NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND MODELLING OF A LINK MONITORING MECHANISM FOR THE COMMON DATA LINK (CDL)

by

John W. Eichelberger III

September 1994

Thesis Advisor: Shridhar B. Shukla

Approved for public release, distribution is unlimited.

. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1994	Master's Thesis

. TITLE AND SUBTITLE

)ESIGN AND MODELLING OF A LINK MONITORING MECHANISM FOR THE :OMMON DATA LINK (CDL)

5. FUNDING NUMBERS

. AUTHOR(S)

ohn W Eichelberger, III

. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

laval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

. SUPPLEMENTARY NOTES

he 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 ie U S Government

a. DISTRIBUTION/AVAILABILITY STATEMENT

pproved for public release, distribution is unlimited

12b. DISTRIBUTION CODE

A

. ABSTRACT (maximum 200 words)

he Common Data Link (CDL) is a full duplex, point-to-point microwave communications system used in imagery and signals intelligence ollection systems It provides a link between two remote Local Area Networks (LANs) aboard collection and surface platforms In a hostile nvironment, there is an overwhelming need to dynamically monitor the link and thus, limit the impact of jamming
his work describes steps taken to design, model, and evaluate a link monitoring system suitable for the CDL The monitoring system is based features and monitoring constructs of the Link Control Protocol (LCP) in the Point-to-Point Protocol (PPP) suite The CDL model is based a system of two remote Fiber Distributed Data Interface (FDDI) LANs In particular, the policies and mechanisms associated with onitoring are described in detail An implementation of the required mechanisms using the OPNET network engineering tool is described erformance data related to monitoring parameters is reported Finally, integration of the FDDI-CDL model with the OPNET Internet model is scribed

14. SUBJECT TERMS	15. NUMBER OF PAGES
ATA LINK, FDDI, LAN, PPP, MONITORING	252
	16. PRICE CODE

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

; 7540-01-280-5500

Standard Form 298 (Rev. 2-89)

Approved for public release, distribution is unlimited.

DESIGN AND MODELLING OF A LINK MONITORING MECHANISM FOR THE COMMON DATA LINK (CDL)
by

John W. Eichelberger III
Lieutenant, United States Navy
B.S., U.S Naval Academy, 1988

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1994

Author: John Warren Eichelberger III

Approved By: Shridhar B Shukla, Thesis Advisor

Gilbert Lundy, Second Reader

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

ABSTRACT

The Common Data Link (CDL) is a full duplex, point-to-point microwave communications system used in imagery and signals intelligence collection systems. It provides a link between two remote Local Area Networks (LANs) aboard collection and surface platforms. In a hostile environment, there is an overwhelming need to dynamically monitor the link and thus, limit the impact of jamming.

This work describes steps taken to design, model, and evaluate a link monitoring system suitable for the CDL. The monitoring system is based on features and monitoring constructs of the Link Control Protocol (LCP) in the Point-to-Point Protocol (PPP) suite. The CDL model is based on a system of two remote Fiber Distributed Data Interface (FDDI) LANs. In particular, the policies and mechanisms associated with monitoring are described in detail. An implementation of the required mechanisms using the OPNET network engineering tool is described. Performance data related to monitoring parameters is reported. Finally, integration of the FDDI-CDL model with the OPNET Internet model is described.

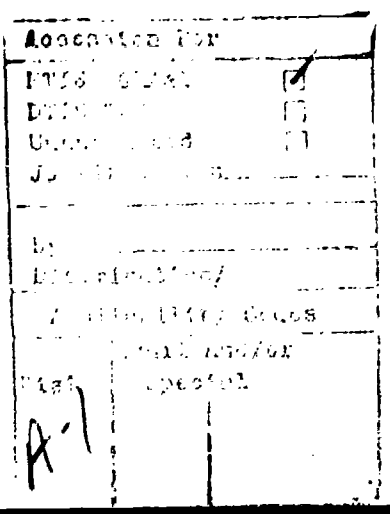


TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. CDL OVERVIEW

This thesis deals with the Common Data Link (CDL), a project of the Defense Support Project Office (DSPO). The CDL is a full duplex, point-to-point microwave data link designed to provide jam resistant communications between two remote platforms, such as an airborne asset and a surface platform. The purpose of the airborne platform is to collect signal and image intelligence data for transmission over the CDL. In turn, the surface platform evaluates this data while transmitting command, control, and communications information back to the aircraft.

The downlink, or return link, operates at a data rate of 274.176 Megabits per second (Mbps) as a high rate, or can be scaled down to a low rate of 10.71 Mbps. It is a multiplexed data stream of bit pipes, ranging in rates from 25 Kilobits per second (Kbps) to 42.84 Mbps. The bit pipes are hierarchically multiplexed for the 274.176 Mbps configuration as shown in Figure 1.

The uplink, or command link, operates at 200 Kbps. The transmitted data is comprised of mostly equipment commands multiplexed with audio and synchronization bits. For the purposes of this work, it is assumed to be unjammable. This assumption is supported by current microwave link implementations.

B. OBJECTIVES

The primary objective of this work is to design a link monitoring mechanism and its associated constructs for the CDL. Secondly, the mechanism must be modelled in a computer simulation, in this case using MIL 3, Inc.'s Optimized Network Engineering Tool (OPNET), in order to evaluate the faithfulness of the monitoring design. Additional model changes will also be presented to integrate OPNET models of Transmission Control Protocol/Internet Protocol (TCP/IP) into the existing CDL-FDDI model. This integrated CDL network model will serve as a building block to evaluate the link's susceptibility to jamming, and will aid in development of CDL-specific applications.

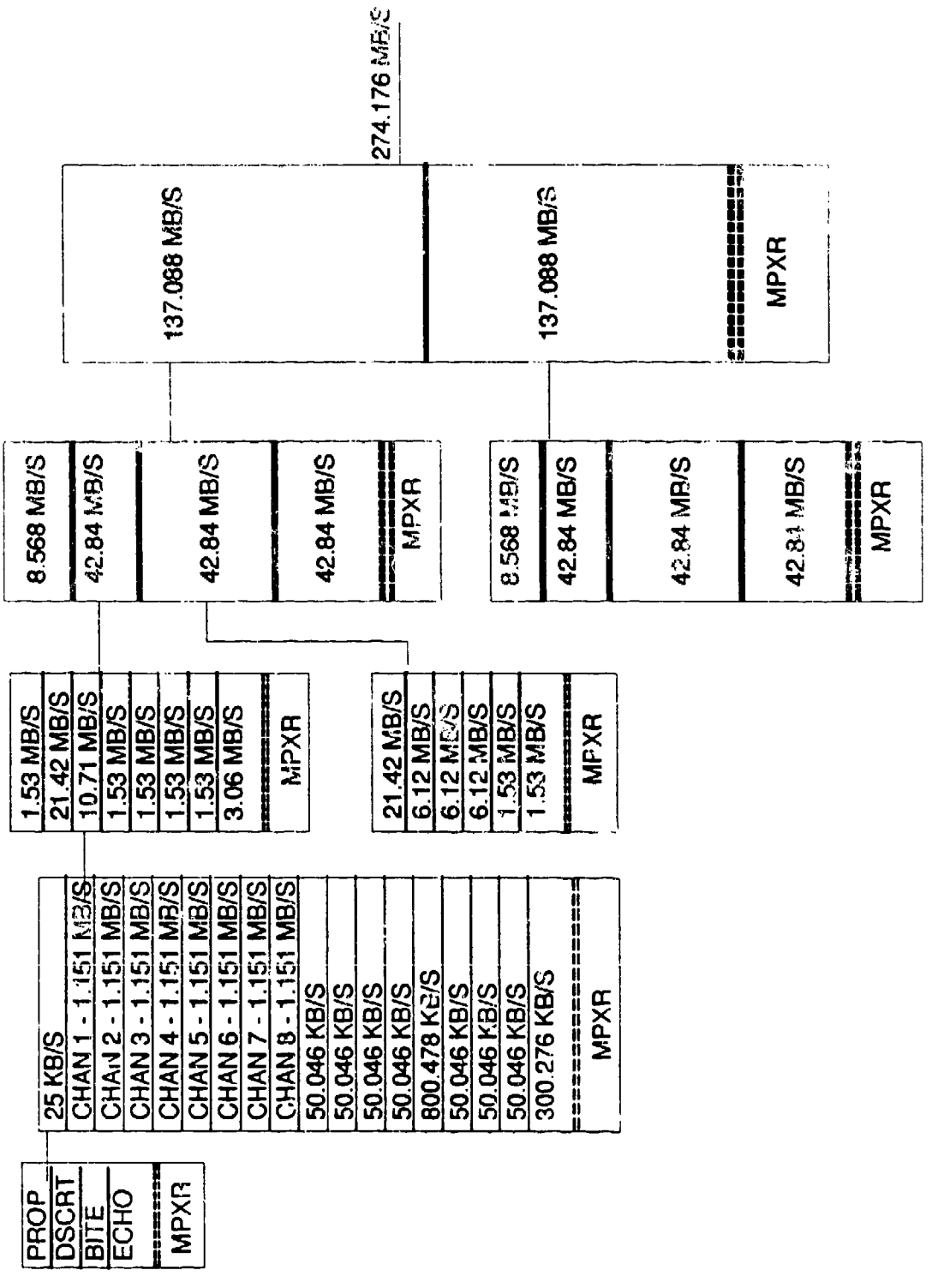


Figure 1: CDL Multiplexer Hierarchy [1]

C. SCOPE

The scope of this thesis includes the following:

- Introduce the Point-to-Point Protocol (PPP) and support its use as a basis for a link monitoring algorithm.
- Present an approach to link monitoring and the issues surrounding the chosen approach.
- Discuss and evaluate the link monitoring additions made to the OPNET model of the remote LAN interconnection using CDL.
- Present changes to existing TCP/IP OPNET models to utilize them with the enhanced network model.

D. ORGANIZATION

Chapter II provides an introduction to PPP and its integration into the CDL scheme. Chapter III delineates issues involved in developing the chosen link monitoring algorithm as well as an overview of the process itself. Chapters IV through VI deal with the OPNET implementation of both link monitoring and the TCP/IP models. Conclusions and recommendations are listed in Chapter VII.

II. POINT-TO-POINT PROTOCOL (PPP)

Requirements for the CDL network interface (NI) have been established and evaluated by [2]. The recommendations include a remote bridge between the collection and surface platforms, which is actually a pair of half bridges, each of which connects a LAN to a point-to-point link. Figure 2 illustrates this remote bridge connection.

The connectivity is to be implemented in the Primary Communication Elements (PCE) and Surface Communication Elements (SCE) rather than the network layer due to requirements that the CDL may need to route datagrams from the network layer and bridge frames from the Media Access Control (MAC). Therefore, some type of link-level protocol is necessary for communication and data transmission between the two bridges, and the Point-to-Point Protocol (PPP) is seen as an ideal choice. Detailed discussion of the level of connectivity is also included in [2].

A. GENERAL DESCRIPTION

PPP is a data link protocol standard adopted by the Internet Engineering Task Force (IETF), the governing body for standards related to the Internet. The goal of PPP is to provide a standard method for transporting protocol data units of multiple protocols over point-to-point links through a simple encapsulation scheme.

PPP has two sub-protocols associated with it. First is the Link Control Protocol (LCP), which is the administrative arm of PPP. The LCP allows a multitude of establishment, configuration, and testing options. The second sub-protocol is the Network Control Protocol (NCP). Its purpose is to smooth out incompatibilities between different types of net[illegible]level protocols (e.g. IP, Appletalk, Novell IPX, etc.) and enable their data units to be transmitted over one point-to-point link.

B. RATIONALE FOR CHOOSING PPP

Using PPP within the remote bridge scheme deals with the bridge communication and data transmission issues, but further analysis uncovers many additional benefits to the

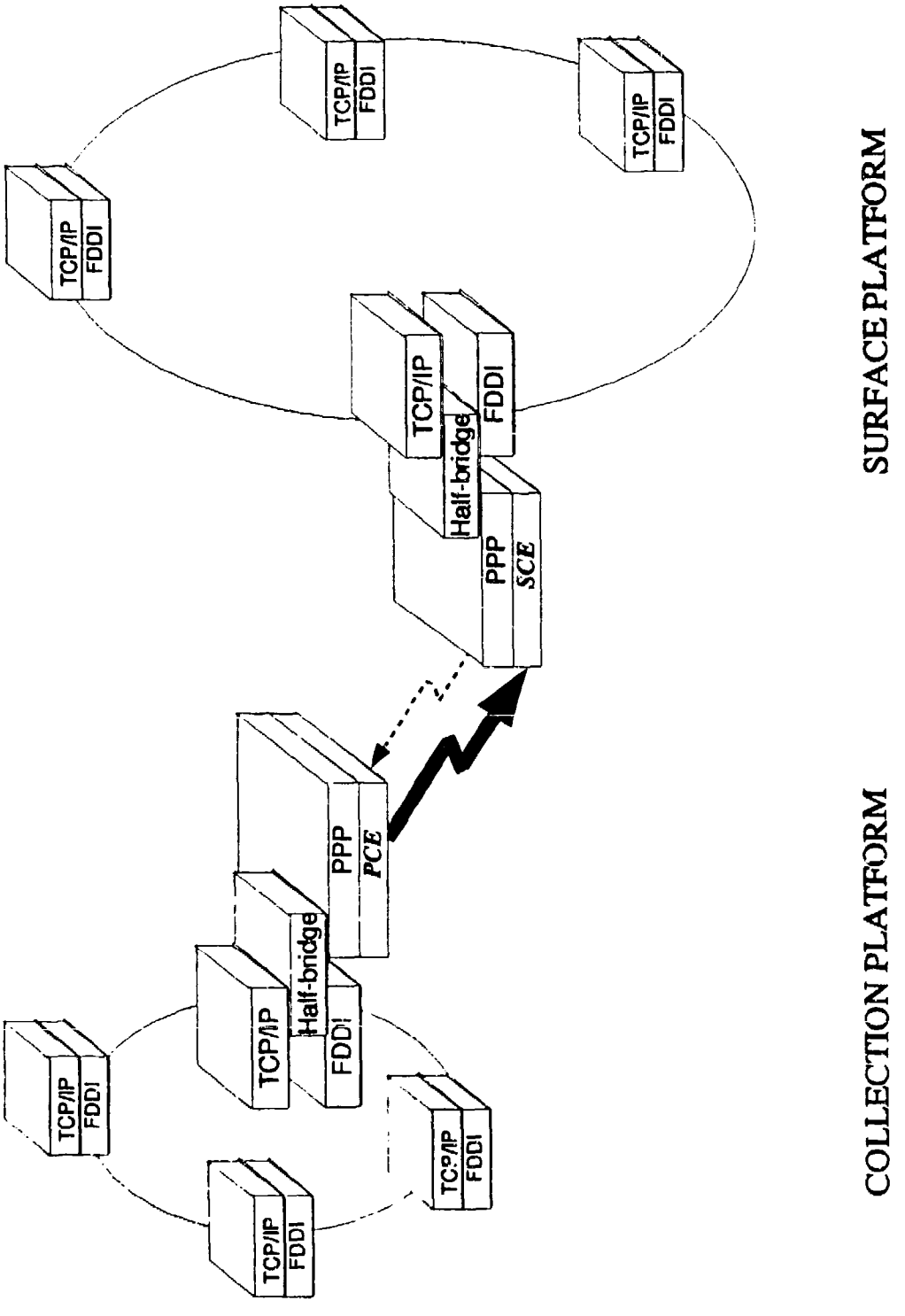


Figure 2: The Role of PPP in a CDL-based Remote LAN Interconnection

CDL. These include increased bandwidth utilization, link monitoring capabilities, dynamic reconfiguration, and future expandability to network-level interconnection. Each of these reasons further supports PPP as the protocol of choice.

1. Maximum Bandwidth Utilization

Current CDL specifications describe a fixed multiplexer hierarchy on the return link. This limits the type of equipment from which information can be transmitted on the CDL because the equipment must conform to a hard-wired data rate. If the nonconforming equipment is installed anyhow, a slower unit will waste bandwidth, while a faster unit will cause overhead in time and extra hardware due to fragmentation requirements. Through LCP reconfiguration of PPP packet sizes and built-in fragmentation constructs, PPP can make this multiplexer hierarchy invisible to the applications.

Another difficulty posed by a fixed multiplexer hierarchy is the inability of the senders to utilize extra bandwidth that may be available in the downlink. For instance, if a piece of equipment breaks or is not being used, the bandwidth it would normally use is filled with useless padding in order to maintain link synchronization.

PPP provides a resequencing capability to distribute frames to any multiplexer input and resequence the data at the receiving end, effectively using all available bandwidth. In jamming scenarios, non-realtime data which can tolerate resequencing-related delays can be buffered at the source to be sent as bursts over multiple multiplexer inputs during non-jamming periods.

2. Dynamic Configuration Changes

In most military applications, the effectiveness of equipment relies on its ability to adapt. Different situations call for different capabilities, and in the case of the CDL, its deployment in the joint services may force a wide array of platforms to be interfaced with the link for a variety of missions with different data requirements. Even in any one mission, information such as imagery may need a higher resolution at different points in time.

Though the equipment may have that capability, in a fixed multiplexer hierarchy the application-to-link connection may not be easily altered

PPP offers a solution to this problem through dynamic reconfiguration of the data link using LCP reconfiguration constructs already in place. Furthermore, PPP can handle data coming from the onboard FDDI ring, or from a network-level connection with another platform, all through dynamic configuration.[3]

3. Flexibility and Upward Compatability

Another problem caused by the current fixed hierarchy is that deployment of newer networking technologies would not be easily possible. Hard-wired data rates and a fixed multiplexer hierarchy hinder any long-term expansion plan, which will be necessary in the quickly expanding field of computer networks. Current research into gigabit-speed networks and beyond could cause CDL obsolesence even before its full-scale deployment, if the approach to the CDL network interface lacks proper vision of the future.

Work reported here is based on our position that PPP is the point-to-point data link protocol of the future, both for the CDL and the networking community, in general. It is an adopted standard, widely used, and easily adaptable to proprietary implementations. It has been used on T1 and T3 links, high-delay satellite links, and is being considered for gigabit-speed SONET links. Most importantly, PPP's flexibility will make CDL portable and upgradable, allowing for greater deployment, interoperability, and expanding capabilities in line with future technology.

4. Link Monitoring

Jamming is a potential threat in any military application. Even without jamming, minor interference can play havoc with data intensive links transmitting through the atmosphere. Some form of error detection, such as frame check sequences (FCS), must be used to alleviate this problem.

Any data link protocol has some form of error detection. For instance, FDDI has an FCS on every frame. This is examined and evaluated at the ultimate destination. The

error handling could therefore be left as an end-to-end problem between the source and destination.

However, in a CDL scenario, errors have a much higher probability of occurrence over the link itself than on the FDDI rings at either platform. Thus, it makes sense to check each frame as it comes across the link. Unfortunately, for the bridging stations to check each frame would require breaking up each frame, FCS calculations, and reframing in order to transmit on the FDDI ring. This presents an unacceptable overhead and violates the end-to-end principle.[4]

The solution proposed is to encapsulate every outgoing FDDI frame in another frame, namely PPP, with error checking being conducted on the higher level instead of breaking apart the underlying FDDI frame. Although PPP is a data link layer protocol just as FDDI, it can be used to encapsulate the FDDI frame for point-to-point transmission.

Link monitoring also involves reporting the link status. When an FCS is evaluated, the receiving station knows how the transmission is being affected, but to be of use in flow control, the information must be sent to the transmitting site. PPP has built-in link quality monitoring constructs, including a Link Quality Report (LQR), whose transmission frequency can be negotiated at link establishment time.

C. PPP FRAME FORMATS

Encapsulation in PPP is attained by merely attaching a header to the data to be framed. Any packet being transmitted through PPP will be prepended with a 16 bit Protocol IDentification (PID) which identifies the network protocol which spawned the packet. If PPP is generating a control frame, it identifies the type of control frame. Additional padding bytes may be appended to the data, if desired.

Framing requirements are not specifically delineated in the PPP standard [5] to allow flexibility in implementation. However, a separate standard has been adopted to provide High-Level Data Link Control (HDLC) framing to PPP frames [6]. This is the only current PPP framing standard, but it does not preclude a further study into a more suitable framing

method for CDL purposes. In any case, this work assumes PPP will be implemented with HDLC framing.

1. Multilink Protocol

The Multilink Protocol is a proposed addition to the suite of Internet PPP standards currently in place[7]. Its primary function is to allow for dynamic changes in the number and size of point-to-point links between two PPP servers. If this should occur, the Multilink Protocol enables PPP to maximize its use of the available bandwidth.

It accomplishes this in three steps, illustrated in Figure 3. First, encapsulated PPP packets are fragmented, if desired, depending on the data rates of the individual links. This can be used to transmit each packet across the varying data rates in the shortest time, and ensuring no one link has data in the transmission pipe longer than any other. Second, all PPP fragments and other designated PPP packets are further encapsulated with a Multilink header. In turn, these Multilink packets are encapsulated in another PPP header and finally framed to be transmitted.

The Multilink Protocol format is shown in Figure 4. In the Multilink header, the sequence number is a 24-bit value used to keep fragments in order. The 'B' and 'E' bits designate the beginning and ending fragments of a packet sent in multiple frames. This 4 byte Multilink header may be substituted with a two byte header in which the sequence number is reduced to 12 bits and the 'B' and 'E' bits remain intact.

The address, control, and frame check sequence (FCS) fields are specific to the HDLC framing format, and even so, can be negotiated between the PPP servers to not be transmitted, thereby eliminating unnecessary overhead.

2. Link Quality Report (LQR)

The Link Quality Report (LQR) is an established PPP mechanism which allows a sender to determine how its data is being received at the receiving end of a point-to-point link[8]. It is composed of 12 32-bit fields containing statistics maintained at the receiver,

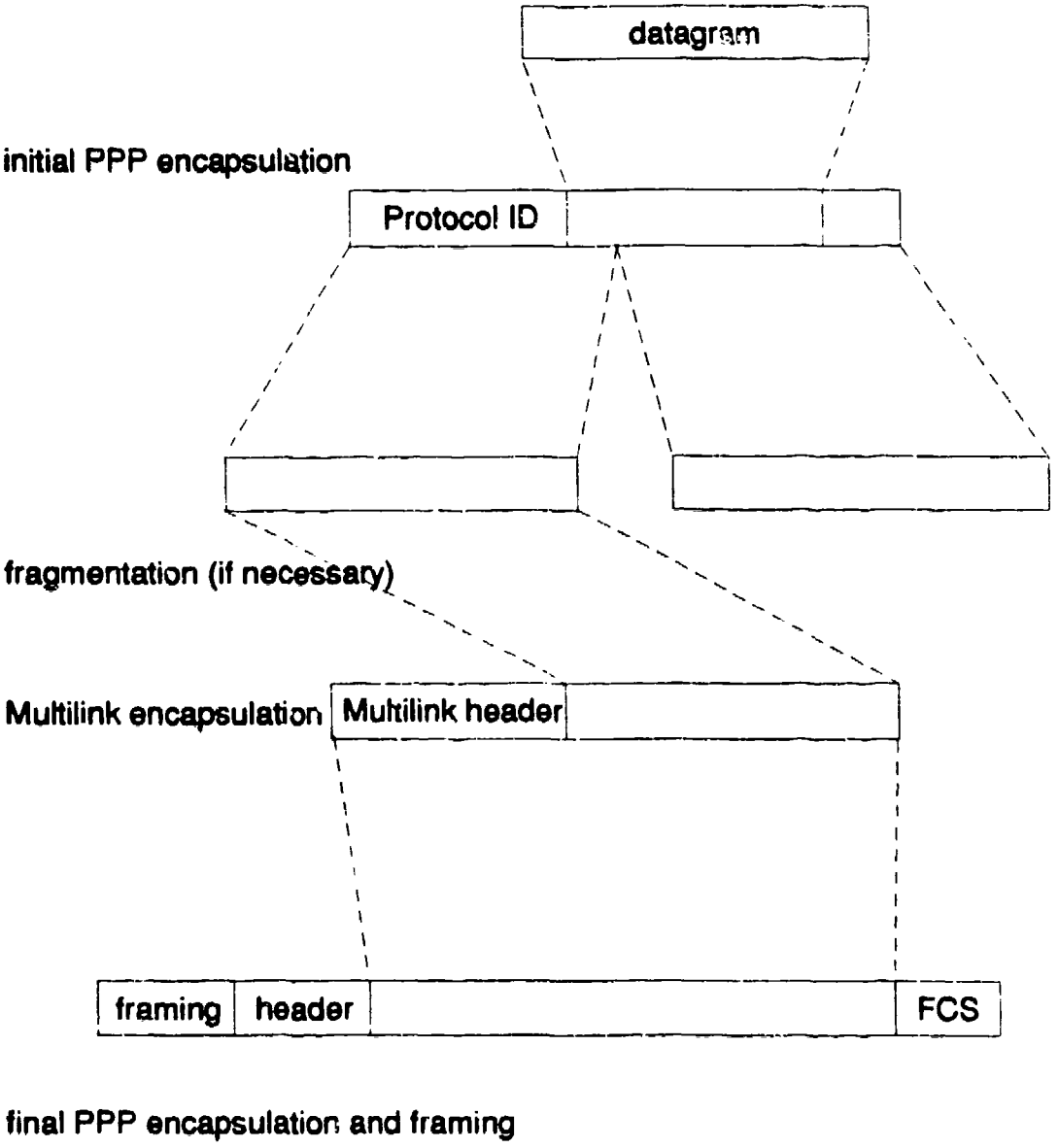
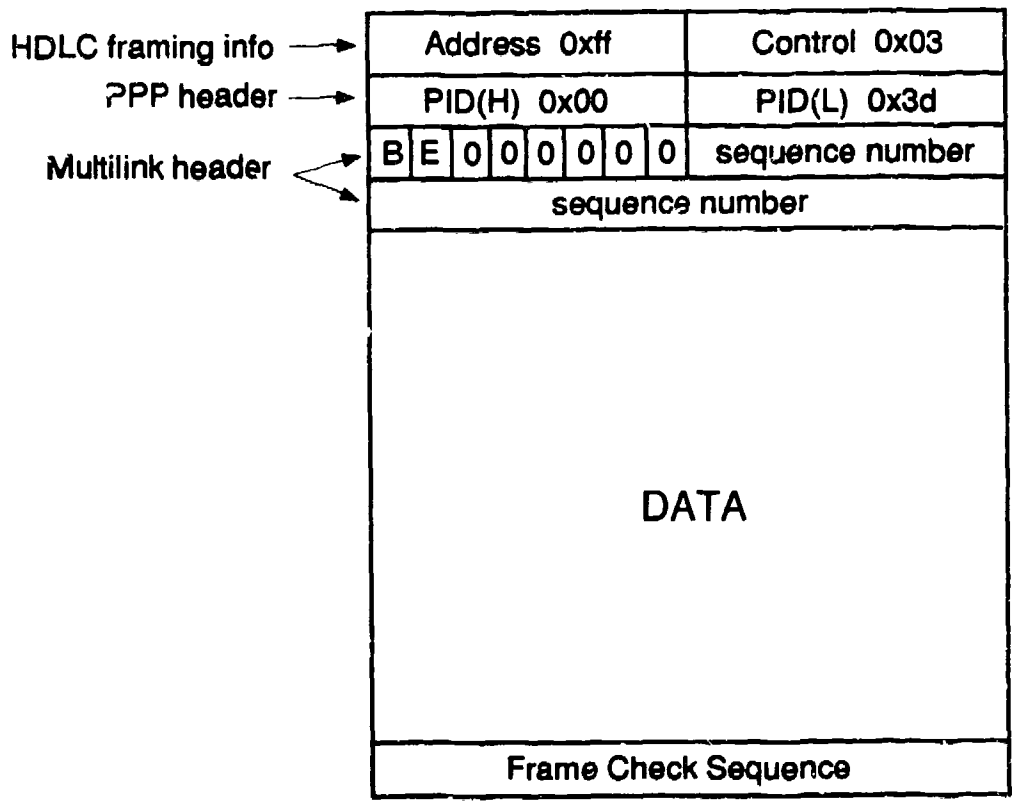


Figure 3: PPP Multilink Encapsulation Procedure



Address: HDLC field - 0xff = all addresses
Control: HDLC field - 0x03 = unnumbered information
PID: PPP field - 0x003d = PPP Multilink Protocol
B/E: Multilink flag - set when this frame is a (B)eginning or (E)nd fragment
Sequence Number: Multilink field - prevents reordering of fragments
FCS: HDLC field - checksum field

The Address, Control, and FCS fields assume HDLC framing, and even so, are completely optional.

