NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

REAL-TIME COMPRESSED VIDEO TRANSMISSION ACROSS THE COMMON DATA LINK

by

Thaddeus Owens Walker III

June, 1995

Thesis Co-Advisors: Murali Tummala
Shridhar B. Shukla

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1995	Engineer's Thesis

4. TITLE AND SUBTITLE REAL-TIME COMPRESSED VIDEO TRANSMISSION ACROSS THE COMMON DATA LINK

5. FUNDING NUMBERS

6. AUTHOR(S) Walker, T. Owens III

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
(1) Space and Naval Warfare Systems Command
2457 Crystal Park #5, Arlington, VA 22202-5100
(2) Defense Airborne Reconnaissance Office
Washington, D.C. 20330-1000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The advances in high speed computer networks and digital communication techniques have enabled the rapid and extensive dissemination of information throughout the modern defense infrastructure. One of the challenges in networking today is real-time dissemination of information. This thesis proposes a solution for a specific aspect of this challenge, namely, the transmission of real-time compressed video data across the Common Data Link (CDL), a high-speed military data link designed to operate in high error environments. Current research is primarily focused on the transmission of real-time data across low-error links.

This thesis proposes, simulates, and analyzes a mechanism which guarantees that (1) delay bounds are met for real-time flows despite network overload and (2) a minimum acceptable image quality is maintained despite the presence of highly correlated errors. These highly correlated errors are characteristic of the type of electromagnetic jamming likely to be encountered by the CDL. This mechanism consists of four fundamental requirements: (1) a hierarchical image compression scheme, (2) rate control at the source, (3) bandwidth allocation within all encountered network nodes, and (4) dynamic forward error correction. The proposed solution is modeled in the OPNET simulation environment, and the validity and feasability of the mechanism are verified. In addition, the simulation is interfaced with a compression/decompression algorithm running in MATLAB to enable the subjective analysis of actual images before and after transmission in various jamming scenarios. The results demonstrate the effectiveness of the proposed solution in meeting delay guarantees and maintaining image quality.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Common Data Link (CDL), Real-Time, Image Compression, Video Transmission	117
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

REAL-TIME COMPRESSED VIDEO TRANSMISSION ACROSS THE COMMON DATA LINK

T. Owens Walker III
Lieutenant, United States Navy
B.S., Cornell University, 1987

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
and
ELECTRICAL ENGINEER

from the

NAVAL POSTGRADUATE SCHOOL
June 1995

Author: _____
T. Owens Walker III

Approved by: _____
Murali Tummala, Thesis Co-Advisor

_____
Shridhar B. Shukla, Thesis Co-Advisor

_____
Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

The advances in high speed computer networks and digital communication techniques have enabled the rapid and extensive dissemination of information throughout the modern defense infrastructure. One of the challenges in networking today is real-time dissemination of information. This thesis proposes a solution for a specific aspect of this challenge, namely, the transmission of real-time compressed video data across the Common Data Link (CDL), a high-speed military data link designed to operate in high error environments. Current research is primarily focused on the transmission of real-time data across low-error links.

This thesis proposes, simulates, and analyzes a mechanism which guarantees that (1) delay bounds are met for real-time flows despite network overload and (2) a minimum acceptable image quality is maintained despite the presence of highly correlated errors. These highly correlated errors are characteristic of the type of electromagnetic jamming likely to be encountered by the CDL. This mechanism consists of four fundamental requirements: (1) a hierarchical image compression scheme, (2) rate control at the source, (3) bandwidth allocation within all encountered network nodes, and (4) dynamic forward error correction. The proposed solution is modeled in the OPNET simulation environment, and the validity and feasability of the mechanism are verified. In addition, the simulation is interfaced with a compression/decompression algorithm running in MATLAB to enable the subjective analysis of actual images before and after transmission in various jamming scenarios. The results demonstrate the effectiveness of the proposed solution in meeting delay guarantees and maintaining image quality.

TABLE OF CONTENTS

I. INTRODUCTION

A. COMMON DATA LINK OVERVIEW

The Common Data Link (CDL) is being developed as part of a multipurpose network infrastructure by the Defense Airborne Reconnaissance Office (DARO). It is designed as a full duplex, jam-resistant point-to-point microwave communication system. The CDL is intended to provide a real-time, as well as reliable, data transfer facility between an airborne data collection platform and a surface intelligence platform. The link comprises a downlink, called the return link, and an uplink, called the command link.

The command link is designed to transmit asset, link and sensor command and control information to the airborne platform and operates at 200 kbps. This link employs binary phase shift keying spread spectrum modulation.

The return link is designed to transmit sensor data, link status and airborne asset reports to the surface platform. The return link employs offset quadrature phase shift keying and can operate at 10.71 Mbps, 137.088 Mbps, or 274.176 Mbps. It can be configured into a hierarchy of time division multiplexed channels depending on the aggregate bit rate selected [Ref. 1].

B. OBJECTIVES

The main objective of this thesis is to develop and analyze a communication mechanism to provide real-time compressed video transmission across the CDL. This mechanism must function correctly in the presence of the high error rates associated with jamming. In the process, a comprehensive CDL network model, under development at the Naval Postgraduate School and designed within MIL 3, Inc.'s Optimized Network Engineering Tool (OPNET), has been completed.

C. CONTRIBUTION

This thesis makes the following contributions to the ongoing CDL research effort. It

(1) proposes and analyzes a solution to the problem of real-time constraints on a packet-switched network in the presence of highly correlated errors,

(2) produces a CDL model in OPNET,

(3) provides a mechanism to subjectively evaluate the transmission of still imagery across the CDL, and

(4) provides a user's guide for the CDL model.

D. ORGANIZATION

This thesis is organized in the following manner. The problem addressed and the proposed solution are presented in Chapter II, along with a discussion of the real-time communication theory behind the development of the proposed solution. Chapter III presents the application and network support required by the proposed solution. This chapter describes the video compression algorithm utilized, an exploration of link monitoring techniques, and a justification of the error correction model employed. The overview of the network model is provided in Chapter IV. Chapter V discusses the model and its associated routines in detail. Sample results showing the capabilities of the model are presented in Chapter VI. Chapter VII presents our conclusions and proposes possible follow-on areas of research. Appendix A contains a condensed user's guide to the CDL model, detailing the step-by-step procedures for using the model. Appendix B is a listing of all the programs for both the OPNET simulation models and the MATLAB image generation routines.

II. REAL-TIME TRANSMISSION OF IMAGERY

This chapter presents the theoretical basis upon which the work reported here has been developed. Specifically, the first section poses the problem to be addressed while the third section presents the proposed solution to this problem. The intervening section discusses the current state of research in the field of real-time transmission and outlines the specific issues that must be addressed by any real-time communication scheme.

A. COMMUNICATION REQUIREMENTS

This thesis investigates how real-time video image data can be transferred across the CDL. The video images to be transmitted are encoded using a five level hierarchical coding scheme based on a discrete wavelet decomposition algorithm. The presence of a fixed playback point at the receiving application defines the need for a maximum real-time delay bound, which will be discussed in the next section. The playback point is used by the receiving application to determine how much data needs to be buffered prior to forwarding it to the user. This thesis proposes, implements, and analyzes a mechanism which guarantees that

(1) delay bounds are met for real-time flows despite network overload and

(2) a minimum acceptable image quality is maintained despite the presence of highly correlated errors.

These highly correlated errors are characteristic of the type of electromagnetic jamming that is expected to be encountered by the CDL.

B. REAL-TIME COMMUNICATION

1. Quality of Service Metrics

a. Delay

Delay is one of the primary metrics for measuring the quality of service in a packet-switched network [Ref. 2]. It can be measured as the per-packet delay or the inter-packet delay. The former is known as the end-to-end delay while the latter is referred to as the jitter. A maximum and minimum bound on the end-to-end packet delay determines a maximum and minimum bound on the jitter [Ref. 3]. Therefore, this thesis primarily deals with the end-to-end delay. The delay bounds provided by a network can be either deterministic or statistical [Ref. 4]. A deterministic bound is one that is never exceeded while a statistical bound provides a probability that the given bound will not be exceeded. A deterministic bound can be thought of as a statistical bound where the probability that the specified bound will not be exceeded is one. The mechanism presented in this thesis enforces a deterministic end-to-end delay bound, which represents a firm upper (and lower) bound on the maximum delay any packet will encounter.

b. Reliability

Reliability is a measure of the probability that a packet is received error free. In a real-time environment, there is a trade-off between delay constraints and reliability. As a result, current work in the field of real-time transmission typically assumes an underlying error-free network or, alternately, assumes that the data is not error sensitive. Error sensitive data transfers subjected to highly correlated errors in the presence of tight real-time constraints have not been studied extensively [Ref. 5]. This is exactly the scenario encountered by the CDL when operating under the influence of

hostile jamming. The thesis addresses the highly correlated errors imposed by jamming and incorporates a forward error correction scheme to ensure that a minimum image quality is maintained.

c. Fidelity

Fidelity is a measure of how faithfully the original image is reproduced at the destination. It is the result of the above mentioned metrics and serves as a comprehensive evaluation of the performance of the communication mechanism over a given network. Typically, fidelity is measured statistically by recording the number of packets lost per image or by observing the signal-to-noise ratio of the reconstructed image [Ref. 6]. While these metrics are useful, a more effective measure of the quality of a transmitted image is a subjective evaluation of the reconstructed image itself. This is particularly true when the end-user is a human, as in some of the CDL deployment scenarios. Although both the packet loss rate and the signal-to-noise ratio will be examined as in most existing studies, this thesis makes the additional contribution of providing the reconstructed images for subjective evaluation by the user.

2. Real-Time Applications

We first define, for a packet-switched network, what is meant by real-time traffic. In their paper, Shenker, Clark and Zhang [Ref. 2] divide applications into two general service types: *elastic* and *real-time*. Elastic applications always wait for late data to arrive. As a result, no prior characterization of an elastic application's traffic is required for it to function properly. Examples of elastic applications are Telnet, E-mail, and FTP. A real-time application, on the other hand, is sensitive to delay. Late packets are discarded.

Real-time applications are further divided into two subtypes: *tolerant* and *intolerant*. A tolerant application can vary its playback point to allow for changes in the actual delays experienced by the packets. Not surprisingly, these applications are referred to as adaptive playback applications and require a predictive service, which provides a statistical delay bound rather a deterministic one. An intolerant application requires a fixed maximum delay because it makes use of a fixed playback point. Thus, this type of application requires a guaranteed maximum delay bound. While the tolerant type is more flexible, the intolerant type requires more stringent guarantees from the network. This thesis assumes intolerant applications with a fixed playback point. Additionally, it assumes that the receiver has ample buffer space availability to accommodate the required playback point.

3. Techniques for Real-Time Application Support

Presently, there are two main approaches to support real-time traffic. Both are based on the fundamental idea of reserving resources and performing admission control to ensure the quality of service. The two approaches differ in that one requires the periodic "update" of the real-time flow [Ref. 7] while the other fixes the flow parameters at the real-time flow establishment [Ref. 8]. Several variations of the second approach that offer some sort of feedback to control the output of the sender have also been proposed [Ref. 6].

The idea of providing delay guarantees across a packet-switched network carrying real-time as well as non-real-time traffic is based on the work of Parekh and Gallager [Ref. 3]. They have demonstrated that a form of weighted fair queuing [Ref. 9] at every network node, coupled with rate control at the source, is both necessary and sufficient to

provide a maximum delay bound on the transmission of packets in a packet-switched network. It is shown that this maximum bound is:

$$\tau_{max} = \frac{\beta_i}{g_i} + \frac{(h_i - 1)l_i}{g_i} + \sum_{m=1}^{h_i} \frac{l_{max}}{r_m} \qquad (1)$$

where β_i is the application's token bucket size, g_i is the weighted rate assigned to the application, h_i is the number of hops in the connection, l_i is the application's maximum packet size, l_{max} is the maximum packet size of the network, and r_m is the bandwidth outbound on hop m. The token bucket size is the maximum number of tokens (described in bits) that can be held by the token/leaky bucket at any given time. The weighted rate is the transmission rate assigned by the network to the leaky/token bucket. The bandwidth is a measure (in bps) of the transmission rate assigned to the flow at each node along the route. It is important to emphasize that this delay bound is valid whether or not the source traffic is regulated (or shaped) for the other active applications attached to the network.

Five elements are required to enforce the Parekh and Gallager delay bound and are present in all the existing real-time communication schemes:

(1) *A flow specification*: The application must inform the network of the quality of service it requires. This is in the form of a description of the traffic it intends to generate. [Ref. 7]

(2) *Admission control*: Based on the flow specification, the network must decide whether or not it can provide the quality of service required by the application. If not, the network must deny the application access to the network. [Ref. 10]

(3) *A real-time flow establishment*: To apply the resource reservation required to protect the real-time data from general data, the network must, in effect, setup a channel for the real-time traffic to follow. [Ref. 8]

(4) *Resource reservation*: This real-time channel is set up by reserving various resources along the chosen route. These resources include bandwidth and buffer capacity.

(5) *Packet scheduling*: To make use of the resources reserved, a packet scheduling mechanism must be in place at the network node to differentiate between the various real-time and elastic data flows.

In the solution presented in this thesis, the admission control is done manually for simulation purposes. In other words, an application is not allowed to request to establish a real-time flow unless the resources exist to accommodate it. It is assumed that follow-on work will include the implementation of an effective automatic admission control strategy.

C. PROPOSED SOLUTION

The mechanism proposed in this thesis ensures that the Parekh and Gallager maximum delay bound calculated as above will be maintained in the presence of network overload. This solution differs from the existing proposals in that it introduces the notion of dynamic, network-controlled resource reservation and allocation. Although many of the existing mechanisms seek to dynamically adapt the sending application's output, none allows the network to dynamically alter its own resource allocations. An overview of the proposed solution follows.

To fulfill the requirement to maintain the guaranteed quality of service despite errors highly correlated in time, the solution presented in this thesis utilizes a link quality monitoring mechanism to determine the status of the link. As link conditions deteriorate, the network applies forward error correction to the real-time traffic. This adds bandwidth to each of the flows and forces the network to reallocate its resources and inform the sending applications of their new flow requirements. These new flow requirements imply that the original flow specification must include both desired and minimum quality of

service requirements to ensure the application can meet the new flow specifications. The remainder of this section will outline the specific details of this solution.

1. Rate Control and Bandwidth Reservation

Rate control at the source is provided by a token-leaky bucket that is used to regulate the flow of image data from the various compression levels. Weighted fair queuing is achieved by implementing intelligent bandwidth allocation and reservation at the Common Data Link-Network Interface (CDL-NI). This involves dedicated buffers assigned to the real-time application data, as well as a general queue for non-real-time applications. The rate out of these queues to the transmitter is controlled to guarantee a certain bandwidth to the real-time application. These rates are assigned individually and will typically vary between the different queues. The only constraint is that the sum of the transmission rates of the real-time queues and the non-real-time queue must not exceed the aggregate bandwidth of the attached CDL. It is assumed that the airborne and surface LANs, modeled as FDDI rings, provide the bandwidth necessary over the LANs by proper selection of the synchronous allotments assigned to the sending node and the receiving network interface.

2. Forward Error Correction

Forward error correction (FEC) is applied to the transmitted data when jamming is reported by the link monitoring mechanism. The application transmission rate is reduced to the minimum acceptable data rate that is agreed upon in the flow specification when the application requested to establish a real-time flow. In this thesis, this corresponds to sending only resolution level one image data. To enforce this dynamic response, a CDL manager at the network interface is required. This CDL manager informs the leaky bucket-token bucket mechanism of the need to send only the minimum

acceptable data flow and updates the CDL bandwidth allocation at the queue outputs to reflect the addition of the forward error correction overhead. With the detection of the end of jamming, the application is allowed to return to its desired transmission rate with the necessary coordination once again being performed by the CDL manager. To ensure timely response to the presence of jamming, the above control interactions consist entirely of communications within the confines of the LAN containing the sending application. It is crucial to realize, that, in all scenarios, the maximum delay bound guaranteed by the network to a particular flow does not change.

3. Implementation Issues

The mechanism proposed above requires the implementation of the following components in OPNET: (1) a token-leaky bucket mechanism at the source, (2) multiple queues at the CDL-NI, (3) a CDL manager, (4) forward error correction at the CDL-NI, and (5) real-time compressed video applications at each end. The real-time video applications require the implementation of both the sending and the receiving entities. The sending application must make the five layers of coded information available to the transmission mechanism while the receiver must extract the data from the packets and separate it for use in a reconstruction algorithm. The packets generated by these applications must be in a standard format with fields specifying the compression level and sequence number of the information the packet contains.

Finally, it is essential that the simulation involve actual video data to allow subjective, as well as, objective analysis of the received data. This requires integration of the compression/decompression algorithm implemented in MATLAB and the OPNET simulation environment. The data is analyzed in terms of visual image quality (determined by the user) as well as packet/data loss rate and signal-to-noise ratio.

III. REQUIRED APPLICATION AND NETWORK SUPPORT

This chapter provides the application and network support required by the solution to the problem presented in Chapter II. Specifically, the first section addresses the compression method applied to the image data to generate the data to be transmitted. The second section looks at the method of link quality monitoring utilized by the simulation model and presents some of the issues that need to be addressed when designing a link monitoring mechanism. The third section describes the general error correction model employed by the simulation.

A. IMAGE COMPRESSION

This thesis treats a video session as the transmission of a series of individual images. Although the work specifically deals with still images, it is assumed that multiple transmissions of these images constitute a single video session. The term *video* is used throughout this work to emphasize this notion.

The image compression algorithm employed in this thesis has been proposed by Carvahlo [Ref. 11]. This algorithm applies a multi-resolution coding scheme to the output of a two-dimensional discrete wavelet transform. This transform is used to generate a pyramidal decomposition of the original image data. This section discusses the suitability of the compression scheme for the proposed real-time mechanism.

The continuous wavelet transform was developed as an improvement over the short time Fourier transform [Ref. 13]. Both methods seek to transform a non-stationary signal into the frequency domain by breaking the signal into blocks that are then assumed to be stationary. The major drawback of the short time Fourier transform is that the higher frequencies suffer from poor time resolution while the lower frequencies suffer from poor frequency resolution. This is due to the fixed size of the time windows applied to the data. The wavelet transformation overcomes this difficulty by varying the size of

the time windows while still maintaining a fixed time-bandwidth product. Thus, smaller time windows in the higher frequencies provide better time resolution, and larger time windows (smaller frequency windows) provide better frequency resolution in the lower frequencies. An illustration of this approach is provided in Figure 1.

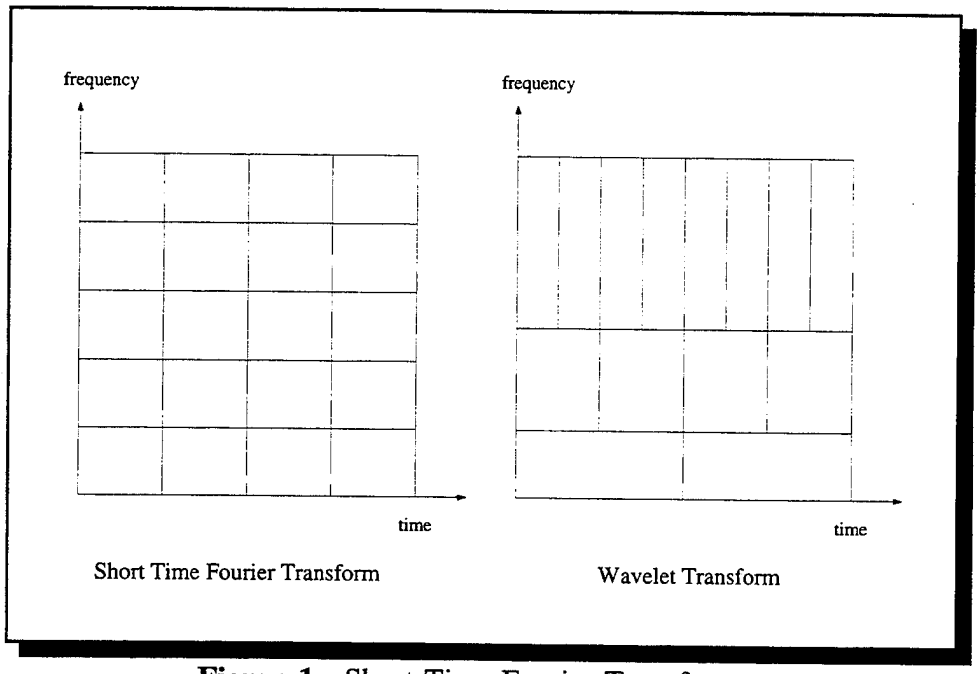


Figure 1. Short Time Fourier Transform vs Wavelet Transform

The continuous transform must be discretized to allow the digital processing of signals. This is accomplished using a multi-level pyramidal decomposition proposed by Mallat [Ref. 12]. This method uses a series of orthogonal high-pass and low-pass filters to generate the decomposition of the signal into a set of orthonormal basis functions [Ref. 13]. It turns out that better results can be achieved by relaxing the orthogonality requirement in the filters. This allows the use of a set of linear phase filters known as biorthogonal filters. The use of these linear phase filters allows for theoretically exact reconstruction [Ref. 14]. In practice, both the quantization error and the need to extend the image at the boundaries limit the actual performance of the decomposition [Ref. 15].

This wavelet decomposition can be extended into two dimensions by assuming that the filters are separable and applying four separate filtering operations.

A multi-resolution sub-band coding method is utilized to achieve the desired compression. The resultant wavelet decomposition is divided into five resolution levels. The resolution levels are based on the energy levels in the decomposition, one being the lowest and five being the highest. Experimental image data has shown that the lower frequency components comprise the majority of the information in an image [Ref. 15]. As a result, the lowest resolution level is always comprised of the low frequency components. Level two adds the next three highest energy components to the transmitted image while the remaining three levels each add four additional components. Figure 2 illustrates this compression scheme. Thus, transmitting at resolution level one corresponds to the maximum compression (16:1) while transmitting at level five represents zero compression.

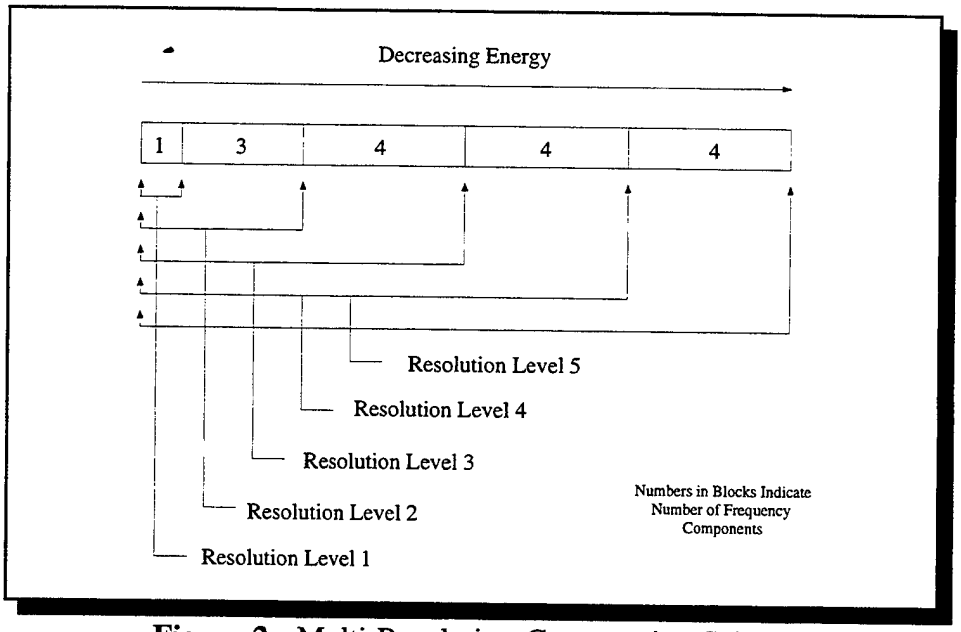


Figure 2. Multi-Resolution Compression Scheme

B. LINK MONITORING

The success of the solution proposed in Chapter II relies on a safe and effective link monitoring mechanism to accurately report the status of the CDL. A safe mechanism can be defined as a mechanism that ensures acceptable link quality by declaring the link "down" before performance degrades below guaranteed parameters. An effective mechanism is tolerant to short-term phenomena occurring on the link [Ref. 16]. There are two fundamental problems that must be addressed in the design of a link monitoring mechanism. The first is inaccuracy in the measurement of the metric used to determine link quality while the second is the occurrence of artificial and undesirable changes in the link status. The link monitoring mechanism utilized in the CDL model was originally proposed by Eichelberger [Ref. 17]. The remainder of this section presents thise mechanism, as well as how it addresses these issues.

