmtmp;

  struct tm *evtmajtm;

  int histogram[25];

  int stopFlag = 0;

  int tooLargeCounter = 0;

  int numberOfSamples = 0;

  int frameReference = 2;

  struct videoLatStat{

    float encodeTime;
    float decodeTime;
    int frameColor;
    int encodeTrue;
    int decodeTrue;
  };

  float tempEncodeTime;
```

```
   FILE *outputFile;

   char outputFileName[] = "uavSimulationReport.txt";

   struct videoLatStat testResults[60000];

/*

query time captured caused by an external event (video generator creates new frame)
on Symmetricom BC635PCI card; using Symmetricom interface library method

*/
    float videoGeneratorFrameTime()
   {

      BYTE stat;

   //bcReadEventTime: Symmetricom interface library method
      if(bcReadEventTime (hBC_PCI, &evtmaj, &evtmin, &stat) == TRUE)

      {

         evtmajtm = gmtime( &evtmaj );

         if(evtmaj != evtmajtmp)
         {
           fprintf(outputFile,"\n\n\nNewly Created Frame @ %02d/%02d/%d
%02d:%02d:%02d.%06lu   Status: %d\n",
                evtmajtm->tm_mon+1, evtmajtm->tm_mday, evtmajtm->tm_year+1900,
                evtmajtm->tm_hour, evtmajtm->tm_min, evtmajtm->tm_sec, evtmin, stat);


         //stored time
            evtmajtmp = evtmaj;

            evtmintmp = evtmin;

            evtmajtmtmp = gmtime(&evtmaj);


         //convert stored time
            int msVG = evtmin;

            float tempVGFloat = evtmajtm->tm_hour*3600000 + evtmajtm->tm_min*60000
+ evtmajtm->tm_sec*1000 + (float)(msVG/1000);

            return tempVGFloat;

         }

         else
         {
            return 0;
         }

      }

   }

/*

function derived from Shawn M. Constance (NO-Reference Blockiness Metric
implementation)
determines average buffer value from specified target value

*/

    int averageBlockValue()
   {

   //buffer target values for color box

      int i = 84;

      int j = 84;

      int x = 4;

      int y = 4;

      unsigned char val;

      int index = (x+i) + (y+j) * image_width;

      unsigned char* buffer = xvimage[current_buf]->data;

      //buffer value
      val = buffer[index];
      return val;

   }

/*

display test results including: number of samples and the histogram representing the
video latency
time over a certain number of samples. Number of samples can be compared to video
generator application
to determine number of frames that were not detected.

*/

    void displayTestResults()
   {

      printf("*******Video latency Test Results*******\n Samples Received: %d\n
Samples Too Large: %d\n Samples Recorded: %d\n Histogram:\n", numberOfSamples,
tooLargeCounter, numberOfSamples - tooLargeCounter);

      printf("0ms ~ 19ms = %d\n",histogram[0]);

      printf("20ms ~ 39ms = %d\n",histogram[1]);

      printf("40ms ~ 59ms = %d\n",histogram[2]);

      printf("60ms ~ 79ms = %d\n",histogram[3]);

      printf("80ms ~ 99ms = %d\n",histogram[4]);

      printf("100ms ~ 119ms = %d\n",histogram[5]);

      printf("120ms ~ 139ms = %d\n",histogram[6]);

      printf("140ms ~ 159ms = %d\n",histogram[7]);

      printf("160ms ~ 179ms = %d\n",histogram[8]);

      printf("180ms ~ 199ms = %d\n",histogram[9]);

      printf("200ms ~ 219ms = %d\n",histogram[10]);

      printf("220ms ~ 239ms = %d\n",histogram[11]);
      printf("240ms ~ 259ms = %d\n",histogram[12]);

      printf("260ms ~ 279ms = %d\n",histogram[13]);

      printf("280ms ~ 299ms = %d\n",histogram[14]);

      printf("300ms ~ 319ms = %d\n",histogram[15]);

      printf("320ms ~ 339ms = %d\n",histogram[16]);

      printf("340ms ~ 359ms = %d\n",histogram[17]);

      printf("360ms ~ 379ms = %d\n",histogram[18]);