Figure 4: PPP Multilink Frame Format

as shown in Figure 5. Upon receipt of an LQR, a station appends another 5 32-bit fields of its own statistics, bringing the total size to 544 bits.

The basic premise in the development of the LQR format and transmission characteristics is that the transmitter will evaluate its own downlink based on statistics of the link accumulated at the receiving end. These values are grouped into a PPP frame which will be sent to the transmitting site. The transmitting site can then independently evaluate the statistics and reconfigure the link as it sees fit, assuming the receiving side agrees with the changes. This procedure supports independence between the transmitter and receiver wherein the receiver does not need to know any capabilities or characteristics of the transmitter other than those established during link configuration. For CDL purposes, independence is indeed necessary, but other factors call for significant changes in both the LQR format and implementation, as well as other aspects of link monitoring in general.

D. SUMMARY

This chapter has presented PPP as a suitable choice for the data link protocol to interconnect the PCE and SCE over the CDL. This decision is further supported by numerous additional PPP benefits including increased bandwidth utilization, link monitoring, dynamic link configuration, and future expandability. Relevant PPP frame formats have also been introduced, namely the basic PPP format, Multink protocol, and the Link Quality Report. These data frame formats, as well as PPP link monitoring constructs, form the basis for developing a CDL link monitoring mechanism.

TRANSMITTED FORMAT

Address	Control	PID
Magic Number		
LastOutLQRs		
LastOutPackets		
LastOutOctets		
PeerInLQRs		
PeerInPackets		
PeerInDiscards		
PeerInErrors		
PeerInOctets		
PeerOutLQRs		
PeerOutPackets		
PeerOutOctets		

APPENDED STATISTICS

SaveInLQRs
SaveInPackets
SaveInDiscards
SaveInErrors
SaveInOctets

Figure 5: PPP Link Quality Report Format

III. LINK MONITORING APPROACHES/ISSUES

A. OVERVIEW

This chapter deals with the development of a link monitoring mechanism, based on PPP constructs, for the CDL. Figures 6 and 7 graphically present the proposed approach, which is outlined below:

1) FDDI frames/datagrams are encapsulated with a PPP header and framed.
2) If necessary, PPP frames are fragmented.
3) PPP frames and fragments are reframed in the PPP Multilink protocol.
4) Pseudo-random monitoring frames are injected into the data stream at a fixed rate.
5) The stream of PPP frames is distributed amongst multiplexer inputs.
6) PPP frames are pulled off demultiplexer outputs at the receiving end.
7) PPP data frames are decapsulated and the FDDI frames/datagrams are sent to the appropriate layer for continuing transmission.
8) Errors are detected in monitoring frames, and the results are maintained in a fixed size history.
9) Using the history length, the fraction of bad packets is calculated.
10) Based on this fraction, LQRs are transmitted at fixed intervals with information regarding link status (good/bad) and trend since last LQR (up/down). Link status is determined (at the surface platform) based on hysteresis, to eliminate excessive link fluctuations.
11) LQRs are received and used to determine the necessary actions by the transmitting station.

B. MONITORING PACKETS VERSUS FRAME CHECK SEQUENCES

The first issue confronting CDL link monitoring is whether to use the FCS built into HDLC-based framing, or use another form of error detection such as individual monitoring

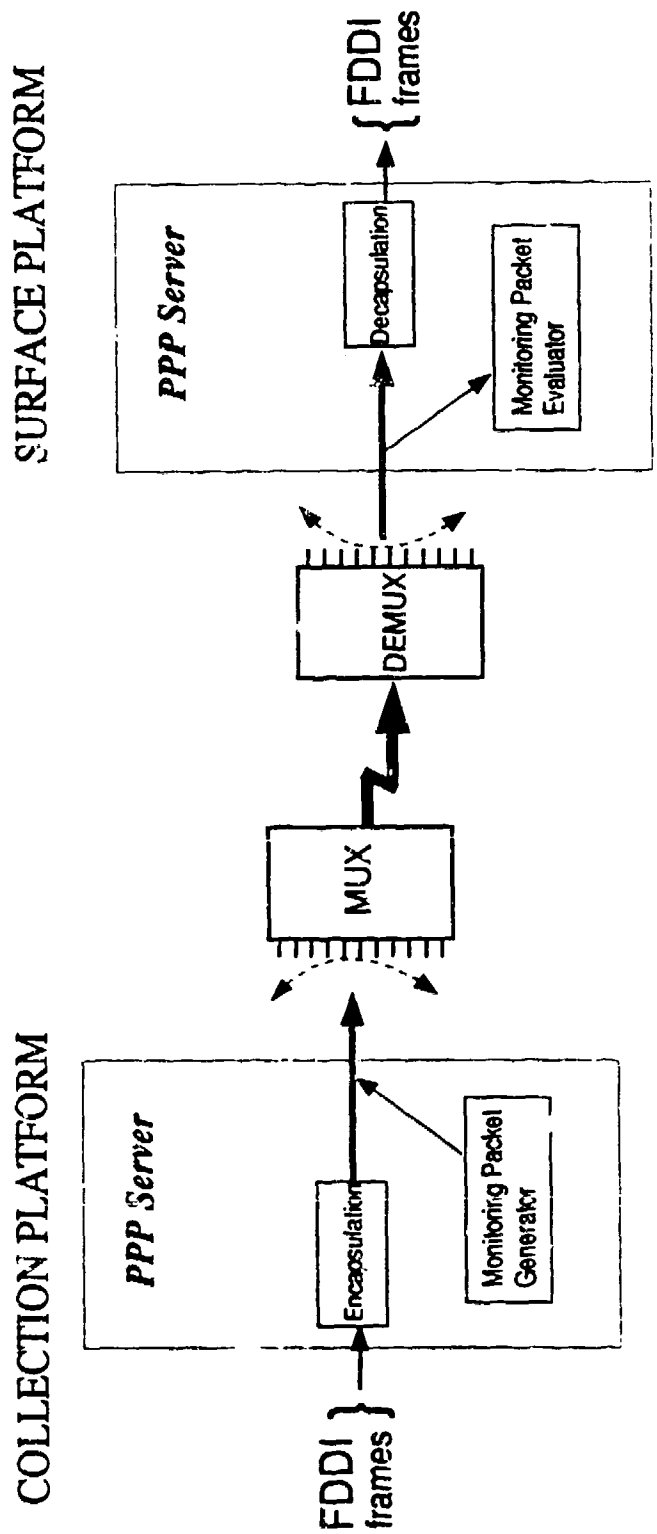


Figure 6: CDL Return Link Monitoring

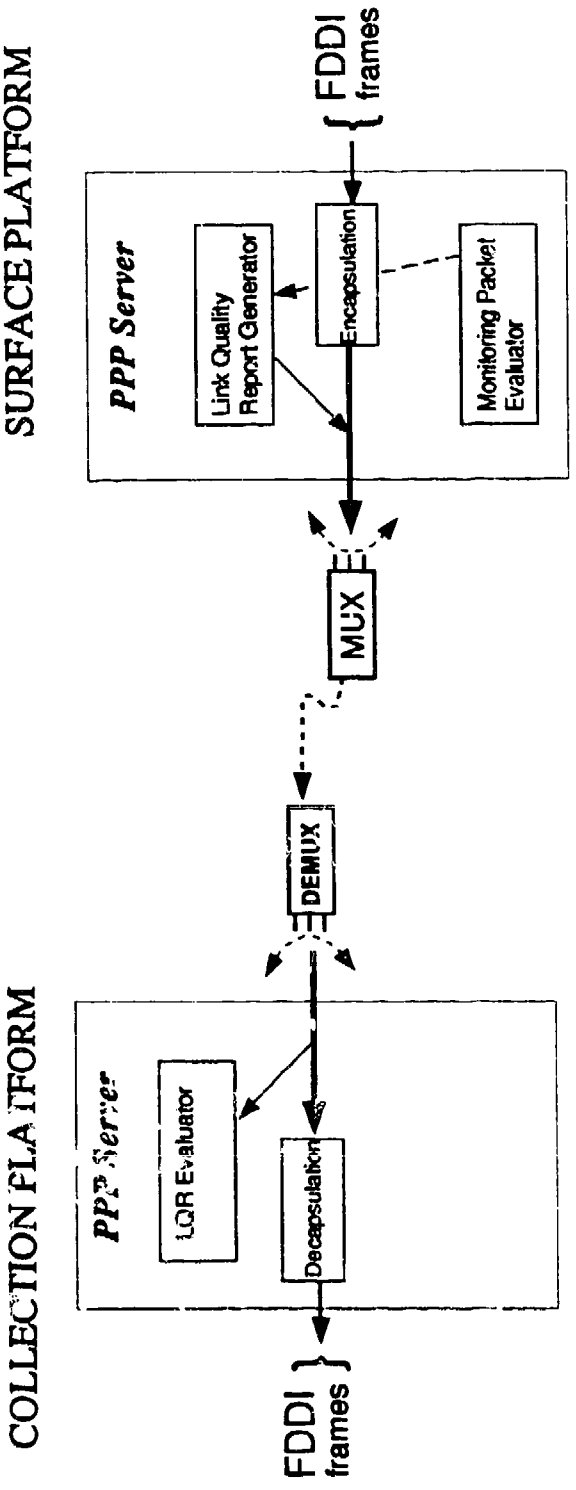


Figure 7: CDL Command Link LQR Transmission

packets. The monitoring packet approach uses either fixed or pseudo-random data streams packetized, framed, and interleaved with regular PPP data frames. At the receiving end, the monitoring packets are easily compared to the expected contents, and errors per packet can be quickly counted without excessive computation.

On the other hand, an FCS requires detailed calculations, and the complexity of the calculations increases as the number of bit errors to be detected increases. These computations must be done twice, once before transmission and once after reception. Also, FCS can never guarantee 100% error detection, whereas preset monitoring packets always guarantee perfect detection.

C. MONITORING PACKET GENERATION

Three factors play a major role in monitoring packet generation. They are packet size, insertion rate and packet format. Individually, each has a major impact on the monitoring effectiveness, but they must be balanced to avoid unnecessary overhead or undetected link interference.

1. Packet Size

Monitoring packet size has a clear impact on error detection: the bigger the packet, the better the BER estimation. However, this factor is limited due to the data multiplexing at the hardware level. Since each framed monitoring packet is broken up in the aggregate data stream, monitoring bits are, in essence, uniformly distributed throughout the data.

This is much more effective than a pattern of a certain number of data packets followed by one monitoring packet. If the jamming pulse duty cycle was low (short pulses with longer intervals between pulses), data could become garbled at a much higher rate than the monitoring packets, yet an evaluation of the monitoring packets would only show slight degradation. In contrast, a uniformly dispersed monitoring packet would increase its detection capabilities in proportion to the number of monitoring packet fragments within

the data, and the resulting evaluation would be more faithfully indicative of the correct link status.

2. Packet Insertion

There are two factors with regard to packet insertion: insertion rate and pattern of insertion. Insertion rate is the rate at which monitoring frames are created and inserted into the output data stream. A pattern of insertion refers to the manner in which a monitoring frame is placed into the data stream at individual multiplexer inputs so as to effect maximum error detection.

a. Insertion Rate

As in the case of packet size, insertion rate has a clear impact on monitoring: the faster and more frequently monitoring packets are transmitted, the better the monitoring. Unfortunately, where the distribution of monitoring bits in the aggregate data stream limited the impact of packet size on monitoring effectiveness, this is not the case with the insertion rate. Thus, the insertion rate will determine, to a great degree, whether there is excessive monitoring overhead or sampling of the error events on the aggregate data stream is insufficient.

b. Pattern of Insertion

Insertion of monitoring frames into the CDL multiplexer hierarchy can take on three primary patterns. First, monitoring frames may be input only on a specific multiplexer input. Secondly, frames may simply follow a round robin scheme, oblivious to the data rate of the particular input. Finally, frames can be input to the least loaded multiplexer input - that input with the smallest queue waiting to be transmitted.

For the first strategy, the main benefit is simplicity. Unfortunately, this scheme depends too heavily on the load balancing between multiplexer inputs and the inputs' data rates. If there is no load balancing, the selected input pipe may get backlogged, forcing important data to compete for transmission time. On the other hand, even if load

balancing is in place, the input may simply be too slow to handle all the monitoring traffic, again causing backlog in the transmission queue and worse, inhibiting the actual monitoring transmission rate and thereby, the monitoring effectiveness.

A round robin procedure has some of the same limitations as the fixed input strategy. While the individual load on each input would have less effect, the disparity between multiplexer input data rates could cause clustering of monitoring bits in the final data stream. For example, it would be possible to transmit a continuous stream of multiplexed monitoring bits followed by a stream of data bits. Such a scenario is undesirable because the errors in the data part of the stream will not be caught by the monitoring packets. While this extreme scenario is unlikely, a strictly round robin distribution on disparate data rates decreases the probability of error detection.

While round robin and fixed input insertion patterns are fundamentally flawed with regard to a CDL situation, a better pattern can be found by evaluating the discrepancies with each. With fixed input, the primary concern was backlogging any one input, while round robin was not robust enough to ensure uniform spreading of monitoring bits. The empty selection insertion strategy solves both problems at once.

Using empty selection as an insertion pattern is effective, but not without its downfalls. For instance, determining the emptiest multiplexer input requires constant monitoring of all input queues and relating queue size to the input's data rate. Also, a slow pipe could still be chosen, causing non-uniform distribution of monitoring bits.

The packet insertion strategy must address a trade-off between effectiveness and simplicity of implementation. Queue monitoring and non-uniform distribution can be dramatically reduced by using smaller monitoring packets and increasing the insertion rate, thereby keeping the effective monitoring frame transmission rate constant. To this end, empty selection is an efficient insertion pattern.

3. Overhead and Effective Sampling

In summary, packet size and insertion characteristics are critical parameters in any monitoring policy. The total transmission overhead can be calculated using these two criteria (Equation 1). Assuming uniform distribution of monitoring bits, the total amount of

$$\textit{Transmission Overhead} = \textit{Packet Size}\ (\text{bits/pkt}) \times \textit{Insertion Rate}\ (\text{pkts/s}) \qquad (1)$$

overhead is the only factor that will determine if the monitoring bits will be an effective sampling of the true bit error rate (BER). This implies that as long as the interval between monitoring bits in the aggregate data stream is less than the jamming pulse duration, the monitoring status will accurately reflect the link status.

Now that a quantitative value for overhead can be set, the only question is how to vary packet size and insertion rate to maintain acceptable overhead. Based on the earlier evaluation of both factors, we may conclude that overhead policy should be controlled by varying the monitoring packet insertion rate and maintaining a small monitoring packet size. This shall ensure a quick monitoring response time while still utilizing the maximum allowable quantity of overhead.

4. Packet Format

The only issue in relation to packet format was the contents: would it be a fixed data pattern or some random pattern. Clearly both approaches have merits: a fixed data pattern would be easily compared at the receiving end, but would lack true probabilistic independence; a random pattern could pass the independence test, but would have no basis for comparison at the receiving end.

The solution is again a hybrid of both formats: a pseudo-random generator which would be initialized by both PPP servers. Now, the data pattern is random for all practical purposes, but is still easily compared to an expected value at the receiving end.

D. MONITORING PACKET EVALUATION

1. History

When a monitoring packet is received, the number of errors are counted and this value is placed in an array of fixed length. A pointer is incremented to the next element in the array, and the next error count is placed in the next element. When the pointer reaches the end, it resets to the beginning of the array. Each time a new element is entered, a new ratio of packets in error (in the history) to total packets (in the history) is computed, ranging from 0 to 1.

Another idea in evaluating the ratio would be to add up the total errors in the history, and calculate an average BER and act on that. A running total can easily be maintained, just as when counting bad packets. The only additional requirement would be memory increasing from one bit per history element to two bytes per history element. This is a viable option, but it has not been further explored in this work.

The ratio generated from the history is the basis for transmitting LQRs and overall link status, so the length of the history is a crucial element in the reporting process of link monitoring. For instance, a history of length 1 would cause the link to bounce between good and bad each time the current packet was different than the previous packet. Expanding that idea, a history of length 10 will oscillate, on the average, at a rate 1/10 less than a history of 1; a history of length 100 at a rate 1/10 that of length 10, and so on. However, a data link that changes status too slow is as much a detriment as a link which changes too fast. Slow changes in history cause sluggish response in transmitting LQRs, and result in inefficient monitoring feedback to the transmitting station. Requirements of a quick response time and limited bounce force another compromise in order to achieve both desired results.

2. Hysteresis

Hysteresis acts as a history for the history, allowing a small history length for quick response, and at the same time preventing too much oscillation. Figure 8 illustrates the principle for the CDL.

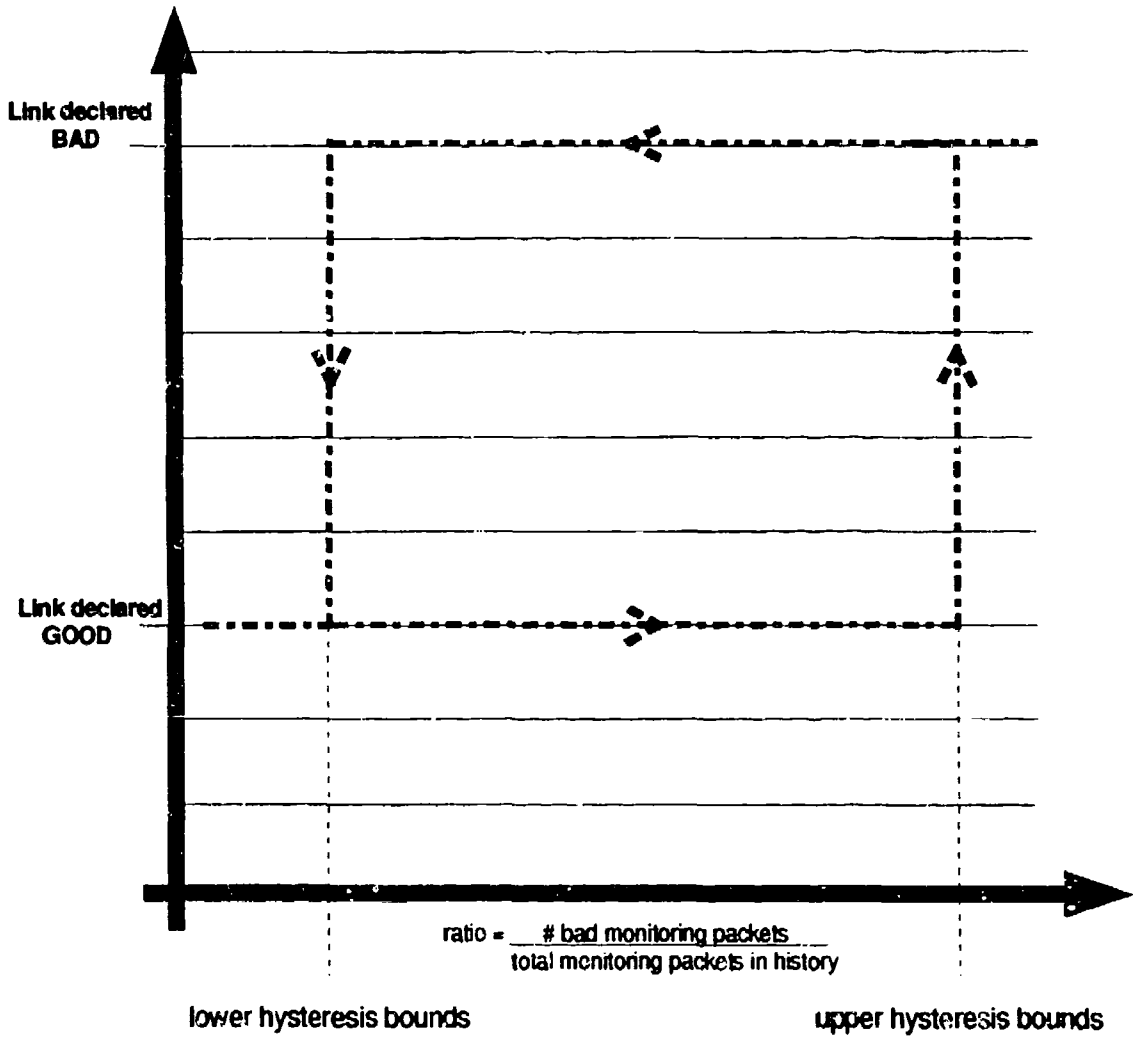


Figure 12: Hysteresis in the CDL

Initially, upper and lower hysteresis bounds are set for the ratio, which will determine when the link is declared good or bad. The link starts out 'good', and as bad packets are received, the ratio goes up. If the ratio oscillates between values without crossing the upper hysteresis bound, the link remains in a 'good' status. Once the ratio passes the upper bound, the link is immediately declared 'bad' and will remain bad until the lower hysteresis bound is crossed.

Hysteresis bounds must be chosen carefully, as narrow bounds will cause excessive fluctuation just as a small history length, and wide bounds will create long delays in link status changes. It is recommended that this be a dynamic attribute, as there is no method to alleviate erroneous results caused by fixed hysteresis bounds.

3. LQR Transmission Frequency

LQRs can be transmitted at a fixed or variable rate depending on the criteria on which transmission is based. For example, a fixed rate transmission may occur if an LQR is sent for every three monitoring packets received. On the other hand, a variable rate transmission may occur if an LQR is sent when the last two monitoring packets are bad. Clearly, a fixed LQR transmission is easy to implement, while a variable rate allows for flexibility and better trend evaluation. Using the positive aspects of both transmission methods, another compromise is seen as the best approach for the CDL.

The method recommended is a fixed rate based on the ratio. The interval between the ratio bounds of 0 to 1 is evenly divided into a certain number of divisions. An LQR will be sent as each threshold is passed, either increasing or decreasing. This translates the fixed rate into a variable rate, as LQRs will be transmitted at a rate commensurate with how fast the ratio is changing, i.e. the slope of the ratio curve. This can impart an urgency to the transmitting station that something is causing the ratio to change quickly, and the link may go bad soon.

This process can only accurately perceive a quick change in the ratio if the monitoring frames are being transmitted at a constant rate. On the other hand, another

approach is where the monitoring frame transmission rate could change based on a fixed LQR transmission rate. However, this option was disregarded because evaluation logic and dynamic transmission capabilities would be necessary on the collection platform. As explained in the next section, it is desirable to keep the collection platform as simple as possible.

E. LQR EVALUATION

1. Evaluation Location

When constructing the monitoring algorithm, the physical limitation which stood out was the disparate data rates of the downlink and uplink. Secondly, weight, space, and equipment requirements added to the notion that most of the link monitoring actions and evaluations should be conducted at the surface platform, and as little information as possible should be transmitted on the low data rate uplink that was not concerned with mission command and control. Therefore, while the collection platform actually evaluates the LQR, the LQR construction must be conducted on the surface with as much information packed into as few bits as possible.

2. Information Content

Due to the minimal bandwidth available for transmission, the actual PPP LQR construct is not a viable option. instead, a proprietary LQR is recommended which must contain, at a minimum, the status of the link. Additional requirements would be based on mission type and applications being used. For general purposes, a two-bit report was deemed sufficient, which included link status and the trend of the link.

In a different context, minimal information transmitted to the collection platform can have a multitude of possibilities. For instance, a transmission containing n bits could lead to 2^n possible actions to be taken. Secondly, each different value could signify a threshold at which certain BER-sensitive applications could sleep or wake up. A final

scenario can be seen where the bits could be interpreted as a preset contingency action, such as a frequency shift.

3. Acting Upon an LQR

No study has been done to determine actions to be conducted upon receipt of an LQR. Minimum requirements for LQR content have been established, but no hard and fast evaluation implementation has been recommended. However, it is recommended that LQR information content should be based on a general mission scenario common to CDL deployment situations, while evaluation methods should be variable based on specific mission and platform scenarios.