The current link monitoring mechanism utilizes fixed length, pseudo-random monitoring packets that are periodically transmitted across the link. The receiving CDL Manager checks the arriving monitoring packet and records the result in a fixed length history. Using this history, in conjunction with a predetermined hysteresis plot, the receiving CDL Manager determines the status of the link and transmits it to the sending CDL Manager via a link quality report. This link quality report contains information on both the status and trend of the link quality.

The monitoring packets are an attempt to address the measurement problem. These monitoring packets provide a measurement of the bit error rate (BER) across the link. Three parameters must be specified: the size, insertion rate, and pattern of insertion for the packets. All three of these are adjustable at simulation set-up time and subsequently fixed during the simulation. The packet size can be chosen to be large to give a better estimate of the BER at the expense of overhead across the CDL. Correspondingly, the packet insertion rate should be chosen to be just high enough to detect jamming pulses of minimum width while minimizing overhead. The pattern of

insertion is a by-product of the configuration of the CDL into a number of transmission pipes. To be most effective, the monitoring packets should be inserted into the available pipes based on an empty allocation scheme [Ref. 18].

The history and hysteresis plot are used to minimize undesired changes in the link status. Both, the length of the history and the thresholds of the hysteresis plot, are determined at simulation set-up time and subsequently fixed during the simulation. The length of the history maintained determines how "quickly" the mechanism responds to changes in the quality of the link. For example, a history of one would cause the link status to oscillate between up and down with every good/bad packet received. Thus, the length of the history must be balanced to prevent the status from changing too rapidly or too slowly in response to changes in the link quality. The hysteresis provides a means to further refine this response. In essence, it serves as a history of the history. Just as in the history itself, the hysteresis thresholds affect the response time of the system. If the thresholds are too close together, the status tends to oscillate unnecessarily and if the thresholds are too far apart, the mechanism reacts too slowly.

There are two major deficiencies in the present version of the link quality monitoring mechanism. The first is that the use of periodic monitoring packets does not provide an exact measurement of the current BER across the link. The second concerns the history, which does not completely eliminate undesired changes in the reported link status. These need to be explored and are included in the suggestions for follow-on work provided in the last chapter.

C. ERROR CORRECTION CODING

This section briefly describes the validity of the general error-correction model chosen for the CDL network model. It is important to note that the validity of the solution proposed in Chapter II itself is independent of the error-correction method chosen. For the purposes of the simulation, this thesis makes use of a general model that

corrects a number of errors proportional to the overhead of the error-correction mechanism itself. In other words, the larger the overhead introduced by the error-correcting mechanism, the more errors it will detect.

The error-correction model utilized can be viewed as a convolutional encoder. A convolutional encoder has the property that for every k bits shifted into the encoder, n bits ($n > k$) are shifted out. Thus, a 1/2 rate convolutional code adds one bit of overhead for each information bit. This allows the overhead to be modeled as a percentage of the size of the information packet. The CDL simulation model uses a 2/3 rate convolutional code. Thus, a half bit of overhead is added for each information bit. It is assumed that the resulting convolutional code is strong enough to correct any errors generated by the imposed jamming. Given the extremely high link margin of the CDL, a 2/3 convolutional code should be able to produce a bit error rate less than 10^{-7}. This makes the above assumption reasonable given the sizes of the packets used in the simulation. At the expense of increased delay, an interleaver should be added to the system to minimize the impact of burst errors.

IV. CDL MODEL OVERVIEW

This chapter provides an overview of the model of an internetwork that uses the CDL constructed in OPNET. Together with the user's guide in Appendix A, it provides the reader with the information necessary to understand and utilize this model. This chapter also provides the background and motivation for the various decisions made during the development of the model. It is vital that the reader cover this material, as well as the detailed explanations of the next chapter, prior to making any modifications to the model. The first section presents the flow of data in the model, which is a presentation of the components and connectivity encountered in the actual image data transmission. The second section outlines the control information flow, which is a description of the components and connectivity required to facilitate the establishment and maintenance of the data flow. The third section describes the critical network procedures performed by the model.

Chapters IV and V make use of italics and quotes to differentiate between the different types of components within an OPNET model. *Processes* and *states* will be expressed in italics, while *"packets"*, *"fields"*, and *"attributes"* will make use of quoted italics. Throughout this chapter reference is made to specific components of the simulation model which will be discussed in greater depth in the next chapter. The reader should treat this chapter as an overview, setting the stage for the next chapter and, upon completion, is encouraged to revisit this chapter.

A. DATA FLOW

The data flow for the CDL model is illustrated in Figure 3. Without a loss of generality, this figure and the following discussion assume the transmission of real-time image data from a station on the collection platform to a station on the surface platform.

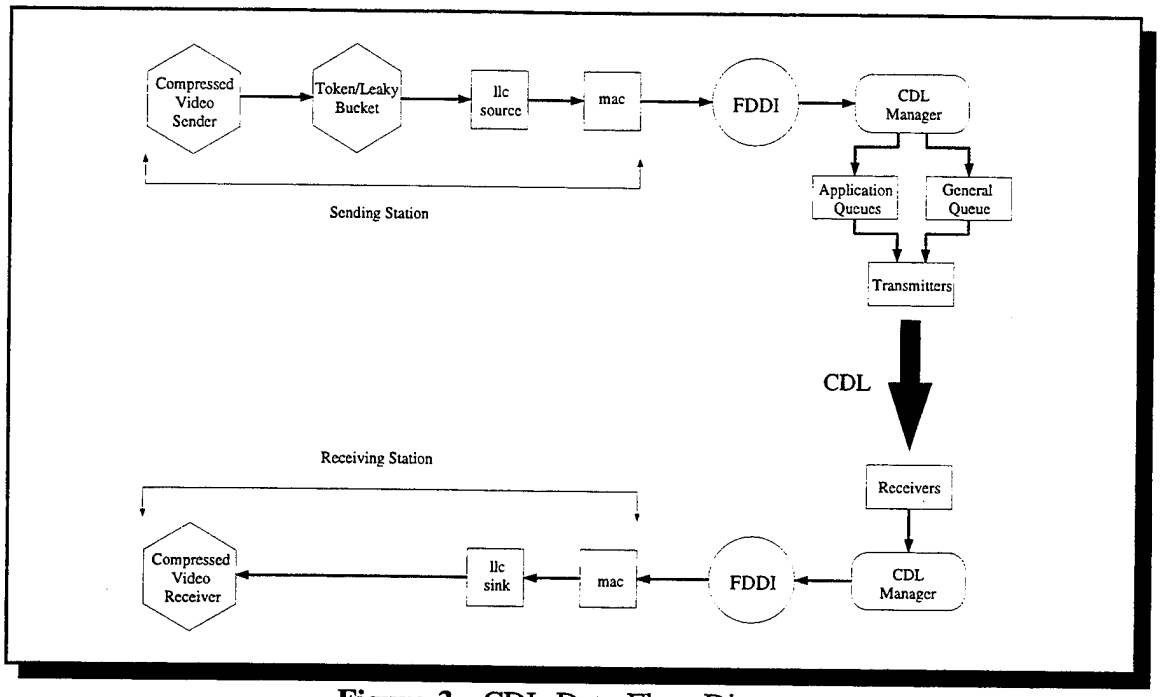


Figure 3. CDL Data Flow Diagram

The image data is placed into the source queue located within the token-leaky bucket of the sending real-time application. This queue is assumed to possess the capacity to buffer the maximum amount of data contained within a single image. The output from the decomposition/compression routine is loaded into this queue according to resolution level. The level one resolution packets are placed at the head of the queue, while the level five packets are placed at the tail of the queue. The queue is flushed and reloaded with the arrival of each new image. Upon transmission, the packets pass through the token-leaky bucket and are presented to the logical link control (llc) and medium access control (mac) layers. In the case of the CDL model, a FDDI LAN resides at this mac layer. The outgoing packets are encapsulated in FDDI frames and forwarded to the local CDL manager.

At the local CDL manager, incoming frames destined for the opposite LAN are demultiplexed according to the source address. A real-time data packet is placed in the queue reserved for the traffic from its sending application. Forward error correction is

applied to the packet if the forward error correction mechanism is activated. Finally, the packet is assigned to a CDL pipe based on an empty allocation scheme and transmitted across the CDL according to the transmission rate for its particular queue.

At the receiving CDL manager, the packet is placed on the attached LAN and sent to the destination application. Upon reaching the destination, the real-time data packet is forwarded to the compressed video receiver where the appropriate statistics are gathered and the packet is discarded.

B. CONTROL INFORMATION FLOW

The control information flow for the CDL model is illustrated in Figure 4. This figure and the subsequent discussion outline the control information flow required to transmit real-time data packets between two LANs across the CDL.

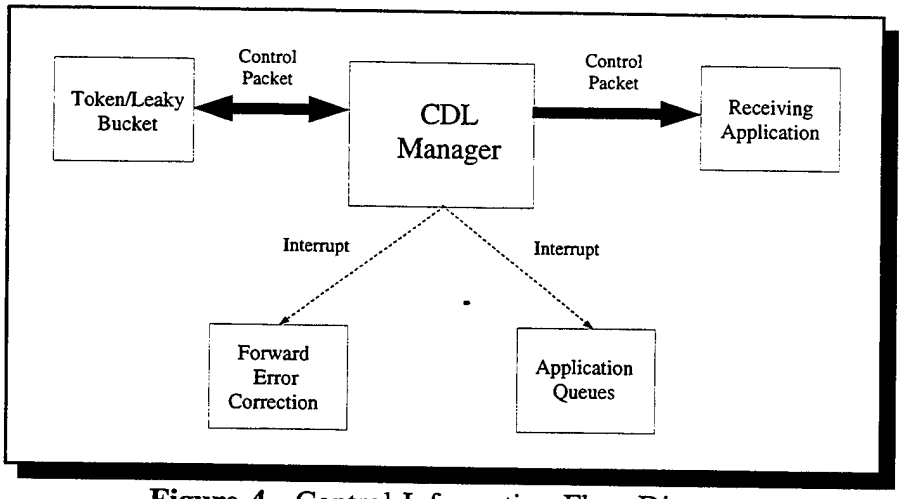


Figure 4. Control Information Flow Diagram

The CDL manager exchanges control information with the sender's token-leaky bucket and the receiver's compressed video receiver through the use of control packets transmitted across the LAN. For the receiving application, the control packets must be

transmitted across the CDL. This control packet specifies the maximum delay bound that is encountered by the real-time data packets that are transmitted through this flow establishment. For the sending application, the token-leaky bucket sends a request for a real-time data transfer to the local CDL manager when the application generates image data for transmission. The CDL manager replies with a control packet containing the average and peak data rates assigned to the application. This control packet is also used to dynamically alter these rates when required by the CDL manager. If these rates represent a reduction in the bandwidth assigned to the application, the application must respond with an acknowledgement packet. This acknowledgement is only required for a reduction in transmission rate because the CDL manager can increase its bandwidth before the sender increases the transmission rate, but it cannot decrease its bandwidth before the sender decreases the transmission rate.

The CDL manager uses interrupts to exchange control information with the local processes resident within the CDL manager. It enables/disables the forward error correction algorithm by setting or clearing an FEC flag that is read by the forward error correction mechanism. Similarly, the CDL manager controls the rates out of the application queues by designating the transmission rate variables for the appropriate queues.

C. EXPLANATION OF PROCEDURES

This section presents the four essential procedures that must be executed by this network model. They are:

(1) The request for a real-time data transfer initiated by the sending application.

(2) The normal transmission of real-time image data across the CDL.

(3) The network's response to the detection of jamming across the CDL.

(4) The network's response to the suspension of jamming across the CDL.

This section serves as an overview, laying out the steps the model goes through without going into the details of how they are accomplished. The details specific to each module are covered in the next chapter.

1. Request to Establish a Real-Time Flow

The following is a description of the procedure undertaken when an application seeks to establish a real-time data flow across the CDL. The sending application sends a request for a real-time data transfer to the CDL Manager responsible for the LAN. This request is a control packet of the format *"compr_vdo_cntrl_pkt"* that contains the source and destination addresses for the flow, as well as the desired and minimum acceptable data rates. For simulation purposes, the desired rate is calculated to allow the transmission of all five layers of decomposed image data in the period between image arrivals. The minimum acceptable rate represents the transmission of only level one in this time frame. Upon reception of the request, the CDL Manager accepts or rejects the request based on available bandwidth at the CDL interface. Presently, the admission control algorithm is not implemented and, accordingly, the CDL Manager accepts all requests. It is envisioned, as part of the required follow-on work, that a rejection will trigger a renegotiation process.

If the request is accepted, the CDL Manager reserves the required bandwidth based on the minimum data rate and the overhead associated with the implemented forward error correction algorithm. The CDL Manager stores the source and destination addresses as well as the desired and minimum data rates in a real-time application database. It uses this information and its knowledge of the network topology to calculate a maximum delay bound based on the work of Parekh and Gallager [Ref. 3]. This delay bound is transmitted to the destination application, which uses it to set its playback point. The CDL Manager sets up a separate queue for the new application and initializes its transmission rate based on the flow's average packet size combined with the overhead added by the FDDI and PPP protocols. The leaky and token bucket rates are calculated and transmitted to the sending application within a *"CDL_mgr_cntrl_pkt"*. The leaky bucket rate is set to the maximum data rate supported by the attached LAN, while the token bucket rate is set to the desired rate, or the minimum rate in the presence of jamming, of the application. Upon receiving the data rates from the CDL manager, the token and leaky buckets convert them to token rates and begin transmission of data packets. At this point, a real-time flow has been established between the appropriate applications. A summary of the procedure for the establishment of a real-time flow is shown in Figure 5.

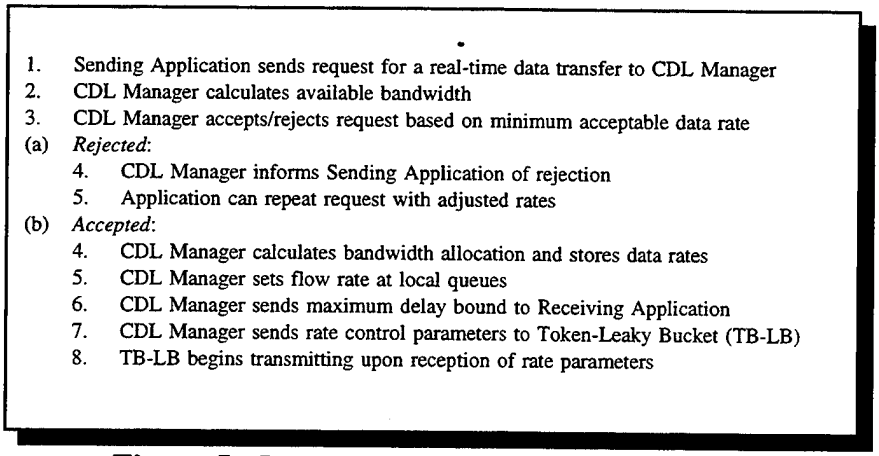


Figure 5. Request to Establish a Real-time Flow

2. Normal Transmission

The following is a description of the procedure for normal transmission of real-time compressed video packets across the CDL. Initially, the MATLAB data files are generated by the appropriate MATLAB video compression algorithm. These files consist of a vector for each component generated by the decomposition routine. These components are ordered according to the resolution levels specified by the compression routine. Packets are created from these files based on two parameters: (1) the image number and (2) the sequence number of the packet within the image. These parameters are placed in the header of each fixed size packet and placed in the application queues within the *token bucket* module. The packets are then loaded into the application queue based upon resolution level. Thus, the number of levels transmitted is determined by the rate at which the queue is emptied. These queues are flushed and reloaded at the arrival of each new image. The packets are encapsulated into FDDI frames by the *mac* process and transmitted to the CDL network interface (cdl-ni) via the local FDDI ring. Upon arrival at the cdl-ni, the packet is sent to the appropriate queue within the CDL Manager based on its source address. Upon departure from the CDL Manager, the packet is encapsulated into the PPP protocol and FEC is applied, if appropriate. As the individual queue rate permits, the packet is allocated to a bit pipe and transmitted across the CDL. If the packet survives the transmission across the CDL, the FDDI packet is removed from the PPP protocol and forwarded to the destination node along the destination's FDDI ring. At the receiving application, the data packet is decapsulated and passed to the *cmpr_vdo_rcvr*. The *cmpr_vdo_rcvr* logs the packet into its output data file, gathers the appropriate statistics, and destroys the packet. This data file is subsequently used by the MATLAB reconstruction routines to reproduce the transmitted images. A summary for the procedure for normal transmission of a real-time application across the CDL is shown in Figure 6.

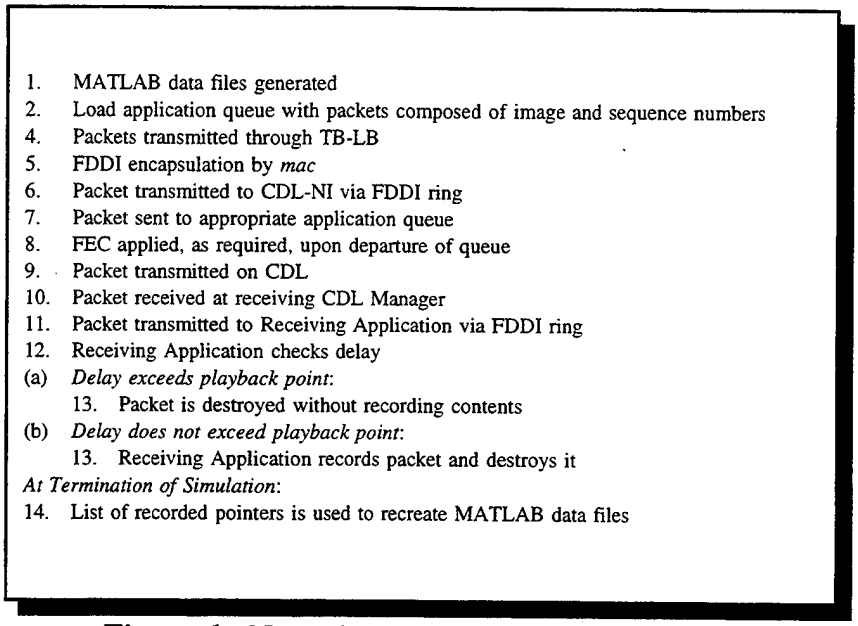


Figure 6. Normal Transmission Across the CDL

3. Detection of Jamming

The link monitoring mechanism embedded within the CDL Manager allows it to detect the presence of jamming on the CDL. When the link monitoring mechanism reports that the link is "bad", the CDL manager activates the forward error correction algorithm. This causes the manager to loop through the real-time application database and send control messages to all the current real-time applications. These control packets contain the new leaky and token bucket rates to account for the additional overhead of the FEC bits. The leaky bucket rate typically remains the same, while the token bucket rate is changed to the application's minimum acceptable rate. Upon reception of the lower data rate, the token bucket converts this rate to a token rate by dividing by the size of a token (in bits) and sends back an *ack* to the CDL Manager. The process flushes its token bucket and application queue and subsequent token arrivals are ignored until the

next image arrives. This has the effect of resetting the application, allowing the new rate to take effect immediately upon the arrival of the next image.

Upon reception of the *ack* from the real-time application, the CDL Manager implements the forward error correction at the CDL interface. The appropriate queue transmission rate is set to the minimum data rate plus the overhead corresponding to the FDDI and PPP packets and the forward error correction scheme. The queue is flushed to prevent the backlog of old image data and a flag is set within the real-time application database to denote the use of FEC for this particular application queue. Thus, as the FDDI packet is transmitted, it is encapsulated in PPP and the size is adjusted to reflect the FEC overhead. An *"FEC"* field attached to the packet is set to denote the level of FEC protection. This level is expressed in terms of a ratio of acceptable errors per bit. This *"FEC"* field would not exist in an actual implementation, it is used in the simulation environment to artificially provide error protection for the packet. A summary for the procedure for the detection of jamming is shown in Figure 7.

1. CDL Manager detects unacceptable BER from link monitoring mechanism
2. CDL Manager sends minimum rate control parameters to LB-TB
3. LB-TB sends acknowledgement to CDL Manager and sets a new transmission rate
4. Upon reception of *ack*, CDL Manager sets flow rate at local queues
5. CDL Manager enables FEC

Figure 7. Detection of Jamming

4. Suspension of Jamming

The CDL Manager recognizes the suspension of jamming when the link monitoring mechanism reports the link as "good." The CDL Manager then initiates the termination of the FEC algorithm. The manager loops through the real-time application database and sends control packets to the current real-time applications on its LAN.

These packets contain the original desired token bucket rate for each application. No acknowledgement is required because the FDDI ring is assumed error-free. Thus, the CDL manager immediately terminates FEC for the application queues and sets the queue transmission rates to reflect the desired data rate without the overhead of forward error correction. Upon reception of the control packet, the token bucket converts the data rate to a token rate and flushes the token bucket and application queues. A summary for the procedure for the suspension of jamming is shown in Figure 8.

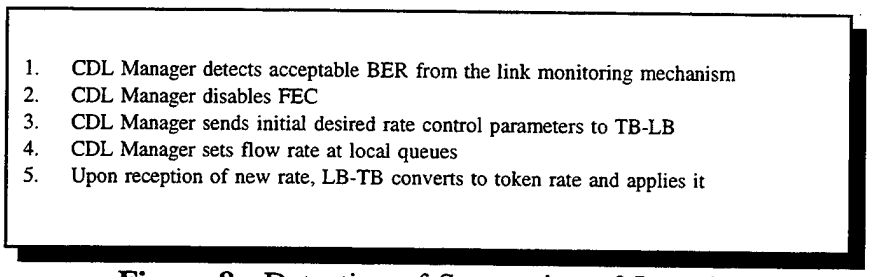


Figure 8. Detection of Suspension of Jamming

V. CDL MODEL DETAILS

This chapter provides detailed explanations of the various components of the Common Data Link model in OPNET. In addition, it looks at the programs required to interface the OPNET output with the MATLAB compression/decompression algorithms. This thesis has completely overhauled the model as it existed [Ref. 17, 19, 20] to make it more efficient and logical. As a result, all aspects and components of the model are discussed and explained in detail. The only major portion of the model that has not been revamped is the underlying model of the FDDI network that serves as the local LAN at both ends of the physical link. In addition, although reorganized, most of the link monitoring mechanism remains intact. For a detailed discussion of the FDDI network and its underlying philosophy, the reader should refer to the OPNET Models Manual [Ref. 21] and the theses of Nix [Ref. 20] and Karayakaylar [Ref. 19].

The chapter is divided into three major sections corresponding to the major divisions of the model. The first section addresses the real-time FDDI station. This module models the real-time compressed video application and its token-leaky bucket interface to the FDDI ring. The next section investigates the CDL Network Interface. This module provides all the elements necessary to interface the local FDDI ring to the Common Data Link. It is composed of an FDDI station with an attached CDL Manager. The final section deals with the MATLAB code generated to allow the results of the OPNET simulations to be applied to the MATLAB reconstruction algorithms written by Carvalho [Ref. 11].