      printf("380ms ~ 399ms = %d\n",histogram[19]);

      printf("400ms ~ 419ms = %d\n",histogram[20]);

      printf("420ms ~ 439ms = %d\n",histogram[21]);

      printf("440ms ~ 459ms = %d\n",histogram[22]);

      printf("460ms ~ 479ms = %d\n",histogram[23]);

      printf("480ms ~ 499ms = %d\n",histogram[24]);

      fprintf(outputFile,"*******Video latency Test Results*******\n Samples Received:
%d\n Samples Too Large: %d\n Samples Recorded: %d\n Histogram:\n",
numberOfSamples, tooLargeCounter, numberOfSamples - tooLargeCounter);

      fprintf(outputFile,"0ms ~ 19ms = %d\n",histogram[0]);

      fprintf(outputFile,"20ms ~ 39ms = %d\n",histogram[1]);

      fprintf(outputFile,"40ms ~ 59ms = %d\n",histogram[2]);

      fprintf(outputFile,"60ms ~ 79ms = %d\n",histogram[3]);

      fprintf(outputFile,"80ms ~ 99ms = %d\n",histogram[4]);

      fprintf(outputFile,"100ms ~ 119ms = %d\n",histogram[5]);

      fprintf(outputFile,"120ms ~ 139ms = %d\n",histogram[6]);

      fprintf(outputFile,"140ms ~ 159ms = %d\n",histogram[7]);
```

```
      fprintf(outputFile,"160ms ~ 179ms = %d\n",histogram[8]);

      fprintf(outputFile,"180ms ~ 199ms = %d\n",histogram[9]);

      fprintf(outputFile,"200ms ~ 219ms = %d\n",histogram[10]);

      fprintf(outputFile,"220ms ~ 239ms = %d\n",histogram[11]);

      fprintf(outputFile,"240ms ~ 259ms = %d\n",histogram[12]);

      fprintf(outputFile,"260ms ~ 279ms = %d\n",histogram[13]);

      fprintf(outputFile,"280ms ~ 299ms = %d\n",histogram[14]);

      fprintf(outputFile,"300ms ~ 319ms = %d\n",histogram[15]);

      fprintf(outputFile,"320ms ~ 339ms = %d\n",histogram[16]);

      fprintf(outputFile,"340ms ~ 359ms = %d\n",histogram[17]);

      fprintf(outputFile,"360ms ~ 379ms = %d\n",histogram[18]);

      fprintf(outputFile,"380ms ~ 399ms = %d\n",histogram[19]);

      fprintf(outputFile,"400ms ~ 419ms = %d\n",histogram[20]);

      fprintf(outputFile,"420ms ~ 439ms = %d\n",histogram[21]);

      fprintf(outputFile,"440ms ~ 459ms = %d\n",histogram[22]);

      fprintf(outputFile,"460ms ~ 479ms = %d\n",histogram[23]);

      fprintf(outputFile,"480ms ~ 499ms = %d\n",histogram[24]);

      exit(0);

   }

/*

capture current time on Symmetricom BC635PCI
card using Symmetricom interface library method

*/
```

```
   float videoReceiverFrameTime()
  {
    DWORD maj, min;

    struct tm *majtime;

    BYTE stat;

  //bcReadBinTime: Symmetricom interface library method
    if ( bcReadBinTime (hBC_PCI, &maj, &min, &stat) == TRUE )
    {

      majtime = gmtime( &maj );

      fprintf(outputFile, "Frame decoded @ %02d/%02d/%d  %02d:%02d:%02d.%06lu
Status: %d\n",
          majtime->tm_mon+1, majtime->tm_mday, majtime->tm_year+1900,
          majtime->tm_hour, majtime->tm_min, majtime->tm_sec, min, stat);


      int msVR = min;

      //convert current time

      float tempVRFloat = majtime->tm_hour*3600000 + majtime->tm_min*60000 +
majtime->tm_sec*1000 + (float)(msVR/1000);

      return tempVRFloat;
    }
  }


/*
  once each frame is decoded and latency is computed,
  place latency value in correct "20ms bucket"
  if latency value is higher than 500ms, ignore and
  increment tooLargeCounter to signify a latency value
  outside of the scope of the histogram
*/
   void processHistogram(float calc)