IV. OPNET MODELLING OF PPP AND LINK MONITORING

A. BASIC OPNET CDL MODEL DESCRIPTION

The CDL has been modelled in OPNET as two remote FDDI LANs connected by a number of point-to-point links Figure 9 shows the OPNET network level representation of the model. Ring 0 represents the Collection Platform (CP) LAN and ring 1 portrays the Surface Platform (SP) LAN. The four point-to-point links terminating at ring1 denote components of the 274 Mbps return link, while the one link back to ring 0 models the 200 Kbps command link.

Each basic FDDI station, illustrated in Figure 10, has the capability to transmit synchronous and prioritized asynchronous traffic.Additionally, the station collects statistical data particular to the OPNET implementation. *Llc_src* and *llc_sink* are OPNET model entities of the Logical Link Control (LLC), while *mac* represents the FDDI MAC

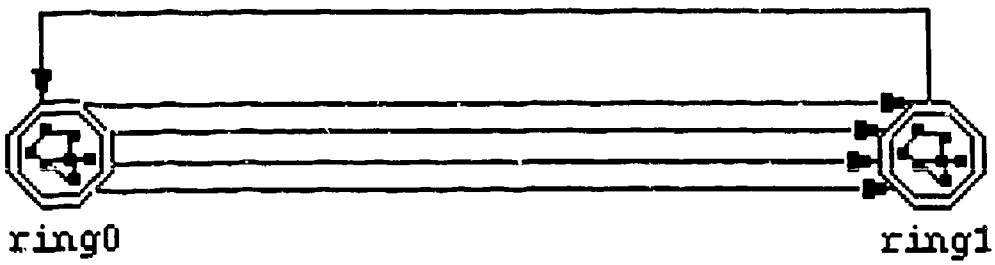


Figure 9: Top Level CDL Model

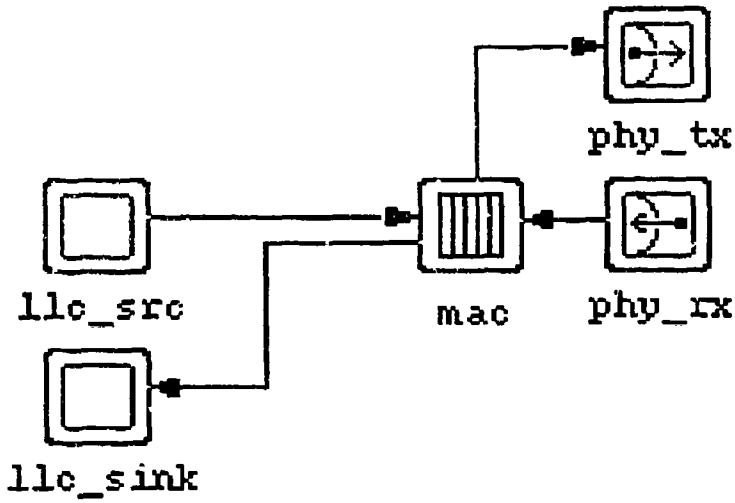


Figure 10: Basic FDDI Station

layer. Finally, *phy_tx* and *phy_rx* represent the physical layer transmitter and receiver. Additional details of this FDDI station model can be found in [9].

Application connections are represented by 10 stations on each ring, each of which can transmit data on the ring destined for any station on either ring. Each LAN has a bridging station, designated F9, which is the LAN's connection to the CDL. Figure 11 shows the 10 station ring, while Figure 12 shows the collection platform bridging station. The additional modules in Figure 12 represent four point-to-point transmitters and one receiver for CDL communication.

The command link is modelled as one point-to-point link with a data rate of 200 Kbps. The return link is modelled as four links with different data rates, currently set at 1.53, 3.06, 21.42, and 42.84 Mbps. The links are connected to transmitters/receivers patched into the Logical Link Control (LLC) of the bridging stations.

Jamming is introduced over the point-to-point links as a function of varying bit error rate (BER). Two jamming types are currently implemented: a channel-swept jammer with

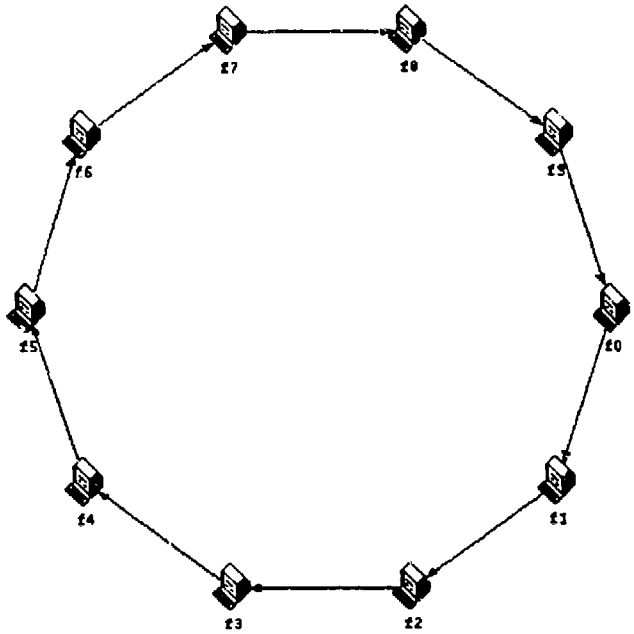


Figure 11: 10 Station FDDI Ring

fixed durations and intervals, and a pulse jammer with random durations and intervals. Both jamming configurations use a random, uniformly distributed BER with maximum values defined for jamming periods and intervals between jamming.

There are also two load balancing algorithms used: round robin, which distributes incoming frames in a cyclic fashion, and empty selection, which puts the current frame on the link with the smallest send queue. Details of both jamming and load balancing algorithms and their implementations may be found in [10].

B. OVERVIEW OF CDL MODEL CHANGES

Changes made to the existing OPNET model are very basic with the intent of maintaining as much previously written code as possible. Actual code modifications are done on only five files in the model:

- collection platform LLC sink process model (*cp_fddi_sink.pr.m*)

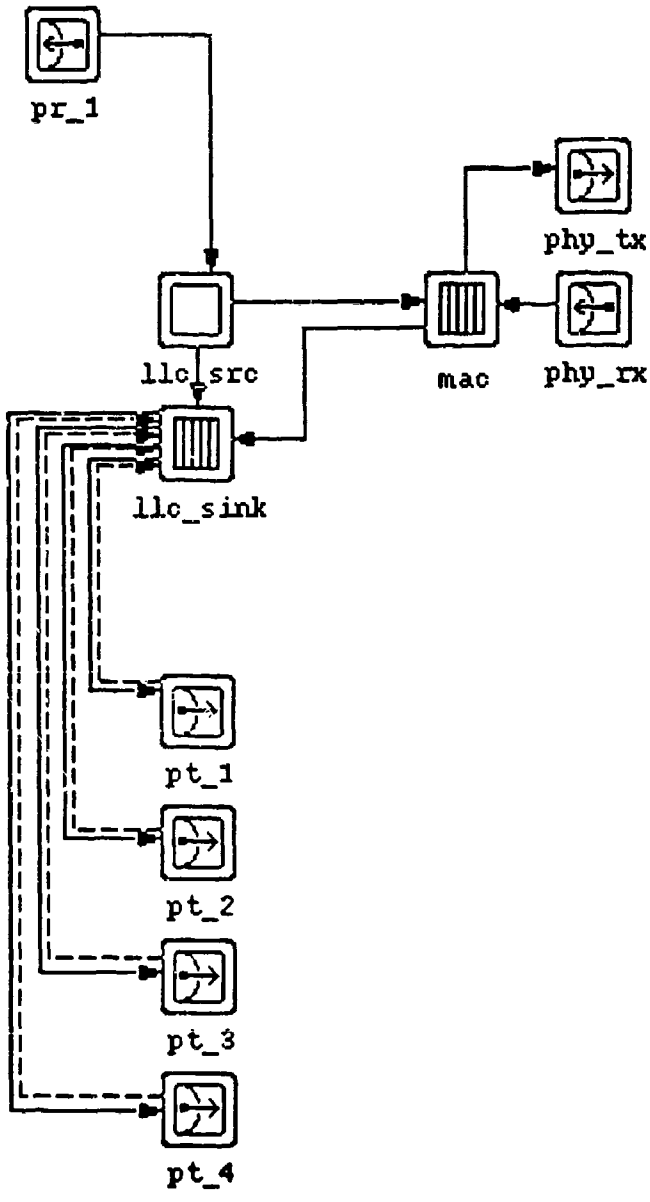


Figure 12: Collection Platform Bridging Station

- collection platform LLC source process model (*cp_fddi_gen.pr.m*)
- surface platform LLC sink process model (*sp_fddi_sink.pr.m*)
- surface platform LLC source process model (*sp_fddi_gen.pr.m*)
- point-to-point error allocation pipeline stage (*cdl_pt_error.ps.c*)

The entire list of changes can be broken down into 3 areas. The first area is data encapsulation, which deals mostly with the new PPP packet formats and their various fields. The link monitoring algorithm is the second area, which caused most of the code changes. This includes not only the model files listed above, but also new attributes in the simulation environment file. Finally, statistic and probe additions were necessary to record relevant data. For completeness, a section is included on how to use this model for experimentation.

C. DATA ENCAPSULATION

Code changes to encapsulate data are minimal. They include creating a formatted PPP packet and setting the PID. Any actions to be taken are based on the assigned PID, and coded using a switch() statement. This structure allows for individual additions of PID options and easy debugging of packet contents.

Two different packet formats are used for this implementation: one for standard PPP frames, and one for Multilink frames. Ideally, one packet should have been sufficient, since both are actually PPP frames. However, the separate packet formats allow for simpler simulation coding, since otherwise each information field would have had to be dynamically created. Figure 13 shows both formats as they are seen in OPNET's Parameter Editor.

1. Standard PPP Format

The standard PPP packet format includes all the basic fields of an HDLC-framed PPP packet. The only editable fields are the high and low bytes of the PID. The "information" field is present for higher level encapsulation, but none is currently used.

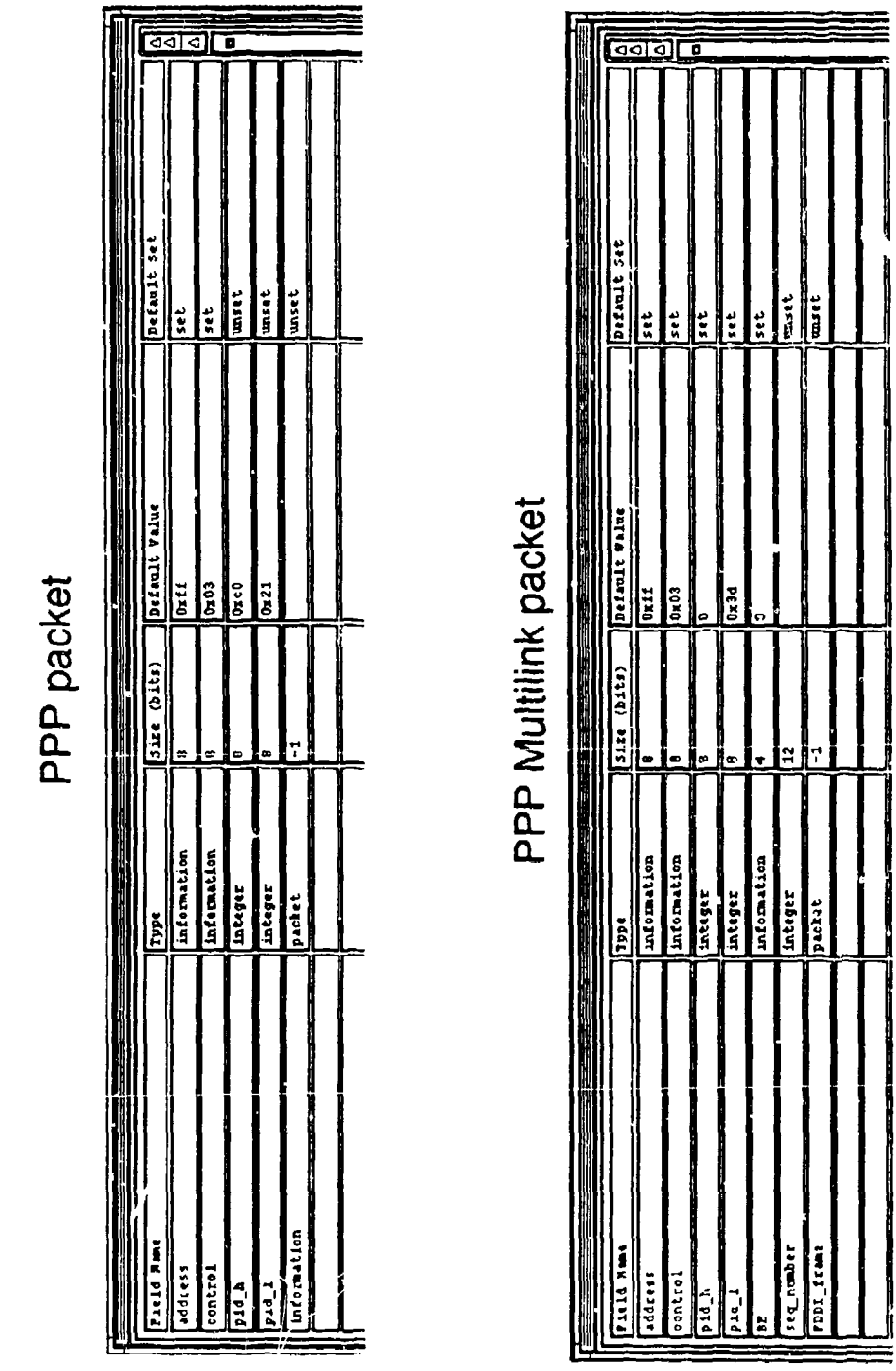


Figure 13: PPP Packet Formats in OPNET

Other than a higher level packet, any information to be transmitted in this type of PPP packet, such as LQR data or other PID-specific fields, must be hard-coded using a kernel procedure, *op_pk_fd_set*. An example can be found in the file *sp_fddi_gen.pr.m*, where the LQR data is added to the PPP packet.

2. PPP Multilink Format

The Multilink packet has the same basic elements as the PPP packet, with the addition of fields specific to a Multilink packet. The overall packet is formatted with the optional Multilink Protocol header using a 12 instead of 24-bit sequence number and a B, E, and filler sequence of 4 bits instead of 8.

The "BE" field is listed as information because a fragmenting procedure has not been implemented with this model. Also, the field name "FDDI_frame" is only for semantics; any frame or datagram can be inserted into this field, as long as it is in an OPNET packet format.

D. LINK MONITORING MODEL

1. Collection Platform (CP) Logical Link Control (LLC) Sink

All FDDI frames destined for the SP come up through the MAC layer to the Logical Link Control (LLC) sink on the bridge station. Also, any frames generated by the bridge station which are destined for the SP also pass to the sink module from the bridge station's own LLC source. These frames are reframed in PPP frames and queued for transmission to the SP bridging station's LLC source based on the load balancing algorithm selected in the environment file.

For monitoring packet generation, the insertion rate (named *link_mon_trans_rate* in the code) is retrieved from the environment file. Upon process initialization, the *'INIT'* state schedules a process self-interrupt at each increment for the duration of the simulation. When this self-interrupt occurs, the *'DISCARD'* state creates a monitoring frame of a fixed size (*mon_pkt_size*), which is read in from the environment file. The frame is then enqueued based on the load balancing algorithm in place for the regular PPP data frames.

The monitoring frame itself is a PPP frame structure with a fixed size assigned via a kernel procedure equal to the desired monitoring frame size. Due to OPNET's error allocation constructs and the methods used in designing the model, there is no need to generate a pseudo-random bit stream as the contents of the frame have no bearing on the number of bits deemed to be in error.

2. Pipeline Error Allocation Stage

In OPNET, each link, whether a point-to-point, bus, or radio link, is defined as a number of stages where certain actions take place. For instance a point-to-point link uses three stages, each represented by a separate C program: propagation delay, error allocation, and error detection. None of these programs is used in the OPNET graphic interface except to assign their names to the stage attributes of the appropriate link in the network, as shown in Figure 14. Also, these programs must be edited through a separate line editor and individually compiled before the overall simulation can be archived and bound.

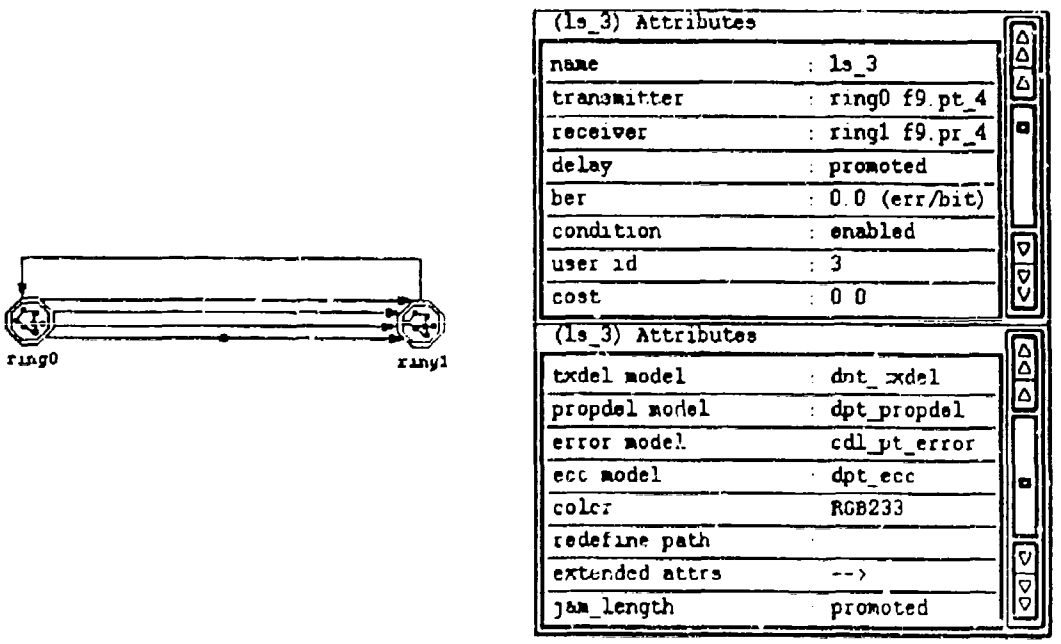


Figure 14: Pipeline Stage Name Setup in OPNET Network Level

Each packet is acted on by these stages, with the resulting values assigned to a fixed set of kernel variables unique to each packet. For instance, each packet has a kernel variable named OPC_TDA_PT_NUM_ERRORS, but the value contained is unique to the packet.

The errors in a packet are calculated using a timestamp, the size of the packet, and the jamming pattern set up in the environment file. [10] details this error allocation procedure. Consequently, this total quantity of errors is put into OPC_TDA_PT_NUM_ERRORS and is carried with the packet until it is read in the surface platform LLC source process.

3. Surface Platform (SP) Logical Link Control (LLC) Source

PPP frames are received at the SP LLC source on all four point-to-point links from the CP and are evaluated based on their PID's high-order byte. Data frames are decapsulated and sent to MAC layer for forwarding, or if the frame is destined for the SP bridging station, it is passed to the LLC sink layer for disposition. Currently, when frames reach the final destination's LLC sink, they are discarded without prejudice.

All monitoring frames have a high-order PID byte of 0xc0. Once a monitoring frame is detected, the low-order PID byte is evaluated to determine what type of monitoring frame it is. Current implementation assumes all LCP frames (low-order PID byte = 0x21) are monitoring packets. The number of errors computed from the point-to-point link is then read and put in the history.

The history is generated in the *'INIT'* state by a function call named create_history. The call requires an integer size, which is obtained from the environment file. A dynamically allocated circular linked list of integers is created, and a pointer is returned to the first element in the list.

When an error value is added to the history, the pointer is incremented to the next element. A counter is then incremented or decremented to reflect if the frame had errors in it. A new error ratio is computed, and it is compared to the ratio when the last LQR was

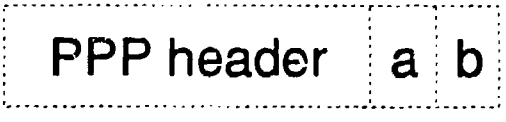
sent. If the difference is greater than the LQR reporting criteria, the link status is reevaluated and an LQR is sent with the latest information.

The link status is coded as one integer value, ranging from 0 to 3, which imparts two characteristics of the link. These characteristics are the actual status (i.e., good or bad), and the trend since the last LQR transmission (i.e., up or down). The integer value reflects the decimal equivalent of a two-bit representation where the status is the high bit and the trend is the low bit. This is shown in Figure 15.

Hysteresis is implemented by comparing the new ratio to each of the hysteresis boundaries. This will only be checked if an LQR is due to be sent, so the hysteresis bounds should be placed at a multiple of the LQR reporting criteria. If either of these boundary conditions have forced a status change, the good/bad status is reset; otherwise, the status bit remains unchanged. In any case, the old trend is then stripped and reset based on the latest comparison. Finally, if an LQR is to be sent, a PPP frame is formatted in the LLC source and sent to the LLC sink for transmission to the collection platform.

4. Surface Platform (SP) Logical Link Control (LLC) Sink

This process, just as the collection platform LLC sink, encapsulates all FDDI frames from the MAC layer and any data from the SP LLC source to be transmitted to the



Bit a: Link Status
0 - Good
1 - Bad
Bit b: Link Trend
0 - Down
1 - Up

Figure 15: LQR Format in OPNET

CP. Since PPP-encapsulated LQRs are sent from the LLC source, a check is introduced to bypass all FDDI data collection and PPP encapsulation if the incoming packet is already encapsulated. All PPP frames are then transmitted to the collection platform.

5. Collection Platform (CP) Logical Link Control (LLC) Source

Once a PPP data frame is received at the CP, it is decapsulated and the data is sent to the MAC layer for further transmission. If the PPP frame is a control frame, it is expected to be an LQR, since there is no need for monitoring the 'unjammable' 200 kbps uplink.

The LQRs contents are evaluated and a message is printed out to the screen giving the link status and trend. Since no policy has been determined for LQR implementation, the frame is then discarded with no action taken.

E. ENVIRONMENT FILE CHANGES

All attribute changes and additions in the environment file, as with the code changes, pertain only to the bridging stations. An example environment file used to generate data can be found in Appendix F. The added attributes include:

- *link_monitor_trans_rate* - the rate at which monitoring frames are inserted into the downlink bit stream (secs/pkt)
- *monitoring pkt size* - the physical size of the monitoring packet (bits)
- *LQR transmission delta* - the ratio change required before an LQR is transmitted
- *upper (lower) hysteresis threshold* - the boundary ratio values which determine if the link is good or bad
- *history length* - the number of monitoring packet values maintained to calculate the ratio

Additionally, receiver *ecc threshold* and transmitter/receiver *data rate* built-in attributes have been promoted to the environment file. For *ecc threshold*, if the value is zero, any incoming packets with errors will be discarded at the receiver, and no monitoring would ever be conducted. As for data rates, promoting these attributes allowed quick changes to determine the effect of transmitting monitoring frames on various data rates.

F. PROBE/ANALYSIS CHANGES

Six new statistics were added to the simulation to monitor simulation performance and ensure proper operation of the link monitoring algorithm. Code changes to implement these statistics are found in the CP LLC sink, the pipeline error allocation stage, and the SP LLC source. Finally, reading the local statistics required additional probes in the simulation's probe file, *fddi_cdl_probe_special.pb.m.*

1. Jamming Tracking

In order to compare the monitoring results with the jamming taking place, an accurate picture of the jamming had to be generated. Since jamming is implemented as a random value within fixed bounds of time and maximum BER, the jamming input parameters could present the basic jamming pattern but not the precise BERs associated with each time slot. Therefore it was necessary to generate a statistic when the jamming is actually taking place in the code. Unfortunately, this action occurs in the pipeline error allocation stage, which can not utilize local statistics as a normal processor module such as a transmitter or queue would.

Another option would have been to send the BER and a timestamp to a processor which can maintain local statistics. However, the only data which can leave a pipeline stage is assigned to the kernel variables described in the previous section, and each of those has a distinct purpose in the operation of the pipeline.

In the end, global statistics are used to maintain the data. These are able to read data generated anywhere in the simulation, but must be initialized by generating a "global statistic handle", which is to be maintained using a state variable. Pipeline stages have no mechanism for declaring state variables, so the handles are created as an array of handles in the *'INIT'* state of the CP LLC sink module and externally declared in the error allocation C program. A global flag is also set when the handles are created to prevent the error allocation program from writing to an undeclared statistic.

Since each of the four pipelines uses the same error allocation stage, a distinction must be made between each pipeline for data collection purposes. The solution was to use the user id, a built-in, user-defined attribute assigned to each pipeline. For simplicity, the user id was assigned, at the network level, to the corresponding global statistic array index value, numbered 0 to 3. Meanwhile, the command link pipeline has a user id of 10, one order higher than any of the return link pipelines. Besides being set in the OPNET network model, the value is assigned as a constant in the error allocation stage for comparison purposes when saving statistical data.

2. Link Status Statistics

In the SP LLC source, two local statistics are added: RATIO_OUTSTAT and LINK_STATUS_OUTSTAT. These names are defined in the header file to correspond to two output statistic values, which in turn are probed to display the bad packet ratio and the link status (good/bad). RATIO_OUTSTAT is the straightforward ratio, while LINK_STATUS_OUTSTAT is scaled to display a 0 or 1 for good or bad, respectively.

3. Additional Changes

Numerous lines of previously created data collection code have been commented out. The purpose of this was to shrink resulting vector files containing as much as 8 MB of data for a 10 second simulation run. It should be noted that global statistics, which account for most of the vector data in this case, are saved in a vector file each time a simulation is run, and do not need to be probed to be saved. To alleviate this, the OPNET modelling manual recommends using a condition flag to decide at runtime whether certain global statistics should be saved.

G. RUNNING EXPERIMENTS

Running experiments requires changes to two files: the environment file to be used, and the script file to be used. Examples of each are shown in Appendices F and G,

respectively. Environment file changes for link monitoring consist of altering attributes listed in section D of Chapter V.

The purpose of the script file is to allow the user to run the simulation without using the OPNET graphical interface. This frees up valuable memory, and allows for background execution. It is recommended that only one simulation be run at a time because simulations running concurrently compete for resources, and this can lead to both simulations crashing with no useful data to show for the 30 minutes of execution time. However, the script file does allow for successive executions without user interaction.