A. REAL-TIME FDDI STATION

For applications requiring real-time guarantees, a modification must be made to the standard OPNET FDDI station. This new station is referred to as a real-time FDDI station and includes a token/leaky bucket to shape the traffic flow at the FDDI network

interface. In this thesis, the real-time compressed video application resides within this station. The mac (medium access control) serves as the interface to the FDDI network. A packet is generated by an ideal source (a source with no associated delays or overhead) and sent to the token bucket to mark the arrival of each new image. The token bucket creates the application queues and initiates the flow establishment procedure. Once the connection has been established, the token bucket is responsible for enforcing the average data rate. The token bucket transmits the application data packets to the leaky bucket, where the peak data rate is enforced. The packet is then processed by *llc_src*, which acts as a service access point for the application into the *mac*. The *mac* encapsulates the packet in the FDDI protocol and is responsible for transmitting the packet along the FDDI ring.

Upon the arrival of a packet from the FDDI ring, the *mac* decapsulates it and sends it to the *llc_sink*. The *llc_sink* acts as a filter, forwarding real-time packets to the compressed video application, while recording statistics and destroying the remaining general traffic packets. The *cmpr_vdo_rcvr* is responsible for the processing of all the received real-time packets. This includes logging the image information needed to reconstruct the image and recording the appropriate statistics to generate the required throughput and delay data.

The control path within this module is very similar to the data path. Control packets are generated in the token bucket process, where they are designated as control packets and filled with the appropriate information. The packet proceeds to the mac, where it is encapsulated and addressed to the local CDL manager. The packet is subsequently transmitted along the FDDI ring. Arriving control packets are routed through the sink and the compressed video receiver. The compressed video receiver sends them to the leaky and token bucket processes, where the information is extracted. Figure 9 presents the real-time FDDI station module.

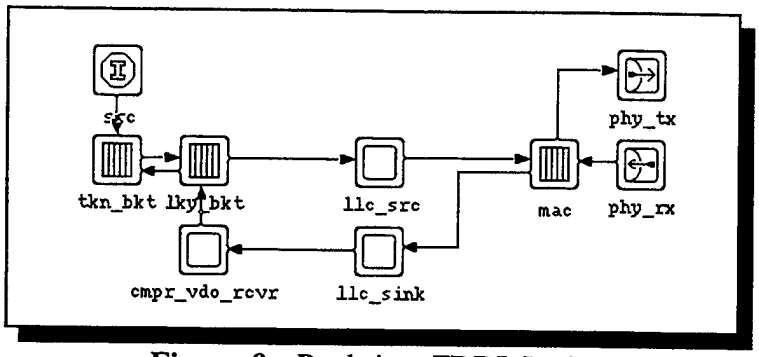


Figure 9. Real-time FDDI Station

The processes *llc_src* and *llc_sink* are modifications of the corresponding processes in the original OPNET FDDI model. The source has been modified to allow the presentation of an independent application to the FDDI ring. In addition to being able to generate non-real-time simulation packets, the *llc_src* can also receive packets from a higher layer and pass them onto the mac process. The sink process has been modified to achieve the same goal in the reverse direction, passing designated packets to the higher application, while processing and destroying the general simulation packets destined for this station. The *phy_tx* and *phy_rx* are the point-to-point transmitter and receiver processes, respectively, that represent the physical FDDI ring. The following sections will discuss the token/leaky bucket processes and the compressed video receiver process in detail. The *mac* process will be discussed in the following chapter as part of the network interface to the CDL.

1. Token/Leaky Bucket

The token/leaky bucket implementation is realized by two separate processes: (1) *tl_bkt_tow* (the token bucket) and (2) *tl_bkt_std* (the leaky bucket). Both of these processes are extensively modified versions of a *tl_bkt* process originally designed by

Nishimura[1]. Combined, these processes form the token/leaky bucket traffic shaping mechanism. Essentially, this mechanism enforces a predetermined peak and average data flow for the attached application.

a. Token Bucket

The token bucket process (*tl_bkt_tow*) is tasked with three major functions: (1) enforcing the mean data rate for the attached application, (2) maintaining the application data points, and (3) creating/destroying the application's control packets. These functions are discussed in the context of the process state diagram, shown in Figure 10.

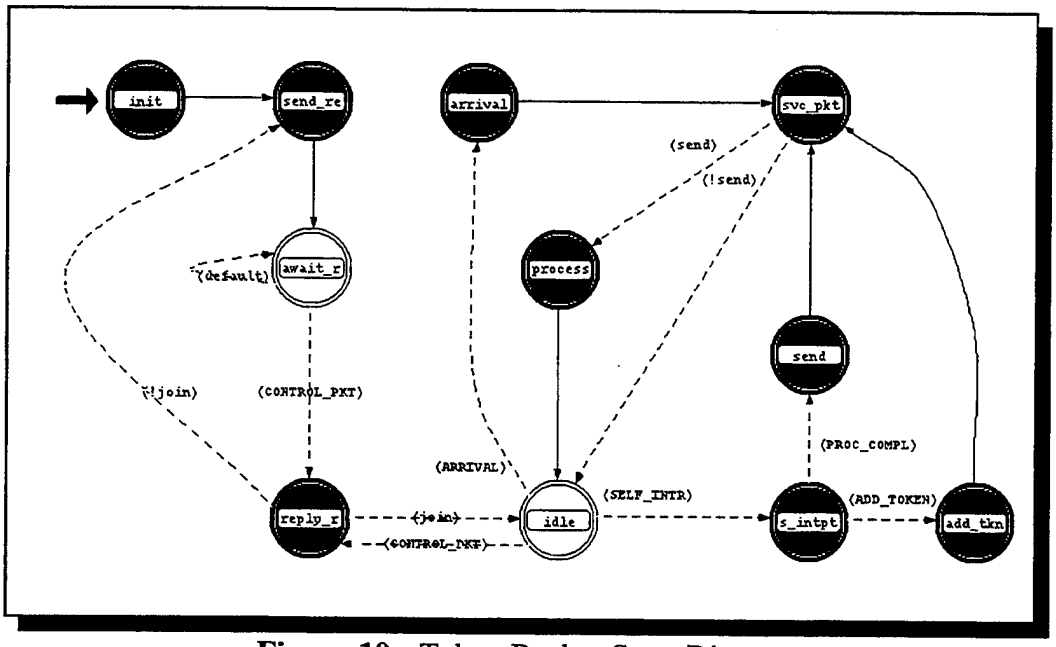


Figure 10. Token Bucket State Diagram

[1]This was designed as part of an EC4850 class project entitled "Traffic Shaping on a TCP/IP Based Internet Model" by LT Bryan Nishimura at the Naval Postgraduate School, September 19, 1994.

Upon simulation initiation, the process moves from the *init* state to the *send_request* state. In this state, a control packet, *"cmpr_vdo_cntrl_pkt"* (Figure 11), is generated and sent to the local CDL Manager as a request to establish a real-time data transfer. The *"desired_rate"* and *"min_acceptable_rate"* parameters are user-defined. The process then moves to the *await_reply* state, where it sits idle, awaiting the CDL Manager's reply. The arrival of a control packet, *"CDL_manager_cntrl_pkt"* (Figure 12), forwarded from the leaky bucket process, triggers a transition to the *reply_rcvd* state. If the *"join"* field is set to "1", then the process records the *"token_rate"* and transforms it into a token rate by dividing it by the token size (in bits). This token rate is used to schedule an interrupt for the arrival of the first token into the bucket. If the specified rate is less than the current rate, the process sends an acknowledgement back to the CDL manager prior to transitioning to the *idle* state. In addition, the token bucket and application queue are flushed and subsequent token arrivals are masked until the arrival of the next image in the *arrival* state. If the *"join"* attribute is not set to "1", then the process returns to the *send_request* state, reinitializing the joining procedures.

Field Name	Type	Size (bits)	Default Value	Default Set
control_packet	integer	0	1	set
src_addr	integer	16	0	unset
dest_addr	integer	16	0	unset
desired_rate	double	16	0	unset
min_acceptable_rate	double	16	0	unset
packet_size	integer	16	0	unset
ack	integer	1	0	set

Figure 11. *"cmpr_vdo_cntrl_pkt"*

Field Name	Type	Size (bits)	Default Value	Default Set
control_packet	integer			set
token_rate	double	16	1000000	set
leaky_rate	double	16	1000000	set
join	integer	1	0	set

Figure 12. *"CDL_manager_cntrl_pkt"*

The *idle* state is used to allow the process to await the arrival of the next interrupt. The three possible interrupts are: (1) arrival of a control packet, (2) arrival of a packet from the ideal source process, and (3) a self-interrupt. The control packet forces a transition back to the *reply_rcvd* state which allows processing of the control packet from the CDL Manager. A packet from the ideal source generator signifies the arrival of a new image, while the self-interrupt could signal the arrival of a new token or the completion of transmission of a particular packet.

Field Name	Type	Size (Bits)	Default Value	Default Set
img_num	integer	8	0	set
lvl_num	integer	8	0	set
sgmt_num	integer	24	0	set
sgmt_size	integer	16	0	set
cr_time	double	0	0.0	set
control_packet	integer	0	0	unset

Figure 13. *"cmpr_vdo_fr"*

The arrival of a new image results in a transition to the *arrival* state. The arrival state flushes the current application queue and refills it with data packets of the form *"cmpr_vdo_fr"* (Figure 13). The physical packet size is determined by the following equation:

$$pkt\ size = (current\ pkt\ size) - (size\ of\ sgmt\_size\ fld) + (value\ of\ sgmt\_size\ fld) \quad (2)$$

Once created, the packets are inserted into the application queue in an order corresponding to the applicable resolution level. The process then moves to the *svc_pkt* state, which determines if there are enough tokens in the bucket to send the packet at the head of the queue. If there are enough tokens, the process transitions to the *process* state, otherwise it returns to the *idle* state, awaiting the arrival of more tokens. The process also transitions directly to the *idle* state when *svc_pkt* is entered and there are no packets queued for transmission. The *process* state calculates the time of transmission for the packet at the head of the queue based on the packet size and the user-defined *"send_rate"*.

The state then schedules a self-interrupt to mark the end of the transmission time and transitions back to the *idle* state.

A self-interrupt signifying the end of a packet transmission causes the process to transition into the *send* state through the *s_intpt* state. The *send* state removes the packet at the head of the queue and forwards it to the leaky bucket process. From this state, the process returns to the *svc_pkt* state to determine if another packet can be transmitted.

A self-interrupt marking the arrival of a new token causes the process to transition into the *add_tkn* state through the *s_intpt* state. Provided the user-defined maximum *bucket_size* is not exceeded, the token is added to the bucket. Prior to leaving this state, an interrupt is scheduled for the next token arrival. This state also exits to the *svc_pkt* state.

b. Leaky Bucket

The leaky bucket process (*tl_bkt_std*) is tasked with enforcing the peak data rate. This peak data rate is normally set to the maximum data rate supported by the attached LAN (100 Mbps for a FDDI ring). This process will be discussed in the context of its state diagram, shown in Figure 14.

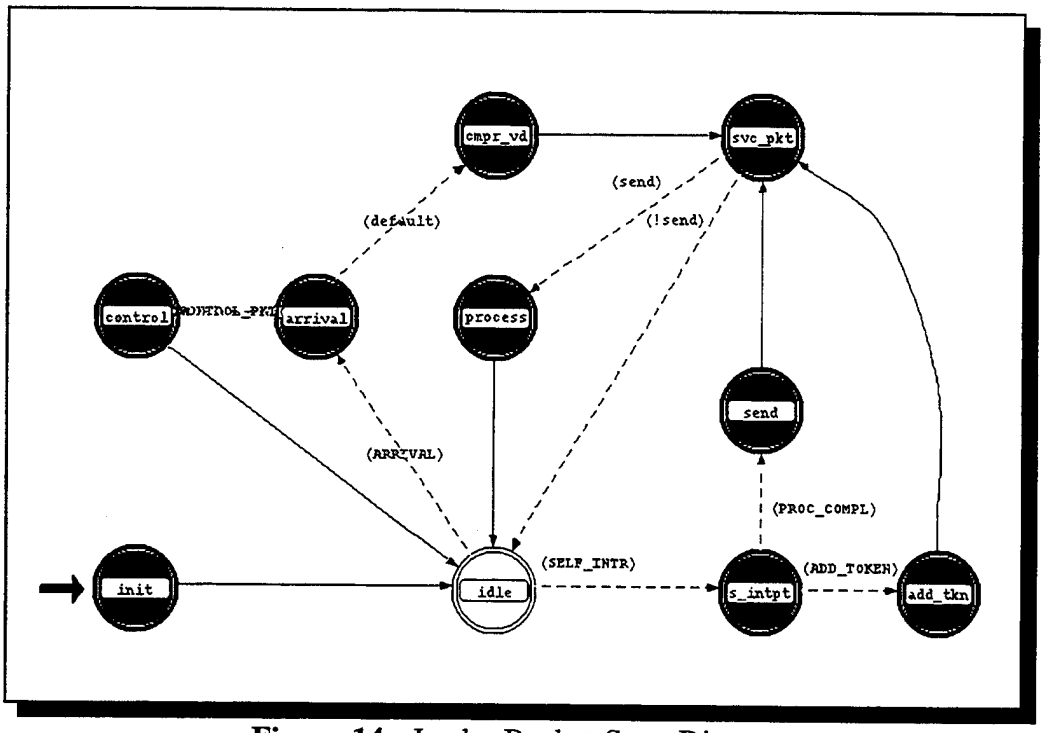


Figure 14. Leaky Bucket State Diagram

Upon simulation initiation, the process moves directly from the *init* state to the *idle* state. In the *idle* state there are two possible events that can occur: (1) the arrival of a packet or (2) a self-interrupt. The arriving packet can be a control packet from the local CDL Manager or a data packet from the token bucket process. The self-interrupt can signify the completion of the transmission of a packet or the arrival of a new token. The arrival of a control packet results in transition to the *control* state where, if the *"join"* field is set to "1", the *"leaky_rate"* is extracted and transformed into a token rate (explained in the previous section). This token rate is used to schedule the arrival of the first token. In addition, the control packet is then forwarded to the token bucket process. The leaky bucket process handles arriving data packets and tokens in the same manner as the token bucket process (described above). The only exception is that an arriving token clears any previous tokens in the bucket. As a result, the leaky bucket will never contain more than one token. This allows the mechanism to enforce a peak data rate.

2. Compressed Video Receiver

The compressed video receiver process (*cmpr_vdo_rcvr*) is tasked with the functions of (1) gathering the desired compressed video packet statistics and (2) creating a file that can be used by MATLAB to reconstruct the transmitted images. The process consists of three states as illustrated in Figure 15.

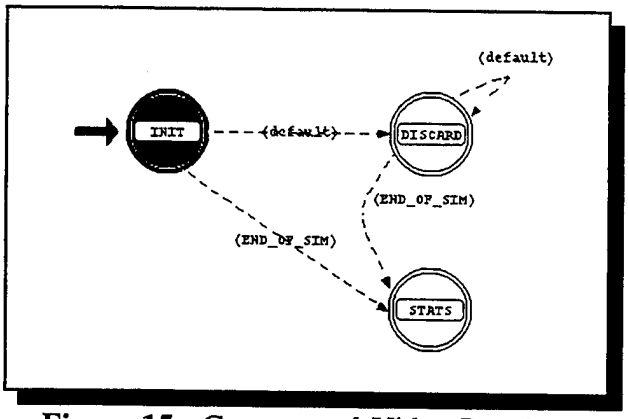


Figure 15. Compressed Video Receiver State Diagram

The *discard* state serves as the central state within the compressed video receiver. This state is re-entered with the arrival of each new compressed video packet, until the *"end_of_simulation"* signal causes a transition to the *stats* state. The *discard* state records the creation time and the image and sequence numbers contained within each packet. Using the creation time, coupled with the current time, the mean and instantaneous delays are calculated and recorded for future analysis. The mean throughput is also calculated and recorded. The image and sequence numbers for each packet are recorded in an output file specific to each compressed video application and specified by the user at runtime. This file is in a vector format suitable for integration into the MATLAB recomposition routines. It is opened in the *init* state and closed in the *stats* state.

B. CDL NETWORK INTERFACE

The Common Data Link Network Interface (CDL-NI) serves as the interface between the local LAN and the Common Data Link. The module is essentially composed of an FDDI station connected to the CDL through a process called the *CDL Manager*. The *phy_tx*, *phy_rx*, *mac*, *llc_src*, and *llc_sink* make up the FDDI station. The *mac* and *llc_sink* have been modified to allow connection to the *CDL Manager*. The *CDL Manager* is responsible for handling all traffic coming from and going to the CDL, which is modeled by the *pr_#* and *pt_#* processes.

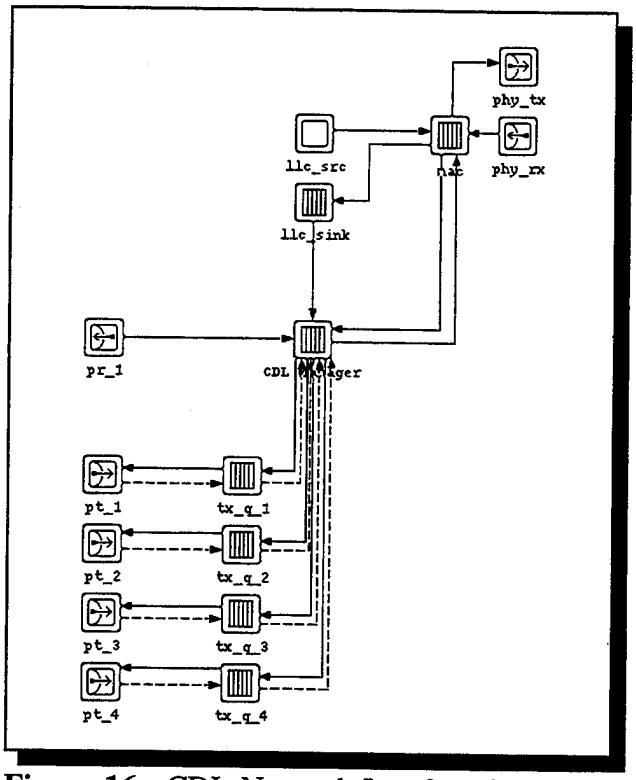


Figure 16. CDL Network Interface for the Collecting Platform

The *llc_sink* has been modified to allow it to send control packets to the *CDL Manager* as required. This is achieved by a filtering operation that detects control packets and forwards them to the *CDL Manager* via the data path between the two processes.

Other general traffic packets destined for this station are processed by the sink. The *mac* is discussed in a subsequent subsection.

The physical data link is modeled by the point-to-point transmitters and receivers (*pr_#* and *pt_#*, respectively). The single receiver in the collection platform cdl_ni (shown in Figure 16) represents the command link from the surface to the collecting platform. This is a 10.71 Mbps link that has forward error correction applied to it. The four transmitters represent the 274 Mbps return link. These different receivers are used to model a few of the channels, or pipes, available in the return link. The sum of the bandwidth of the four receivers is equal to the total bandwidth of the link. The transmitter queue processes (*tx_q_#*) are used to queue up PPP packets scheduled for the respective transmitters. The queues receive a statistic from the attached transmitter notifying them that the transmitter is either busy or idle. When the transmitter is idle, the queue will send the packet at its head. In addition, the queue sends a statistic to the *CDL Manager* informing it of the number of bits currently buffered in the queue. The number of transmitters can be altered by placing the appropriate number of point-to-point transmitters and transmitter queues in the cdl_ni module and entering the number of transmitters in the *"number_of_xmtrs"* attribute of the *CDL Manager* process. A corresponding number of receivers must be added to the cdl_ni at the other end of the link. The *CDL Manager* is discussed in a subsequent subsection.

1. MAC

The *mac* process serves as the interface to the local FDDI ring. All FDDI packets that arrive at the cdl_ni, whether from the local ring or the CDL, are processed by the *mac*. The *mac* at this interface is a modified version of the *mac* present at a standard FDDI station. The fundamental difference is that the mac resident at the cdl_ni has the additional capability to handle packets destined for, or arriving from, the CDL. The following discussion briefly covers the basic functionality that is common to all *mac*

processes and then expand upon the items that have been added by this thesis. Figure 17 presents the *mac* process state diagram.

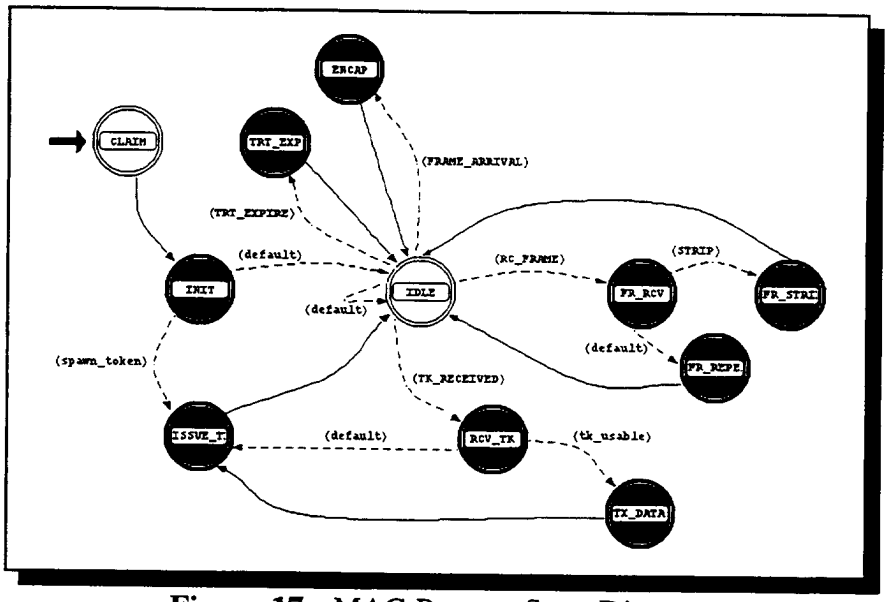


Figure 17. MAC Process State Diagram

The functions directly related to the FDDI ring are discussed in the OPNET Models Manual [Ref. 21] and elaborated on in the thesis of Karayakaylar [Ref. 19]. Essentially, the process, if designated as a spawn_station, spawns a token and the token is circulated around the ring. When the token arrives at the current station, it is captured and the station begins transmitting its synchronous and asynchronous data in accordance with the specifications of a FDDI ring. The synchronous bandwidth is a user-defined attribute, *"mac_sync_bandwidth."*. Upon completion of transmission, the station forwards the token to the next station on the ring. The *mac* is responsible for maintaining all counters associated with the FDDI ring. When a packet destined for this station arrives on the ring, it is removed, decapsulated, and forwarded to the *llc_sink*. When a packet arrives from the *llc_src*, it is encapsulated and queued up for transmission on the ring.

Each packet contains a flag specifying whether it represents synchronous or asynchronous data.

The *mac* process has been modified to allow it to process packets associated with the CDL. The *mac* treats the CDL as another physical transmission medium much like the attached FDDI ring. FDDI packets arriving over the CDL are forwarded to the *mac* process, where the destination address is examined. If the packet is destined for a station on the local LAN, the packet is queued for transmission over the local FDDI ring. Otherwise, it is returned to the *CDL Manager*. An identical filtering process is applied to packets arriving from the local FDDI ring.

2. CDL Manager

The *CDL Manager* is the heart of the CDL network interface. This process serves as the manager for both the CDL and the attached local LAN, by providing the bandwidth allocation for the CDL and the attached FDDI ring. In addition, the *CDL Manager* contains the link monitoring and forward error correction mechanisms. These two mechanisms are discussed in greater detail in the following sections. This section discusses the details of the *CDL Manager* itself. The process state diagram (Figure 18) was designed to permit easy extensions to facilitate follow-on work. It was also designed to be symmetric in the sense that the *CDL Manager* is identical at both surface platform and the collection platform interfaces.