  {

    printf("\nTime Diff: %f\n", calc);
    if(calc < 500)
    {

      int div = calc/20;

      histogram[div] = histogram[div] + 1;

    }

    else
    {

      tooLargeCounter++;
    }


  }


   static void flip_page(void)
  {
    //stop frame detected
    if(stopFlag == 1)
    {
      printf("\nSimulation Complete\n");
      displayTestResults();
      fclose(outputFile);
      exit(0);
    }


    if(frameReference < 60000)

    {


    //load encodeTime, decodeTime, and frame color buffer for each frame
    //if available
      testResults[frameReference].encodeTime = videoGeneratorFrameTime();
      testResults[frameReference].decodeTime = videoReceiverFrameTime();
      testResults[frameReference].frameColor = averageBlockValue();
```

```
    //determine if decode time should be recognized
    //or if frame is an intermediate frame
      if( testResults[frameReference].frameColor >= 0 &&
testResults[frameReference].frameColor <= 40){

        testResults[frameReference].decodeTrue = 1;
      }

      else{

        testResults[frameReference].decodeTrue = 0;
      }

    //determine if encode time should be recognized
    //or if multiple event occurred and should be ignored
      if( testResults[frameReference-1].encodeTime == 0 &&
testResults[frameReference].encodeTime > 0){
        testResults[frameReference].encodeTrue = 1;
      }

      else{
        testResults[frameReference].encodeTrue = 0;
      }

    //store results of simulation into text file for
    //cross reference to histogram and more detailed
    //information on the frames and timestamps
      if(testResults[frameReference].encodeTrue == 1){

        fprintf(outputFile,"\n\n****ENCODE****\n");

        fprintf(outputFile,"\n\nFrame Number: %d\n",frameReference);

        fprintf(outputFile,"Block Value: %d\n",
testResults[frameReference].frameColor);

        fprintf(outputFile,"Encode Time: %f\n",
testResults[frameReference].encodeTime);

        tempEncodeTime = testResults[frameReference].encodeTime;

      }

      else if(testResults[frameReference].decodeTrue == 1){
        fprintf(outputFile,"\n\n****DECODE****\n");

        fprintf(outputFile,"Frame Number: %d\n",frameReference);

        fprintf(outputFile,"Block Value: %d\n",
testResults[frameReference].frameColor);

        fprintf(outputFile,"Decode Time: %f\n",
testResults[frameReference].decodeTime);

        float timeDelta = (testResults[frameReference].decodeTime - tempEncodeTime);

        processHistogram(timeDelta);

        numberOfSamples++;

        fprintf(outputFile,"Time Difference (seconds): %f\n", timeDelta/1000);

        fprintf(outputFile,"Time Difference (milliseconds): %f\n\n", timeDelta);

      }

      frameReference++;

    }

    else{
      stopFlag = 1;

    }

    put_xvimage( xvimage[current_buf] );


  /* remember the currently visible buffer */
    visible_buf = current_buf;

    if (num_buffers > 1)
    {
      current_buf =
        vo_directrendering ? 0 : ((current_buf + 1) % num_buffers);
      XFlush(mDisplay);
    }
    else
      XSync(mDisplay, False);
    return;
  }



/*

MPlayer initialization -- initialize and start Symmetricom bc635PCI
for use in functions

*/
   static int preinit(const char *arg)
  {

    DWORD maj, min;

    // Start the device
    hBC_PCI = bcStartPci();

   if (!hBC_PCI)
    {
      printf ("Error Opening Device Driver\n");
    }


    // Init the time format
    if ( bcReqTimeFormat (hBC_PCI, &tmfmt) != TRUE )
    {
      printf ("Error Getting Time Format!!!\n");
    }