The script file in Appendix G is set up to run the same simulation executable, *fddi_cdl.sim*, four times. On each run, a different environment file is used, and the results are saved in a different vector file. The probe file and simulation duration are also specified on the command line. The shell environment variable, debug, can be used by the OPNET simulations to invoke the OPNET on-line debugger, but as this requires user interaction, it defeats the purpose of the script file.

Lastly, there are two other requirements to ensure a smoothly running script file. First, the .sim file is an executable, so when executing the script, the .sim file must be in the current directory, or the entire path must precede the name of the executable in the script file. Secondly, an up-to-date env_db file must be present in $HOME/op_admin. If this file is not found, the simulation will not run at all.

V. MODEL EVALUATION

A. OVERVIEW

This chapter provides test runs to demonstrate the reliability of the OPNET computer model. The tests show a changing BER, the resulting monitoring packet error ratio, and the resulting link status changes. Additionally, parameters are varied to demonstrate the effects of history, hysteresis, and insertion rate on the overall link status.

B. TESTING SCENARIO

A baseline model was developed for the CDL in [10]. This model was used as the testbed for link monitoring testing. A minor change was required to reevaluate the load balancing algorithms, as well as adding a situation where all monitoring packets are directed to one specific multiplexer input. Otherwise, the original model code remained intact.

Tests were only conducted on the return link, since monitoring is only needed on that link. For these tests, most attributes listed in the simulation environment file were retained from the baseline model. Any additionally required attributes were promoted to the environment file. In summary, any attribute which was varied for simulation purposes was put in the environment file.

The 11 variable environment attributes listed in Section IV.E, as well as the two load balancing algorithms and the two jamming techniques, led to 28 different model runs to ensure reliability. For a basic reliability test, a periodic jamming pulse is introduced over all four return link point-to-point links. Figure 16 shows the jamming pulse over one of the links, however the jamming is identical over all four links. The duration of each pulse is long enough to ensure that it would be detected by the monitoring system regardless of the parameter settings. Figures 17 and 18 represent the monitoring output in the form of the bad packet ratio over all four links, and the resulting link status (0-good, 1-bad). Clearly, these figures show that the basic model operates as expected.

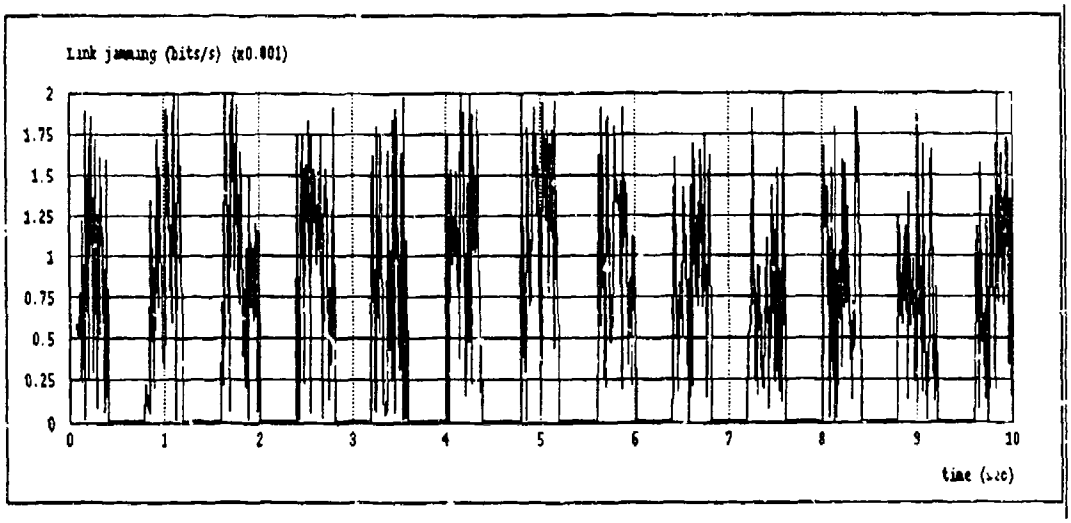


Figure 16: Jamming Introduced for Basic Monitoring Test

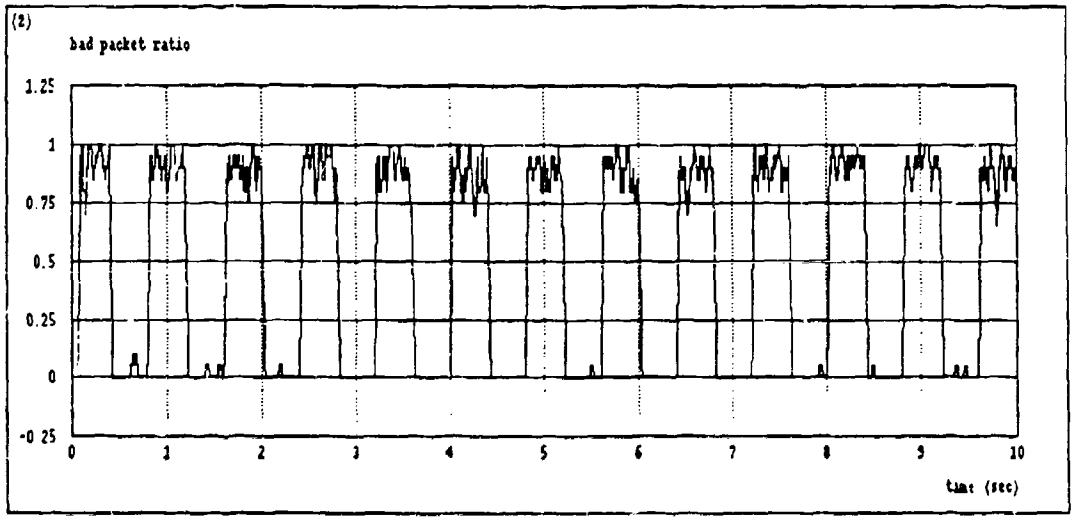


Figure 17: Bad Packet Ratio for Basic Monitoring Test

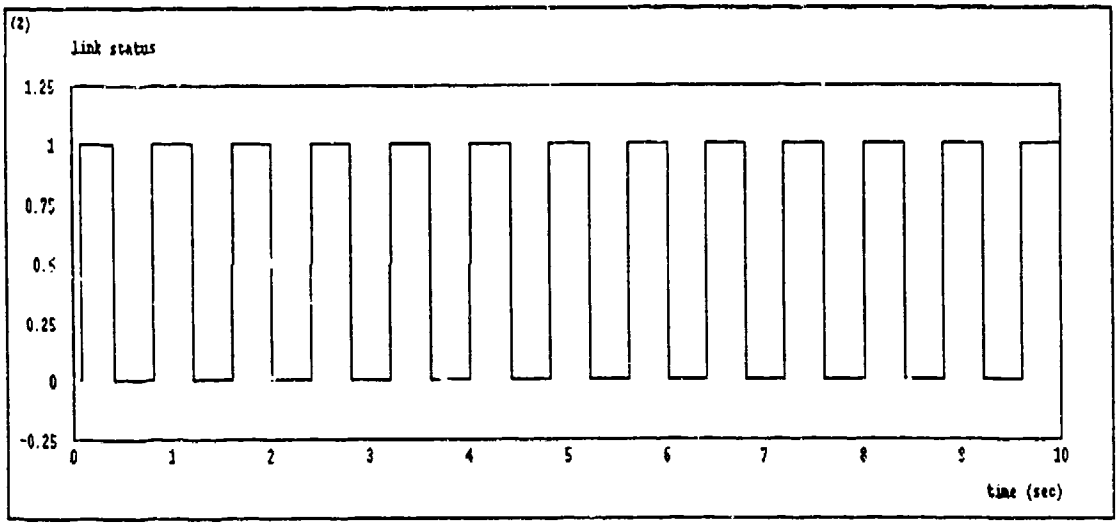


Figure 18: Link Status for Basic Monitoring Test

C. PARAMETER VARIATION EFFECTS

Subsequent tests now need to verify that trends in certain attributes affect the model as expected. This is accomplished by varying only one attribute for 3-4 trials and comparing the results. For purposes of discussion, one set of data is presented to support each case; however, numerous data sets were evaluated to ensure reliable model performance.

1. Insertion Rate

As previously discussed, insertion rate has a tremendous impact on the resulting bad packet ratio, and, therefore, the overall monitoring scheme. It is assumed that insertion rate is directly proportional to the change in bad packet ratio: the faster the insertion rate, the quicker a change may occur in the ratio. To demonstrate that idea, four runs are shown in Figures 19 through 22. Each of these examples are exactly the same except that the insertion rate is decreased by a factor of 10 in each subsequent graph.

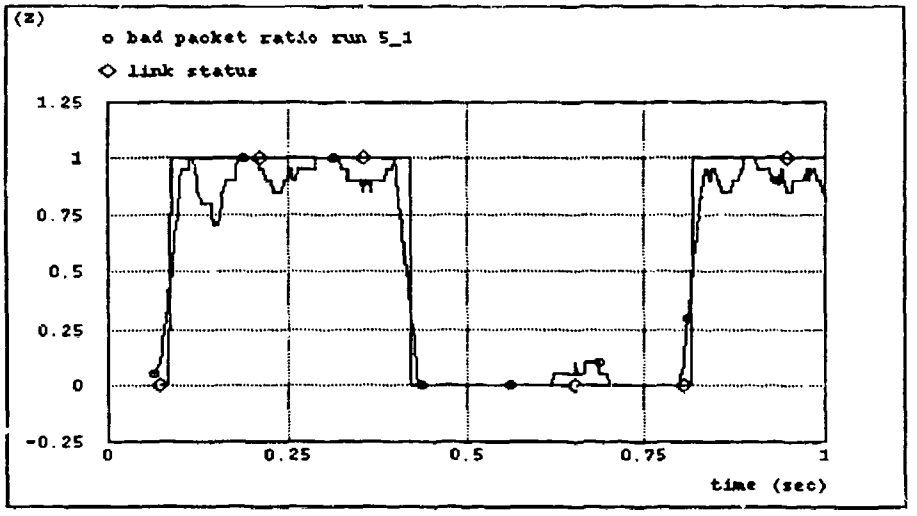


Figure 19: Insertion Period of 729 Microseconds

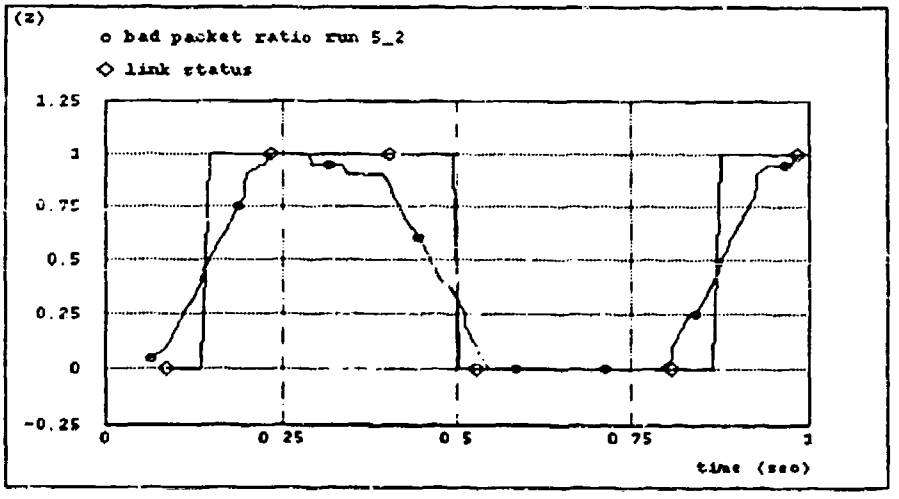


Figure 20: Insertion Period of 7.29 Milliseconds (ms)

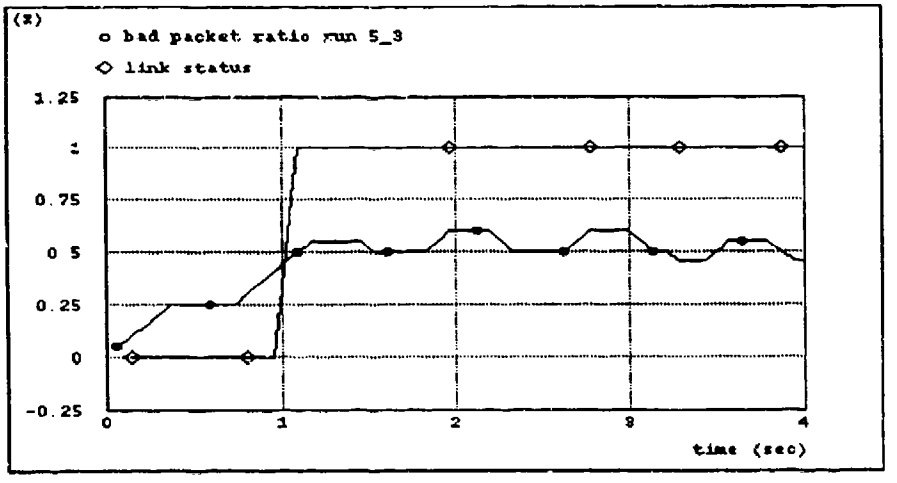


Figure 21: Insertion Period of 72.9 ms

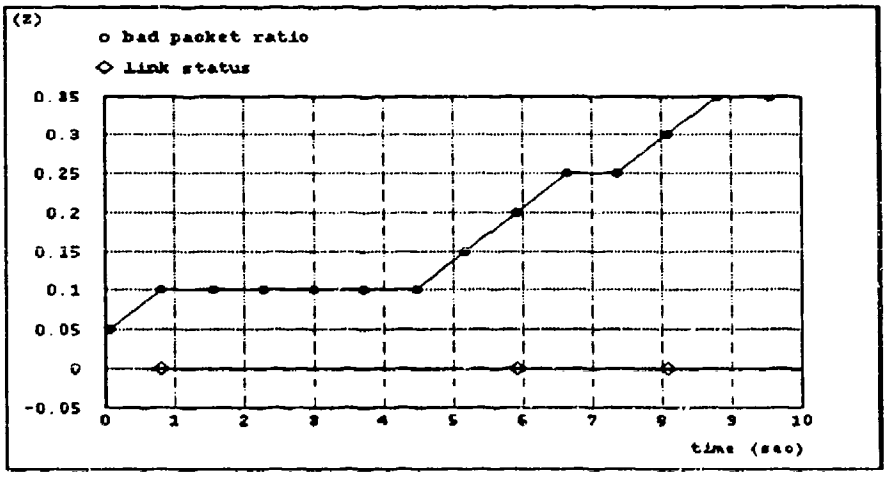


Figure 22: Insertion Period of 729 ms

Figure 4 has an insertion rate of 1 monitoring frame every 729 microseconds. Assuming full utilization of the aggregate bandwidth, that equals one 5,000-bit monitoring frame for every 10 20,000-bit data frames, or an overhead of approximately 2.5%. Likewise, Figure 5 shows an insertion rate of 7.29 milliseconds, equalling .25% overhead, and so on.

The jamming pulse used here cycles every 0.8 seconds with a 0.4 second pulse duration. Clearly Figure 19, with the highest insertion rate, has the best picture of the jamming. In Figure 20, slower response is evident due to the lower insertion rate, yet a decent picture is maintained. However, Figures 21 and 22 show increasingly severe cases of undersampling, where the monitoring frame insertion rate is so slow that the ratio can not accurately reflect the jamming. This conclusively shows the expected results of decreasing insertion rate, verifying the model for this attribute.

2. History Length

It is assumed that the history length is inversely proportional to the ratio rate of change: the longer the history, the less effect each monitoring frame has on the overall bad packet ratio. This fact can be verified by simple calculation, while the graphical representations shown in Figures 23, 24, and 25 show two characteristics related to increasing history length.

Figure 23 is a scenario with a history length of 20, while Figures 24 and 25 show the same test run with history lengths of 30 and 40, respectively. The jamming is different over each of the four links, to simulate a widely varying jammer, while the monitoring packets are inserted in a round robin fashion.

The first point of interest is the expected decrease in the rate of change in the ratio. At the same point in time, each successive graph takes slightly longer to register a drop in the jamming. This is a direct result of increasing history length.

Of more significance is the fact that the rate of change for each successive run is only *slightly* lower than the previous run. One may assume the rate of ratio change should

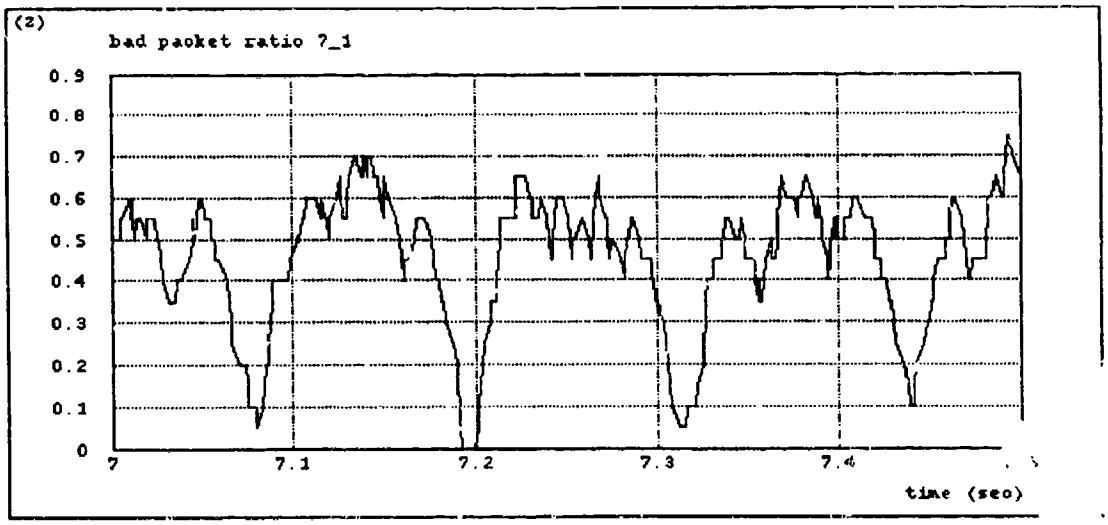


Figure 23: History of Length 20

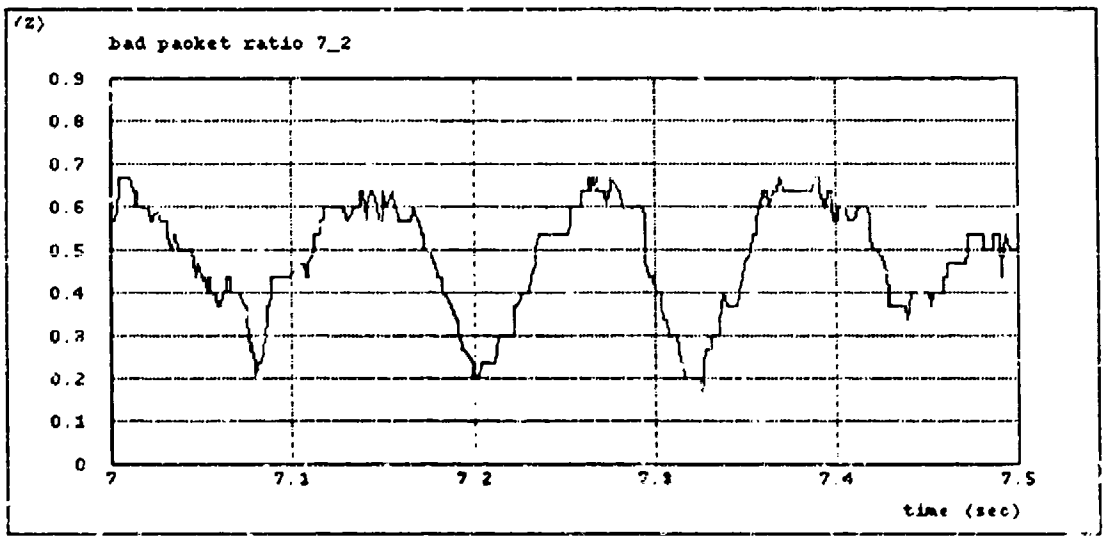


Figure 24: History of Length 30

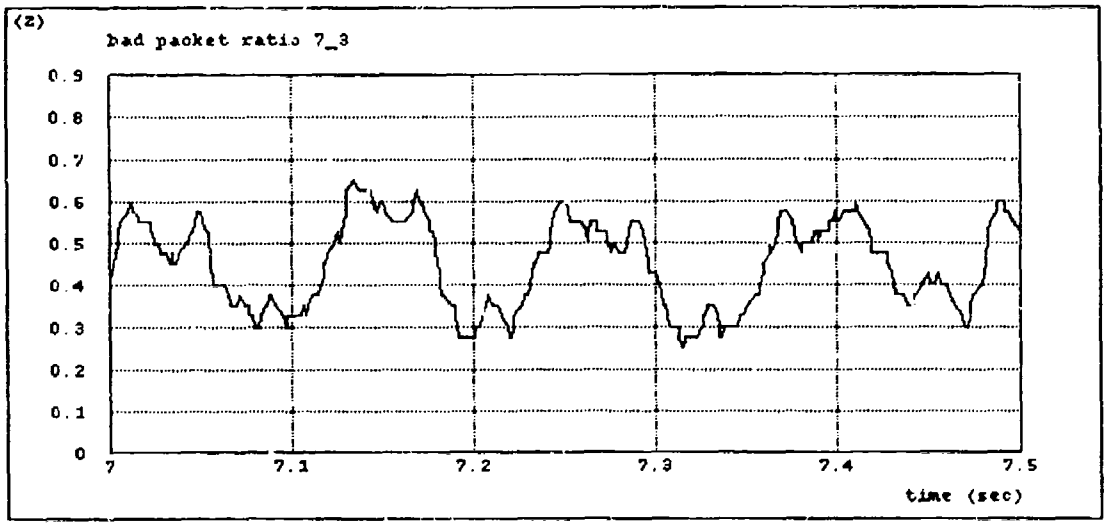


Figure 25: History of Length 40

be more significant, and it would, except that the increased history length brings the extreme values (high and low) closer. As a result, the packet ratio's trend changes more quickly than if the peaks were the same as a run with a smaller history. Although this is a benefit of longer histories, the closing of the peak values causes a significant impact on hysteresis and the determination of appropriate hysteresis bounds.

In any case, it can be seen that the model accurately represents the effects of changing history length.

3. Hysteresis Bounds

The implementation of hysteresis is a simple comparison of the current bad packet ratio to bounds input into the simulation through the environment file. While that is not significant and can be visually evaluated, what is important is the effect hysteresis has on link status.

In a simple scenario, where jamming is fixed and periodic (not highly likely), hysteresis would have very little impact. Figure 26 which illustrates such a situation, shows

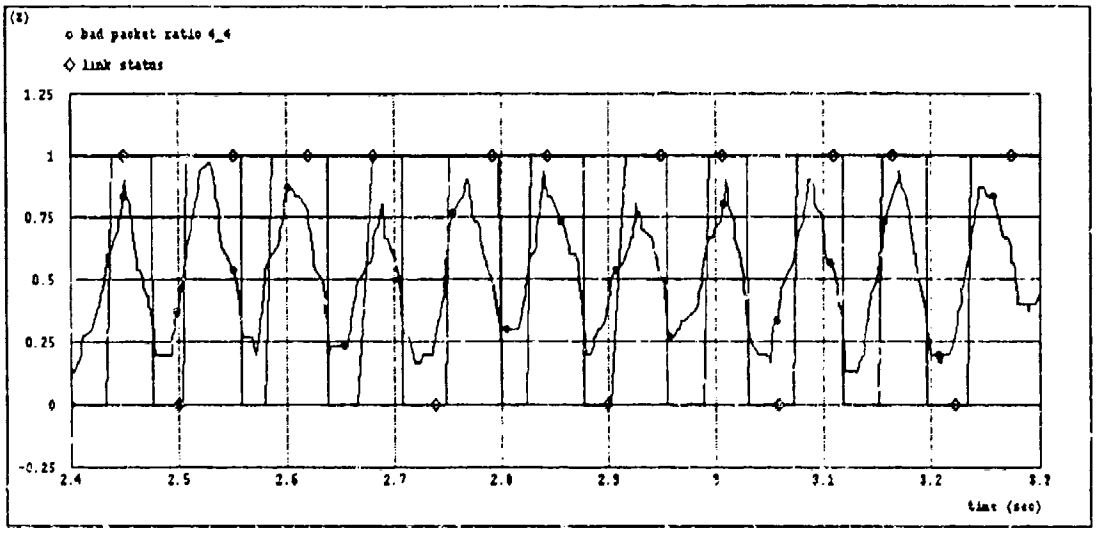


Figure 26: Hysteresis Used in Conjunction With Periodic Jamming

that periodic jamming is accurately reflected in the bad packet ratio. Although this test run was conducted with hysteresis bounds of 0.5 and 0.3, it is easy to see that replacing the hysteresis with a single pivot value higher than 0.3 would actually improve the amount of time that the link would be declared good, as well as eliminate the need for any hysteresis.

On the other hand, non-periodic jamming, as Figure27 shows, does not lend itself to a single pivot value. Here it can be seen that without hysteresis, but with a single pivot value set at 0.4, excessive oscillation would occur that could affect equipment performance and long term wear on circuitry.