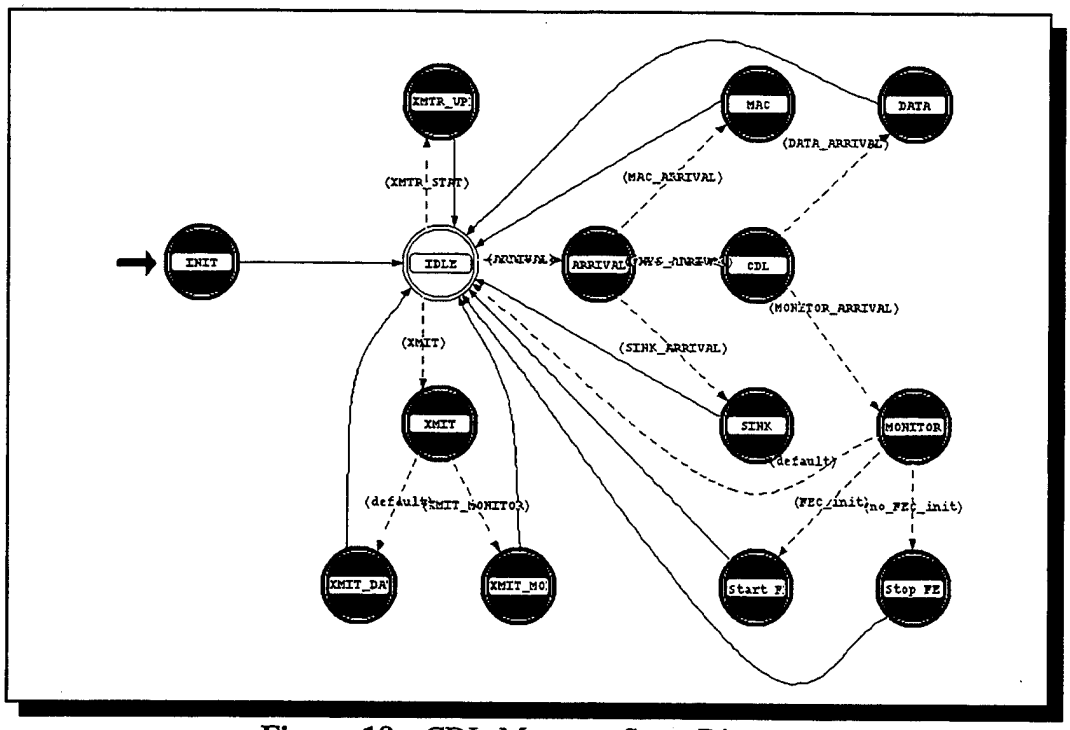


Figure 18. CDL Manager State Diagram

Upon simulation initiation, the process moves from the *init* state into the *idle* state. There are three events that cause a transition out of this state: (1) the arrival of a packet, (2) an update from the transmitter queues, and (3) a self-interrupt to mark the completed transmission of a previous packet.

a. Packet Arrivals

An arriving packet can come from one of three possible sources: the mac, the sink, or the CDL. Each arrival forces a transition into a correspondingly named state. In the *mac* state, the arriving packet is filtered by source address to determine if it is a real-time application packet. If the source address is contained in the real-time database, the packet is added to the appropriate real-time queue. Otherwise, the packet is placed in the general application queue (subqueue[0] within the process). If the queue was

previously empty and no packets are presently being transmitted, a self-interrupt is generated to trigger the transmission cycle. In the *sink* state, the arriving packet is assumed to be a control packet. This packet can either be a request for real-time data transfer from a real-time application or an acknowledgement of a reduction in allowed transmission rate from an existing real-time application. In the former case, the source and destination addresses, the desired and minimum acceptable data rates and the packet size are removed from the *"cmpr_vdo_cntrl_pkt"* and stored in the real-time application database. The *CDL Manager* creates a *"CDL_manager_cntrl_pkt"* and sets the *"join"* field to one to signify that a real-time connection will be established for the application in question. The *"token_rate"* field is set to the desired data rate, while the *"leaky_rate"* is set to the maximum data rate provided by the local LAN, in this case: 100 Mbps. The control packet is encapsulated in a FDDI frame and forwarded to the *mac* process. A real-time queue is set up for the applicable application, its data rate is calculated using the overhead of the FDDI ring and the PPP protocol, coupled with the packet size information stored in the database. The general application queue data rate is reduced by a corresponding amount. If the arriving control packet is an acknowledgement of the reduced transmission rate, the *CDL Manager* applies the forward error correction scheme to all subsequent packets from that real-time application prior to transmission.

A packet arriving from the CDL forces a transition into one of two different states based on the type of packet: data packet vs link monitoring packet. A data packet causes a transition to the *data* state, which decapsulates the FDDI packet and forwards it to the *mac* process. A link monitoring packet causes a transition into the *monitor* state, which will be explained in greater detail in the next section.

b. Transmitter Update

An transmitter queue update is triggered by a change in the queue statistics supplied by the transmitters. This causes a transition into the *xmtr_update*

state. In this state, a buffer is maintained that has an entry for each transmitter queue. The received statistic informs the *CDL Manager* of the number of bits currently contained in the applicable transmitter queue and is stored in the appropriate buffer entry. This information is utilized by the empty allocation transmission scheme discussed in the next paragraph.

c. Completion of Packet Transmission

A self-interrupt is generated whenever the *CDL Manager* is capable of transmitting another packet to the transmitter queues. This interrupt causes a transition into the *xmit* state followed by a transition into either the *xmit_data* or the *xmit_monitor* state. The former is the default transition, while the latter is entered when the interrupt signals the periodic transmission of a link monitoring packet, which is initialized in the *init* state. In the *xmit_monitor* state, a *"ppp"* packet is generated and the fields *"pid_h"* and *"pid_l"* are set to indicate a link quality monitoring packet. This packet is placed at the head of the general application subqueue to force earliest possible transmission and guarantee that the link monitoring mechanism receives the required bandwidth dictated by its transmission rate. If no packets are presently in awaiting transmission, a transmit interrupt is generated prior to transitioning back to the *idle* state. In the *xmit_data* state, the interrupt code number is used to determine which subqueue is available for transmission. The packet at the head of the applicable subqueue is removed and encapsulated in a *"ppp_ml"* packet for transmission on the CDL. Forward error correction is applied based on a FEC flag maintained for each subqueue within the real-time application database. (The FEC mechanism is discussed in greater detail in the next section.) Two load balancing algorithms are available to determine the destination transmitter queue. The first uses a round robin scheme, while the second, called the empty allocation scheme, chooses the transmitter queue currently buffering the smallest number of bits. To effectively make use of the

concept of bandwidth allocation, the second algorithm is utilized. An interrupt is set for the applicable subqueue to signal the end of the transmission. This transmission time is based on the size of the packet and the subqueue transmission rate (contained in the real-time application database). A flag is used to mark this particular subqueue as busy. This flag is cleared when the *xmit_data* state is entered and no packets are contained in the applicable subqueue for transmission.

3. FEC Mechanism

The forward error correction mechanism is embedded within the *CDL Manager*. When the *CDL Manager* is informed that the link has gone "bad" via a link quality report, the *CDL Manager* initiates the forward error correction algorithm. The *CDL Manager* cycles through its real-time application database, sending each application a control message updating its transmission data rate to the minimum acceptable rate advertised by that particular application. This rate is provided at connection establishment and stored in the real-time application database by the *CDL Manager* when it accepted the request. Upon reception, the leaky bucket and token bucket modules record updated transmission rates and convert them to token rates. In its role as the control module for the token-leaky bucket, the token bucket module returns an acknowledgement to the *CDL Manager*. This is a *"compr_vdo_cntrl_pkt"* with the *"ack"* field set to one. The token bucket process flushes its token bucket and application queue, allowing the revised data rate to take effect immediately. Upon reception of the acknowledgement, the *CDL Manager* cycles through its real-time application database to locate the appropriate application. The corresponding subqueue is flushed and its transmission rate is set to this minimum data rate. The FEC flag is set for this subqueue, which allows the forward error correction scheme to be applied to all subsequent packets from this subqueue.

The forward error correction scheme is modeled by increasing the size of the PPP packet by a ratio appropriate to the type of FEC utilized. An *"FEC"* field (integer, of size

zero, for simulation only) is set to denote the level of FEC protection. This field contains the acceptance threshold, expressed in correctable number of errors per bit, and is used by the error correction communication pipeline stage to determine whether or not the packet should be accepted. If the threshold is exceeded, the packet is discarded at the receiver.

4. Link Monitoring Mechanism

The link monitoring mechanism of [Ref. 17] has been redesigned to make it symmetric at both ends of the CDL and has been placed within the *CDL Manager* for consistency of functional organization. A link quality monitoring packet is sent at a user-defined rate across the CDL. The *CDL Manager* at the receiving end determines the number of errors within the packet and records the value in its history database. A ratio is calculated that determines the number of packets in error over the length of the maintained history. When the ratio exceeds a user-defined threshold, the status is declared as BAD. When the change in this ratio exceeds some user-defined threshold, a link quality report is transmitted back across the CDL to the sending *CDL Manager*. This link quality report provides the status of the link. This status is reported as either GOOD or BAD with a packet error ratio trend that is either going UP or DOWN. In addition to the mentioned parameters, the history length and reporting criteria are also user-defined. For a more detailed discussion of the theory behind the implemented method of link monitoring, refer to Eichelberger [Ref. 17].

This link monitoring mechanism is embedded in the *monitor* state of the *CDL Manager* process. This state is entered when a monitoring packet or a link quality report arrives at the *CDL Manager*. Upon the arrival of a link monitoring packet, the number of errors are determined and stored in the history database. This value is determined in the CDL pipeline (discussed in the next section) and maintained by OPNET in the constant OPC_TDA_NUM_ERRORS. In an actual implementation, this value would be

determined by comparing the received packet to a packetized pseudo-random bit stream maintained by each *CDL Manager*. The packet error ratio is calculated and if the value differs from the previous ratio by a user-defined threshold, a link quality report is generated. This is accomplished by creating a *"ppp"* packet with the appropriate values in the *"pid_l"* and *"pid_h"* fields. The *"LQR_info"* field is set to the current link status and trend. The status is set to GOOD or BAD based on a user-defined hysteresis. An UP trend signifies an increasing packet error ratio, while a DOWN trend signifies a decreasing trend. This packet is placed at the head of the general application subqueue for transmission.

When a link quality report arrives at the *CDL Manager*, the link status information is retrieved from the *"LQR_info"* field. As required, this information is used to initiate or suspend the forward error correction algorithm.

5. CDL Physical Pipeline

The physical Common Data Link is modeled in OPNET through the use of a point-to-point transmitter and receiver connection. In OPNET, a point-to-point link is realized as a four stage pipeline designed to reflect the characteristics of a physical link. The pipeline is composed of four separate C routines that are integrated into the OPNET simulation environment. The four stages are as follows:

(1) *Transmission Delay* - This stage models the transmission delay encountered by each packet. The default C routine, *dpt_txdel*, implements this delay by making use of the channel attribute *"data rate"* and the length of each packet. This routine has not been modified in the CDL implementation.

(2) *Propagation Delay* - This stage models the propagation delay experienced by a packet travelling across the link. The default C

routine, *dpt_propdel*, makes use of the channel attribute *"delay"* to apply a constant delay to all the packets. Once again, this routine has not been modified in the CDL implementation.

(3) *Error Allocation* - This stage determines the number of errors generated in each of the packets. The default C routine, *dpt_error*, makes use of the channel attribute *"ber"* to determine the fixed bit error rate used in the stochastic error generation process. Coupled with the length of the packet, the routine is able to calculate a number of errors to apply to each packet. This routine has been replaced by the C routine *cdl_pt_error* which contains the jamming mechanism.

(4) *Error Detection and Correction* - This stage determines whether or not the packet will be accepted by the receiver. The default C routine, *dpt_ecc*, uses the *"ecc"* attribute of the receiver as a threshold to determine the acceptable percentage of bits in error per packet. This routine has been replaced by the routine *cdl_pt_ecc*, which implements the forward error correction scheme.

The modified routines are discussed in detail in the following paragraphs.

a. cdl_pt_error

The *cdl_pt_error* was originally created as part of the thesis work of Karayakaylar [Ref. 19]. Unfortunately, the model of jamming presented in the original thesis is cumbersome and difficult to work with. As a result, the jamming model has been completely overhauled. The new model is presented in the state diagram of Figure 19. Essentially, the jamming model consists of two states, ON and OFF, representing the

status of the modeled jammer. Each state has a user-defined bit error rate, *"no_jam_ber"* and *"jam_ber"*, associated with it. This bit error rate is applied uniformly to all channels modeled within the CDL structure. Thus, the jamming can be seen as an over-all reduction in the total signal-to-noise ratio of the system which causes a corresponding increase in the bit error rate. The states are linked by the user-defined probability transitions, *"no_jam_trans"* and *"jam_trans"*.

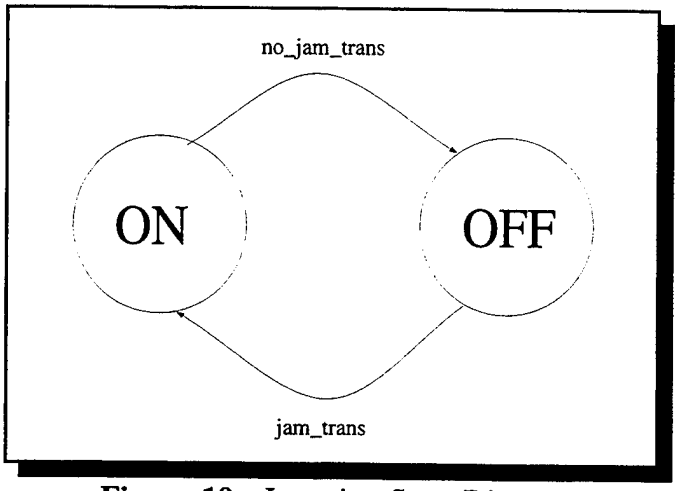


Figure 19. Jamming State Diagram

The *cdl_pt_error* routine begins by determining if a change in jammer status has occurred. This is accomplished by randomly generating a number based on a uniform distribution from zero to one, inclusive. If the number is less than or equal to the probability of a state transition, then the jammer is switched to the new state, otherwise the jammer remains in the current state. The global variable *"jamming"* is set to one to signify ON and zero to signify OFF. The appropriate bit error rate is used to compute the number of errors based on the size of the packet. The OPNET constant OPC_TDA_PT_NUM_ERRORS is set to this value and assigned to the packet. It is this value that is used in the next stage to determine whether or not the packet should be accepted.

b. cdl_pt_ecc

The final stage of the pipeline, the C routine *cdl_pt_ecc*, uses a threshold to determine if the ratio of bits in error to packet length exceeds an acceptable value. The default model has been modified to alter the manner in which the threshold is acquired. An incoming packet is checked to determine if the forward error correction field, *"FEC"*, has been set. If the field has been set, the value stored in it is used as the error ratio threshold. If the field has not been set, the *"ecc"* attribute of the receiver is used as the threshold. The ratio of bits in error to packet length is computed, and, if the applicable threshold has been exceeded, the packet is discarded. This is accomplished by setting the OPNET packet constant OPC_TDA_PT_PK_ACCEPT to OPC_TRUE if the packet is to be accepted, or OPC_FALSE if the packet is to be rejected. The point-to-point receiver will handle the packet accordingly. The value in the *"FEC"* field is set by the *CDL Manager* prior to transmission of the packet across the CDL. Its value comes from the *CDL Manager* attribute, *"FEC Protection"*.

C. MATLAB INTERFACE

This section describes in detail the MATLAB 4.1 routines used for the compression and decompression of the transmitted images. The compression/decompression algorithms are based on a five level hierarchical compression scheme utilizing a 15^{th} order biorthogonal filter. The original compression/decompression code was written by Carvahlo as part of his thesis [Ref. 11]. This section focuses on the code generated to implement the interface between the OPNET simulation and the MATLAB image processing code. It is divided into two parts, a broad overview of the interface between OPNET and MATLAB, followed by a detailed discussion of the code generated to implement this interface.

The network simulation takes place within the OPNET simulation environment, which is not optimally designed to send actual data through the resultant model. In OPNET, the packet information and appropriate packet parameters are maintained in a global database. Only the pointer to the database entry is passed around the simulation. This thesis takes that idea one step further, maintaining a separate database external to OPNET for the transmitted image data. Thus, the OPNET simulation needs only to transmit the pointers within this external database. Each compressed video packet contains a pointer to one hundred data values within the image database. Since each image value is representable by eight bits, this corresponds to a packet length of 800 bits plus the *"cmpr_vdo_pkt"* header, for a total of 840 bits. The pointers contained in each received packet are logged into an output file that is eventually be read by MATLAB. The output file contains the image number, the level number, and the sequence number within each level for the data contained within the packet. A MATLAB routine sorts through this file and breaks it into separate image files. These image files are further sorted and missing pointers cause their corresponding data values to be zeroed out. Thus, the end product is a database representing the data that was successfully transmitted and zeros for the data that was lost in the transmission process. This database serves as the input to the reconstruction routine.

The compression and decompression routines make use of the following MATLAB routines written by Carvahlo:

(1) *Compression*:

decomp.m - the main decomposition routine, calls the other routines

dwt2rDEC.m - the Quadrant Pyramidal 2D discrete wavelet transform used to compute the wavelet sequence for transmission

(2) *Decompression*:

recomp.m - the main recomposition routine, calls the other routines

res_scale_DEC.m - discards wavelet coefficients by comparing their energies to a relative threshold based on the resolution desired

idwt2DEC.m - 2D inverse discrete wavelet transform, returns the reconstructed image

Two MATLAB routines were written to interface these algorithms with the results generated by the OPNET simulation. The first, create_mask.m, is the routine used to create an image mask from the data supplied by OPNET. The second, which is embedded in the function idwt2DEC.m, is used to apply the generated mask to the image data.

The above decomposition routines generate sixteen blocks of data, each of which is 16384 values long. These blocks are the result of bandpass filtering, and, therefore, represent various frequency levels within the image. The number and size of the blocks are a byproduct of the number and size of the filters used in the decomposition. Each value is representable by eight bits. These sixteen blocks are grouped into five resolution levels, based on energy content. Level one always contains the lowest frequency block and serves as a foundation upon which the other levels can build. Each complete image represents (16 x 16384 =) 262144 bytes to be transmitted. These are queued up and transmitted, as explained earlier, and an output file of image, level, and sequence numbers is generated.

1. create_mask

The function create_mask.m reads this output file into MATLAB as a matrix called "mask_data". The rows of this matrix represent the pointers correctly received by the receiver and have three columns: image, level, and sequence number. This matrix is then broken into separate files, named "image#_mask.mat", based on the image number. The files are MATLAB workspaces that consist of a matrix, denoted "mask", that

represents the mask to be applied to generate that particular transmitted image. "Mask" is a 16x16384 matrix of ones and zeros. Each row corresponds to a particular frequency block. A one represents a data value that was successfully received, while a two represents a data value that was lost. Create_mask.m calls a routine, eval_mask.m, to evaluate each mask and return the number of values and packets lost in that particular image. Create_mask uses this information to generate plots of the number of values and packets lost per image for all the images transmitted in that session.

2. idwt2DEC.m modification

The resultant mask is applied to the image data during the recomposition algorithm. Specifically, it is applied just prior to the image reconstruction in the function idwt2DEC.m. The modified code requests the user to input the number of the particular image to be generated and uses this to load the appropriate mask workspace file, of the form "image#_mask.mat". This loads the "mask" applicable to this image into MATLAB. The embedded code performs an element by element multiplication between the image data and the received mask. As a result, data values corresponding to zeros in the mask are erased. This multiplication is done on a block by block basis, with each row of the mask matrix representing a block. A matrix containing the sorted energy levels of the original blocks is used to ensure that the frequency blocks are masked according to their order of transmission. In other words, it is crucial to know which blocks were transmitted with which levels. The sixteen blocks resulting from this element by element multiplication are subsequently used by the reconstruction algorithm to generate the transmitted image.

VI. RESULTS

This chapter presents the analysis of representative results that can be produced using the work presented in the previous chapters. The results provided permit both quantitative and subjective evaluation of the effectiveness of real-time transmission across the CDL using the proposed approach. Specifically, the simulation data shows the utility of the proposed solution by providing plots of the various critical indicators in a network environment. The MATLAB algorithms provide a visual display of the resultant images transmitted through the network model, allowing the user to subjectively evaluate the effectiveness of the solution.

The first section provides a broad overview of the simulations, including the justification for the choice of indicators. The second section illustrates the success of the transmission scheme without the presence of errors. The final section introduces jamming to the simulation and demonstrates the success of the mechanism under high bit error rates.

A. SIMULATION OVERVIEW

Two simulations are presented in this chapter. The first one demonstrates the operation of the mechanism in the presence of overloaded network conditions, but no imposed jamming. These results validate the ability of the mechanism to guarantee real-time constraints. The second simulation imposes jamming on the link. The results demonstrate the effectiveness of the mechanism to reproduce the minimum acceptable image quality and still maintain the real-time constraints.

Numerous performance metrics are evaluated to ensure the validity of the results. The instantaneous delay for the received compressed video packets is recorded to ensure that the real-time bounds on maximum and minimum end-to-end delay are not exceeded. The sizes of the queues along the nodes establishing the real-time flow are examined to

verify the realistic nature of the solution. In addition, the size of the general application queue at the *CDL Manager* is provided to verify that the network was subjected to an overload condition. For completeness, the throughput as well as a record of the arrival of the time of the various real-time packets are presented.

Three metrics are utilized to evaluate the received image. The first is a plot of the number of packets lost per image for the real-time flow. The second is a quantitative measure of the signal-to-noise ratio of the reconstructed image. Finally, the reconstructed image itself is presented to allow a subjective evaluation of the transmission scheme. Where applicable, a display of the missing components in the reconstructed image is also provided. Figure 20 displays the original image as it is presented to the compressed video application at the transmitting end of the connection.