    BYTE stat;

    if ( bcReadBinTime (hBC_PCI, &maj, &min, &stat) == TRUE )
    {
      evtmajtmtmp = gmtime( &maj );
    }

    outputFile = fopen(outputFileName, "w");

    if(outputFile == NULL)
    {
      printf("Error opening output file\n");
    }
    XvPortID xv_p;
    int busy_ports = 0;
    unsigned int i;
    strarg_t ck_src_arg = { 0, NULL };
    strarg_t ck_method_arg = { 0, NULL };

    opt_t subopts[] =
    {
    /* name         arg type     arg var          test */
    {  "port",      OPT_ARG_INT, &xv_port,       (opt_test_f)int_pos },
    {  "ck",        OPT_ARG_STR, &ck_src_arg,    xv_test_ck },
    {  "ck-method", OPT_ARG_STR, &ck_method_arg, xv_test_ckm },
    {  NULL }
    };

    xv_port = 0;

   /* parse suboptions */
    if ( subopt_parse( arg, subopts ) != 0 )
    {
      return -1;
    }

   /* modify colorkey settings according to the given options */
    xv_setup_colorkeyhandling( ck_method_arg.str, ck_src_arg.str );

    if (!vo_init())
      return -1;

  /* check for Xvideo extension */
    if (Success != XvQueryExtension(mDisplay, &ver, &rel, &req, &ev, &err))
    {
      mp_msg(MSGT_VO, MSGL_ERR,
          "Sorry, Xv not supported by this X11 version/driver\n");
      mp_msg(MSGT_VO, MSGL_ERR,
          "******** Try with  -vo x11  or  -vo sdl  *********\n");
      return -1;
    }

   /* check for Xvideo support */
    if (Success !=
     XvQueryAdaptors(mDisplay, DefaultRootWindow(mDisplay), &adaptors,
               &ai))
    {
      mp_msg(MSGT_VO, MSGL_ERR, "Xv: XvQueryAdaptors failed\n");
      return -1;
    }

   /* check adaptors */
    if (xv_port)
    {
      int port_found;

      for (port_found = 0, i = 0; !port_found && i < adaptors; i++)
      {
        if ((ai[i].type & XvInputMask) && (ai[i].type & XvImageMask))
        {
          for (xv_p = ai[i].base_id;
               xv_p < ai[i].base_id + ai[i].num_ports; ++xv_p)
          {
            if (xv_p == xv_port)
            {
              port_found = 1;
              break;
            }
          }
        }
      }
      if (port_found)
      {
        if (XvGrabPort(mDisplay, xv_port, CurrentTime))
          xv_port = 0;
      }
      else
      {
        mp_msg(MSGT_VO, MSGL_WARN,
               "Xv: Invalid port parameter, overriding with port 0\n");
        xv_port = 0;
      }
    }

    for (i = 0; i < adaptors && xv_port == 0; i++)
    {
      if ((ai[i].type & XvInputMask) && (ai[i].type & XvImageMask))
      {
        for (xv_p = ai[i].base_id;
             xv_p < ai[i].base_id + ai[i].num_ports; ++xv_p)
          if (!XvGrabPort(mDisplay, xv_p, CurrentTime))
```

```
            {
              xv_port = xv_p;
              break;
            }
            else
            {
              mp_msg(MSGT_VO, MSGL_WARN,
                    "Xv: could not grab port %i\n", (int) xv_p);
              ++busy_ports;
            }
        }
      }
      if (!xv_port)
      {
        if (busy_ports)
          mp_msg(MSGT_VO, MSGL_ERR,
               "Could not find free Xvideo port - maybe another process is already using
it.\n"
               "Close all video applications, and try again. If that does not help,\n"
               "see 'mplayer -vo help' for other (non-xv) video out drivers.\n");
        else
          mp_msg(MSGT_VO, MSGL_ERR,
               "It seems there is no Xvideo support for your video card available.\n"
               "Run 'xvinfo' to verify its Xv support and read
DOCS/HTML/en/video.html#xv!\n"
               "See 'mplayer -vo help' for other (non-xv) video out drivers. Try -vo x11\n");
        return -1;
      }

      if ( !vo_xv_init_colorkey() )
      {
        return -1; // bail out, colorkey setup failed
      }
      vo_xv_enable_vsync();
      vo_xv_get_max_img_dim( &max_width, &max_height );

      fo = XvListImageFormats(mDisplay, xv_port, (int *) &formats);

      return 0;
    }
```