D. SUMMARY OF RESULTS

Each of the previous examples shows that the OPNET computer model accurately converts changes in BER into a link monitoring ratio. In turn, this ratio is properly

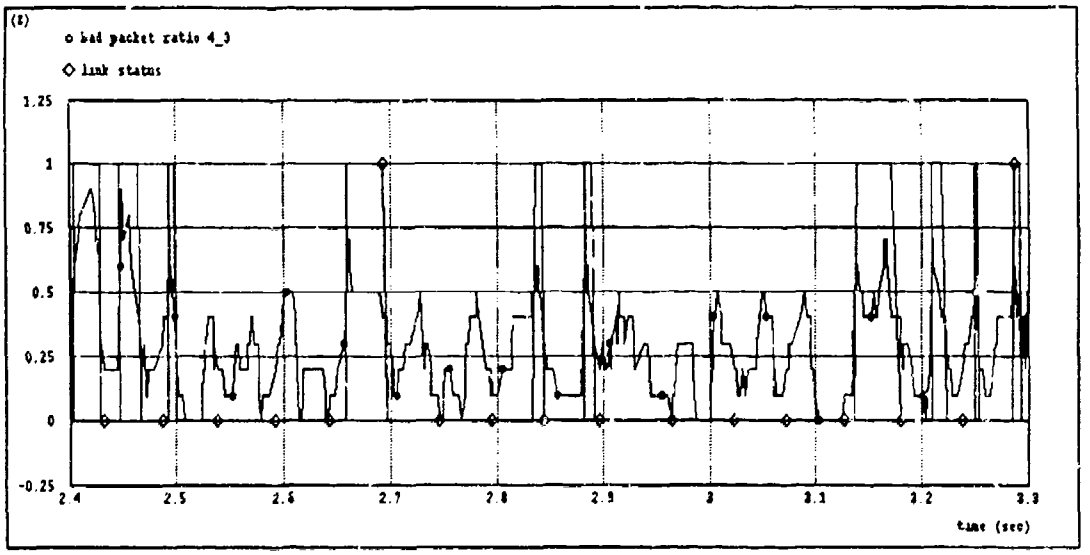


Figure 27: Hysteresis Used in Conjunction With Non-periodic Jamming

evaluated in a manner consistent with the proposed link monitoring mechanism. More specifically, significant characteristics of the modelled link monitoring mechanism behave as expected. Put together, these factors lead to the conclusion that the link monitoring system, as implemented in conjunction with the baseline OPNET CDL model, accurately reflects the link monitoring design presented in Section III of this thesis.

VI. OPNET TCP/IP MODELLING

A. PURPOSE/SCOPE

The long term goal of the OPNET network model is to integrate it with a working network, transport, and application layer in order to better measure user-to-user performance over the CDL. TCP/IP has been chosen for the model because of its widespread implementation. MIL 3 supplies a baseline TCP/IP model within an Internet model, but the TCP model is very simple in that it is based on the original 1981 TCP standard and has not integrated any of the improvements used in present-day TCP suites. These improvements include Karn's algorithm, Jacobsen header compression, and round-trip time estimation, among others.

The work for this portion of the project is the first step towards full-scale integration of the CDL network model with TCP/IP. Additional changes are recommended to enhance the TCP model and change the application models into CDL-specific applications.

The scope of this section is limited to OPNET code and model changes designed to integrate the CDL network model with the OPNET generic Internet model. No evaluations are conducted except to verify that the model runs correctly.

B. SUMMARY OF CHANGES

The changes to integrate the two models revolve entirely around communications between layers, primarily the network protocol (IP) and the data link protocol (FDDI LLC/ MAC layers). Since each model is self-contained, and substantial changes have been made to the network model, additional changes must be made to both models to work together properly.

For instance, the basic Internet model station's IP layer is designed to communicate directly with the OPNET's basic FDDI MAC layer, as shown in Figure 28, while the integrated basic station model calls for an LLC layer between them as well as a substantially different MAC layer, shown in Figure 29. On the other hand, the CDL

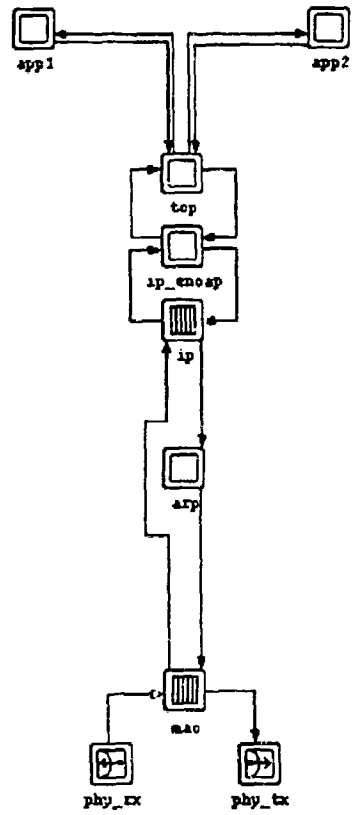


Figure 28: Basic OPNET Internet Station Model

network station model, previously shown in Figure 10, has no layers above the LLC, so all traffic is created and destroyed in the LLC instead of at the application layer.

To alleviate communications problems, the OPNET data packets and Interface Control Information packets (ICIs) must be revamped to communicate correct and relevant data between the LLC and the MAC. Also, code must be introduced into the pertinent processor modules to enable usage of the new data packets and ICIs.

C. PACKET/ICI FORMAT CHANGES

In OPNET, data communication between any two processors is done by sending a packet and/or ICI to an adjoining processor through a stream. A packet is used to model the flow of actual data, while an ICI is used to model control information between layers.

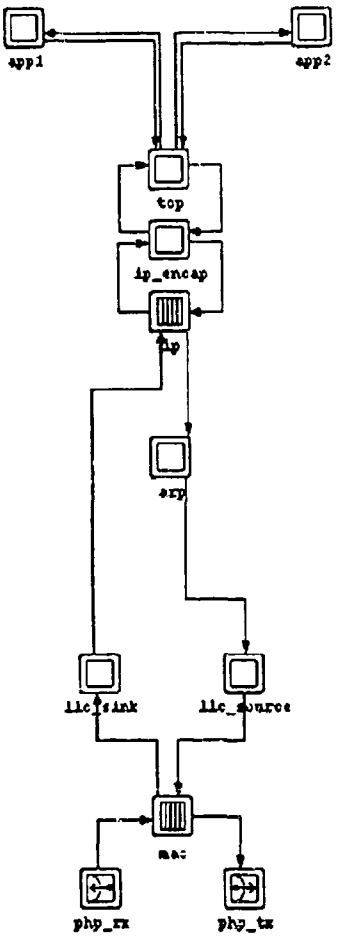
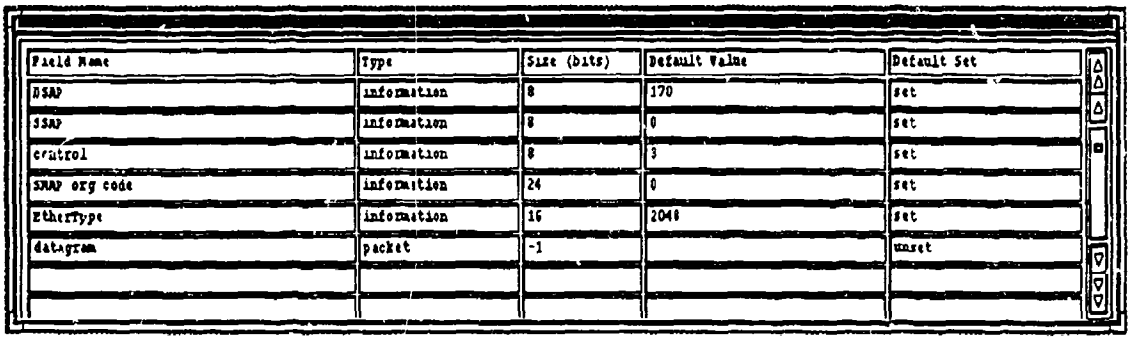


Figure 29: Integrated CDL-TCP Basic Station Model

OPNET allows direct packet communication between any two processors anywhere in the model, bypassing the streams, but utilizing this feature violates the model's integrity and is discouraged.

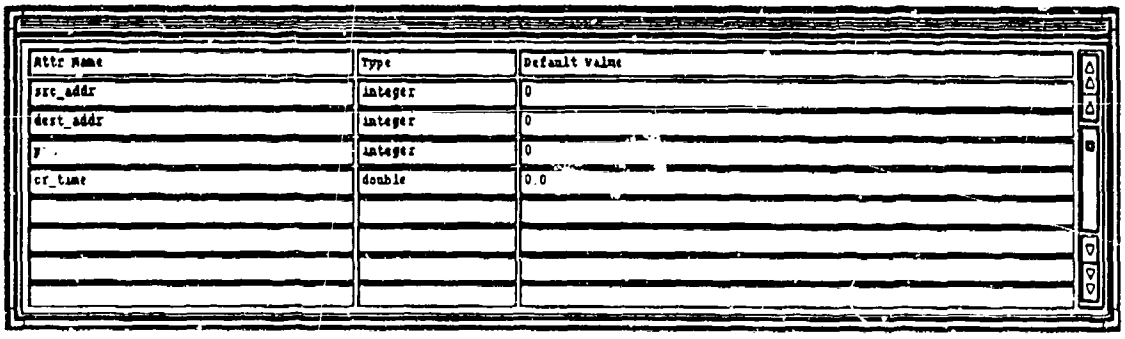
The first area of interest is the stream connecting the arp module to the llc_source. Originally, the arp was designed to communicate directly with the MAC layer using a packet, *ip_dgram*, and an ICI, *ip_mac_req*. These formats are satisfactory and are left intact, forcing the *llc_source* module to conform to the new inputs.

The llc_source module used in the network CDL model transmits a packet using the *fddi_llc_fr* format, and an ICI using the *fddi_mac_req* format to the MAC layer. However, this packet did not conform to standards put forth in [11]. So, the packet was modified to contain the appropriate information, while all other data, still necessary to be passed to the MAC layer, was placed in the ICI. Figures 30 and 31 show the new packet, *fddi_llc_fr_tcp*, and ICI, *fddi_mac_req_tcp*.



Field Name	Type	Size (bits)	Default Value	Default Set
DSAP	information	8	170	set
SSAP	information	8	0	set
control	information	8	3	set
SNAP org code	information	24	0	set
EtherType	information	16	2048	set
datagram	packet	-1		unset

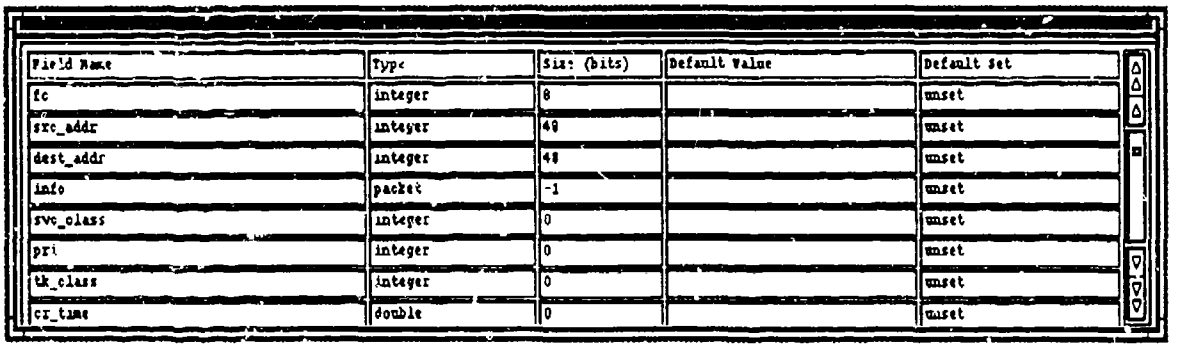
Figure 30: LLC Frame Format *fddi_llc_fr_tcp*



Attr Name	Type	Default value
src_addr	integer	0
dest_addr	integer	0
[illegible]	integer	0
cr_time	double	0.0

Figure 31: LLC Source to MAC ICI Format *fddi_mac_req_tcp*

Inside the MAC and physical layers, an OPNET packet is passed from station to station without an ICI. However, data not included in the MAC frame header, needed for FDDI performance calculations, must be transmitted with the data to be collected at the final destination. Normally, this information would be placed in an ICI, which in turn would be attached to the MAC frame. In this case, though, since there were only two pieces of data, the information was simply appended to the MAC frame format, as shown in Figure 32. It is important to note that the fields in question, pri and cr_time, have a size of 0 bits.



Field Name	Type	Size (bits)	Default Value	Default Set
fc	integer	8		unset
src_addr	integer	48		unset
dest_addr	integer	48		unset
info	packet	-1		unset
svc_class	integer	0		unset
pri	integer	0		unset
tk_class	integer	0		unset
cr_time	double	0		unset

Figure 32: FDDI MAC Frame Format *fddi_mac_fr_tcp*

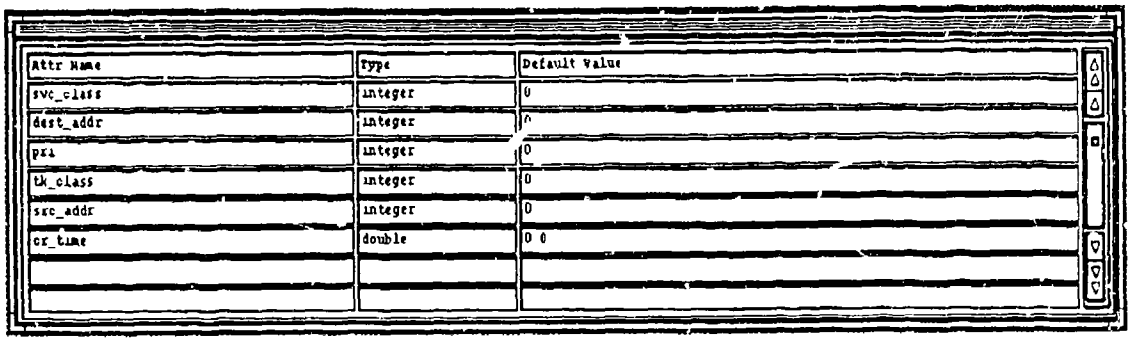
This is so the contents of those fields will not be counted towards the total FDDI bits transmitted, transmission delay, or error calculations.

Up from the MAC to the LLC sink, the same packet format is used as in Figure 30, but a different ICI is used. The new ICI format, *fddi_mac_ind_tcp*, (shown in Figure 33), is used because most of the fields passed in *fddi_mac_req_tcp* pertain to the FDDI token and type of service. Most of this information is unnecessary above the MAC layer, although the *pri* and *cr_time* fields are still required for data collection.

D. CODING CHANGES

Changes were made to the following CDL network model files:

fddi_cdl.nt.m --> fddi_cdl_tcp2.nt.m



Attr Name	Type	Default Value
svc_class	integer	0
dest_addr	integer	0
pri	integer	0
tk_class	integer	0
src_addr	integer	0
cr_time	double	0.0

Figure 33: MAC to LLC Sink ICI Format *fddi_mac_ind_tcp*

- *fddi_station.nd.m --> inet_fddinix.nd.m*
- *cpni.nd.m --> cpni_tcp.nd.m*
- *spni.nd.m --> spni_tcp.nd.m*
- *cp_fddi_gen.pr.m --> cp_fddi_gen_tcp.pr.m*
- *cp_fddi_sink.pr.m --> cp_fddi_sink_tcp.pr.m*
- *sp_fddi_gen.pr.m --> sp_fddi_gen_tcp.pr.m*
- *sp_fddi_sink.pr.m --> sp_fddi_sink_tcp.pr.m*
- *fddi_mac.pr.m --> fddi_mac_stat5.pr.m*
- *cp_fddi_mac.pr.m --> cp_fddi_mac_tcp.pr.m*
- *sp_fddi_mac.pr.m --> sp_fddi_mac_tcp.pr.m*

The network level changes consisted of new icons for each of the rings, and a different icon for each LAN station, as illustrated in Figures 34 and 35. Each non-bridging station in an *inet_fddinix* node model as shown in Figure 29. The CP and SP bridging stations are shown in Figures 36 and 37, respectively.

All coding changes for this model addition were done to implement the packet and ICI changes delineated in the previous section and are explicitly commented in the appendices. The only noteworthy changes occurred in the bridging stations.

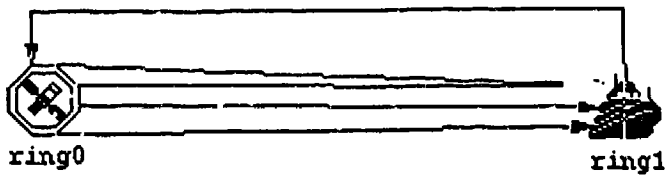


Figure 34: TCP-enhanced CDL Network Model

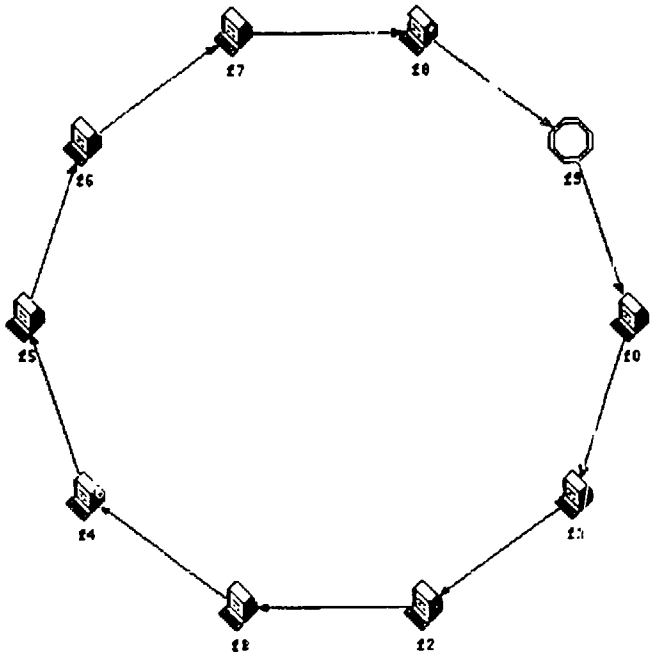


Figure 35: TCP-enhanced FDDI Ring

The bridging stations are unlike the basic ring stations in that they also use PPP. This means that the LLC source and sink on each station may be handling PPP frames, IP

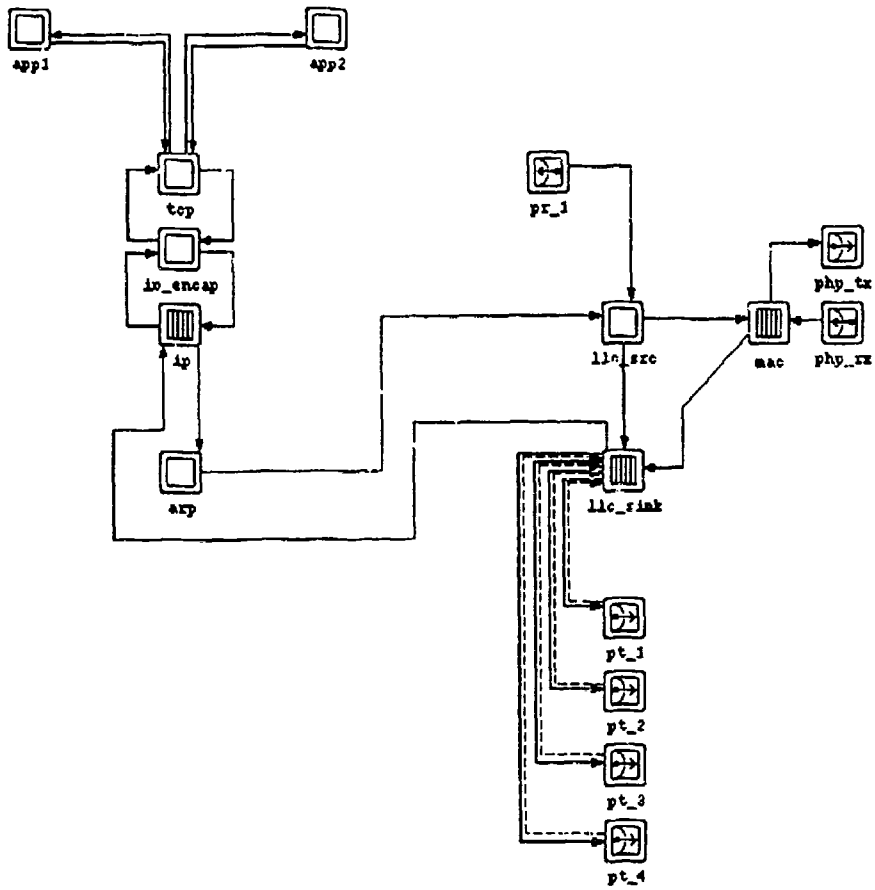


Figure 36: TCP-enhanced CP Bridging Station

datagrams, and LLC frames simultaneously. Furthermore, the added stream between the LLC source and sink requires special ICI formatting considerations.

Many different situations arise because of the multiple formats entering and exiting the LLC. At the LLC source, each PPP frame could be a control frame, an LLC frame for the local LAN, or an encapsulated IP datagram for the bridging station. Likewise, LLC frames arriving at the LLC sink could be destined for the remote LAN or the higher layers of the bridging station.

In the CDL network model, any LLC frame coming from the MAC uses *fddi_mac_ind* as its ICI format. However, from the LLC source to the sink, the ICI used is *fddi_mac_req*.

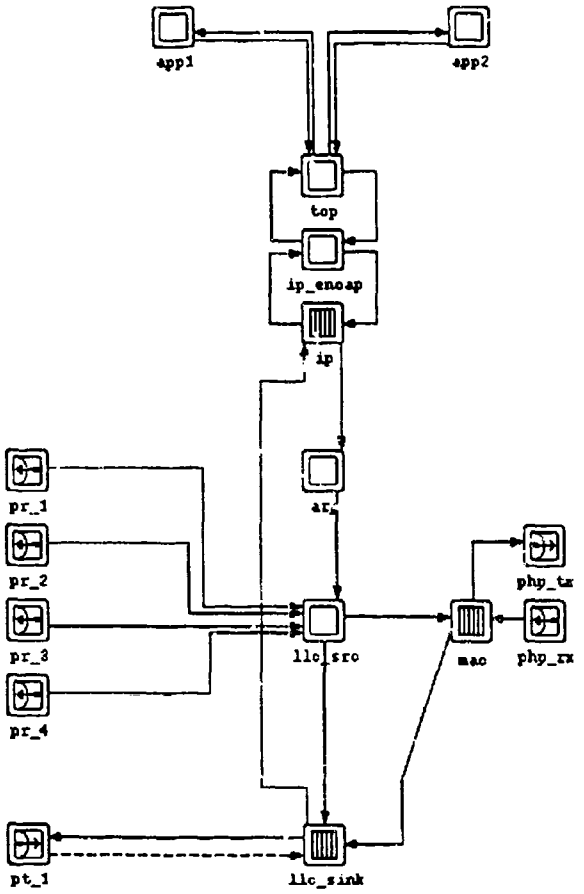


Figure 37: TCP-enhanced SP Bridging Station

This conflict is solved by new fields already added to the *fddi_mac_ind_tcp* ICI. Due to changes already in place because of the new LLC frame format (Figure 30), all ICIs going into the LLC sink are identical, eliminating a significant portion of code.

Another problem with the new LLC frame format can be seen as frames are transmitted over the CDL. As can be seen in Figure 3, once an LLC frame is transmitted to the remote LAN, there is no data field directly accessible to state the ultimate destination. An ICI would be beneficial, but the ICI would be tied to the transmission of the PPP-encapsulated LLC frame, and would have to be retrieved at the remote bridging station,

reinstalled to the decapsulated LLC frame, and then sent to the destination address. This is feasible, but cumbersome.

A solution is found using the Kernel Procedure op_pk_ici_set. This procedure ties the ICI to the packet rather than the transmission event. Therefore, an fddi_mac_req_tcp ICI can be installed to the LLC frame, encapsulated in PPP, transmitted, decapsulated, and sent to the LLC source layer. There it can finally be accessed for proper routing of the LLC frame.

E. CONCLUSION

The TCP/IP changes are very basic, yet they involve every facet of the LLC source, LLC sink, and MAC layers. The changes are small and spread out throughout the code, but well documented. It is recommended that this model be used as the primary building block to achieve full TCP/IP functionality in the CDL network model.

VII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

In this thesis, we have presented and modelled a link monitoring mechanism for the CDL. The effectiveness of this mechanism is measured by normal test runs, and by varying critical monitoring parameters in order to demonstrate expected results. Additionally, changes are presented to integrate OPNET's TCP/IP model with the existing CDL network model.

Specifically, the contributions made by this thesis are as follows:

1) We have identified the critical parameters relevant to the CDL status monitoring.
2) We have successfully shown their impact by implementing a monitoring mechanism in OPNET.
3) We have identified how the monitoring mechanism can be fitted into the PPP and CDL network interface framework.
4) We have integrated OPNET's TCP/IP stack into the network interface and LAN station models.
5) This thesis has also completed the gradual development of a remote LAN interconnection model in OPNET, setting the stage for detailed experimentation.

B. RECOMMENDATIONS

The link monitoring mechanism and TCP/IP framework presents a number of possible research avenues:

1) Evaluation of HDLC bit-stuffing overhead over a multiplexed microwave link, such as the CDL.
2) Experimentation with the link monitoring parameters to determine their optimal settings for dealing with potential jammers.
3) Development of CDL-specific applications to attach to the OPNET computer model for further evaluation.

4) Evaluation of CP output buffer size and implementation in conjunction with jamming and real-time data.