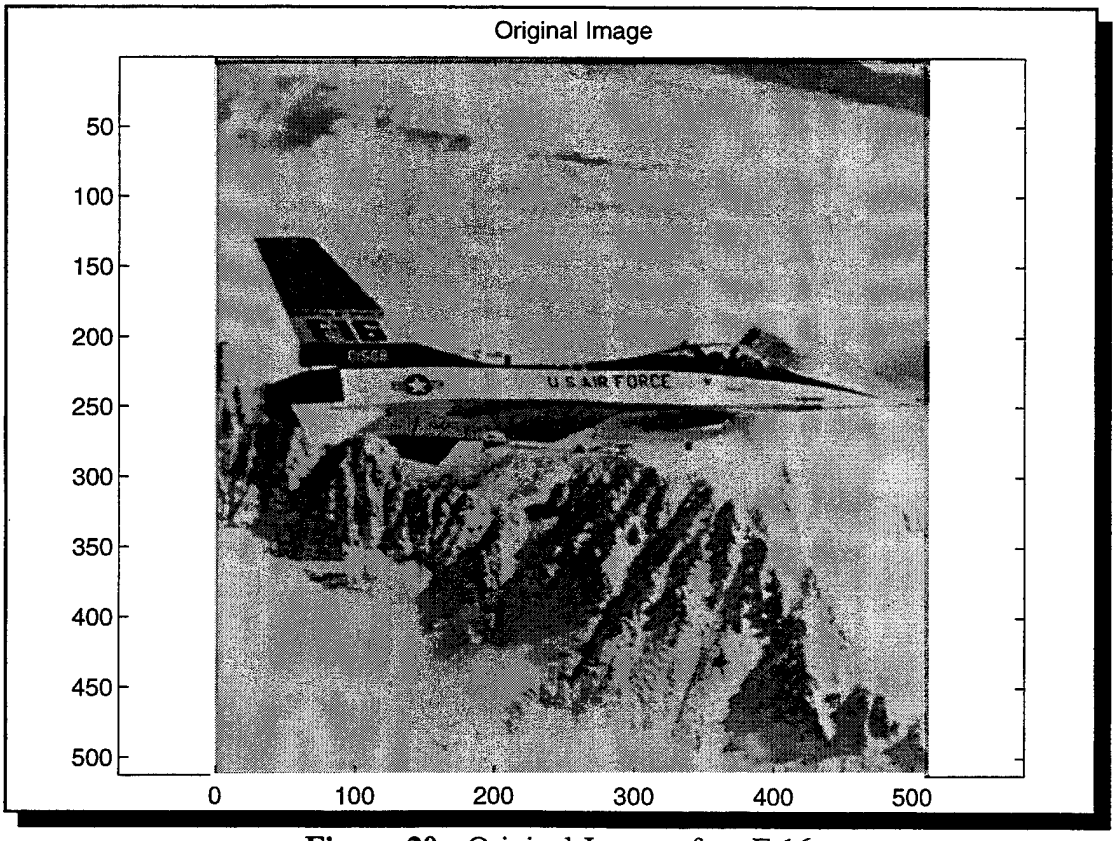


Figure 20. Original Image of an F-16

B. TRANSMISSION WITHOUT JAMMING

The first simulation demonstrates the operation of the real-time transmission mechanism in the absence of jamming. The objective is to demonstrate that the mechanism realistically maintains the real-time transmission constraints. The following is a listing of the vital parameters of this simulation:

simulation duration:	1 second
sending application:	station 0 on the airborne LAN (node 0)
receiving application:	station 0 on the surface LAN (node 10)
image interarrival time:	0.1 seconds
image size:	262,144 values
	2,097,152 bits
	26,215 packets (100 values/packet)
desired transmission rate:	22.021 Mbps
minimum acceptable rate:	1.3763 Mbps (represents only level one)

1. Queue Sizes

Figures 21 through 23 present the size of the queues located at the transmitting station. Figure 21 is a plot of the size of the queue in the token bucket process. The image arrival time of 0.1 seconds can clearly be seen in the plot. The initial massive transmission that occurs with the arrival of each new image is caused by the tokens accumulated during the idle time between the transmission of one image and the arrival of the next image. After this burst, the image data gets sent at the rate corresponding to the sloping portions of the curve in Figure 21. In Figure 22, the leaky bucket transmission rate is only limited by the maximum rate supported by the FDDI ring, 100 Mbps. The transmission rate of the mac process (Figure 23) is limited by the availability

of the FDDI ring. Figure 24 is a plot of the real-time application subqueue assigned to the flow at the CDL Manager. Together, these four queue plots demonstrate that the queue requirements for the reserved resources along the real-time flow never exceed the size of a single image.

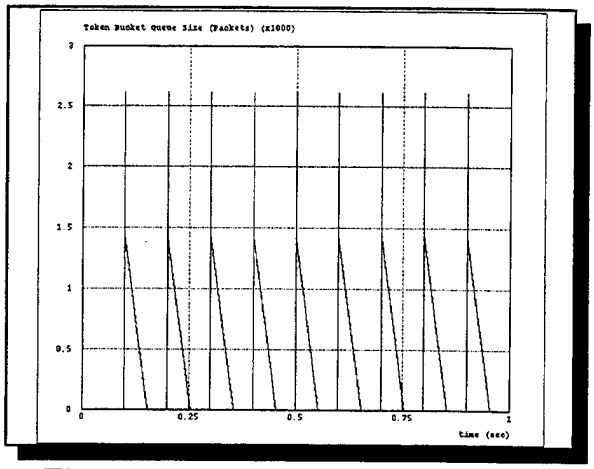


Figure 21. Token Bucket Queue Size

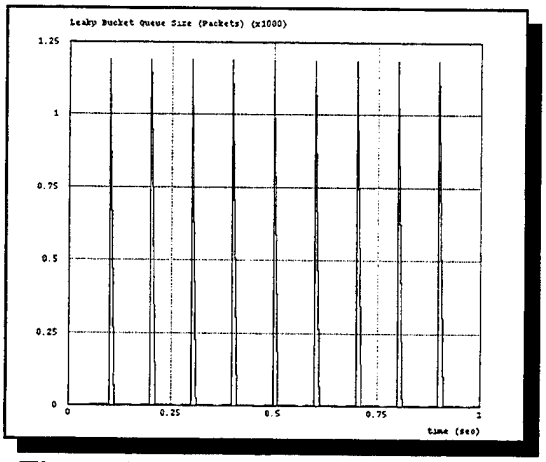


Figure 22. Leaky Bucket Queue Size

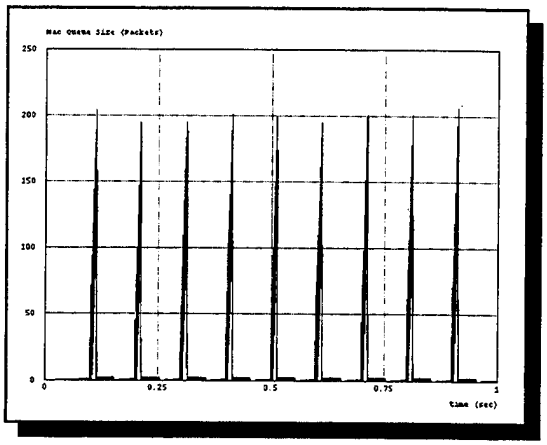


Figure 23. Mac Queue Size

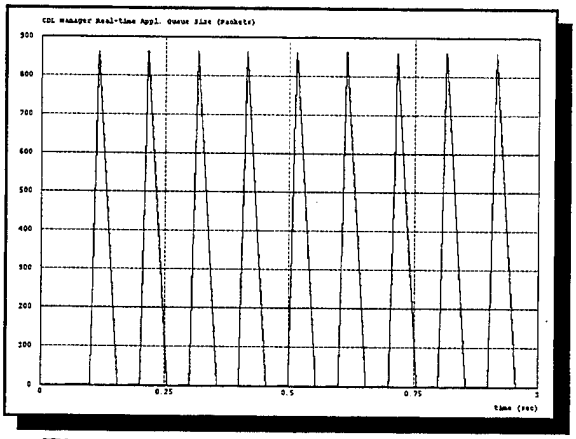


Figure 24. CDL Manager Real-time Application Subqueue Size

The next figure, Figure 25, is a plot of the size of the general application subqueue at the CDL Manager. The monotonic nature of this plot verifies the overloaded condition of the CDL. The subqueue is backing up linearly over time with the arrival of excess

packets. In an actual implementation, this queue size would be limited and excess packets would be discarded. It should be noted that this overloaded condition was artificially imposed on the network by forcing the non-real-time stations to generate a load that exceeded the CDL capacity used in the simulation.

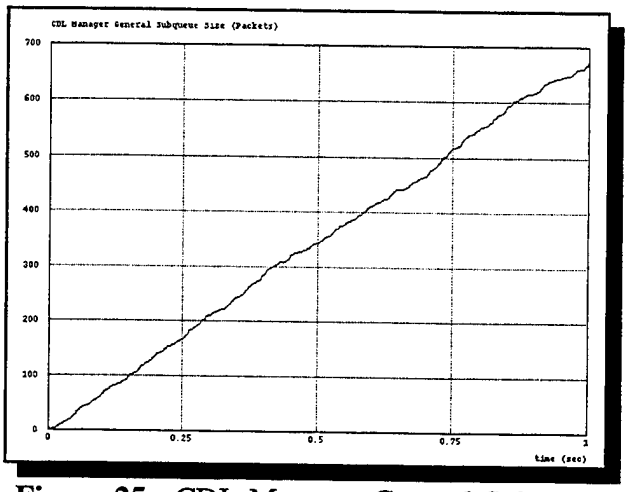


Figure 25. CDL Manager General Subqueue Size - Illustrating Overload Condition

2. Image Data Transmission

The next three figures relate to the statistics gathered from the compressed video receiver at the destination station. Figure 26 is a record of the image and sequence numbers of the packets received by the process. It shows that eight complete resolution (including all five levels) images were received. These are considered complete because the images were received at the destination prior to the expiration of the simulation time. The mean and instantaneous end-to-end delay values for the received packets are plotted in Figure 27. The plots clearly illustrate the maximum and minimum bound on this delay, validating the effectiveness of the proposed mechanism. The delay is at a minimum value for the first packet received and rises to a maximum value for the final packet received from an image. This is because the delay is measured from the time the

entire image arrives at the sender, which is the same for all packets within a given image. The throughput is shown in Figure 28 for completeness. As expected, this average throughput approaches the desired transmission rate of the sending application, 22.021 Mbps.

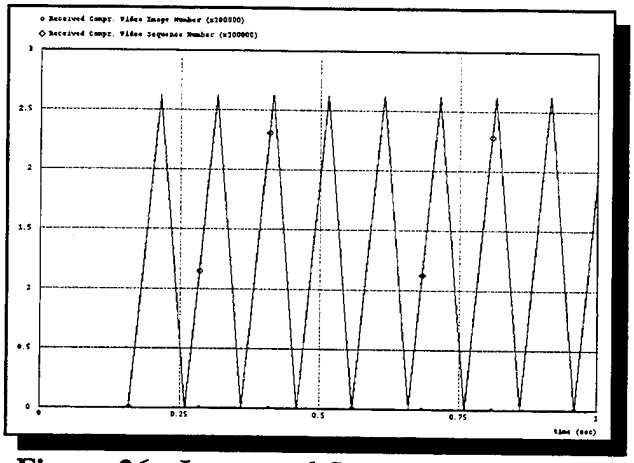


Figure 26. Image and Sequence Numbers of Received Compressed Video Packets

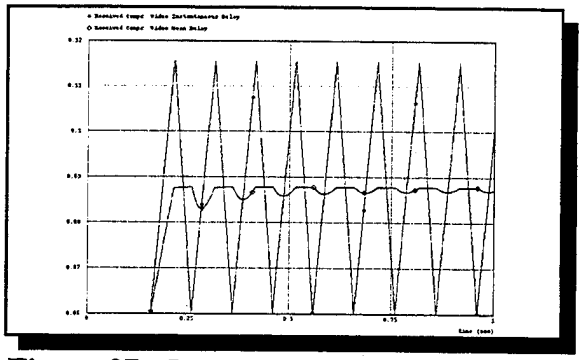


Figure 27. Instantaneous and Mean Delay Values for Received Compressed Video Packets

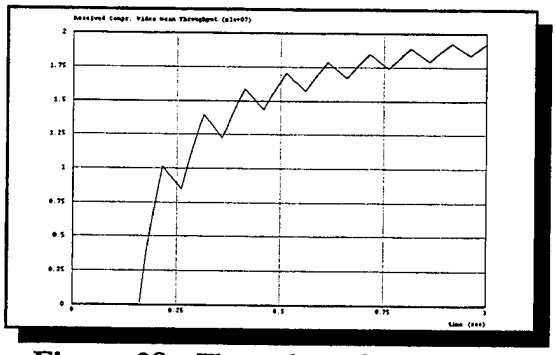


Figure 28. Throughput for Received Compressed Video Packets

3. Received Image Quality

The final two figures present the reconstructed image and its performance metrics. Figure 29 shows that the eight transmitted images (images 0 to 7) were received without packet loss. This is not surprising because of the large link margin of the CDL when jamming is not present. The ninth and final image (image 8) suffered packet loss due to the termination of the simulation. Finally, the reconstructed image is displayed in Figure 30. As expected, the reconstructed image is identical to the transmitted image. The signal-to-noise ratio of 48.83 dB is also included in the graphic.

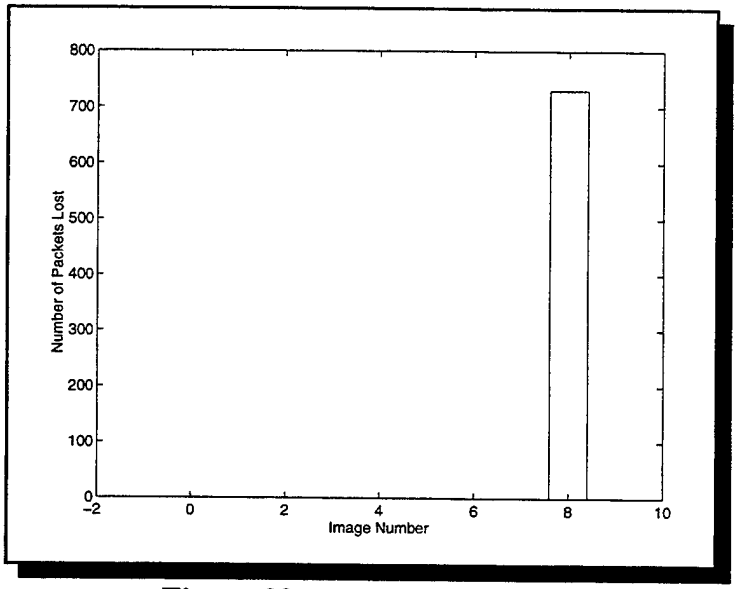


Figure 29. Number of Packets Not Received at Destination

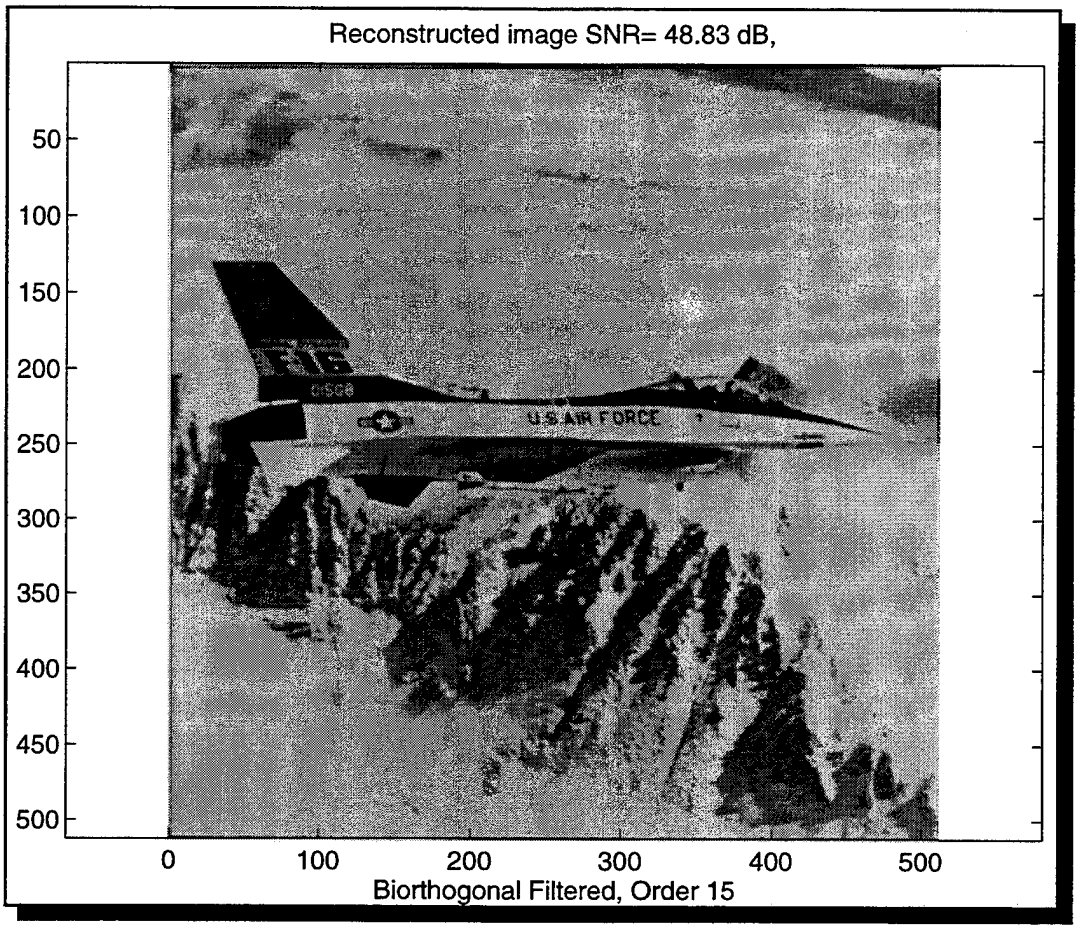


Figure 30. Received Image: Overloaded Network with No Jamming

C. TRANSMISSION WITH JAMMING

The second simulation run is designed to validate the effectiveness of the real-time mechanism in the presence of jamming. The parameters of concern for the second run are:

simulation duration:	1 second
sending application:	station 0 on the airborne LAN (node 0)
receiving application:	station 0 on the surface LAN (node 10)
image interarrival time:	0.1 second
image size:	262,144 values
	2,097,152 bits
	26,215 packets (100 values/packet)
desired transmission rate:	22.021 Mbps
minimum acceptable rate:	1.3763 Mbps (represents only level one)
probability of bit error	
in jamming state:	10^{-3}

1. Jamming

Figures 31 and 32 display the state of both the jamming and the CDL Manager's forward error correction mechanism. Jamming is experienced by the CDL at 0.25 seconds into the simulation and the CDL Manager responds by initiating the forward error correction mechanism just prior to 0.5 seconds. This lag is a result of the history and the hysteresis built into the link monitoring mechanism. By adjusting these, the CDL Manager's speed of response to the commencement of the jamming can be adjusted. The combined plots show that from simulation startup to 0.25 seconds, no jamming is experienced by the CDL. From 0.25 to 0.5 seconds jamming is present, but the CDL

Manager has not responded. After 0.5 seconds, the CDL Manager has initiated the forward error correction mechanism.

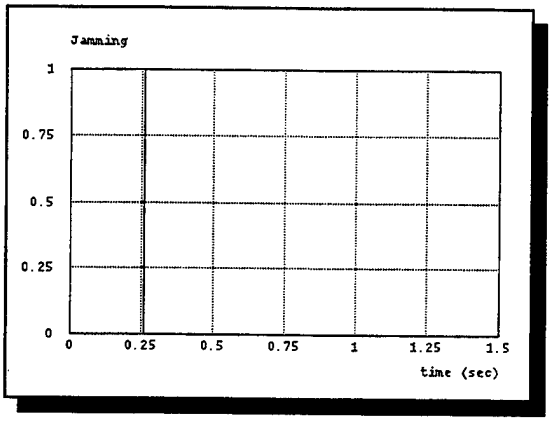


Figure 31. Plot of Jamming:
0 corresponds to OFF
1 corresponds to ON

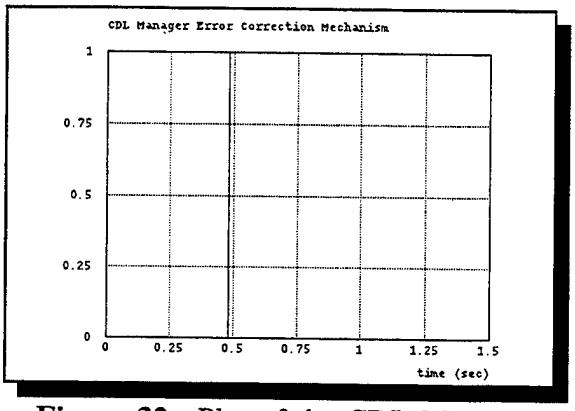


Figure 32. Plot of the CDL Manager Forward Error Correction Mechanism:
0 corresponds to OFF
1 corresponds to ON

2. Queue Sizes

The next three figures are an indication of the queue requirements along the path of the flow. Once again, the queue requirements at any resource never exceed the amount of data contained in a single image. At the CDL Manager, this is achieved by adding the overhead of the forward error correction just prior to transmission of the packet. The token bucket queue (Figure 33) is seen not to completely empty with each transmission once the forward error correction has been activated. This is because the bucket's average transmission rate has been reduced to the minimum acceptable rate by the CDL Manager corresponding to the transmission of resolution level one. In addition, the tokens no longer have a period where they can accumulate in the token bucket. Thus, the token bucket no longer experiences the bursts seen in the earlier case. This removes the backlog caused by this burst at the queues in the leaky bucket, mac, and CDL Manager processes. This can be clearly seen in Figures 34, 35, and 36, where the queues are

reduced to the size of a single packet because the rate out of the queue is greater than or equal to the rate into the queue.

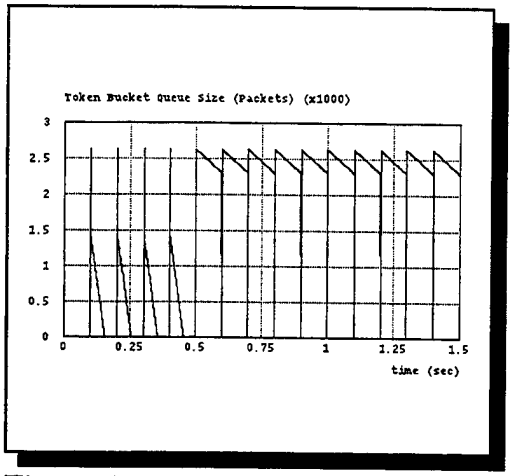


Figure 33. Token Bucket Queue Size (Packets)

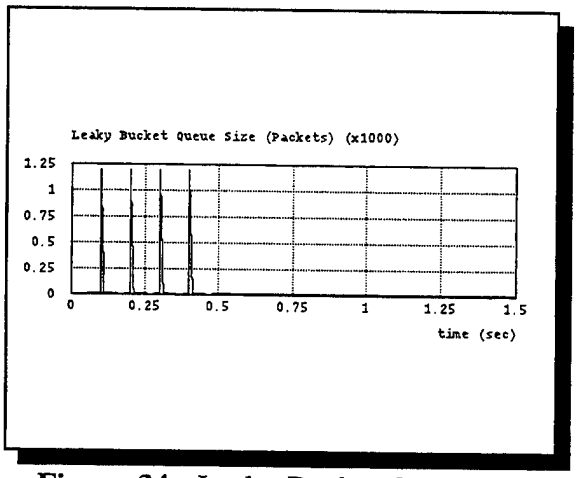


Figure 34. Leaky Bucket Queue Size (Packets)

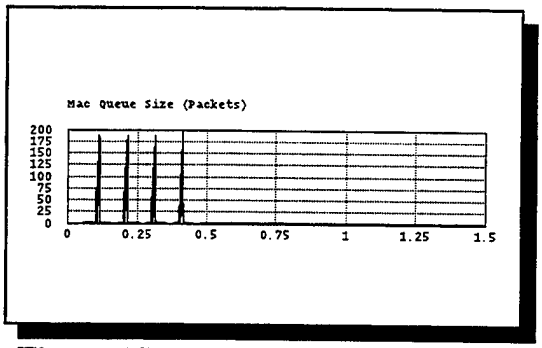


Figure 35. Mac Queue Size (Packets)

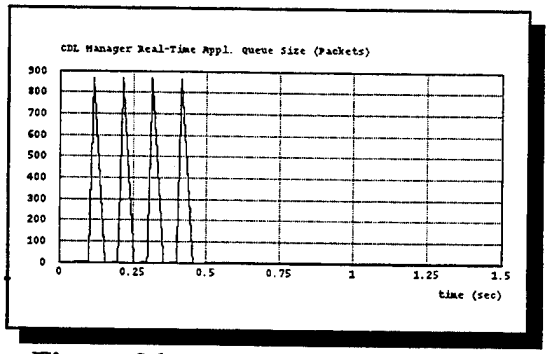


Figure 36. CDL Manager Real-time Subqueue Size (Packets)

3. Image Data Transmission

The next set of three figures presents the statistics gathered from the received compressed video packets. The first figure, Figure 37, plots the image and sequence numbers of the received packets. Once the forward error correction mechanism is

activated and the sender's transmission rate is reduced, only level one packets are received. The mean and instantaneous end-to-end delay values are plotted in Figure 38. The instantaneous delay is bounded and it is this bound that is used by the receiver for its playback point. It should be noted that the results from the previous section conform to these maximum and minimum delay bounds. Thus, the real-time constraints on both the end-to-end delay and jitter are satisfied. Finally, the average throughput is plotted in Figure 39. As expected, this value approaches the minimum acceptable rate of 1.3763 Mbps corresponding to the transmission of only level one data.