APPENDIX A

RING 0 LLC_SRC MODULE CODE "cp_fddi_gen.pr.c"

```
/* Process model C form file: cp_fddi_gen.pr.c */
/* Portions of this file Copyright (C) MIL 3, Inc. 1992 */



/* OPNET system definitions */
#include <opnet.h>
#include "cp_fddi_gen.pr.h"
FSM_EXT_DECS



/* Header block */
#define MAC_LAYER_OUT_STREAM        0
#define LLC_SINK_OUT_STREAM         1 /*18APR94*/

/* define possible service classes for frames */
#define FDDI_SVC_ASYNC              0
#define FDDI_SVC_SYNC               1

/* define token classes */
#define FDDI_TK_NONRESTRICTED       0
#define FDDI_TK_RESTRICTED          1



/* State variable definitions */
typedef struct
        {
        FSM_SYS_STATE
        Distribution*           sv_inter_dist_ptr;
        Distribution*           sv_len_dist_ptr;
        Distribution*           sv_dest_dist_ptr;
        Distribution*           sv_pkt_priority_ptr;
        Objid                   sv_mac_objid;
```

```
        Objid                   sv_my_id;
        int                     sv_low_dest_addr;
        int                     sv_high_dest_addr;
        int                     sv_station_addr;
        int                     sv_src_addr;
        int                     sv_low_pkt_priority;
        int                     sv_high_pkt_priority;
        double                  sv_arrival_rate;
        double                  sv_mean_pk_len;
        double                  sv_async_mix;
        Ici*                    sv_mac_iciptr;
        Ici*                    sv_mac_iciptr1;
        Ici*                    sv_llc_ici_ptr;
        Packet*                 sv_pkptr1;
        } cp_fddi_gen_state;

#define pr_state_ptr            ((cp_fddi_gen_state*) SimI_Mod_State_Ptr)
#define inter_dist_ptr          pr_state_ptr->sv_inter_dist_ptr
#define len_dist_ptr            pr_state_ptr->sv_len_dist_ptr
#define dest_dist_ptr           pr_state_ptr->sv_dest_dist_ptr
#define pkt_priority_ptr        pr_state_ptr->sv_pkt_priority_ptr
#define mac_objid               pr_state_ptr->sv_mac_objid
#define my_id                   pr_state_ptr->sv_my_id
#define low_dest_addr           pr_state_ptr->sv_low_dest_addr
#define high_dest_addr          pr_state_ptr->sv_high_dest_addr
#define station_addr            pr_state_ptr->sv_station_addr
#define src_addr                pr_state_ptr->sv_src_addr
#define low_pkt_priority        pr_state_ptr->sv_low_pkt_priority
#define high_pkt_priority       pr_state_ptr->sv_high_pkt_priority
#define arrival_rate            pr_state_ptr->sv_arrival_rate
#define mean_pk_len             pr_state_ptr->sv_mean_pk_len
#define async_mix               pr_state_ptr->sv_async_mix
#define mac_iciptr              pr_state_ptr->sv_mac_iciptr
#define mac_iciptr1             pr_state_ptr->sv_mac_iciptr1
#define llc_ici_ptr             pr_state_ptr->sv_llc_ici_ptr
#define pkptr1                  pr_state_ptr->sv_pkptr1



/* Process model interrupt handling procedure */


void
cp_fddi_gen ()
        {
```

```
Packet  *pkptr, *ppp_pkptr;
int     pklen;
int     dest_addr;
int     i, restricted;
int     pkt_prio;
int     ppp_pid_h, ppp_pid_l;
int     status;


FSM_ENTER (cp_fddi_gen)

FSM_BLOCK_SWITCH
        {
        /*---------------------------------------------------------*/
        /** state (INIT) enter executives **/
        FSM_STATE_ENTER_UNFORCED (0, state0_enter_exec, "INIT")
        {
        /* determine id of own processor to use in finding attrs */
        my_id = op_id_self ();

        /* determine address range for uniform desination assignment */
        op_ima_obj_attr_get (my_id, "low dest address", &low_dest_addr);
        op_ima_obj_attr_get (my_id, "high dest address", &high_dest_addr);

        /* determine object id of connected 'mac' layer process */
        mac_objid = op_topo_assoc (my_id, OPC_TOPO_ASSOC_OUT,
        OPC_OBJMTYPE_MODULE, MAC_LAYER_OUT_STREAM);

        /* determine the address assigned to it */
        /* which is also the address of this station */
        op_ima_obj_attr_get (mac_objid, "station_address", &station_addr);



         /* set up a distribution for generation of addresses */
        dest_dist_ptr = op_dist_load ("uniform_int", low_dest_addr,
         high_dest_addr);

        /* added 26DEC93 */
        /* determine priority range for uniform traffic generation */
        op_ima_obj_attr_get (my_id, "high pkt priority", &high_pkt_priority);
        op_ima_obj_attr_get (my_id, "low pkt priority", &low_pkt_priority);
```

```
/* set up a distribution for generation of priorities */
pkt_priority_ptr = op_dist_load ("uniform_int", low_pkt_priority,
high_pkt_priority);

/* above added 26DEC93 */


/* also determine the arrival rate for packet generation */
op_ima_obj_attr_get (my_id, "arrival rate", &arrival_rate);

/* determine the mix of asynchronous and synchronous */
/* traffic. This is expressed as the proportion of */
/* asynchronous traffic. i.e a value of 1.0 indicates */
/* that all the produced traffic shall be asynchronous. */
op_ima_obj_attr_get (my_id, "async_mix", &async_mix);

/* set up a distribution for arrival generations */
if (arrival_rate != 0.0)
{
/* arrivals are exponentially distributed, with given mean */
inter_dist_ptr = op_dist_load ("constant", 1.0 / arrival_rate, 0.0);

/* determine the distribution for packet size */
op_ima_obj_attr_get (my_id, "mean pk length", &mean_pk_len);

/* set up corresponding distribution */
len_dist_ptr = op_dist_load ("constant", mean_pk_len, 0.0);

/* designate the time of first arrival */
fddi_gen_schedule ();

/* set up an interface control information (ICI) structure */
/* to communicate parameters to the mac layer process */
/* (it is more efficient to set one up now and keep it */
/* as a state variable than to allocate one on each packet xfer) */
mac_iciptr = op_ici_create ("fddi_mac_req");
}

}


/** blocking after enter executives of unforced state. **/
FSM_EXIT (1,cp_fddi_gen)
```

```
/** state (INIT) exit executives **/
FSM_STATE_EXIT_UNFORCED (0, state0_exit_exec, "INIT")
{
}


/** state (INIT) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
/*---------------------------------------------------------*/



/** state (ARRIVAL) enter executives **/
FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "ARRIVAL")
{
/* This station should receive frames from the other lan as long as */
/* there are frames in the input streams addressed to this lan */
/*check if the interrupt type is stream interrupt *//*12APR94*/
if(op_intrpt_type() == OPC_INTRPT_STRM)
{
/* if it is, get the packet in the input stream causing interrupt */
ppp_pkptr = op_pk_get(op_intrpt_strm());
op_pk_nfd_get (ppp_pkptr, "pid_h",&ppp_pid_h);
switch (ppp_pid_h)
{
case 0x00: /* data frame */
op_pk_nfd_get (ppp_pkptr, "FDDI_frame", &pkptr1);
op_pk_destroy (ppp_pkptr);
/* get the destination address of the frame */
/* 16APR94 */
op_pk_nfd_get(pkptr1, "dest_addr", &dest_addr);
/* check if this frame destined for the local bridge station */
if(dest_addr == station_addr)
/* if it is, send the packet to llc_sink directly */
/* in order to prevent overhead of mac access */
op_pk_send(pkptr1, LLC_SINK_OUT_STREAM);/*19APR94*/
else
/* this packet is to send to mac */
{
/* determine the source address of the frame */
op_pk_nfd_get(pkptr1, "src_addr", &src_addr);
```

```
/* set up an ICI structure to communicate parameters to */
/* MAC layer process */
mac_iciptr1 = op_ici_create("fddi_mac_req");
/* place the original source address into the ICI *//* 15APR94 */
/* "fddi_mac_req" is modified so that it contains the original */
/* source address from the remote lan */
op_ici_attr_set(mac_iciptr1, "src_addr", src_addr);
/* place the destination address into the ICI */ /*12APR94*/
op_ici_attr_set(mac_iciptr1, "dest_addr", dest_addr);
/* assign the service class and requested token class */
/* At this moment the frames coming from the remote lan are assumed to have*/
/* the same priority as synchronous frames in order not to accumulate */
/* packets on the bridge station mac and instead to deliver their destinations */
/* as soon as possible */
op_pk_nfd_set(pkptr1, "pri", 8);
op_ici_attr_set(mac_iciptr1, "svc_class", FDDI_SVC_SYNC);
op_ici_attr_set(mac_iciptr1, "pri", 8);
op_ici_attr_set(mac_iciptr1, "tk_class", FDDI_TK_NONRESTRICTED);
/* send the packet coupled with the ICI */
op_ici_install(mac_iciptr1);
op_pk_send(pkptr1, MAC_LAYER_OUT_STREAM),
 }
break;

case 0xc0: /* ppp control packet - either LQR or monitoring pkt*/
/* Since the command link is deemed unjammable, this should */
/* only be for LQR's. Simply delete them for now until a*/
/* policy is determined on how to handle the LQR's.*/
 op_pk_nfd_get (ppp_pkptr, "LQR_info", &status);
printf ("LQR received at cp ");
switch (status)
{
case 0. printf ("status GOOD trend DOWN\n");
case 1: printf ("status GOOD trend UP\n");
case 2: printf ("status BAD trend DOWN\n");
case 3: printf ("status BAD trend UP\n");
}
op_pk_destroy (ppp_pkptr);
break;

default:
printf ("ERROR: packet rcvd at cp: neither data nor control\n");
break;
```

```
} /* end switch */
} /* end if (op_intrpt_type()==OPC_INTRPT_STRM) */
/* otherwise, generate the frame :12APR94 */
else
{
/* determine the length of the packet to be generated */
pklen = op_dist_outcome (len_dist_ptr);

/* determine the destination */
/* dont allow this station's address as a possible outcome */
gen_packet:
dest_addr = op_dist_outcome (dest_dist_ptr);
if (dest_addr != -1 && dest_addr == station_addr)
goto gen_packet;

/* 26DEC94 & 29JAN94: determine its priority */
pkt_prio = op_dist_outcome (pkt_priority_ptr);

/* create a packet to send to mac */
pkptr = op_pk_create_fmt ("fddi_llc_fr");

/* assign its overall size. */
op_pk_total_size_set (pkptr, pklen);

/* assign the time of creation */
op_pk_nfd_set (pkptr, "cr_time", op_sim_time ());

/* place the destination address into the ICI */
/* (the protocol_type field will default) */
op_ici_attr_set (mac_iciptr, "dest_addr", dest_addr);

/* place the source address into the ICI *//* 17APR94*/
op_ici_attr_set (mac_iciptr, "src_addr", station_addr);

/* assign the priority, and requested token class */
/* also assign the service class; 29JAN94: the fddi_llc_fr */
/* format is modified to include a "pri" field. */
if (op_dist_uniform (1.0) <= async_mix)
{
op_pk_nfd_set (pkptr, "pri", pkt_prio), /* 29JAN94 */
op_ici_attr_set (mac_iciptr, "svc_class", FDDI_SVC_ASYNC);
op_ici_attr_set (mac_iciptr, "pri", pkt_prio); /* 29JAN94 */
}
```

```
else{
op_pk_nfd_set (pkptr, "pri", 8); /* 29JAN94 */
op_ici_attr_set (mac_iciptr, "svc_class", FDDI_SVC_SYNC);
op_ici_attr_set (mac_iciptr, "pri", 8); /* 29JAN94 */
}

/* Request only nonrestricted tokens after transmission */
op_ici_attr_set (mac_iciptr, "tk_class", FDDI_TK_NONRESTRICTED);

/* Having determined priority, assign it; 26DEC93 */
/* op_ici_attr_set (mac_iciptr, "pri", pkt_prio); */

/* send the packet coupled with the ICI */
op_ici_install (mac_iciptr);
/* check if destination address is in the remote lan */
if(dest_addr > 9)
/* if it is, this packet is to send llc_sink directly */
op_pk_send (pkptr, LLC_SINK_OUT_STREAM); /*18APR94*/
else
/* if not, the packet is destined for local lan, so send to mac */
op_pk_send (pkptr, MAC_LAYER_OUT_STREAM);

/* schedule the next arrival */
fddi_gen_schedule ();
} /* end else (if opc_intrpt_type()==OPC_INTRPT_STRM) */
}


/** blocking after enter executives of unforced state. **/
FSM_EXIT (3,cp_fddi_gen)


/** state (ARRIVAL) exit executives **/
FSM_STATE_EXIT_UNFORCED (1, state1_exit_exec, "ARRIVAL")
{
}


/** state (ARRIVAL) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
/*---------------------------------------------------------*/
```

```
        }


    FSM_EXIT (0,cp_fddi_gen)
    }



void
cp_fddi_gen_svar (prs_ptr,var_name,var_p_ptr)
    cp_fddi_gen_state*prs_ptr;
    char    *var_name, **var_p_ptr;
    {

    FIN (cp_fddi_gen_svar (prs_ptr))

    *var_p_ptr = VOS_NIL;
    if (Vos_String_Equal ("inter_dist_ptr" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_inter_dist_ptr);
    if (Vos_String_Equal ("len_dist_ptr" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_len_dist_ptr);
    if (Vos_String_Equal ("dest_dist_ptr" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_dest_dist_ptr);
    if (Vos_String_Equal ("pkt_priority_ptr" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_pkt_priority_ptr);
    if (Vos_String_Equal ("mac_objid" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_mac_objid);
    if (Vos_String_Equal ("my_id" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_my_id);
    if (Vos_String_Equal ("low_dest_addr" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_low_dest_addr);
    if (Vos_String_Equal ("high_dest_addr" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_high_dest_addr);
    if (Vos_String_Equal ("station_addr" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_station_addr);
    if (Vos_String_Equal ("src_addr" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_src_addr);
    if (Vos_String_Equal ("low_pkt_priority" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_low_pkt_priority);
    if (Vos_String_Equal ("high_pkt_priority" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_high_pkt_priority);
```

```
        if (Vos_String_Equal ("arrival_rate" , var_name))
                *var_p_ptr = (char *) (&prs_ptr->sv_arrival_rate);
        if (Vos_String_Equal ("mean_pk_len" , var_name))
                *var_p_ptr = (char *) (&prs_ptr->sv_mean_pk_len);
        if (Vos_String_Equal ("async_mix" , var_name))
                *var_p_ptr = (char *) (&prs_ptr->sv_async_mix);
        if (Vos_String_Equal ("mac_iciptr" , var_name))
                *var_p_ptr = (char *) (&prs_ptr->sv_mac_iciptr);
        if (Vos_String_Equal ("mac_iciptr1" , var_name))
                *var_p_ptr = (char *) (&prs_ptr->sv_mac_iciptr1);
        if (Vos_String_Equal ("llc_ici_ptr" , var_name))
                *var_p_ptr = (char *) (&prs_ptr->sv_llc_ici_ptr);
        if (Vos_String_Equal ("pkptr1" , var_name))
                *var_p_ptr = (char *) (&prs_ptr->sv_pkptr1);

        FOUT;
        }



void
cp_fddi_gen_diag ()
        {
        Packet  *pkptr, *ppp_pkptr;
        int     pklen;
        int     dest_addr;
        int     i, restricted;
        int     pkt_prio;
        int     ppp_pid_h, ppp_pid_l;
        int     status;

        FIN (cp_fddi_gen_diag ())


        FOUT;
        }



void
cp_fddi_gen_terminate ()
```

```
        {
        Packet  *pkptr, *ppp_pkptr;
        int     pklen;
        int     dest_addr;
        int     i, restricted;
        int     pkt_prio;
        int     ppp_pid_h, ppp_pid_l;
        int     status;

        FIN (cp_fddi_gen_terminate ())


        FOUT;
        }



Compcode
cp_fddi_gen_init (pr_state_pptr)
        cp_fddi_gen_state**pr_state_pptr;
        {
        static VosT_Cm_Obtypeobtype = OPC_NIL;

        FIN (cp_fddi_gen_init (pr_state_pptr))

        if (obtype == OPC_NIL)
                {
                if (Vos_Catmem_Register ("proc state vars (cp_fddi_gen)",
                sizeof (cp_fddi_gen_state), Vos_Nop, &obtype) == VOSC_FAILURE)
                FRET (OPC_COMPCODE_FAILURE)
                }

        if ((*pr_state_pptr = (cp_fddi_gen_state*) Vos_Catmem_Alloc (obtype, 1)) == OPC_-
NIL)
                FRET (OPC_COMPCODE_FAILURE)
        else
                {
                (*pr_state_pptr)->current_block = 0;
                FRET (OPC_COMPCODE_SUCCESS)
                }
        }
```

```
/* static added 2DEC93, on advice from MIL3 */
static
fddi_gen_schedule ()
        {
        doubleinter_time;

        /* obtain an interarrival period according to the */
        /* prescribed distribution */
        inter_time = op_dist_outcome (inter_dist_ptr);

        /* schedule the arrival of next generated packet */
        op_intrpt_schedule_self (op_sim_time () + inter_time, 0);
        }
```

APPENDIX B

ERROR ALLOCATION PIPELINE STAGE CODE "cdi_pt_error.ps.c"

```
/* cdl_pt_error.ps.c */
/* Customized error allocation model for point-to-point link transceiver pipe' ne */
/*****************************************/
/* Last modified : 29MAR94, 4APR94,22APR94,20AUG94 */
/*****************************************/
#include <opnet h>
#include <ma' .h>
/*20AUG94   monitor jamming using global statistics set up in cp_fddi_sink*/
/*send stats o it only on links with id < COMMAND_LINK (user id of cmd link)*/
/*All links on the downward path should be monitored*/
extern int jammer_stats_init;
extern Gshandle link_gshandle[4];
/* Define a convenient macro for computing*/
/* factorials using the gamma unction.*/
#define log_factorial(n)lgamma ((double) n + 1.0)

void cdl_pt_error (pkptr)
Packet*  pkptr;
    {
    Objid    link_objid;
    double   pe, r, p_accum, p_exact;
    double   log_p1, log_p2, log_a ange;
    double  duty_cycle, /* 31MAR94 */
    double  jam_length, jam_ber,int_ber_jamlen, ber_ber_jamlen;/*29MAR94*/
    double  time_stamp; /* time sta mp for the packet arriving time */
    double   offset; /* 1APR94 */
    int      invert_errors = OPC_FALSE, seg_size, num_errs;
    int      jammer_type;/* 26APR94*/
    /*int      channel_index;*//* 4APR94 */
    int      user_id; /* 20AUG94 - l  id number defined in network model*/
    int      COMMAND_LINK = 1;
```

```
/** Compute the number of errors assigned to the given packet **/
/** based on its length and the bit error probability. **/
FIN (cdl_pt_error (pkptr))

/*Make a time stamp to see whether the packet is in jamming period or not */
time_stamp = op_sim_time(); /*printf("time_stamp = %16.12f\n", time_stamp);*/

/* Obtain object id of point-to-point link carrying transmission. */
link_objid = op_td_get_int (pkptr, OPC_TDA_PT_LINK_OBJID);
/* Obtain the channel index for the particular link */
/* Determine which channel the packet is on *//*30MAR*/
/* channel_index = op_td_get_int (pkptr,OPC_TDA_PT_CH_INDEX);*//*4APR94*/
/* Obtain the bit-error probability of the channel. */
/* op_ima_obj_attr_get (link_objid, "ber", &pe); *//*ignore this attribute 31MAR94*/
/* Obtain the extended attributes for the point-to-point link */
/* These attributes are appended in order to simulate jamming features */
/* 29MAR94 */
op_ima_obj_attr_get (link_objid, "jam_length", &jam_length);
op_ima_obj_attr_get (link_objid, "jam_ber", &jam_ber);
op_ima_obj_attr_get (link_objid, "interval_bet_jam_len", &int_bet_jamlen);
op_ima_obj_attr_get (link_objid, "ber_bet_jam_len", &ber_bet_jamlen);
op_ima_obj_attr_get (link_objid, "in_jam_offset", &offset);
op_ima_obj_attr_get (link_objid, "jammer_type", &jammer_type);
op_ima_obj_attr_get (link_objid, "user id", &user_id);
/* Obtain the length of the packet. */
seg_size = op_pk_total_size_get (pkptr);
/* Determine the jammer type in use;26APR94 */
/* Check if pulsed jammer is in use */
if (jammer_type == 0)
    {
    /* Randomize the jamming durations */
    /* These durations are randomized with uniform distribution */
    /* in range [0, duration]. User should be aware of these */
    /* attributes specified in the environment file. They are max values */
    /* for those particular durations */
    jam_length = op_dist_uniform(jam_length);
    int_bet_jamlen = op_dist_uniform(int_bet_jamlen);
```

```
    }
/* Otherwise, channel swept jammer is in use. Jamming durations */
/* should not be randomized to keep consecutive pulses in order. */
/* Compute duty cycle for jamming */
duty_cycle = jam_length + int_bet_jamlen;
/* Check time stamp if it is in the initial jam offset period */
/* All BER's are uniformly distributed in range [0, ber[, so that */
/* realistic representation is provided; User should be aware of */
/* these attributes specified in the environment file. They are max values */
/* for those particular bers*/
if (time_stamp < offset)
    {
    pe = op_dist_uniform(ber_bet_jamlen); /* the packet is still not in the jamming period */

    if ((jammer_stats_init==OPC_TRUE)&& (user_id < COMMAND_LINK))
        op_stat_global_write (link_gshandle[user_id], pe);
    }
else
    {
    /* Check packet is in jamming period */
    if ( fmod(time_stamp,duty_cycle) <= jam_length )
        {
        pe = op_dist_uniform(jam_ber); /* the packet is in jamming period, */
        /* random "pe" to be computed as jam_ber */
        if ((jammer_stats_init==OPC_TRUE)&& (user_id < COMMAND_LINK))
            op_stat_global_write (link_gshandle[user_id], pe);
        }
    else
        {
        pe = op_dist_uniform(ber_bet_jamlen); /* packet is in unjammed period */
        /*    random "pe" to be computed as ber_bet_jamlen */
        if ((jammer_stats_init==OPC_TRUE)&& (user_id < COMMAND_LINK))
        op_stat_global_write (link_gshandle[user_id], pe);
        }
    }

/*This part computes num_errs for the packet */
/* Case 1: if the bit error rate is zero, so is the number of errors */
```

```
if (pe == 0.0 || seg_size == 0)
    num_errs = 0;
/* Case 2: if the bit error rate is 1.0, then all the bits are in error.*/
/* (note however, that bit error rates should not normally exceed 0.5).*/
else if (pe >= 1.0)
    num_errs = seg_size;
/* Case 3: The bit error rate is not zero or one. */
else
    {
    /* If the bit error rate is greater than 0.5 and less than 1.0, invert*/
    /* the problem to find instead the number of bits that are not in error*/
    /* in order to accelerate the performance of the algorithm. Set a flag*/
    /* to indicate that the result will then have to be inverted.*/
    if (pe > 0.5)
        {
        pe = 1.0 - pe;
        invert_errors = OPC_TRUE;
        }
    /* The error count can be obtained by mapping a uniform random number */
    /* in [0, 1[ via the inverse of the cumulative mass function (CMF) */
    /* for the bit error count distribution. */
    /* Obtain a uniform random number in [0, 1[ to represent */
    /* the value of the CDF at the outcome that will be produced. */
    r = op_dist_uniform (1.0);
    /* Integrate probability mass over possible outcomes until r is exceeded. */
    /* The loop iteratively corresponds to "inverting" the CMF since it finds*/
    /* the bit error count at which the CMF first meets or exceeds the value r.*/
    for (p_accum = 0.0, num_errs = 0; num_errs <= seg_size; num_errs++)
        {
        /* Compute the probability of exactly 'num_errs' bit errors occurring.*/
        /* The probability that the first 'num_errs' bits will be in error */
        /* is given by pow (pe, num_errs). Here it is obtained in logarithmic*/
        /* form to avoid underflow for small 'pe' or large 'num_errs_jam'.*/
        log_p1 = (double) num_errs * log (pe);
        /* Similarly, obtain the probability that the remaining bits will not */
        /* be in error. The combination of these two events represents one */
        /* possible configuration of bits yielding a total of 'num_errs' errors.*/
        log_p2 = (double) (seg_size - num_errs) * log (1.0 - pe);
```

```
            /* Compute the number of arrangements that are possible with the same */
            /* number of bits in error as the particular case above. Again obtain */
            /* this number in logarithmic form (to avoid overflow in this case).*/
            /* This result is expressed as the logarithmic form of the formula for*/
            /* the number N of combinations of k items from n: N = n!/(n-k)!k!  */
            log_arrange = log_factorial (seg_size) -
            log_factorial (num_errs) -
            log_factorial (seg_size - num_errs);
            /* Compure the probability that exactly 'num_errs' are present */
            /* in the segment of bits, in any arrangement.*/
            p_exact = exp (log_arrange + log_p1 + log_p2);
            /* Add this to the probability mass accumulated so far for previously */
            /* tested outcomes to obtain the value of the CMF at outcome = num_errs.*/
            p_accum += p_exact;
            /*'num_errs_jam' is the outcome for this trial if the CMF meets or exceeds */
            /* the uniform random value selected earlier. */
            if (p_accum >= r)
                break;
            }
        /* If the bit error rate was inverted to compute correct bits instead, then */
        /* reinvert the result to obtain the number of bits in error. */
        if (invert_errors == OPC_TRUE)
            num_errs = seg_size - num_errs;
        }
    /*printf("num_of_errors = %5d\n", num_errs);*/
    /* Set number of bit errors in packet transmission data attribute. */
    op_td_set_int (pkptr, OPC_TDA_PT_NUM_ERRORS, num_errs);
    FOUT
    }
```

APPENDIX C

RING 0 LLC_SINK MODULE CODE
"cp_fddi_sink.pr.c"

```
/* Process model C form file: cp_fddi_sink.pr.c */
/* Portions of this file Copyright (C) MIL 3, Inc. 1992 */



/* OPNET system definitions */
#include <opnet.h>
#include "cp_fddi_sink.pr.h"
FSM_EXT_DECS



/* Header block */
/* Globals */
/* array format installed 20JAN94; positions 0-7 represent the asynch priority levels, PRIORITIES
+ 1 */
/* represents synch traffic, and grand totals are as given in the original. */

#define PRIORITIES 8 /* 20JAN94 */
#define         XMITTER_ONE0 /*10MAY94*/
#define         XMITTER_TWO1
#define         XMITTER_THREE2
#define         XMITTER_FOUR3


static /* 05FEB94 */
double fddi_sink_accum_delay = 0.0;
static /* 05FEB94 */
double          fddi_sink_accum_delay_a[PRIORITIES + 1] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0};
static /* 05FEB94 */
int fddi_sink_total_pkts = 0;
static /* 05FEB94 */
int             fddi_sink_total_pkts_a[PRIORITIES + 1] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
static /* 05FEB94 */
double fddi_sink_total_bits = 0.0;
static /* 05FEB94 */
double          fddi_sink_total_bits_a[PRIORITIES + 1] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```

```
0.0};
static /* 05FEB94 */
double fddi_sink_peak_delay = 0.0;
static /* 05FEB94 */
double          fddi_sink_peak_delay_a[PRIORITIES + 2] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0};
static /* 05FEB94 */
int             fddi_sink_scalar_write = 0;
static /* 05FEB94 */
int pri_set = 20; /* 20JAN94 */
static
int             subq_index = 0; /* 5APR94 */
static
double          buffer[4]={0.0,0.0,0.0,0.0}; /*10MAY94*/


/* statistics used for CDL throughput */
static /* 20APR94 */
int fddilp1_total_pkts = 0;
static
int             fddilp1_total_pkts_a[PRIORITIES + 1] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
static
double fddilp1_total_bits = 0.0;
static
double          fddilp1_total_bits_a[PRIORITIES + 1] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0};


/* Externally defined globals. */
extern doublefddi_t_opr [];

/*12JAN94:attributes from the Environment file */
double Offered_Load; /* 12JAN94 */
double Asynch_Offered_Load; /* 12JAN94 */

/* transition expressions */
#define END_OF_SIM op_intrpt_type() == OPC_INTRPT_ENDSIM

/* get picture of jamming. this variable allows anyone who*/
/* is writing to the global statistic to ensure it has */
/* been initialized. 20AUG94*/
int             jammer_stats_init=OPC_FALSE;
Gshandle link_gshandle[4];



/* State variable definitions */
```

```
typedef struct
        {
        FSM_SYS_STATE
        Gshandle sv_thru_gshandle;
        Gshandle sv_m_delay_gshandle;
        Gshandle sv_ete_delay_gshandle;
        Gshandle sv_thru_gshandle_a[10];
        Gshandle sv_m_delay_gshandle_a[10];
        Gshandle sv_ete_delay_gshandle_a[9];
        Gshandle sv_t_gshandle;
        Gshandle sv_t_gshandle_a[10];
        Objid    sv_my_id;
        int      sv_PPP_seq_number;
        double   sv_time;
        } cp_fddi_sink_state;

#define pr_state_ptr            ((cp_fddi_sink_state*) SimI_Mod_State_Ptr)
#define thru_gshandle           pr_state_ptr->sv_thru_gshandle
#define m_delay_gshandle        pr_state_ptr->sv_m_delay_gshandle
#define ete_delay_gshandle      pr_state_ptr->sv_ete_delay_gshandle
#define thru_gshandle_a         pr_state_ptr->sv_thru_gshandle_a
#define m_delay_gshandle_a      pr_state_ptr->sv_m_delay_gshandle_a
#define ete_delay_gshandle_a    pr_state_ptr->sv_ete_delay_gshandle_a
#define t_gshandle              pr_state_ptr->sv_t_gshandle
#define t_gshandle_a            pr_state_ptr->sv_t_gshandle_a
#define my_id                   pr_state_ptr->sv_my_id
#define PPP_seq_number          pr_state_ptr->sv_PPP_seq_number
#define time                    pr_state_ptr->sv_time



/* Process model interrupt handling procedure */


void
cp_fddi_sink ()
        {
        double  delay, creat_time;
        Packet* pkptr;
        Packet* pkptr1 ; /*5APR94*/
        Packet* ppp_pkptr;/*15JUN94*/
        int     src_addr, my_addr;
        int     dest_addr;/*14APR94*/
        Ici*    from_mac_ici_ptr;
        double  fddi_sink_ttrt;
```

```
    int     xmit_subq_index;/*5APR94*/
    int     load_balance_code; /*6APR94*/
    int     i,subq_no; /*25APR94*/
    int     index; /*10MAY94 */
    double  link_mon_trans_rate; /*15JUN94*/
    char    str0[512], str1 [512]; /* for diagnostics*/
    int     mon_pkt_size; /*26JUL*/
    double  sim_duration; /*26JUL*/


FSM_ENTER (cp_fddi_sink)

FSM_BLOCK_SWITCH
  {
  /*--------------------------------------------------*/
  /** state (DISCARD) enter executives **/
  FSM_STATE_ENTER_UNFORCED (0, state0_enter_exec, "DISCARD")
    {
    /* determine type of interrupt:10MAY94 */
    switch (op_intrpt_type())
      {
    case OPC_INTRPT_STAT:
    /* the interrupt is caused by the transmitters' status */
      {
      index = op_intrpt_stat();
      switch(index)
        {
        case XMITTER_ONE:
          {
          buffer[0] = op_stat_local_read(XMITTER_ONE);
          break;
          }
        case XMITTER_TWO:
          {
          buffer[1] = op_stat_local_read(XMITTER_TWO);
          break;
          }
        case XMITTER_THREE:
          {
          buffer[2] = op_stat_local_read(XMITTER_THREE);
          break;
          }
        case XMITTER_FOUR:
          {
          buffer[3] = op_stat_local_read(XMITTER_FOUR);
          break;
          }
```

```
            default:
              {
              op_sim_end("*** FDDI-CDL : FATAL ERROR","Unexpected stat
interrupt","","");
              }
            }
          break;
          }
        case OPC_INTRPT_STRM:
        /* the interrupt is caused by the incoming packets */
          {
        /* get the packet and the interface control info */
        pkptr = op_pk_get (op_intrpt_strm ());
        from_mac_ici_ptr = op_intrpt_ici ();

        /* 20JAN94: get the packet's priority level, which */
        /* will be used to index arrays of thruput and delay */
        /* computations. */
        /* pri_set = op_pk_priority_get (pkptr); doesn't work here */
        op_pk_nfd_get (pkptr, "pri", &pri_set); /* 29JAN94 */

        /* determine the time of creation of the packet */
        op_pk_nfd_get (pkptr, "cr_time", &creat_time);

        /* 18APR94:determine the destination address of the packet */
        op_pk_nfd_get (pkptr, "dest_addr", &dest_addr);

        /* 20APR94:determine the source address of the packet */
        op_pk_nfd_get (pkptr, "src_addr", &src_addr);

        /* 7APR94: determine which load balancing algorithm is in use */
        op_ima_obj_attr_get ( my_id, "load balancing algorithm", &load_balance_code );

        /* 14APR94 : also get my own address */
        op_ima_obj_attr_get ( my_id, "station_address", &my_addr);

        /* destroy the packet */
        /* op_pk_destroy (pkptr); */
        /* 03FEB94: rather, enqueue the packet. This will be the */
        /* first step toward developing a LAN bridging structure. */
        /* -Nix */
        /* op_subq_pk_insert (pri_set, pkptr, OPC_QPOS_TAIL); */

        /* 14APR94: check the frame passed to "llc" is destined for */
        /* this station. If it is destroy the packet and update the local traffic */
        /* statistics; if not, allocate the packets */
        /* to the transmitters since they are destined for the remote lan */
```

```
/* update also incoming return link statistics for the frames */
/* which will be queued in llc_sink to be sent to remote lan.*/
/* -Karayakaylar */

if((dest_addr == my_addr)&&(src_addr < my_addr))
   {
   /* add in its size */
   fddi_sink_total_bits += op_pk_total_size_get (pkptr);
   fddi_sink_total_bits_a[pri_set] += op_pk_total_size_get (pkptr); /* 20JAN-20APR94 */

   /* accumulate delays */
   delay = op_sim_time () - creat_time;
   fddi_sink_accum_delay += delay;
   fddi_sink_accum_delay_a[pri_set] += delay; /* 20JAN-20APR94 */

   /* keep track of peak delay value */
   if (delay > fddi_sink_peak_delay)
      fddi_sink_peak_delay = delay;

   /* 20JAN94: keep track by priority levels as well 23JAN-20APR94 */
   if (delay > fddi_sink_peak_delay_a[pri_set])
   fddi_sink_peak_delay_a[pri_set] = delay;

   op_pk_destroy (pkptr);

   /* increment packet counter; 20JAN94 */
   fddi_sink_total_pkts++;
   fddi_sink_total_pkts_a[pri_set]++;

   /* if a multiple of 25 packets is reached, update stats */
   /* 03FEB94: [0]->[7] represent asynch priorities 1->8, */
   /* respectively; [8] represents synchronous traffic, */
   /* and [9] represents overall asynchronous traffic.-Nix */
   /*if (fddi_sink_total_pkts % 25 == 0)
      {
      op_stat_global_write (thru_gshandle,
            fddi_sink_total_bits / op_sim_time ());

  op_stat_global_write (thru_gshandle_a[pri_set],
fddi_sink_total_bits_a[0] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[0],
fddi_sink_total_bits_a[1] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[1],
fddi_sink_total_bits_a[pri_set] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[2],
fddi_sink_total_bits_a[2] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[3],
```

```
fddi_sink_total_bits_a[3] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[4],
fddi_sink_total_bits_a[4] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[5],
fddi_sink_total_bits_a[5] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[6],
fddi_sink_total_bits_a[6] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[7],
fddi_sink_total_bits_a[7] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[8],
fddi_sink_total_bits_a[8] / op_sim_time());
   */
     /* 30JAN94: gather all asynch stats into one overall figure */
  /*op_stat_global_write (thru_gshandle_a[9],
(fddi_sink_total_bits - fddi_sink_total_bits_a[8]) /
op_sim_time());
*/

/* (fddi_sink_total_bits_a[0] + fddi_sink_total_bits_a[1] + */
/* fddi_sink_total_bits_a[2] + fddi_sink_total_bits_a[3] + */
/* fddi_sink_total_bits_a[4] + fddi_sink_total_bits_a[5] + */
/* fddi_sink_total_bits_a[6] + fddi_sink_total_bits_a[7]) / */
/* op_sim_time()); */

/*
  op_stat_global_write (m_delay_gshandle,
        fddi_sink_accum_delay / fddi_sink_total_pkts);

  op_stat_global_write (m_delay_gshandle_a[0],
fddi_sink_accum_delay_a[0] / fddi_sink_total_pkts_a[0]);
  op_stat_global_write (m_delay_gshandle_a[1],
fddi_sink_accum_delay_a[1] / fddi_sink_total_pkts_a[1]);
  op_stat_global_write (m_delay_gshandle_a[2],
fddi_sink_accum_delay_a[2] / fddi_sink_total_pkts_a[2]);
  op_stat_global_write (m_delay_gshandle_a[3],
fddi_sink_accum_delay_a[3] / fddi_sink_total_pkts_a[3]);
  op_stat_global_write (m_delay_gshandle_a[4],
fddi_sink_accum_delay_a[4] / fddi_sink_total_pkts_a[4]);
  op_stat_global_write (m_delay_gshandle_a[5],
fddi_sink_accum_delay_a[5] / fddi_sink_total_pkts_a[5]);
  op_stat_global_write (m_delay_gshandle_a[6],
fddi_sink_accum_delay_a[6] / fddi_sink_total_pkts_a[6]);
  op_stat_global_write (m_delay_gshandle_a[7],
fddi_sink_accum_delay_a[7] / fddi_sink_total_pkts_a[7]);
  op_stat_global_write (m_delay_gshandle_a[8],
fddi_sink_accum_delay_a[8] / fddi_sink_total_pkts_a[8]);
*/
```

```
    /* 30JAN94: gather all asynch stats into one figure */
  /*op_stat_global_write (m_delay_gshandle_a[9],
(fddi_sink_accum_delay - fddi_sink_accum_delay_a[8]) /
(fddi_sink_total_pkts - fddi_sink_total_pkts_a[8]));
*/

/* (fddi_sink_accum_delay_a[0] + fddi_sink_accum_delay_a[1] + */
/* fddi_sink_accum_delay_a[2] + fddi_sink_accum_delay_a[3] + */
/* fddi_sink_accum_delay_a[4] + fddi_sink_accum_delay_a[5] + */
/* fddi_sink_accum_delay_a[6] + fddi_sink_accum_delay_a[7]) / */
/* (fddi_sink_total_pkts_a[0] + fddi_sink_total_pkts_a[1] + */
/* fddi_sink_total_pkts_a[2] + fddi_sink_total_pkts_a[3] + */
/* fddi_sink_total_pkts_a[4] + fddi_sink_total_pkts_a[5] + */
/* fddi_sink_total_pkts_a[6] + fddi_sink_total_pkts_a[7])); */



    /* also record actual delay values */
    /*op_stat_global_write (ete_delay_gshandle, delay);
    op_stat_global_write (ete_delay_gshandle_a[pri_set], delay),
    }
    */
  } /* end of if(dest_addr == my_addr)&&(src_addr < my_addr) statement */

/* 20APR94:destroy the packets coming from the remote lan destined for this*/
/* station. These packets are not counted for local traffic.*/
/*else */if(dest_addr == my_addr)
  op_pk_destroy(pkptr);

/* 20APR94: check the frame passed to "llc" is destined for remote lan */
/* This will allow only the packets to be counted for CDL traffic.*/
/* -Karayakaylar */
else
  {

  /* add in its size */
  fddilp1_total_bits += op_pk_total_size_get (pkptr);
  fddilp1_total_bits_a[pri_set] += op_pk_total_size_get (pkptr); /* 20APR94 */

  /* increment packet counter; 20APR94 */
  fddilp1_total_pkts++;
  fddilp1_total_pkts_a[pri_set]++;

  /* if a multiple of 25 packets is reached, update stats */
  /* [0]->[7] represent asynch priorities 1->8, */
  /* respectively; [8] represents synchronous traffic, */
  /* and [9] represents overall asynchronous traffic.-Nix */
```

```
  /*if (fddilp1_total_pkts % 25 == 0)
     {
     op_stat_global_write (t_gshandle,
          fddilp1_total_bits / op_sim_time ());

  op_stat_global_write (t_gshandle_a[pri_set],
fddilp1_total_bits_a[0] / op_sim_time());
  op_stat_global_write (t_gshandle_a[0],
fddilp1_total_bits_a[1] / op_sim_time());
  op_stat_global_write (t_gshandle_a[1],
fddilp1_total_bits_a[pri_set] / op_sim_time());
  op_stat_global_write (t_gshandle_a[2],
fddilp1_total_bits_a[2] / op_sim_time());
  op_stat_global_write (t_gshandle_a[3],
fddilp1_total_bits_a[3] / op_sim_time());
  op_stat_global_write (t_gshandle_a[4],
fddilp1_total_bits_a[4] / op_sim_time());
  op_stat_global_write (t_gshandle_a[5],
fddilp1_total_bits_a[5] / op_sim_time());
  op_stat_global_write (t_gshandle_a[6],
fddilp1_total_bits_a[6] / op_sim_time());
  op_stat_global_write (t_gshandle_a[7],
fddilp1_total_bits_a[7] / op_sim_time());
  op_stat_global_write (t_gshandle_a[8],
fddilp1_total_bits_a[8] / op_sim_time());
   */
     /* gather all asynch stats into one overall figure */
  /*op_stat_global_write (t_gshandle_a[9],
(fddilp1_total_bits - fddiip1_total_bits_a[8]) /
op_sim_time());

*/
/* (fddilp1_total_bits_a[0] + fddilp1_total_bits_a[1] + */
/* fddilp1_total_bits_a[2] + fddilp1_total_bits_a[3] + */
/* fddilp1_total_bits_a[4] + fddilp1_total_bits_a[5] + */
/* fddilp1_total_bits_a[6] + fddilp1_total_bits_a[7]) / */
/* op_sim_time()); */

  /*}*/

     /*15JUN94 : before allocating, encapsulate the FDDI frame into a PPP packet */
     ppp_pkptr = op_pk_create_fmt ("ppp_ml");
     op_pk_nfd_set (ppp_pkptr, "seq_number", PPP_seq_number++);
     op_pk_nfd_set (ppp_pkptr, "FDDI_frame", pkptr),

     /* 14APR94 :allocate the packets to llc_sink subqueues */
```

```
            /* 6APR94 -Karayakaylar*/
            /* check if load balancing algorithm is circular */
            /* zero(0) is the circular load balancing code */
            if (load_balance_code == 0)
              {
          /* 5APR94 */
              /* Apply load balancing to insert the packets in the */
              /* subqueues, in a circular order */
              subq_no = subq_index % 4;
              op_subq_pk_insert(subq_no, ppp_pkptr, OPC_QPOS_TAIL); /*15JUN altered to
send ppp*/
        subq_index++;
          }


        /* 25APR94 */
            /* check if load baiancing algorithm is empty allocation */
            /* one(1) is the empty allocation load balancing code */
            if (load_balance_code == 1)
              {
              /* Apply load balancing to insert the packets in the */
              /* subqueues by choosing the subqueue which has the maximum current */
              /* number of free packet slots */
              subq_no = op_subq_index_map(OPC_QSEL_MAX_FREE_PKSIZE);
              op_subq_pk_insert(subq_no, ppp_pkptr, OPC_QPOS_TAIL); /*15JUN altered to
send ppp*/
            }

          }/*if(dest_addr > my_addr) statement */
          break;
        }/* end of case OPC_INTRPT_STRM statement */

        case OPC_INTRPT_SELF: /*27JUL94*/
        /* if it is a self-interrupt, it is time to send a monitoring packet. Send one. */
          {
          ppp_pkptr = op_pk_create_fmt("ppp");
          op_pk_nfd_set (ppp_pkptr,"pid_h",0xc0);
          op_pk_nfd_set (ppp_pkptr,"pid_l",0x21);
          op_ima_obj_attr_get (my_id,"monitoring pkt size", &mon_pkt_size);
          op_pk_bulk_size_set (ppp_pkptr, mon_pkt_size);
        if (load_balance_code == 1)
            {
            subq_no = op_subq_index_map(OPC_QSEL_MAX_FREE_PKSIZE);
            op_subq_pk_insert (subq_no,ppp_pkptr,OPC_QPOS_TAIL);
            }
          else if (load_balance_code == 0)
            {
```

```
        subq_no = subq_index % 4;
        op_subq_pk_insert (subq_no,ppp_pkptr,OPC_QPOS_TAIL);
        subq_index++;
        }
      break;
      }


  }/* end of switch */

      /* check if each subqueue is not empty and transmitter is not busy */
      /* irregardless if an interrupt is received or not 1AUG94*/
  for (xmit_subq_index = 0;xmit_subq_index <=3; ++xmit_subq_index)
    {
      if ((!op_subq_empty(xmit_subq_index))&&(buffer[xmit_subq_index] == 0.0))
        {
        /*access the first packet in the subqueue */
        pkptr1 = op_subq_pk_remove (xmit_subq_index, OPC_QPOS_HEAD);
      /* forward it to the destination xmitter */
        /* associated with the subqueue index */
        op_pk_send (pkptr1, xmit_subq_index );
        /*if (op_sim_debug()==OPC_TRUE)
          printf ("packet sent to cp xmtr from subqueue %d\n",xmit_subq_index);*/
        }
   }
  }


/** blocking after enter executives of unforced state. **/
FSM_EXIT (1,cp_fddi_sink)


/** state (DISCARD) exit executives **/
FSM_STATE_EXIT_UNFORCED (0, state0_exit_exec, "DISCARD")
  {
  }


/** state (DISCARD) transition processing **/
FSM_INIT_COND (END_OF_SIM)
FSM_DFLT_COND
FSM_TEST_LOGIC ("DISCARD")

FSM_TRANSIT_SWITCH
  {
  FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;)
```

```
  FSM_CASE_TRANSIT (1, 0, state0_enter_exec, ;)
  }
/*--------------------------------------------------------*/


/** state (STATS) enter executives **/
FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "STATS")
  {
  /* At end of simulation, scalar performance statistics */
  /* and input parameters are written out. */

    op_stat_scalar_write ("RL Throughput (bps), Priority 1",
        fddilp1_total_bits_a[0] / op_sim_time ()); /*20APR94*/

  op_stat_scalar_write ("RL Throughput (bps), Priority 2",
  fddilp1_total_bits_a[1] / op_sim_time ());

  op_stat_scalar_write ("RL Throughput (bps), Priority 3",
  fddilp1_total_bits_a[2] / op_sim_time ());

  op_stat_scalar_write ("RL Throughput (bps), Priority 4",
  fddilp1_total_bits_a[3] / op_sim_time ());

  op_stat_scalar_write ("RL Throughput (bps), Priority 5",
  fddilp1_total_bits_a[4] / op_sim_time ());

  op_stat_scalar_write ("RL Throughput (bps), Priority 6",
  fddilp1_total_bits_a[5] / op_sim_time ());

  op_stat_scalar_write ("RL Throughput (bps), Priority 7",
  fddilp1_total_bits_a[6] / op_sim_time ());

  op_stat_scalar_write ("RL Throughput (bps), Priority 8",
  fddilp1_total_bits_a[7] / op_sim_time ());

  op_stat_scalar_write ("RL Throughput (bps), Asynchronous",
  (fddilp1_total_bits - fddilp1_total_bits_a[8]) / op_sim_time ());


  /* (fddilp1_total_bits_a[0] + fddilp1_total_bits_a[1] + */
  /* fddilp1_total_bits_a[2] + fddilp1_total_bits_a[3] + */
  /* fddilp1_total_bits_a[4] + fddilp1_total_bits_a[5] + */
  /* fddilp1_total_bits_a[6] + fddilp1_total_bits_a[7]) / */
  /* op_sim_time ()); */
```

```
op_stat_scalar_write ("RL Throughput (bps), Synchronous",
fddilp1_total_bits_a[8] / op_sim_time ());

op_stat_scalar_write ("RL Throughput (bps), Total",
fddilp1_total_bits / op_sim_time ()); /*20APR94*/



/* Only one station needs to do this */
if (!fddi_sink_scalar_write)
   {
   /* set the scalar write flag */
   fddi_sink_scalar_write = 1;

   op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 1",
        fddi_sink_accum_delay_a[0] / fddi_sink_total_pkts_a[0]);

 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 2",
 fddi_sink_accum_delay_a[1] / fddi_sink_total_pkts_a[1]);

 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 3",
 fddi_sink_accum_delay_a[2] / fddi_sink_total_pkts_a[2]);

 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 4",
 fddi_sink_accum_delay_a[3] / fddi_sink_total_pkts_a[3]);

 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 5",
 fddi_sink_accum_delay_a[4] / fddi_sink_total_pkts_a[4]);

 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 6",
 fddi_sink_accum_delay_a[5] / fddi_sink_total_pkts_a[5]);

 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 7",
 fddi_sink_accum_delay_a[6] / fddi_sink_total_pkts_a[6]);

 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 8",
 fddi_sink_accum_delay_a[7] / fddi_sink_total_pkts_a[7]);

 op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Asynchronous",
 (fddi_sink_accum_delay - fddi_sink_accum_delay_a[8]) /
 (fddi_sink_total_pkts - fddi_sink_total_pkts_a[8]));

 /* (fddi_sink_accum_delay_a[0] + fddi_sink_accum_delay_a[1] + */
 /* fddi_sink_accum_delay_a[2] + fddi_sink_accum_delay_a[3] + */
 /* fddi_sink_accum_delay_a[4] + fddi_sink_accum_delay_a[5] + */
 /* fddi_sink_accum_delay_a[6] + fddi_sink_accum_delay_a[7]) / */
```

```
/* (fddi_sink_total_pkts_a[0] + fddi_sink_total_pkts_a[1] + */
/* fddi_sink_total_pkts_a[2] + fddi_sink_total_pkts_a[3] + */
/* fddi_sink_total_pkts_a[4] + fddi_sink_total_pkts_a[5] + */
/* fddi_sink_total_pkts_a[6] + fddi_sink_total_pkts_a[7])); */


op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Synchronous",
fddi_sink_accum_delay_a[8] / fddi_sink_total_pkts_a[8]);

op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Total",
fddi_sink_accum_delay / fddi_sink_total_pkts);

  op_stat_scalar_write ("Throughput-0 (bps), Priority 1",
       fddi_sink_total_bits_a[0] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Priority 2",
fddi_sink_total_bits_a[1] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Priority 3",
fddi_sink_total_bits_a[2] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Priority 4",
fddi_sink_total_bits_a[3] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Priority 5",
fddi_sink_total_bits_a[4] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Priority 6",
fddi_sink_total_bits_a[5] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Priority 7",
fddi_sink_total_bits_a[6] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Priority 8",
fddi_sink_total_bits_a[7] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Asynchronous",
(fddi_sink_total_bits - fddi_sink_total_bits_a[8]) / op_sim_time ());


/* (fddi_sink_total_bits_a[0] + fddi_sink_total_bits_a[1] + */
/* fddi_sink_total_bits_a[2] + fddi_sink_total_bits_a[3] + */
/* fddi_sink_total_bits_a[4] + fddi_sink_total_bits_a[5] + */
/* fddi_sink_total_bits_a[6] + fddi_sink_total_bits_a[7]) / */
/* op_sim_time ()); */
```

```
op_stat_scalar_write ("Throughput-0 (bps), Synchronous",
fddi_sink_total_bits_a[8] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Total",
fddi_sink_total_bits / op_sim_time ());


  op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 1",
       fddi_sink_peak_delay_a[0]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 2",
fddi_sink_peak_delay_a[1]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 3 ',
fddi_sink_peak_delay_a[2]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 4",
fddi_sink_peak_delay_a[3]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 5",
fddi_sink_peak_delay_a[4]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 6",
fddi_sink_peak_delay_a[5]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 7",
fddi_sink_peak_delay_a[6]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 8",
fddi_sink_peak_delay_a[7]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Synchronous",
fddi_sink_peak_delay_a[8]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Overall",
fddi_sink_peak_delay);


  /* Write the TTRT value for ring 0. This preserves */
  /* the old behavior for single-ring simulations. */
  op_stat_scalar_write ("TTRT (sec.) - Ring 0",
       fddi_t_opr [0]);

/* 12JAN94: obtain offered load information from the Environment */
/* file; this will be used to provide abscissa information that */
/* can be plotted in the Analysis Editor (see "fddi_sink" STATS */
```

```
        /* state. To the user: it's your job to keep these current in */
        /* the Environment File. -Nix */
        op_ima_sim_attr_get (OPC_IMA_DOUBLE, "total_offered_load_0", &Offered_Load);
        op_ima_sim_attr_get (OPC_IMA_DOUBLE, "asynch_offered_load_0", &Asynch_Of-
fered_Load);


        /* 12JAN94: write the total offered load for this run */
        op_stat_scalar_write ("Total Offered Load-0 (Mbps)",
        Offered_Load);

        op_stat_scalar_write ("Asynchronous Offered Load-0 (Mbps)",
        Asynch_Offered_Load);
          }
        }


      /** blocking after enter executives of unforced state. **/
      FSM_EXIT (3,cp_fddi_sink)


      /** state (STATS) exit executives **/
      FSM_STATE_EXIT_UNFORCED (1, state1_exit_exec, "STATS")
        {
        }


      /** state (STATS) transition processing **/
      FSM_TRANSIT_MISSING ("STATS")
      /*---------------------------------------------------------*/