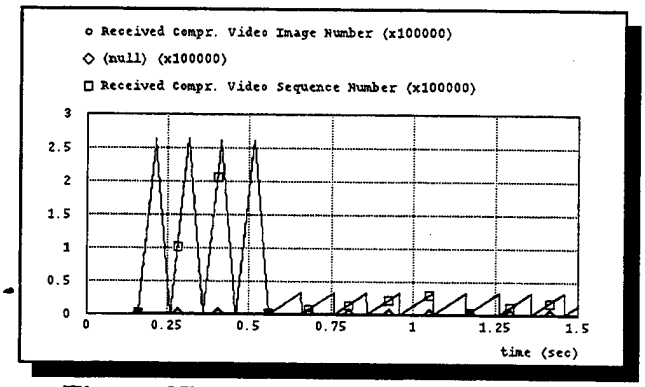


Figure 37. Plot of Image and Sequence Number of Received Compressed Video Packets ($x10^4$)

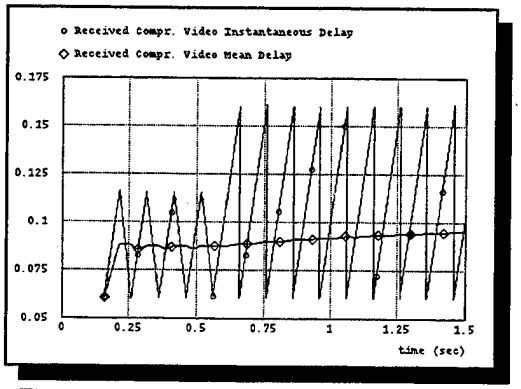


Figure 38. Mean and Instantaneous Delay of Compressed Video Packets (secs)

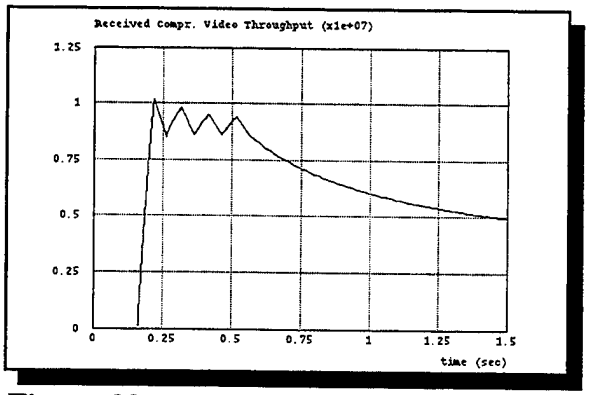


Figure 39. Compressed Video Throughput ($x\ 10^7$)

4. Received Image Quality

The final five figures present the reconstructed images with and without the forward error correction mechanism in operation. Figure 40 displays the number of packets lost per image during the transmission session. The first image (image 0), prior to the jamming, suffers no packet losses and corresponds to the example of the previous section (Figure 29). The next three images (images 1, 2, 3) are exposed to jamming prior to the activation of the error correction mechanism. From the fifth image onwards, the mechanism is activated and the remaining images represent the correct transmission of the complete set of level one packets for each of the images. It should be noted that more packets of the original image are lost once the FEC mechanism is activated; however, all level one packets are received correctly.

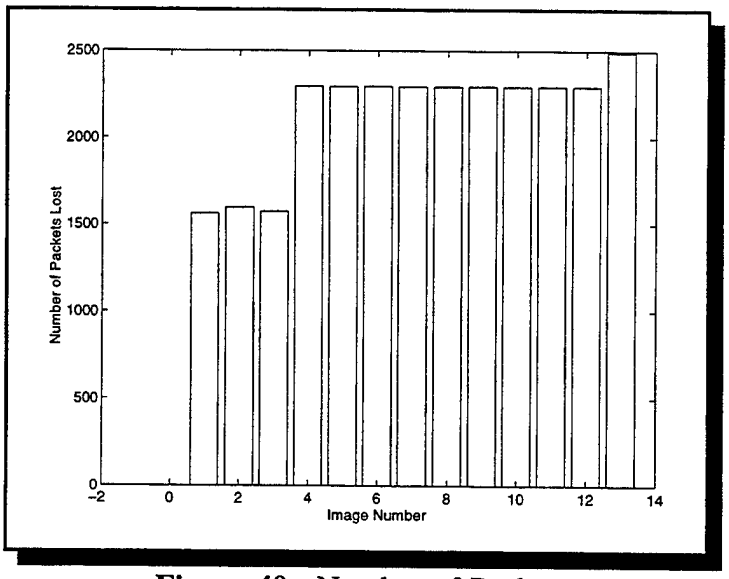


Figure 40. Number of Packets Not Received at Destination

Figures 41 and 42 display the reconstructed images received by the destination application for images 2 and 6. Image 2 was subject to jamming, but did not make use of the error correction mechanism. All five resolution levels continue to be sent with packet losses occurring randomly throughout the transmission. The result is that packets are lost from all frequency components within the spectrum. The dark black areas correspond to lost packets that contained the low frequency components of the image. These omissions make it difficult to analyze or identify the target. The next figure, Figure 42, demonstrates the effectiveness of the communication mechanism. The transmission rate of the sender is reduced and the CDL bandwidth is reallocated to allow the addition of forward error correction to the transmission across the CDL. Thus, while the transmission consists only of the low frequency components in the image, they all are received correctly. This results in a complete and identifiable, although fuzzy, image at the receiver. This is further confirmed by the last two figures, which display the information that was contained in the packets that failed to arrive at the receiver. The former case contains information throughout the entire spectrum, while the latter case contains only the high frequency components. By trading volume for correctness, the mechanism has sacrificed the "sharpness" of the high frequency components for the content of the complete image. The signal-to-noise of the second image is much higher than that of the first, thus quantitatively verifying the qualitative analysis. It is interesting to note that image 2 suffers less packet loss than image 6, but produces an "inferior" result. This underscores the difficulty in relying exclusively on the traditional performance metrics.

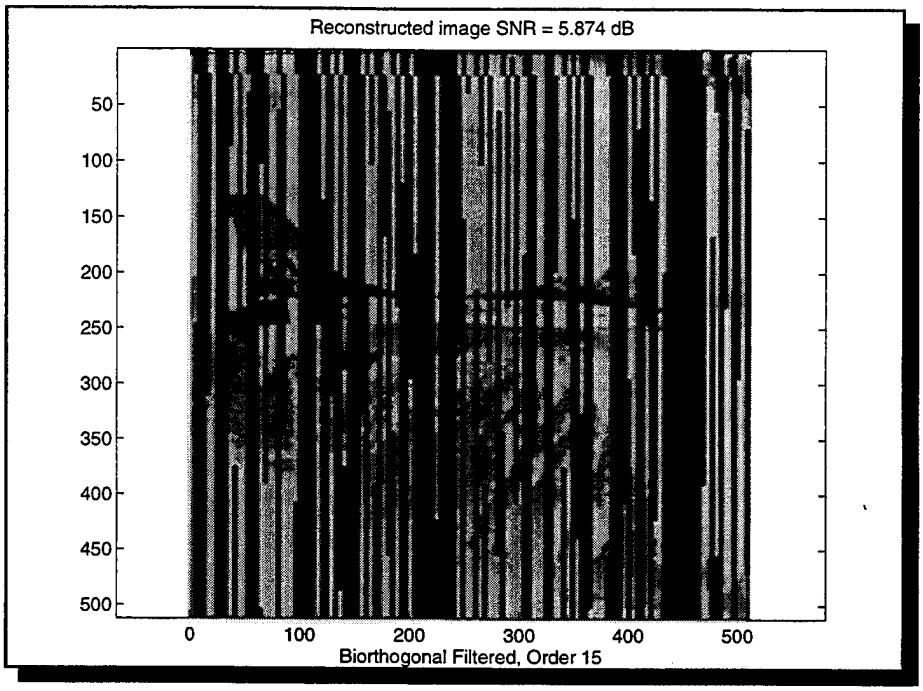


Figure 41. Received Image: Jamming without Proposed Mechanism

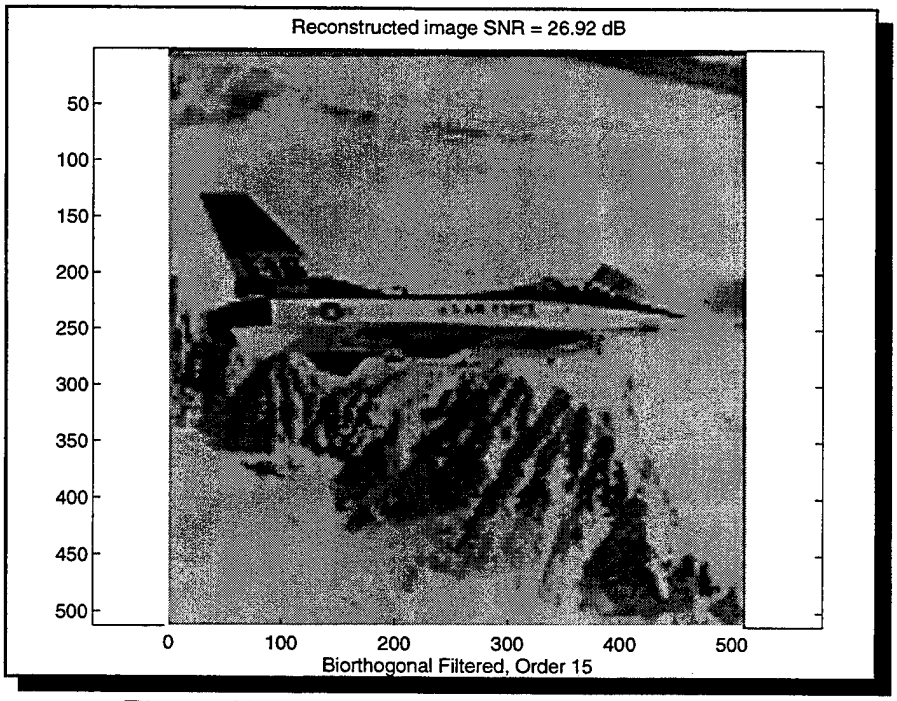


Figure 42. Received Image Received: Jamming with Proposed Mechanism

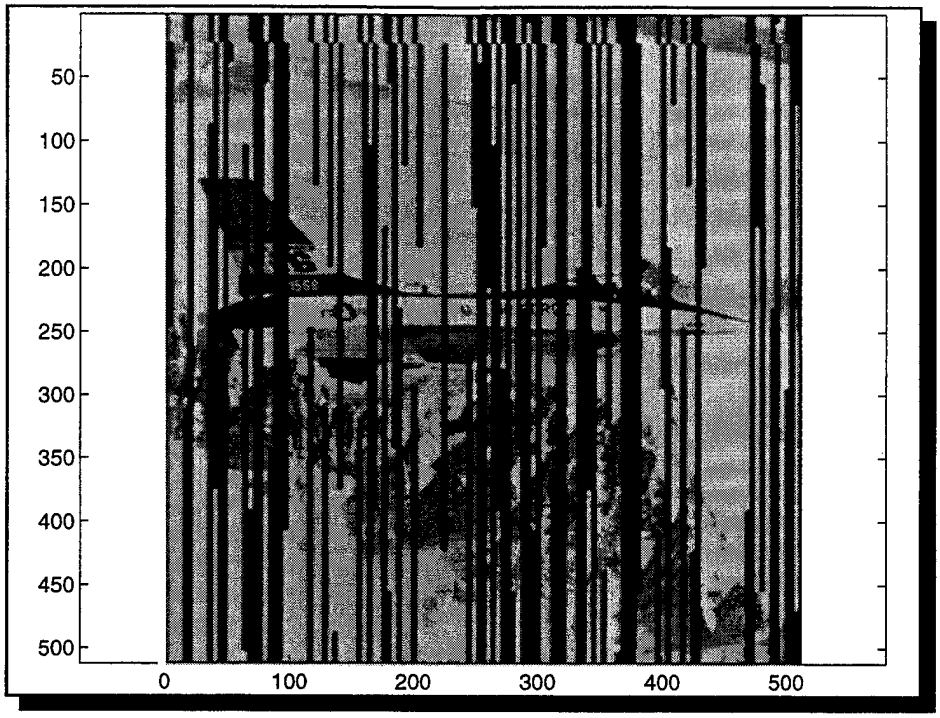


Figure 43. Errors in Received Image without Proposed Mechanism

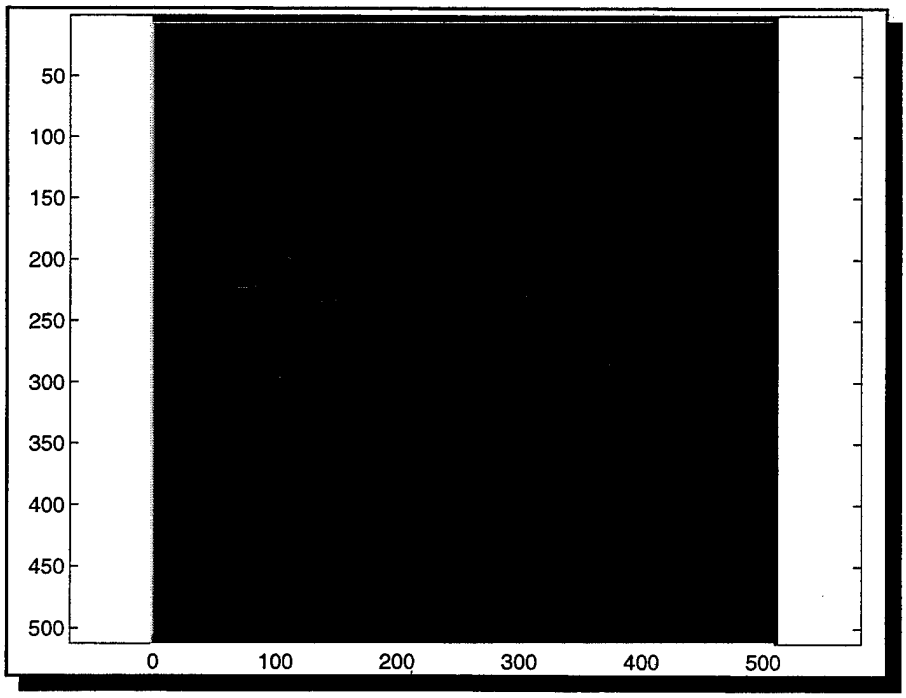


Figure 44. Errors in Received Image with Proposed Mechanism

VII. CONCLUDING REMARKS

A. CONCLUSIONS

In this thesis, we have presented and analyzed a mechanism to transmit real-time compressed video data across a packet-switched network in the presence of highly correlated errors. A model of the Common Data Link was completed in OPNET and served as the simulation testbed for the proposed mechanism. Results were examined quantitatively and qualitatively to verify the effectiveness of the mechanism.

The specific contributions of this thesis are as follows:

(1) A successful real-time transmission scheme is presented and analyzed to allow the Common Data Link to operate effectively and reliably in the presence of jamming. The mechanism to enforce this transmission scheme guarantees that (1) delay bounds are met for real-time flows despite network overload and (2) a minimum acceptable image quality is maintained despite the presence of highly correlated errors. This mechanism is composed of the following elements: (1) a hierarchical image compression scheme, (2) rate control at the source, (3) bandwidth allocation within all encountered network nodes, and (4) dynamic forward error correction.

(2) A complete and operational model of two FDDI LANs interconnected by the CDL is produced in OPNET. This model serves as an expandable test-bed for further CDL development.

(3) An interface between OPNET and MATLAB is designed and implemented to allow the reconstruction and subjective evaluation of transmitted images simulated in OPNET. Unique in the study of real-time transmission schemes, this ability to view the images received during the simulation is essential to the effective evaluation of the proposed mechanism.

(5) A user's guide for the CDL network model in OPNET is produced. This user's guide facilitates the effective and efficient use of the model for follow-on research and development.

B. FOLLOW-ON WORK

The breadth and diversity of the material covered by this thesis leads to numerous possibilities for follow-on research. Among these are:

(1) A comprehensive review of the current link monitoring mechanism. This review would seek to improve the measurement of the bit error rate and eliminate undesirable changes in the link status.

(2) A close examination of the required error correction scheme leading to a more realistic model of both the overhead and the delay associated with the chosen scheme.

(3) An implementation of the developed real-time transmission scheme over a multi-hop CDL network.

(4) An implementation of multiple LAN connections to a single CDL network interface.

(5) An implementation of alternate higher level protocols, specifically TCP/IP, used to encapsulate real-time data over the CDL.

APPENDIX A. CDL MODEL USER'S GUIDE

This appendix is designed to serve as a guide to the effective and efficient use of the Common Data Link simulation model in OPNET. It is broken into four sections: standard model operation, the model interfaces, the model output options, and model modifications. A thorough understanding of this appendix is essential to the productive utilization of the CDL simulation.

The best way to approach this appendix is to use Section B as a guidebook in setting up the simulation to generate the specific environment to be analyzed. Section C is used in conjunction with Section B to specifically recognize the attributes that are of importance in a particular scenario and to provide guidelines in choosing the appropriate values. The desired outputs can be generated as given in Section D. Section E is applicable when changes to the basic architecture of the existing model are attempted.

This appendix assumes a working knowledge base for both OPNET and MATLAB. In addition to the OPNET Manuals, Nix's thesis [Ref. 20] provides an excellent tutorial for those unfamiliar with OPNET. A similar tutorial section is available in the basic MATLAB manual.

Upon completion of this thesis, the working CDL model will be compressed and stored on the Naval Postgraduate School's Electrical and Computer Engineering network and will be made available for follow-on development upon request. Inquiries should be directed to the author (walker@ece.nps.navy.mil) or Shridhar Shukla (shukla@ece.nps.navy.mil).

A. SYSTEM REQUIREMENTS

Two licensed simulation operating environments are required to fully utilize the CDL model. The first is MIL 3, Inc.'s Optimized Network Engineering Tool (version 2.4 or later) and the second is the MathWork's MATLAB (version 4.0 or later). Together,

these software packages determine the system requirements for running the CDL model. In addition, it is recommended that, at a minimum, 50 Mbytes be reserved on the hard drive to facilitate the storage of the large temporary files generated by OPNET during simulation. The simulations reported here were run on a Sun Sparcstation 5 running SunOS4.1.3.

B. CDL MODEL OPERATION

To configure the CDL model simulation, there are four areas that need to be addressed: (1) application parameters, (2) network parameters, (3) CDL Manager parameters, and (4) link parameters. This section discusses these areas and provides guidance in setting the appropriate parameters. In addition, the simulation execution is presented in detail. Parameters that need not be altered are not discussed. These include the network and link attributes which need not be changed unless the user desires to change the underlying model.

1. Applications

The application and its token-leaky bucket interface to the FDDI ring must be set up to model the desired application performance. The major factors are the frequency of image generation and the average image size in bits. In addition, the minimum and desired level of image resolution should be implemented. The frequency of image generation is determined by the *"src.interarrival args"* attribute. This should be set to the interval between image arrivals in seconds. The average image size is used to set the appropriate data rates and data packet and token sizes. The data packet size reflects the size of the information field and is most efficient if it is an integral number of bytes. Each byte is used to represent a single value of the compression output data. The *"tkn_bkt.data segment size"* should be set to the desired number of bits in the information

field. The *"tkn_bkt.desired rate"* and *"tkn_bkt.min_acceptable_rate"* should be calculated based on the amount of data required to produce the desired image resolution and the minimum acceptable image resolution, respectively. These are set according to the following equation:

$$rate = \frac{\left(\frac{number\ of\ values\ for\ resolution}{number\ of\ values\ per\ packet}\right) * (segment\ size + header\ size)}{image\ update\ interval} \qquad (3)$$

The segment size and header are both in bits and the update interval is in seconds. The header size is 40 bits, while the segment size is the value chosen for *"tkn_bkt.data segment size"*. The number of values for the applicable resolution can be determined by multiplying the number of values per level by the number of resolutions levels desired. The *"token_size"* for both the token and leaky buckets should be set to a value greater than or equal to the segment size plus the header size. To guarantee that the FDDI ring will support the application, the *"mac.sync bandwidth"* should be set to a value greater than or equal to the bandwidth determined by the following equation:

$$bandwidth = \frac{\frac{"desired\ rate"}{(segment\ size + 40)} * (segment\ size + 80)}{10^8} \qquad (4)$$

This equation sets the application's synchronous allotment based on the desired data rate, taking into account the overhead of both the compressed video packet and the FDDI packet. The destination address needs to be set and must be another real-time FDDI station. Finally, the *"cmpr_vdo_rcvr.Data_file"* attribute should be set to the name of the output file to record the arrival of compressed video packets at this station.

To simulate the additional load of general traffic on the Common Data Link, multiple non-real-time FDDI stations can be setup on the attached FDDI rings. The stations generate traffic based on a random message generation routine. A station is set

up by specifying the parameters for this message generation routine. The packet generation is based on an exponential pdf with a mean data rate set by the attribute *"llc_src.arrival rate"*. The packet lengths are also set by an exponential pdf with a mean packet length determined by *"llc_src.mean pk length"*. The range of valid destination addresses is specified by setting the attributes *"llc_src.low dest address"* and *"llc_src.high dest address"*. The percentage of synchronous and asynchronous packet generation is determined by the *"llc_src.async_mix"*. Finally, the range of the FDDI packet priorities is determined by the attributes *"llc_src.low pkt priority"* and *"llc_src.high pkt priority"*.

2. CDL Manager

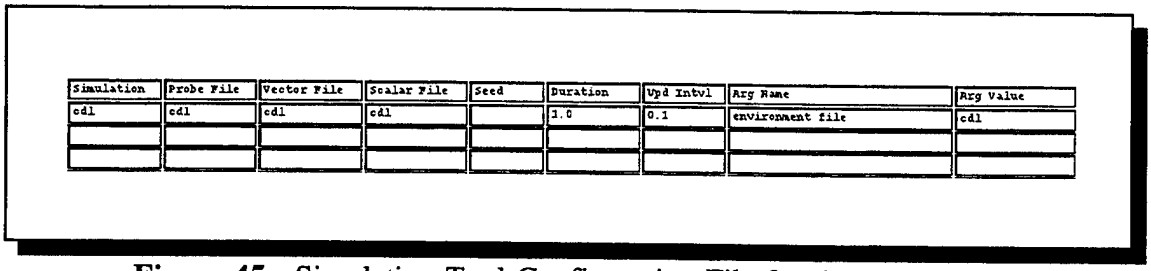
The CDL Manager must be tailored to the individual simulation to achieve the desired operation. The major components are the link monitoring attributes and the forward error correction attributes. The link monitoring must be set up for both ends of the data link. The monitoring packets are generated based on the attributes *"CDL Manager.link_monitoring_trans_rate"* and *"CDL Manager. monitoring pkt size"*. In addition, the transmission criteria and hysteresis plot must be defined for the link monitoring reports. The *"CDL Manager.LQR transmission delta"* is used to set the change in monitoring ratio (bad packets/total packets) that will trigger the transmission of a link quality report. The *"CDL Manager.history length"* sets the length of the link monitoring history to be maintained by the CDL Manager. This history is the record of good and bad link monitoring packets received by this CDL Manager. The link monitoring hysteresis plot is defined by the attributes *"CDL Manager.lower hysteresis threshold"* and *"CDL Manager.upper hysteresis threshold"*, which are the thresholds for the GOOD and BAD link status determined as ratios of the number of bad packets in the history divided by the number of packets in the history length, respectively.

3. Jamming

The jamming must be set up as is appropriate for the desired simulation. The bit error rates for the states of jamming and no jamming are set by the attributes *"ls_#.jam_ber"* and *"ls_#.no_jam_ber"*, respectively. The probability that jamming is initiated and suspended is determined by the *"ls_#.jam_trans"* and *"ls_#.no_jam_trans"* attributes. It is important to note that these attributes must be set for each channel modeled in the CDL link.

4. Simulation Execution

The CDL model simulation is executed through the use of the OPNET simulation tool. Figure 45 presents the simulation tool configuration file, cdl, used in conjunction with the CDL model.



Simulation	Probe File	Vector File	Scalar File	Seed	Duration	Upd Intvl	Arg Name	Arg Value
cdl	cdl	cdl	cdl		1.0	0.1	environment file	cdl

Figure 45. Simulation Tool Configuration File for the CDL Model

The simulation file created by the CDL model is cdl.sim and is entered in the first block. There are two probe files available with the CDL model. The first, cdl_basic.pb, probes almost all of the output statistics produced by the CDL model. This file is used as a template to create the actual probe files used in the simulations. The probe file referred to in Figure 45, cdl.pb, is a subset of cdl_basic.pb. The vector and scalar files are used to gather the output statistics generated by the simulation. The environmental

file used in the simulations is cdl.ef, as presented in Section C. It should be noted that the debugging feature is turned on within cdl.ef and, thus, the command *cont* must be entered on the command line to run the simulation. For more details on the debugging facility, refer to the OPNET Manual.

As shown in Figure 45, the CDL simulation provides updates every 0.1 seconds. (This is adjustable.) Upon simulation termination, the results can be observed through use of the OPNET Analysis Tool on the output file cdl.ov.

C. CDL MODEL INTERFACES

This section provides a detailed list outlining the parameters that are used as inputs to the CDL model. These parameters are normally recorded in the environment file, of the format "*.ef". A sample of the environment file used in these simulations can be found at the end of this section. The best approach to setting up a specific simulation is to use this list as a reference, in conjunction with the steps outlined in the previous section, to determine what parameters must be altered. The user is encouraged to make use of the sample environmental file as a template, making only the changes necessary for the specific simulation.

1. Listing of Simulation Attributes

The following list furnishes the attribute along with a brief description of its meaning and its effect on the model. The standard form for these attributes is: [net].[subnet].[node].[module].[process].[attribute]. For our simulation the net is "top" and the subnet is either "ring0" (the collection platform) or "ring1" (the surface platform). The node is f[node #], ie. "f0" for node 0. Thus, "top.ring0.f5.mac.station_address" would correspond to the station_address, an attribute of the mac process, for the 5th station on ring0. In the environment file, an asterisk is placed at a position that applies to all

particular values for that position in the model. For example, an "*" in the second position would force this parameter value to be applied to both ring0 and ring1. The list contains only the fields in the attribute name that are specific to that particular attribute. In the environmental file, quotes are used to enclose attributes whose names contain spaces. These quotes are dropped for convenience in the following list. Finally, the data type for the attribute is included in parenthesis following the attribute name. The following abbreviations are used: floating point (fp) and integer (int).

General Model Attributes:

(1) *station address* (int) - address of the particular station

(2) *mac.ring_id* (int) - This identifies the subnet this station belongs to

(3) *llc_src.CDL_NI_addr* (int) - This is set to the station address of the local CDL_NI

FDDI Ring Constants:

(1) *spawn station* (int) - address of station designated to spawn the FDDI token

(2) *station_latency* (fp) - delay encountered at each station along the ring (seconds)

(3) *prop_delay* (fp) - delay encountered between stations (seconds)

General Station Attributes (Both real-time and non-real-time):

(1) *mac.sync bandwidth* (fp) - fraction of total synchronous bandwidth that is guaranteed to this station

(2) *mac.T_Req* (fp) - token rotation time requested by this station (seconds)

Real-time Compressed Video Attributes:

(1) *src.interarrival pdf* (type) - type of probability distribution function for image generation

(2) *src.interarrival args* (type) - average interval between image generations (secs)

(3) *tkn_bkt. data segment size* (int) - size of information field in each cmpr_vdo_pkt (bits) This is used to calculate number of image decomposition values per packet = (segment size)/8

(4) *tkn_bkt. desired rate* (int) - desired data transmission rate for compressed video application (bps). This is set to permit the transmission of the entire image between image arrivals:

rate = ((image size)/(values per pkt)) * (sgmnt size + header size)/(image intvl)

The header size used in this equation is 40.

(5) *tkn_bkt.min_acceptable_rate* (int) - minimum acceptable data transmission rate (bps). This is set for transmission of level one data only (desired rate/16)

(6) *cmpr_vdo_rcvr.Data_file* - name of file to store list of received compressed video packets. This file provides the data to MATLAB. Its name must be of the form *.data.

Real-time Station Attributes:

(1) *tkn_bkt. token_size* (int) - size of token bucket tokens (bits). This should be set to maximum packet size = data_sgmt_size + header (The size of the header is 40 bits)

(2) *tkn_bkt. bucket_size* (int) - maximum size of token bucket (bits)

(3) *tkn_bkt. send_rate* (int) - transmission time for packets departing the token bucket (bps)

(4) *tkn_bkt. max_queue_size* (int) - maximum size of token bucket transmission queue (bits). This needs to be bigger than maximum size of image data.

(5) *lky_bkt. token_size* (int) - size of leaky bucket tokens (bits)

should be set to maximum packet size = data_sgmt_size + header (40 bits)

(6) *lky_bkt. bucket_size* (int) - maximum size of token bucket (bits)

(7) *lky_bkt. send_rate* (int) - transmission time for packets departing the leaky bucket (bps)

(8) *lky_bkt. max_queue_size* (int) - maximum size of queue (bits). This needs to be bigger than maximum size of image data

Non-Real-time Station Parameters:

(1) *llc_src.low dest address* (int) - smallest address for this station to send to

(2) *llc_src.high dest address* (int) - largest address for this station to send to

(3) *llc_src.high pkt priority* (int) - maximum FDDI packet priority for this station

(4) *llc_src.low pkt priority* (int) - minimum FDDI packet priority for this station

(5) *llc_src.arrival rate* (int) - average FDDI packet generation rate for this station

(6) *llc_src.mean pk length* (int) - average FDDI packet length for this station

(7) *llc_src.async_mix* (fp) - asynchronous/synchronous mix for FDDI packet generation at this station (1.0 = all asynchronous traffic)

CDL Manager Attributes:

(1) *CDL Manager.load balancing algorithm* (int) - load balancing algorithm (0 = circular, 1 = empty allocation)

(2) *CDL Manager.FEC_level* (fp) - Bit Error Rate threshold (errors/bit) the imposed FEC will protect under (ie. the packet in question will be accepted if it has less than this ratio of errors per bit)

(3) *CDL Manager.FEC_overhead* (fp) - Overhead to be applied to packets when FEC is applied to them (expressed in terms of a ratio to multiply packet size by)

Link Monitoring Attributes:

(1) *CDL Manager.link_monitor_trans_rate* (fp) - rate to transmit link monitoring packets (seconds/packet) from this CDL Manager

(2) *CDL Manager.monitoring pkt size* (int) - size of link monitoring packets transmitted from this CDL Manager

(3) *CDL Manager.LQR transmission delta* (fp) - change in ratio (bad packets/total packets) to trigger transmission of LQR

(4) *CDL Manager.upper hysteresis threshold* (fp) - ratio to trigger link status BAD

(5) *CDL Manager.lower hysteresis threshold* (fp) - ratio to trigger link status GOOD

(6) *CDL Manager.history length* (int) - size of link monitoring history (number of packets)

Jamming Attributes:

(1) *ls_#.jam_ber* (fp) - Bit Error Rate when jamming is present on channel #

(2) *ls_#.no_jam_ber* (fp) - Bit Error Rate when jamming is not present on channel #

(3) *ls_#.jam_trans* (fp) - probability that jamming will commence for channel #

(4) *ls_#.no_jam_trans* (fp) - probability that jamming will stop for channel #

Note: Despite the separate channel designations, once commenced, jamming is applied to all channels simultaneously. Upon suspension, jamming is stopped in all channels simultaneously.

CDL Attributes:

(1) *ls_#.delay* (fp) - delay encountered in channel #

(2) *pt_#[0].data rate* (int) - data rate for channel #

(3) *ecc threshold* (fp) - threshold to be used for packet acceptance (errors/bit)

2. Environmental file: cdl.ef

The following is the version of the environmental file utilized to generate the results of the second simulation in the body of this thesis. It is provided as a template for the model user.

```
#####################################################
#                                                   #
#    cdl.ef - Environmental File for CDL Model      #
#                                                   #
#####################################################


#
#
#      Last Modified:    8 June 1995
#      By:               T. Owens Walker III
#
#      Simulation Set-up:
#
#      Set-up to run a real-time flow between f0 on ring0
#      and f0 on fing1 in the presence of jamming
#
#


# sample simulation configuration file for
# two interconnected 10 station network in the
# existence of pulsed jammer interference (137.088 Mbps channel
hierarchy )
# with circular allocation load balancing algorithm

# Includes real-time flow establishment between
# f0 on ring0 and f0 on ring1


###############################
#                             #
#  General Model Attributes   #
#                             #
###############################


# station addresses

*.ring0.f0.mac.station_address:      0
*.ring0.f1.mac.station_address:      1
*.ring0.f2.mac.station_address:      2
*.ring0.f3.mac.station_address:      3
*.ring0.f4.mac.station_address:      4
*.ring0.f5.mac.station_address:      5
*.ring0.f6.mac.station_address:      6
*.ring0.f7.mac.station_address:      7
*.ring0.f8.mac.station_address:      8
*.ring0.f9.mac.station_address:      9

*.ring1.f0.mac.station_address:      10
*.ring1.f1.mac.station_address:      11
*.ring1.f2.mac.station_address:      12
*.ring1.f3.mac.station_address:      13
*.ring1.f4.mac.station_address:      14
*.ring1.f5.mac.station_address:      15
```

```
*.ring1.f6.mac.station_address:      16
*.ring1.f7.mac.station_address:      17
*.ring1.f8.mac.station_address:      18
*.ring1.f9.mac.station_address:      19


*.ring0.*.mac.ring_id :              0
*.ring1.*.mac.ring_id :              1

*.ring0.*.llc_src.CDL_NI_addr :      9
*.ring1.*.llc_src.CDL_NI_addr :      19


#####################################
#                                   #
#  Non-Real-Time Station Attributes  #
#                                   #
#####################################


# Specific stations may be tailored by specifying the full name:
# for example, top.ring0.f19.llc_src.async_mix : .5
# This means all stations must be specified, or individuals
# may be named after the generic is specified.
# destination addresses for random message generation

#"*.*.llc_src.low dest address" :   9
#"*.*.llc_src.high dest address" :  9
#"top.ring0.f0.llc_src.low dest address"  :
#"top.ring0.f0.llc_src.high dest address" :

"*.ring0.*.llc_src.low dest address":    10
"*.ring0.*.llc_src.high dest address":   19
"*.ring1.*.llc_src.low dest address":    0
"*.ring1.*.llc_src.high dest address":   19


# range of priority values that can be assigned to packets; FDDI
# standards allow for 8 priorities of asynchronous traffic.  MIL3's
# original model is modified to allow each station to generate multiple
# priorities, within a specified range.

 "*.*.llc_src.high pkt priority" :  7
 "*.*.llc_src.low pkt priority" :   0


# arrival rate(frames/sec), and message size (bits) for random message
# generation at each station on the ring.

#"*.*.*.arrival rate" :                  0
#"*.*.*.mean pk length" :                20000
"top.ring0.*.*.arrival rate":            0
"top.ring0.*.*.mean pk length":          1000
"top.ring1.*.*.arrival rate":            0
"top.ring1.*.*.mean pk length":          10
#"ring1.f9.*.arrival rate":              0
#"ring1.f9.*.mean pk length":            0
```

```
################################
#                              #
#  Link Monitoring Attributes  #
#                              #
################################


# determine rate at which link monitoring packets will be sent
(secs/pkt)
# and size of pkt (bits)

"top.ring0.f9.CDL Manager.link_monitor_trans_rate":    .01
#"top.ring0.f9.CDL Manager.link_monitor_trans_rate":   100
"top.ring0.f9.CDL Manager.monitoring pkt size":        10000
"top.ring1.f9.CDL Manager.link_monitor_trans_rate":    100
"top.ring1.f9.CDL Manager.monitoring pkt size":        10000


# attributes related to link monitoring and LQR's
# LQR transmission delta - the change in the ratio of
#      (bad packets/total pkts in history) when an LQR will be
transmitted
# ex: .1 means that an LQR will be sent when the ratio changes by 10%
# hysteresis thresholds determine when to declare a change in the link
status
#      these also are based on the ratio of bad packets/total pkts in
history
# history length is the size of the linked list which holds the number
of errors
#      in the last x monitoring packets

"*.*.f9.*.LQR transmission delta":          0.09
"*.*.f9.*.upper hysteresis threshold":      0.8
"*.*.f9.*.lower hysteresis threshold":      0.4
"*.*.f9.*.history length":                  20
"top.ring1.f9.*.ecc threshold":             0.0


############################
#                          #
#  FDDI Ring Attributes    #
#                          #
############################


# total offered load is the sum of all stations' loads (Mbps).
# Compute this by hand; this value is useful for generating
# scalar plots where offered load is the abscissa.

 total_offered_load_0 :              0.18
 asynch_offered_load_0 :             0.162
 total_offered_load_1 :              0.18
 asynch_offered_load_1 :             0.162


# set the proportion of asynchronous traffic
# a value of 1.0 indicates all asynchronous traffic

"*.*.*.async_mix" :                  0.5
```

```
#"top.ring0.f9.llc_src.async_mix" :        1
"top.ring0.f0.llc_src.async_mix" :         0.0


#*** Ring configuration attributes used by "fddi_mac" ***


# allocate percentage of synchronous bandwidth to each station
# this value should not exceed 1 for all stations combined; OPNET does
not
# enforce this; 01FEB94: this must be less than 1; see equation below

# To determine corresponding data rate:
#     Max Data Rate for Station = (sync bandwidth) * 100Mbps

"*.*.mac.sync bandwidth" :                 0.08955675
#"top.ring0.f9.mac.sync bandwidth":        0.0
#"top.ring0.*.mac.sync bandwidth" :        0.01
#"top.ring1.*.mac.sync bandwidth" :        0.01
#"top.ring0.f0.mac.sync bandwidth":        0.8
#"top.ring1.f9.mac.sync bandwidth":        0.8


# Target Token Rotation Time (one half of maximum synchronous response
time)

# SUM(SAi) + D_Max + F_Max + Token_Time <= TTRT
# Powers gives TTRT = 10 ms as necessary for voice transmission; in
"BONeS",
# D_Max + F_Max + Token_Time = 1.97888 ms.

"*.*.mac.T_Req" :                          .01


# Index of the station which initially launches the token
# This index should be greater than the maximum station number
# Bridge stations spawns token for interconnected simulation by default.

"spawn station":                           20


########################
#                      #
#   CDL Attributes     #
#                      #
########################


# Delay incurred by packets as they traverse a station's ring interface
# see Powers, p. 351 for a discussion of this (Powers gives 1usec,
# but 60.0e-08 agrees with Dykeman & Bux)

station_latency:                           60.0e-08


# Propagation Delay separating stations on the ring.
# If propagation delay is 5.085 microsec/km, this corresponds to
# to a 50 station ring with a circumference of 50 km.
# (The value given for propagation delay corresponds to Powers, and to
```

```
# Dykeman & Bux)

prop_delay:                          5.085e-06


# Jamming Attributes

*.ls_0.jam_ber:                 1e-3
*.ls_1.jam_ber:                 1e-3
*.ls_2.jam_ber:                 1e-3
*.ls_3.jam_ber:                 1e-3
*.ls_4.jam_ber:                 0.0
*.ls_0.no_jam_ber:              1e-12
*.ls_1.no_jam_ber:              1e-12
*.ls_2.no_jam_ber:              1e-12
*.ls_3.no_jam_ber:              1e-12
*.ls_4.no_jam_ber:              0.0
*.ls_0.jam_trans:               0.0001
*.ls_1.jam_trans:               0.0001
*.ls_2.jam_trans:               0.0001
*.ls_3.jam_trans:               0.0001
*.ls_4.jam_trans:               0.0001
*.ls_0.no_jam_trans:            0.0001
*.ls_1.no_jam_trans:            0.0001
*.ls_2.no_jam_trans:            0.0001
*.ls_3.no_jam_trans:            0.0001
*.ls_4.no_jam_trans:            0.0001


# Return and command link propagation delays are specified as 60 msec.

*.ls_0.delay:                                                   0.06
*.ls_1.delay:                                                   0.06
*.ls_2.delay:                                                   0.06
*.ls_3.delay:                                                   0.06
*.ls_4.delay:                                                   0.06


# CDL Channel Rates

"top.ring0.f9.pt_1[0].data rate": 8568000
"top.ring0.f9.pt_2[0].data rate": 42840000
"top.ring0.f9.pt_3[0].data rate": 42840000
"top.ring0.f9.pt_4[0].data rate": 42840000
"top.ring1.f9.pr_1[0].data rate": 8568000
"top.ring1.f9.pr_2[0].data rate": 42840000
"top.ring1.f9.pr_3[0].data rate": 42840000
"top.ring1.f9.pr_4[0].data rate": 42840000
"top.ring1.f9.pt_1[0].data rate": 10240000


# determine which load balancing algorithm is in use in the
# local bridge station (linking node).
# User should specify  the algorithm before simulation.
#
#     0 (zero)  ----> circular load balancing algorithm (default)
#     1 (one)        ----> empty allocation algorithm
#
```

```
"top.ring0.f9.CDL Manager.load balancing algorithm":   1
"top.ring1.f9.CDL Manager.load balancing algorithm":   1


# determine the station address of the bridge node in
# both rings.

"top.ring0.f9.*.station_address":    9
"top.ring1.f9.*.station_address":    19



##############################
#                            #
#   Simulation Attributes    #
#                            #
##############################


# Token Acceleration Mechanism enabling flag.
# It is reccomended that this mechanism be enabled for most situations
# 16APR94 : for bridged fddi_cdl_interconnection network this flag
# must be zero.  Otherwise, program fault occurs.
# error documented on MIL3 bbs - Ike
accelerate_token:                          0


# Run control attributes
seed:                                                  10
#duration:                                             10
verbose_sim:                                           TRUE
#upd_int:                                              0.1
#os_file:                                              cdl_basic_tow


# Opnet Debugger (odb) enabling attribute
debug:                                                         TRUE


##################################
#                              #
#  Real-Time Station Attributes  #
#                              #
##################################


# To set cmpr_vdo_fr packet rate:
#      Packet size = Data Segment Size + 40 bits of header
#      Set token sizes to Packet Size
#      Set token rates to desired packet rate
# Bucket Rate = token_rate * token_size


# ring0.f0 parameters


"top.ring0.f0.llc_src.dest address":               10
```

```
"top.ring0.f0.src.interarrival pdf":            constant
"top.ring0.f0.src.interarrival args":           0.1
"top.ring0.f0.src.pk size pdf":                 constant
"top.ring0.f0.src.pk size args":                1024

"top.ring0.f0.tkn_bkt.token_rate":              0
"top.ring0.f0.tkn_bkt.token_size":              100
"top.ring0.f0.tkn_bkt.bucket_size":             1000000
"top.ring0.f0.tkn_bkt.send_rate":               1000000
"top.ring0.f0.tkn_bkt.max_queue_size":          100000000
"top.ring0.f0.tkn_bkt.data segment size":       800
"top.ring0.f0.tkn_bkt.max_pk_size":             10000
"top.ring0.f0.tkn_bkt.desired_rate":            22021000
"top.ring0.f0.tkn_bkt.min_acceptable_rate":     1376300

"top.ring0.f0.lky_bkt.token_rate":              10000
"top.ring0.f0.lky_bkt.token_size":              840
"top.ring0.f0.lky_bkt.bucket_size":             1000000
"top.ring0.f0.lky_bkt.send_rate":               1000000
"top.ring0.f0.lky_bkt.max_queue_size":          100000000
"top.ring0.f0.lky_bkt.max_pk_size":             10000

"top.ring0.f0.cmpr_vdo_rcvr.Data_file":         ring0_f0.data


# ring1.f0 parameters


"top.ring1.f0.llc_src.dest address":            1

"top.ring1.f0.src.interarrival pdf":            constant
"top.ring1.f0.src.interarrival args":           100
"top.ring1.f0.src.pk size pdf":                 constant
"top.ring1.f0.src.pk size args":                1024

"top.ring1.f0.tkn_bkt.token_rate":              0
"top.ring1.f0.tkn_bkt.token_size":              100
"top.ring1.f0.tkn_bkt.bucket_size":             1000000
"top.ring1.f0.tkn_bkt.send_rate":               1000000
"top.ring1.f0.tkn_bkt.max_queue_size":          100000000
"top.ring1.f0.tkn_bkt.data segment size":       60
"top.ring1.f0.tkn_bkt.max_pk_size":             10000
"top.ring1.f0.tkn_bkt.desired_rate":            0
"top.ring1.f0.tkn_bkt.min_acceptable_rate":     0

"top.ring1.f0.lky_bkt.token_rate":              0
"top.ring1.f0.lky_bkt.token_size":              100
"top.ring1.f0.lky_bkt.bucket_size":             1000000
"top.ring1.f0.lky_bkt.send_rate":               1000000
"top.ring1.f0.lky_bkt.max_queue_size":          100000000
"top.ring1.f0.lky_bkt.max_pk_size":             10000

"top.ring1.f0.cmpr_vdo_rcvr.Data_file":         ring1_f0.data


# ring0.f1 parameters


"top.ring0.f1.llc_src.dest address":            0
```

```
"top.ring0.f1.src.interarrival pdf":            constant
"top.ring0.f1.src.interarrival args":           100
"top.ring0.f1.src.pk size pdf":                 constant
"top.ring0.f1.src.pk size args":                1024

"top.ring0.f1.tkn_bkt.token_rate":              0
"top.ring0.f1.tkn_bkt.token_size":              100
"top.ring0.f1.tkn_bkt.bucket_size":             1000000
"top.ring0.f1.tkn_bkt.send_rate":               100000000000000
"top.ring0.f1.tkn_bkt.max_queue_size":          100000000
"top.ring0.f1.tkn_bkt.data segment size":       60
"top.ring0.f1.tkn_bkt.max_pk_size":             10000
"top.ring0.f1.tkn_bkt.desired_rate":            0
"top.ring0.f1.tkn_bkt.min_acceptable_rate":     0

"top.ring0.f1.lky_bkt.token_rate":              0
"top.ring0.f1.lky_bkt.token_size":              100
"top.ring0.f1.lky_bkt.bucket_size":             1000000
"top.ring0.f1.lky_bkt.send_rate":               1000000000000000
"top.ring0.f1.lky_bkt.max_queue_size":          100000000
"top.ring0.f1.lky_bkt.max_pk_size":             10000

"top.ring0.f1.cmpr_vdo_rcvr.Data_file":         ring0_f1.data


##############################
#                            #
#   CDL Manager Attributes   #
#                            #
##############################


# Forward Error Correction

"top.ring0.f9.CDL Manager.FEC_level":          0.2
"top.ring0.f9.CDL Manager.FEC_overhead":       1.5

"top.ring1.f9.CDL Manager.FEC_level":          1.0
"top.ring1.f9.CDL Manager.FEC_overhead":       1.0
```

D. CDL MODEL OUTPUT

The CDL model is capable of generating a variety of outputs designed to evaluate the performance of the simulation under a myriad of different criteria. This section describes the outputs available within the model and describes how to access them. It is divided into the three logical forms of output generated by the simulation: (1) statistical (or traditional) data, (2) transmitted image reconstruction, and (3) output designed to assist in debugging simulations.

This section assumes a basic knowledge of the forms of output provided by OPNET as well as a general knowledge of the debugging facility available within the simulation environment. Once again, the reader is directed to the OPNET Manuals, if required.

1. Statistical Output

The statistical information provided by the CDL model falls into two major groups: the resource usage statistics and the transmission statistics. Resource statistics are utilized to determine system requirements and the feasibility of implementation. Transmission statistics tend to relate directly to the transmitted packets rather than the system resources.

The resource statistics are gathered through existing OPNET output variables associated with the different processes. The most common ones utilized within the CDL model are those associated with the multiple buffers throughout the system. The output statistics for these queues include the queue size, availability, and delay times experienced by packets passing through the queue. These can be accessed by attaching a probe to the desired statistic within the appropriate module. More information about these standard OPNET output statistics can be found in the OPNET Manual.

The statistics that are gathered to describe the transmission through the CDL model are, for the most part, specific to the CDL model. These user-defined output variables, known as OUTSTATS, are assigned within the process modules and are accessed using the probe facility in a manner identical to that for accessing OPNET's standard output statistics. A descriptive list of the CDL model specific statistics, grouped by process, follows. The list provides the number of the OUTSTAT followed by a brief description. For example, to gather statistics to display the instantaneous throughput into the token bucket, the user would assign a probe to OUTSTAT 0 of the *tl_bkt_tow* process.

Token Bucket (*tl_bkt_tow*):

0 instantaneous throughput into token bucket

1 instantaneous throughput out of token bucket

2 average throughput into token bucket

3 average throughput out of token bucket

4 number of packets dropped by the token bucket

Leaky Bucket (*tl_bkt_std*):

0 instantaneous throughput into leaky bucket

1 instantaneous throughput out of leaky bucket

2 average throughput into leaky bucket

3 average throughput out of leaky bucket

4 number of packets dropped by the leaky bucket

5 cumulative delay experienced by a packet upon arrival to leaky bucket

6 cumulative delay experienced by a packet upon processing by leaky bucket

7 cumulative delay experienced by a packet upon departure from leaky bucket

Real-time LLC Source (*fddi_sender*):

0 cumulative delay experienced by a packet upon arrival to llc source

Real-time LLC Sink (*fddi_rcvr*):

0 creation time of packets received at llc sink

Compressed Video Receiver (*cmpr_vdo_rcvr*):

0 image numbers of packets received at compressed video receiver

1 level numbers of packets received at compressed video receiver

2 sequence numbers of packets received at compressed video receiver

3 creation time of packets received at compressed video receiver

4 instantaneous end-to-end delay of packets received at compressed video receiver

5 mean end-to-end delay of packets received at compressed video receiver

6 average throughput of packets received at compressed video receiver

7 total number of packets received at compressed video receiver

CDL Manager (*CDL_manager*):

0 ratio of bad packets in the history to history length currently maintained in link monitoring history

1 current link status (0 = GOOD, 1 = BAD)

In addition to the statistics listed above, there are a number of user-defined global output statistics specific to the CDL model. These are automatically generated (without the need of probes) and are available in the analysis tool of OPNET. They are referenced by name and are not associated with a particular process.

Jamming	state of jamming (0 = OFF, 1 = ON)
Error Ratio per Packet	errors per bit of current packet (in CDL pipe)

2. Image Generation

This CDL simulation set-up differs from many of the existing set-ups utilized in the study of real-time transmission schemes in that a facility to reconstruct the transmitted image has been provided. This is achieved by a set of MATLAB programs that interface the OPNET model output data files with the MATLAB reconstruction routines. This section describes the procedure to produce the received image.

The CDL simulation produces an output data file for each individual compressed video receiver. The name of this file is assigned in the environment file (see Section B). This file is used as an input to the MATLAB routine create_mask:

create_mask('*filename*', *number of images*)

The filename must be in quotes and does not include the suffix ".data". The number of images is the number of images transmitted during the session and can be determined by examining the image number of the last few entries in the data file. Create_mask creates a series of MATLAB workspaces associated with the transmission session. There is one workspace per transmitted image, each containing a variable, "mask," used to reconstruct that image. The workspace files are named image#_mask.mat and are used by the routine recomp. Create_mask calls a routine eval_mask to produce plots of the number of packets and values lost per image in the transmission session. These can be saved using the MATLAB command print.

Recomp is called without arguments and prompts the user for the following information:

(1) *filename to be reconstructed* - This is the name of the original image file to be used by the routine. The original image used in the body of the thesis is 'airplane'. The entry should be in single quotes and contain no extensions.

(2) *image number to be generated* - This is the number of the transmitted image to be reconstructed and is mapped to the appropriate workspace file.

This file produces three images. The first is the original image transmitted by the sender. the second is the received image and includes the received signal-to-noise ratio. Finally, an image is generated reflecting the errors in the received image. In essence, this image represents the difference between the transmitted image and the received image. Encapsulated postscript files of these images are created if the user requests hard copy images when prompted by the routine. The files are original_image.eps, run_image.eps, and run_image_error.eps, respectively.

3. Debugging

It is often beneficial to be able to view the progress of a simulation while it is running. This is accomplished through the debugging facility provided by OPNET. Specifically, a trace can be requested for the desired object. Numerous user-defined label traces were created to be used in conjunction with the CDL model. These traces are called in the format:

ltrace *trace_name*

This section describes these CDL specific traces and, in general, assumes a basic knowledge of the debugging facility in OPNET. It is worth noting that the debug mode must be active to make use of these traces.

(1) *rates* - traces the process of real-time flow establishment and update in the presence of jamming

(2) *FEC* - traces the establishment of the forward error correction mechanism in the presence of jamming

(3) *mntr* - traces the link monitoring mechanism

(4) *xmtrs* - traces the utilization of the various bit pipes in the CDL implementation

(5) *bkt* - traces the actions of the token and leaky bucket modules

(6) *rt_pkt* - traces the transmission of real-time packets throughout the simulation (It is more efficient to use the debug command pkmap to identify a specific packet by number and use the debug command pktrace to trace it.)

(7) *img* - records the arrival of each new image

The first three traces are the most useful and provide insight into the proposed real-time transmission mechanism.

E. CDL MODEL MODIFICATION

This section discusses the procedures and issues involved in making the following foreseeable changes to the existing CDL model.

(1) The addition of extra real-time FDDI stations to the model.

(2) The reorganization of the channel structure of the CDL.

(3) The implementation of an admission control algorithm.

(4) The implementation of end-to-end user feedback.

(5) The implementation of an alternate high level protocol.

The first two can be thought of as extensions to the existing model, while the last three can be thought of as additions to the existing model.

1. Additional Real-time FDDI Stations

Additional real-time FDDI stations may be added to the existing model by replacing the standard FDDI module with the real-time module, *fddi_station_ts*. The attributes discussed in Sections B and C will need to be initialized and a real-time station must be chosen for the destination. Additional stations may also be added by adding additional nodes to the local FDDI ring. This would necessitate the adjustment of the FDDI attributes as well as the initialization of the new station attributes. The added station could be made a real-time or a non-real-time station by choosing the appropriate module.

2. CDL Channel Reorganization

The CDL is logically organized as a collection of channels, or bit-pipes. The existing CDL model breaks the link into four channels. This can be altered to more

realistically model the actual link by adding the appropriate number of transmitters and receivers at the CDL network interfaces. Each transmitter and receiver pair must be connected by a channel and the appropriate channel attributes must be initialized. In addition, the *"number_of_xmtrs"* attribute of the transmitting *CDL_manager* must be adjusted to reflect the updated configuration.

3. Admission Control

The admission control algorithm is essential for an effective network-wide employment of any real-time transmission scheme. This algorithm would typically reside within the *CDL_manager* for the CDL model. Specifically, the *sink* state would be modified to conduct a screening process prior to establishing a real-time flow. This screening process would require that the process examine available bandwidth on both the attached LAN and the attached CDL. If the bandwidth is not available for the requested flow, the *CDL_manager* would send the requesting application a *"CDL_manager_cntrl_pkt"* with the *"join"* attribute set to zero to signify a rejection of the request.

4. End-to-End User Feedback

End-to-end user feedback can be used to adjust the minimum required image quality at the receiver. To implement this, the receiving application must send a control packet back to the sending application requesting an increase (or decrease) in the number of resolution levels being transmitted. This would trigger a renegotiation between the sending application and the CDL Manager for the new flow parameters. This renegotiation can be modeled as a return to the existing real-time flow establishment procedures using the updated transmission rates.

5. Alternate High Level Protocol

The implementation of an alternate high level protocol, such as TCP/IP, requires extensive additions to the existing model. This alternate protocol would have direct contact with the CDL Manager and llc source/sink in the same manner as the existing FDDI LANs. Such a modification must carefully consider whether the CDL_NI must function as a router or a bridge or both. The issues related to this are discussed in [Ref. 22]. Thus, in the case of TCP/IP, an IP module would have access to the CDL Manager and would provide many of the same services to the CDL Manager as the current *mac* process. All TCP/IP traffic received over the CDL would pass through this IP module to determine whether or not the packet is destined for a station on the attached LAN. Thus, the *CDL_manager* process must be modified to perform a filtering operation based on a protocol field. In addition, because IP resides at a higher level than the FDDI mac, additional issues specific to a network layer protocol must be addressed.

APPENDIX B. PROGRAM LISTINGS

A. OPNET Model

The source code for the CDL simulation model in OPNET is not included in this appendix due to its sheer bulk. In addition, the source code is presented in a more readable format when the user prints the hard-copy of the code from within OPNET itself. This is accomplished by selecting the appropriate process and generating an OPNET report. The interested reader is encouraged to generate reports concerning the processes and modules discussed in this thesis.

B. MATLAB

1. create_mask.m

```
function Losses = create_mask(filename, num_of_images);

%%%%%
%%%%% create_mask - input is filename of simualtion data and
%%%%%                          number of images in file
%%%%%                output is matrix of values and packets lost per image
%%%%%
%%%%%     Losses = create_mask(filename, num_of_images)
%%%%%
%%%%%     Written by T. Owens Walker III
%%%%%                 Spring 1995
%%%%%


% filename = 'ring1_f0';
num_values_per_pkt = 100;
num_sgmts_per_block = 16384;
num_blocks_per_image = 16;

%%%%%  Load data file into mask_data matrix

eval(['load ', filename, '.data']);
eval(['mask_data = ',filename, ';']);
[Rows, Cols] = size(mask_data);
```

```
%%%%%  Break mask_data apart into separate images

for n = 1:num_of_images
      eval(['image',num2str(n-1),'  =  zeros(1,  (num_blocks_per_image  *
num_sgmts_per_block) + num_values_per_pkt);']);
end

for row = 1:Rows
   start_value = mask_data(row,3) + 1;
   end_value = mask_data(row,3) + num_values_per_pkt;

eval(['image',num2str(mask_data(row,1)),'(1,',sprintf('%d',start_value),
':', sprintf('%d',end_value), ') = ones(1,num_values_per_pkt);']);
end

%%%%%  Store masks in the proper *.mat files

Losses = zeros(2, num_of_images);

for n = 1:num_of_images
      eval(['mask  =  image',num2str(n-1),  '(1:(num_blocks_per_image  *
num_sgmts_per_block));']);
   mask = reshape(mask, num_sgmts_per_block, num_blocks_per_image)';
   [Losses(1,n),Losses(2,n)] = eval_mask(mask);
   eval(['save image',num2str(n-1), '_mask mask']);
end

bar(0:(length(Losses(1,:)) - 1),Losses(1,:));
xlabel('Image Number'); ylabel('Number of Values Lost');

figure
bar(0:(length(Losses(2,:)) - 1),Losses(2,:));
xlabel('Image Number'); ylabel('Number of Packets Lost');
```

2. eval_mask.m

```
function [values_lost, packets_lost] = eval_mask(mask);

%%%%%
%%%%% eval_mask - input is a mask matrix
%%%%%              outputs vectors of packets and values lost
%%%%%
%%%%%     [values_lost, packets_lost] = eval_mask(mask);
%%%%%
%%%%%     Written by T. Owens Walker III
%%%%%                Spring 1995
%%%%%

[r,c] = size(mask);
values_lost = (r*c) - sum(sum(mask));
packets_lost = values_lost/100;
```

3. idwt2DEC.m

```
function im=idwt2DEC(file,HH,HV,GH,GV,Nh,Nv,L1,L2,lowest,flt,qq)
% IDWT2     Two-dimensional inverse discrete wavelet transform
%     IDWT2(FILE) returns the reconstructed image IM by
%     taking the inverse wavelet transform of wavelet coefficients
%     stored in FILE_DWT.

%  Version 1.3 by R.M. Carvalho   24 September 1994

% Load the wavelet coefficients and construct the filter matrices
%-------------------------------------------------------------------
eval(['load ',file,'_test'])

% Reconstruction routine
%-------------------------------------------------------------------

whos
[lr,lc]=size(d11);
min_val=d11(1,1);
max_val=d11(1,2);

for index=1:4

   if index>1

      if size(eval(['d',num2str(index),'1']))>=[1,4]

         eval(['mv=(d',num2str(index),'1(1,1));'])
         eval(['Mv=(d',num2str(index),'1(1,2));'])
         eval(['d',num2str(index),'1=d',num2str(index),'1(3:lc);'])

eval(['d',num2str(index),'1=reshape(d',num2str(index),'1,N1,N2);'])

eval(['d',num2str(index),'1=n2mbit2(d',num2str(index),'1,8,8,mv,Mv,0);'])

      end


   else
      N1
      N2
      d11=d11(1,3:lc);
      size(d11)
      d11=reshape(d11,N1,N2);
      d11=n2mbit2(d11,8,8,min_val,max_val,0);

   end

   if size(eval(['d',num2str(index),'2']))>=[1,4]

       eval(['mv=(d',num2str(index),'2(1,1));'])
       eval(['Mv=(d',num2str(index),'2(1,2));'])
       eval(['d',num2str(index),'2=d',num2str(index),'2(3:lc);'])
      eval(['d',num2str(index),'2=reshape(d',num2str(index),'2,N1,N2);'])
```

```
eval(['d',num2str(index),'2=n2mbit2(d',num2str(index),'2,8,8,mv,Mv,0);'])

   end

   if size(eval(['d',num2str(index),'3']))>=[1,4]

      eval(['mv=(d',num2str(index),'3(1,1));'])
      eval(['Mv=(d',num2str(index),'3(1,2));'])
      eval(['d',num2str(index),'3=d',num2str(index),'3(3:lc);'])
      eval(['d',num2str(index),'3=reshape(d',num2str(index),'3,N1,N2);'])

eval(['d',num2str(index),'3=n2mbit2(d',num2str(index),'3,8,8,mv,Mv,0);'])

   end

   if size(eval(['d',num2str(index),'4']))>=[1,4]

      eval(['mv=(d',num2str(index),'4(1,1));'])
      eval(['Mv=(d',num2str(index),'4(1,2));'])
      eval(['d',num2str(index),'4=d',num2str(index),'4(3:lc);'])
      eval(['d',num2str(index),'4=reshape(d',num2str(index),'4,N1,N2);'])

eval(['d',num2str(index),'4=n2mbit2(d',num2str(index),'4,8,8,mv,Mv,0);'])

   end


end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%%%%%  Apply Results of Transmission Simulation
%%%%%              Written by T. Owens Walker III
%%%%%                          Thesis Work
%%%%%                          Spring 1995
%%%%%

%%%%%  Load mask workspace

image_number = input('Enter Image Number to be Generated:  ');
eval(['load image', num2str(image_number),'_mask']);

%%%%%  Apply Mask

[MR,MC] = size(mask);
[DR,DC] = size(d11);
i = 1;
for n=1:4
  for m=1:4
     mask_row = find(I == i);
     mask_temp = reshape(mask(mask_row,:),DR,DC);
       eval(['d',int2str(m),int2str(n),' = d',int2str(m),int2str(n),  '.*
mask_temp;'])
     i = i + 1;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% check for zero matrices, rebuild to size
for coef=1:4

       if size(eval(['d',num2str(coef),'1']))<[4,4]

            eval(['rows=d',num2str(coef),'1(1,1);'])
            eval(['cols=d',num2str(coef),'1(1,2);'])
            eval(['d',num2str(coef),'1=zeros(rows,cols);']);

       end

       if size(eval(['d',num2str(coef),'2']))<[4,4]

            eval(['rows=d',num2str(coef),'2(1,1);'])
            eval(['cols=d',num2str(coef),'2(1,2);'])
            eval(['d',num2str(coef),'2=zeros(rows,cols);']);

       end

       if size(eval(['d',num2str(coef),'3']))<[4,4]

            eval(['rows=d',num2str(coef),'3(1,1);'])
            eval(['cols=d',num2str(coef),'3(1,2);'])
            eval(['d',num2str(coef),'3=zeros(rows,cols);']);

       end

       if size(eval(['d',num2str(coef),'4']))<[4,4]

            eval(['rows=d',num2str(coef),'4(1,1);'])
            eval(['cols=d',num2str(coef),'4(1,2);'])
            eval(['d',num2str(coef),'4=zeros(rows,cols);']);

       end

end

%lvl 2 reconstruction

lvl=2
for coef=1:4
       [HH,HV,GH,GV,Nh,Nv]=filts2(L1/lvl,L2/lvl,flt,qq,1);

       HV=(2)*HV;
       HH=(2)*HH;
       GV=(2)*GV;
       GH=(2)*GH;

       eval(['cwrk=rctver2(d',num2str(coef),'1,HV,Nv);'])
       eval(['dwrk1=rctver2(d',num2str(coef),'2,GV,Nv);'])

       [r,c]=size(dwrk1);

       datv1=cwrk(1:r,1:c)+dwrk1;
       eval(['dwrk2=rctver2(d',num2str(coef),'3,HV,Nv);'])
       eval(['dwrk3=rctver2(d',num2str(coef),'4,GV,Nv);'])
       datv2=dwrk2+dwrk3;
       dath1=rcthor2(datv1,HH,Nh);
```

```
      dath2=rcthor2(datv2,GH,Nh);
      eval(['d0',num2str(coef),'=dath1+dath2;'])
end

% lvl 1 reconstruction
lvl=1
      [HH,HV,GH,GV,Nh,Nv]=filts2(L1/lvl,L2/lvl,flt,qq,1);

      HV=(2)*HV;
      HH=(2)*HH;
      GV=(2)*GV;
      GH=(2)*GH;

        coef=0;
      d01=d01';
      d02=d02';
      d03=d03';
      d04=d04';

      eval(['cwrk=rctver2(d',num2str(coef),'1,HV,Nv);'])
      eval(['dwrk1=rctver2(d',num2str(coef),'2,GV,Nv);'])

      [r,c]=size(dwrk1);

      datv1=cwrk(1:r,1:c)+dwrk1;
      eval(['dwrk2=rctver2(d',num2str(coef),'3,HV,Nv);'])
      eval(['dwrk3=rctver2(d',num2str(coef),'4,GV,Nv);'])
      datv2=dwrk2+dwrk3;
      dath1=rcthor2(datv1,HH,Nh);
      dath2=rcthor2(datv2,GH,Nh);
      im=dath1+dath2;
      im=rot90(im);
      im=flipud(im);
      [Lv Lh]=size(im);
```

LIST OF REFERENCES

[1] Defense Support Project Office, *CDL System Description Document for Common Data Link (CDL)*, Specification Number 7681996, 1993.

[2] S. Shenker, D. Clark, and L. Zhang, "A Service Model for an Integrated Services Internet," Internet Draft, October 1993.

[3] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," *IEEE/ACM Transactions on Networking*, Vol. 1, No. 3, June 1993.

[4] D. Ferrari, "Client Requirements for Real-Time Communication Services," *IEEE Communications Magazine*, Vol. 28, No. 11, pp. 65-72, November 1990.

[5] B. A. Coan and D. Heyman, "Reliable Software and Communication III: Congestion Control and Network Reliability," *IEEE Journal on Selected Areas in Communications*, Vol. 12, No. 1, January 1994.

[6] H. Kanakia, P. P. Mishra, and A. Reibman, "An Adaptive Congestion Control Scheme for Real-Time Packet Video Transport," *Proceedings of ACM SIGCOMM*, pp. 20-31, October 1993.

[7] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network*, September 1993.

[8] D. Ferrari, A. Banerjea, and H. Zhang, "Network Support for Multimedia: A Discussion of the Tenet Approach," *Computer Networks and ISDN Systems*, Vol. 26, pp. 1267-1280, 1994.

[9] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *Proceedings of ACM SIGCOMM*, pp. 13-12, 1989.

[10] D. D. Clark, S. Shenker, and L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," *Proceedings of ACM SIGCOMM*, pp. 14-27, August 1992.

[11] R. M. Carvahlo, "Multi-Resolution Image Compression Using Sub-Band Coding and Wavelet Decomposition," Master's Thesis, Naval Postgraduate School, Monterey, California, December 1994.

[12] S. G. Mallat, "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 7, pp. 674-693, 1989.

[13] O. Rioul and Martin Vetterli, "Wavelets and Signal Processing," *IEEE Signal Processing Magazine*, pp. 14-28, October 1991.

[14] A. Erdemir, "Data Compression by Using Wavelet Transforms and Vector Quantization," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1993.

[15] H. J. Barnard, *Image and Video Coding Using a Wavelet Decomposition*, Ph.D. Dissertation, Delft University, Netherlands, 1994.

[16] D. C. Schmidt, "Safe and Effective Error Rate Monitors for SS7 Signaling Links," *IEEE Journal on Selected Areas in Communications*, Vol. 12, No. 3, April 1994.

[17] J. W. Eichelberger, "Design and Modelling of a Link Monitoring Mechanism for the Common Data Link (CDL)," Master's Thesis, Naval Postgraduate School, Monterey, California, September 1994.

[18] A. Takeshi,, "Distributed Multilink System for Very-High-Speed Data Link Control," *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 4, pp. 540-549, May 1993.

[19] S. Karayakaylar, "Data Link Level Interconnection of Remote Fiber Distributed Data Interface Local Area Networks (FDDI LANs) Through the Critical Data Link (CDL)," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1994.

[20] E. E. Nix, "Modeling and Simulation of a Fiber Distributed Data Interface Local Area Network (FDDI LAN) Using OPNET® for Interfacing Through the Common Data Link (CDL)," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1994.

[21] MIL 3, Inc., *OPNET Modeler*, user's manual in 11 volumes, 3400 International Drive NW, Washington, D.C. 20008, 1993.

[22] S. Shukla, *Design Requirements for the Common Data Link's Network Interface*, Technical Report NPS-EC-94-011, Naval Postgraduate School, September 1994.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5101	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5121	1
4.	Professor Murali Tummala, Code EC/Tu Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5121	4
5.	Professor Shridhar Shukla, Code EC/Sh Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5121	1
6.	Professor Loomis, Code EC/Lm Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5121	1
7.	Professor Paul Moose, Code EC/Me Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5121	1

8.	Professor Gilbert Lundy, Code CS/Ln Department of Computer Science Naval Postgraduate School Monterey, California 93943-5118	1
9.	Professor Van Emden Henson, Code MA/Hv Department of Mathematics Naval Postgraduate School Monterey, California 93943-5216	1
10.	CDR K. Webb, Code SPAWAR 72 Space and Naval Warfare Systems Command Crystal Park #5, 2451 Crystal Dr. Arlington, VA 22202-5100	2
11.	CDR D. Gear, Officer in Charge NISE EAST Detachment Washington 3801 Nebraska Ave N.W. Washington, D.C. 20393	2
12.	LCDR Skinner Advanced Maritime Projects Office Building 659, Box 51 NAS Jacksonville, FL 32212	2
13.	LT T. Owens Walker III PCU John C. Stennis Supervisor of Shipbuilding, Conversion and Repair, USN Newport News, VA 23607-2787	2
14.	Mr. Marc Russon, LORAL Mail Station F2-G14 640 North 2200 West Salt Lake City, UT 84116-2988	3
15.	Program Manager, Common Data Link Defense Airborne Reconnaissance Office Washington, D.C. 20330-1000	2