      /** state (INIT) enter executives **/
      FSM_STATE_ENTER_FORCED (2, state2_enter_exec, "INIT")
        {
        /* get the gshandles of the global statistic to be obtained */
        /* 20JAN94: set array format */
        /*
        thru_gshandle_a[0] = op_stat_global_reg ("pri 1 throughput-0 (bps)");
        thru_gshandle_a[1] = op_stat_global_reg ("pri 2 throughput-0 (bps)");
        thru_gshandle_a[2] = op_stat_global_reg ("pri 3 throughput-0 (bps)");
        thru_gshandle_a[3] = op_stat_global_reg ("pri 4 throughput-0 (bps)");
        thru_gshandle_a[4] = op_stat_global_reg ("pri 5 throughput-0 (bps)");
        thru_gshandle_a[5] = op_stat_global_reg ("pri 6 throughput-0 (bps)");
        thru_gshandle_a[6] = op_stat_global_reg ("pri 7 throughput-0 (bps)");
        thru_gshandle_a[7] = op_stat_global_reg ("pri 8 throughput-0 (bps)");
```

```
thru_gshandle_a[8] = op_stat_global_reg ("synch throughput-0 (bps)");
thru_gshandle_a[9] = op_stat_global_reg ("async throughput-0 (bps)");
thru_gshandle = op_stat_global_reg ("total throughput-0 (bps)");

m_delay_gshandle_a[0] = op_stat_global_reg ("pri 1 mean delay-0 (sec.)");
m_delay_gshandle_a[1] = op_stat_global_reg ("pri 2 mean delay-0 (sec.)");
m_delay_gshandle_a[2] = op_stat_global_reg ("pri 3 mean delay-0 (sec.)");
m_delay_gshandle_a[3] = op_stat_global_reg ("pri 4 mean delay-0 (sec.)");
m_delay_gshandle_a[4] = op_stat_global_reg ("pri 5 mean delay-0 (sec.)");
m_delay_gshandle_a[5] = op_stat_global_reg ("pri 6 mean delay-0 (sec.)");
m_delay_gshandle_a[6] = op_stat_global_reg ("pri 7 mean delay-0 (sec.)");
m_delay_gshandle_a[7] = op_stat_global_reg ("pri 8 mean delay-0 (sec.)");
m_delay_gshandle_a[8] = op_stat_global_reg ("synch mean delay-0 (sec.)");
m_delay_gshandle_a[9] = op_stat_global_reg ("async mean delay-0 (sec.)");
m_delay_gshandle = op_stat_global_reg ("total mean delay-0 (sec.)");

ete_delay_gshandle_a[0] = op_stat_global_reg ("pri 1 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[1] = op_stat_global_reg ("pri 2 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[2] = op_stat_global_reg ("pri 3 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[3] = op_stat_global_reg ("pri 4 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[4] = op_stat_global_reg ("pri 5 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[5] = op_stat_global_reg ("pri 6 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[6] = op_stat_global_reg ("pri 7 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[7] = op_stat_global_reg ("pri 8 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[8] = op_stat_global_reg ("synch end-to-end delay-0 (sec.)");
ete_delay_gshandle = op_stat_global_reg ("total end-to-end delay-0 (sec.)");

t_gshandle_a[0] = op_stat_global_reg ("pri 1 RL throughput (bps)");
t_gshandle_a[1] = op_stat_global_reg ("pri 2 RL throughput (bps)");
t_gshandle_a[2] = op_stat_global_reg ("pri 3 RL throughput (bps)");
t_gshandle_a[3] = op_stat_global_reg ("pri 4 RL throughput (bps)");
t_gshandle_a[4] = op_stat_global_reg ("pri 5 RL throughput (bps)");
t_gshandle_a[5] = op_stat_global_reg ("pri 6 RL throughput (bps)");
t_gshandle_a[6] = op_stat_global_reg ("pri 7 RL throughput (bps)");
t_gshandle_a[7] = op_stat_global_reg ("pri 8 RL throughput (bps)");
t_gshandle_a[8] = op_stat_global_reg ("synch RL throughput (bps)");
t_gshandle_a[9] = op_stat_global_reg ("async RL throughput (bps)");
t_gshandle = op_stat_global_reg ("total RL throughput (bps)");
*/
link_gshandle[0] = op_stat_global_reg ("Link 0 jamming");
link_gshandle[1] = op_stat_global_reg ("Link 1 jamming");
link_gshandle[2] = op_stat_global_reg ("Link 2 jamming");
link_gshandle[3] = op_stat_global_reg ("Link 3 jamming");
jammer_stats_init = OPC_TRUE;

subq_no = 0;
```

```
  /* 7APR94:determine id of own processor to use in finding */
  /* load balancing attribute and station address of the bridge node */
  my_id = op_id_self();

  /* 15JUN94: get rate for link monitoring packet transmission */
  op_ima_obj_attr_get (my_id,"link_monitor_trans_rate", &link_mon_trans_rate );

  /* 26JUL94: get duration of the simulation */
  op_ima_sim_attr_get (OPC_IMA_DOUBLE, "duration", &sim_duration);

  /* 26JUL94:generate a self interrupt at each time that a monitoring packet*/
  /* is to be sent. Since these packets are to be sent at a fixed terate (per */
  /* second), the only way to guarantee packet generation in this event */
  /* driven simulation is to create an interrupt at the appropriate sim */
  /* times. These interrupts will cause a monitoring packet to be sent */
  /* through the discard state.*/

  for (time= op_sim_time(); time<=sim_duration; time+=link_mon_trans_rate)
    {
    op_intrpt_schedule_self (time, 0xc021);
   }


  }


/** state (INIT) exit executives **/
FSM_STATE_EXIT_FORCED (2, state2_exit_exec, "INIT")
  {
  }


/** state (INIT) transition processing **/
FSM_INIT_COND (END_OF_SIM)
FSM_DFLT_COND
FSM_TEST_LOGIC ("INIT")

FSM_TRANSIT_SWITCH
  {
  FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;)
  FSM_CASE_TRANSIT (1, 0, state0_enter_exec, ;)
  }
/*---------------------------------------------------------*/



}
```

```
    FSM_EXIT (2,cp_fddi_sink)
    }



void
cp_fddi_sink_svar (prs_ptr,var_name,var_p_ptr)
  cp_fddi_sink_state*prs_ptr;
  char    *var_name, **var_p_ptr;
  {

  FIN (cp_fddi_sink_svar (prs_ptr))

  *var_p_ptr = VOS_NIL;
  if (Vos_String_Equal ("thru_gshandle" , var_name))
    *var_p_ptr = (char *) (&prs_ptr->sv_thru_gshandle);
  if (Vos_String_Equal ("m_delay_gshandle" , var_name))
    *var_p_ptr = (char *) (&prs_ptr->sv_m_delay_gshandle);
  if (Vos_String_Equal ("ete_delay_gshandle" , var_name))
    *var_p_ptr = (char *) (&prs_ptr->sv_ete_delay_gshandle);
  if (Vos_String_Equal ("thru_gshandle_a" , var_name))
    *var_p_ptr = (char *) (prs_ptr->sv_thru_gshandle_a);
  if (Vos_String_Equal ("m_delay_gshandle_a" , var_name))
    *var_p_ptr = (char *) (prs_ptr->sv_m_delay_gshandle_a);
  if (Vos_String_Equal ("ete_delay_gshandle_a" , var_name))
    *var_p_ptr = (char *) (prs_ptr->sv_ete_delay_gshandle_a);
  if (Vos_String_Equal ("t_gshandle" , var_name))
    *var_p_ptr = (char *) (&prs_ptr->sv_t_gshandle);
  if (Vos_String_Equal ("t_gshandle_a" , var_name))
    *var_p_ptr = (char *) (prs_ptr->sv_t_gshandle_a);
  if (Vos_String_Equal ("my_id" , var_name))
    *var_p_ptr = (char *) (&prs_ptr->sv_my_id);
  if (Vos_String_Equal ("PPP_seq_number" , var_name))
    *var_p_ptr = (char *) (&prs_ptr->sv_PPP_seq_number);
  if (Vos_String_Equal ("time" , var_name))
    *var_p_ptr = (char *) (&prs_ptr->sv_time);

  FOUT;
  }



void
```

```
cp_fddi_sink_diag ()
  {
  double delay, creat_time;
  Packet*pkptr;
  Packet* pkptr1 ; /*5APR94*/
  Packet*ppp_pkptr;/*15JUN94*/
  int        src_addr, my_addr;
  int        dest_addr;/*14APR94*/
  Ici*    from_mac_ici_ptr;
  double fddi_sink_ttrt;
  int     xmit_subq_index;/*5APR94*/
  int        load_balance_code; /*6APR94*/
  int        i,subq_no; /*25APR94*/
  int        index; /*10MAY94 */
  double link_mon_trans_rate; /*15JUN94*/
  char    str0[512], str1 [512]; /* for diagnostics*/
  int        mon_pkt_size; /*26JUL*/
  double sim_duration; /*26JUL*/

  FIN (cp_fddi_sink_diag ())


  /* find out why straight line syndrome */

    op_prg_odb_print_major ("----------DEBUGGING for straight line---------",OPC_NIL);
    sprintf (str0,"count of packets : (%d)",subq_index);
   sprintf (str1,"subqueue no : (%d)",subq_no);
    op_prg_odb_print_minor (str0,str1, OPC_NIL);


  FOUT;
  }



void
cp_fddi_sink_terminate ()
  {
  double delay, creat_time;
  Packet*pkptr;
  Packet* pkptr1 ; /*5APR94*/
  Packet*ppp_pkptr;/*15JUN94*/
  int        src_addr, my_addr;
  int        dest_addr;/*14APR94*/
  Ici*    from_mac_ici_ptr;
  double fddi_sink_ttrt;
```

```
  int     xmit_subq_index;/*5APR94*/
  int       load_balance_code; /*6APR94*/
  int       i,subq_no; /*25APR94*/
  int       index; /*10MAY94 */
  double link_mon_trans_rate; /*15JUN94*/
  char    str0[512], str1 [512]; /* for diagnostics*/
  int       mon_pkt_size; /*26JUL*/
  double sim_duratic-- /*26JUL*/

  FIN (cp_fddi_sink_terminate ())


  FOUT;
  }



Compcode
cp_fddi_sink_init (pr_state_pptr)
  cp_fddi_sink_state**pr_state_pptr;
  {
  static VosT_Cm_Obtypeobtype = OPC_NIL;

  FIN (cp_fddi_sink_init (pr_state_pptr))

  if (obtype == OPC_NIL)
    {
    if (Vos_Catmem_Register ("proc state vars (cp_fddi_sink)",
      sizeof (cp_fddi_sink_state), Vos_Nop, &obtype) == VOSC_FAILURE)
      FRET (OPC_COMPCODE_FAILURE)
    }

  if ((*pr_state_pptr = (cp_fddi_sink_state*) Vos_Catmem_Alloc (obtype, 1)) == OPC_NIL)
    FRET (OPC_COMPCODE_FAILURE)
  else
    {
    (*pr_state_pptr)->current_block = 4;
    FRET (OPC_COMPCODE_SUCCESS)
    }
  }
```

APPENDIX D

RING 1 LLC_SRC MODULE CODE "sp_fddi_gen.pr.c"

```
/* Process model C form file: sp_fddi_gen.pr.c */
/* Portions of this file Copyright (C) MIL 3, Inc. 1992 */



/* OPNET system definitions */
#include <opnet.h>
#include "sp_fddi_gen.pr.h"
FSM_EXT_DECS



/* Header block */
#define MAC_LAYER_OUT_STREAM0
#define LLC_SINK_OUT_STREAM1 /*18APR94*/

/* define possible service classes for frames */
#define FDDI_SVC_ASYNC0
#define FDDI_SVC_SYNC1

/* define token classes */
#define FDDI_TK_NONRESTRICTED0
#define FDDI_TK_RESTRICTED1

/* define output statistics */
#define RATIO_OUTSTAT0
#define LINK_STATUS_OUTSTAT 1

/* link_status constants 30 JUL94*/
#define GOOD        0
#define BAD         2

/* history trend constants 30JUL94*/
#define DOWN        0
#define UP          1
```

```
/* redefine absolute value function for floating pt values 28JUL94*/
#define abs(i) (i)<0 ? -(i) : (i)

struct history_element
/* format for linked list of history values 26JUL94*/
{
    int num_errs;
    struct history_element *next;
} *history;

/* function declaration 28JUL94*/
struct history_element *create_history();



/* State variable definitions */
typedef struct
    {
    FSM_SYS_STATE
    Distribution* sv_inter_dist_ptr;
    Distribution* sv_len_dist_ptr;
    Distribution* sv_dest_dist_ptr;
    Distribution* sv_pkt_priority_ptr;
    Objid       sv_mac_objid;
    Objid       sv_my_id;
    int         sv_low_dest_addr;
    int         sv_high_dest_addr;
    int         sv_station_addr;
    int         sv_src_addr;
    int       sv_low_pkt_priority;
    int       sv_high_pkt_priority;
    int         sv_ppp_pid_h;
    int         sv_ppp_pid_l;
    double      sv_arrival_rate;
    double      sv_mean_pk_len;
    double      sv_async_mix;
    Ici*        sv_mac_iciptr;
    Ici*        sv_mac_iciptr1;
    Ici*        sv_llc_ici_ptr;
    Packet*     sv_pkptr1;
    Packet*     sv_ppp_pkptr1;
    Packet*     sv_ppp_pkptr2;
```

```
    int             sv_hist_len;
    int             sv_link_status;
    int             sv_pkts_in_error;
    double          sv_old_ratio;
    } sp_fddi_gen_state;

#define pr_state_ptr            ((sp_fddi_gen_state*) Siml_Mod_State_Ptr)
#define inter_dist_ptr          pr_state_ptr->sv_inter_dist_ptr
#define len_dist_ptr            pr_state_ptr->sv_len_dist_ptr
#define dest_dist_ptr           pr_state_ptr->sv_dest_dist_ptr
#define pkt_priority_ptr        pr_state_ptr->sv_pkt_priority_ptr
#define mac_objid               pr_state_ptr->sv_mac_objid
#define my_id                   pr_state_ptr->sv_my_id
#define low_dest_addr           pr_state_ptr->sv_low_dest_addr
#define high_dest_addr          pr_state_ptr->sv_high_dest_addr
#define station_addr            pr_state_ptr->sv_station_addr
#define src_addr                pr_state_ptr->sv_src_addr
#define low_pkt_priority        pr_state_ptr->sv_low_pkt_priority
#define high_pkt_priority       pr_state_ptr->sv_high_pkt_priority
#define ppp_pid_h               pr_state_ptr->sv_ppp_pid_h
#define ppp_pid_l               pr_state_ptr->sv_ppp_pid_l
#define arrival_rate            pr_state_ptr->sv_arrival_rate
#define mean_pk_len             pr_state_ptr->sv_mean_pk_len
#define async_mix               pr_state_ptr->sv_async_mix
#define mac_iciptr              pr_state_ptr->sv_mac_iciptr
#define mac_iciptr1             pr_state_ptr->sv_mac_iciptr1
#define llc_ici_ptr             pr_state_ptr->sv_llc_ici_ptr
#define pkptr1                  pr_state_ptr->sv_pkptr1
#define ppp_pkptr1              pr_state_ptr->sv_ppp_pkptr1
#define ppp_pkptr2              pr_state_ptr->sv_ppp_pkptr2
#define hist_len                pr_state_ptr->sv_hist_len
#define link_status             pr_state_ptr->sv_link_status
#define pkts_in_error           pr_state_ptr->sv_pkts_in_error
#define old_ratio               pr_state_ptr->sv_old_ratio



/* Process model interrupt handling procedure */


void
```

```
sp_fddi_gen ()
    {
    Packet    *pkptr;
    int           pklen;
    int           dest_addr;
    int           i,j, restricted;
    int pkt_prio;
    int           num_errors;
    double    new_ratio, upper_thresh, lower_thresh;
    double    LQR_trans_delta;


  FSM_ENTER (sp_fddi_gen)

  FSM_BLOCK_SWITCH
    {
    /*--------------------------------------------------------*/
    /** state (INIT) enter executives **/
    FSM_STATE_ENTER_UNFORCED (0, state0_enter_exec, "INIT")
      {
      /* determine id of own processor to use in finding attrs */
      my_id = op_id_self ();

      /* determine address range for uniform desination assignment */
      op_ima_obj_attr_get (my_id, "low dest address", &low_dest_addr);
      op_ima_obj_attr_get (my_id, "high dest address", &high_dest_addr);

      /* determine object id of connected 'mac' layer process */
      mac_objid = op_topo_assoc (my_id, OPC_TOPO_ASSOC_OUT,
          OPC_OBJMTYPE_MODULE, MAC_LAYER_OUT_STREAM);

      /* determine the address assigned to it */
      /* which is also the address of this station */
      op_ima_obj_attr_get (mac_objid, "station_address", &station_addr);



       /* set up a distribution for generation of addresses */
         dest_dist_ptr = op_dist_load ("uniform_int", low_dest_addr,
       high_dest_addr);

         /* added 26DEC93 */
         /* determine priority range for uniform traffic generation */
```

```
op_ima_obj_attr_get (my_id, "high pkt priority", &high_pkt_priority);
op_ima_obj_attr_get (my_id, "low pkt priority", &low_pkt_priority);

/* set up a distribution for generation of priorities */
pkt_priority_ptr = op_dist_load ("uniform_int", low_pkt_priority, high_pkt_priority);

/* above added 26DEC93 */


/* also determine the arrival rate for packet generation */
op_ima_obj_attr_get (my_id, "arrival rate", &arrival_rate);

/* determine the mix of asynchronous and synchronous */
/* traffic. This is expressed as the proportion of */
/* asynchronous traffic. i.e a value of 1.0 indicates */
/* that all the produced traffic shall be asynchronous. */
op_ima_obj_attr_get (my_id, "async_mix", &async_mix);

/* set up a distribution for arrival generations */
if (arrival_rate != 0.0)
   {
   /* arrivals are exponentially distributed, with given mean */
   inter_dist_ptr = op_dist_load ("constant", 1.0 / arrival_rate, 0.0);

   /* determine the distribution for packet size */
   op_ima_obj_attr_get (my_id, "mean pk length", &mean_pk_len);

   /* set up corresponding distribution */
   len_dist_ptr = op_dist_load ("constant", mean_pk_len, 0.0);

   /* designate the time of first arrival */
   fddi_gen_schedule ();

   /* set up an interface control information (ICI) structure */
   /* to communicate parameters to the mac layer process */
   /* (it is more efficient to set one up now and keep it */
   /* as a state variable than to allocate one on each packet xfer) */
   mac_iciptr = op_ici_create ("fddi_mac_req");
   }

/* set up history array (dynamically allocated) which will maintain */
/* number of errors in each monitoring packet rcvd. The number of */
/* values (length of the array) saved will be determined by an */
```

```
  /* environment attribute. 21JUL94 */
  op_ima_obj_attr_get (my_id, "history length", &hist_len);
  history = create_history(hist_len);
  printf ("HISTORY\n");
  if (op_sim_debug() == OPC_TRUE)
    {
    for (i=1;i<=hist_len;++i)
      {
     printf ("%d ",history->num_ers);
      history = history->next;
      }
    printf ("\n");
   }
  }


/** blocking after enter executives of unforced state. **/
FSM_EXIT (1,sp_fddi_gen)


/** state (INIT) exit executives **/
FSM_STATE_EXIT_UNFORCED (0, state0_exit_exec, "INIT")
  {
  }


/** state (INIT) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
/*---------------------------------------------------------*/



/** state (ARRIVAL) enter executives **/
FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "ARRIVAL")
  {
  /* This station should receive frames from the other lan as long as */
  /* there are frames in the input streams addressed to this lan */
  /*check if the interrupt type is stream interrupt *//*12APR94*/
  if(op_intrpt_type() == OPC_INTRPT_STRM)
    {
    /* if it is, get the packet in the input stream causing interrupt */
    /* modified for PPP 12JUL94 */
    ppp_pkptr1 = op_pk_get(op_intrpt_strm());
```

```
  /* determine type of PPP packet 12JUL94 */
  op_pk_nfd_get(ppp_pkptr1,"pid_h", &ppp_pid_h);
  /* case on pid_h: 00 - data, 0xc0 - control 13JUL*/
switch (ppp_pid_h)
    {
    case 0x00: /* if data, strip header and send as FDDI frame */
      /* strip off PPP header 12JUL94*/
      if (op_sim_debug()==OPC_TRUE)
        {
        printf ("pkt rcvd at sp: data\n");
        num_errors =op_td_get_int(ppp_pkptr1,OPC_TDA_PT_NUM_ERRORS);
        printf ("number of errors in data - %d\n",num_errors);
        }
      op_pk_nfd_get(ppp_pkptr1,"FDDI_frame",&pkptr1);
      op_pk_destroy(ppp_pkptr1);
      /* get the destination address of the frame : 16APR94 */
      op_pk_nfd_get(pkptr1, "dest_addr", &dest_addr);
      /* check if this frame is for the remote bridge station (bridge in surface lan) */
      if(dest_addr == station_addr)
        /* if it is, send the packet to llc_sink directly */
        /* in order to prevent overhead of mac access */
        op_pk_send(pkptr1, LLC_SINK_OUT_STREAM);/*19APR94*/
      else
        /* this packet is to send to mac */
        {
        /* determine the source address of the frame */
        op_pk_nfd_get(pkptr1, "src_addr", &src_addr);
        /* set up an ICI structure to communicate parameters to */
        /* MAC layer process */
        mac_iciptr1 = op_ici_create("fddi_mac_req");
        /* place the original source address into the ICI *//* 16APR94 */
        /* "fddi_mac_req" is modified so that it contains the original */
        /* source address from the local lan(collection platform) */
        op_ici_attr_set(mac_iciptr1, "src_addr", src_addr);
        /* place the destination address into the ICI */ /*12APR94*/
        op_ici_attr_set(mac_iciptr1, "dest_addr", dest_addr);
        /* assign the service class and requested token class */
        /* At this moment the frames coming from the remote lan are assumed */
        /* to have the same priority as synchronous frames in order not to accumulate */
        /* packets on the bridge station mac and instead to deliver their destinations */
        /* as soon as possible */
        op_pk_nfd_set(pkptr1, "pri", 8);
        op_ici_attr_set(mac_iciptr1, "svc_class", FDDI_SVC_SYNC);
```

```
                op_ici_attr_set(mac_iciptr1, "pri", 8);
                op_ici_attr_set(mac_iciptr1, "tk_class", FDDI_TK_NONRESTRICTED);
                /* send the packet coupled with the ICI */
                op_ici_install(mac_iciptr1);
                op_pk_send(pkptr1, MAC_LAYER_OUT_STREAM);
              }
            break;
          case 0xc0: /* either monitoring packet or LQR */
            op_pk_nfd_get (ppp_pkptr1,"pid_1",&ppp_pid_1);
            switch (ppp_pid_1)
              {
              case 0x21: /*monitoring packet*/
                printf ("MONITORING PACKET RECEIVED\n");
                num_errors =op_td_get_int(ppp_pkptr1,OPC_TDA_PT_NUM_ERRORS);
                printf (" NUMBER OF ERRORS -->%d\n",num_errors);
                if ((num_errors != 0) && (history->num_errs==0))
                   pkts_in_error++;
                else if ((num_errors == 0) && (history->num_errs != 0))
                   pkts_in_error--;
                printf (" TOTAL NUMBER OF PACKETS IN ERROR --
>%d\n",pkts_in_error);

                history->num_errs = num_errors;
                history = history->next;

                if (op_sim_debug()==OPC_TRUE)
                /* print out history values 30JUL94 */
                   {
                   for (i=1;i<=hist_len;++i)
                   {
                   printf ("%d " history->num_errs);
                   history = history->next;
                   }
                   printf ("\n");
      }

                op_pk_destroy (ppp_pkptr1);

                new_ratio = (double)pkts_in_error/(double)hist_len;
                /* outstat(0) will be a record of the ratio 8AUG94*/
                op_stat_local_write (RATIO_OUTSTAT, new_ratio);
                op_ima_obj_attr_get (my_id, "LQR transmission delta", &LQR_trans_delta);
                if (op_sim_debug()==OPC_TRUE)
```

```
                    printf ("LQR delta - %f new_ratio - %f old_ratio -
%f\n",LQR_trans_delta,new_ratio,old_ratio);
                /* 8AUG94 This condition allows for tolerance in the ratio
calculation(division) */
                if (abs(new_ratio - old_ratio) >= LQR_trans_delta)
                    {
                    ppp_pkptr2 = op_pk_create_fmt ("ppp");
                    op_pk_nfd_set (ppp_pkptr2, "pid_h", 0xc0);
                    op_pk_nfd_set (ppp_pkptr2, "pid_l", 0x25);

                    op_ima_obj_attr_get (my_id, "upper hysteresis threshold",
&upper_thresh);
                    op_ima_obj_attr_get (my_id, "lower hysteresis threshold",
&lower_thresh);

                    /* status = GOOD 0 or BAD 2 trend = UP 1 or DOWN 0 2AUG94*/
        if (new_ratio >= upper_thresh)
                    link_status = BAD;
                    else if (new_ratio <= lower_thresh)
                    link_status = GOOD;
                    else link_status = link_status - (link_status % 2);/* remove trend value*/

                    /* outstat(1) will monitor link status 0-GOOD 1-BAD 8AUG94*/
                    op_stat_local_write (LINK_STATUS_OUTSTAT, (double) (link_status/
2));

                    /* trend will change each time */
                    if (new_ratio > old_ratio) /*up-trend */
                    link_status += UP;
                    else link_status += DOWN;



                    op_pk_nfd_set (ppp_pkptr2, "LQR_info", link_status);
                    if (op_sim_debug()==OPC_TRUE)
                    {
                    printf ("resulting LQR pkt\n");
                    op_pk_print(ppp_pkptr2);
                    }
                    old_ratio = new_ratio;
                    op_pk_send (ppp_pkptr2, LLC_SINK_OUT_STREAM);
                    }
                if (op_sim_debug()==OPC_TRUE)
```

```
                printf (" Latest ratio: %f\n",new_ratio);
            break;

            case 0x25:/*LQR from cp - not implemented yet*/
            break;
            default:
              printf ("ERROR: UNKNOWN PPP PACKET - pid_l=0x%x\n",ppp_pid_l);
              op_pk_destroy (ppp_pkptr1);
            break;
            }
     break;
       default:
         printf ("ERROR: UNKNOWN PPP PACKET - pid_h=0x%x\n",ppp_pid_h);
         op_pk_destroy (ppp_pkptr1);
       break;
       }

    }
/* otherwise, generate the frame */
else
    {
    /* determine the length of the packet to be generated */
    pklen = op_dist_outcome (len_dist_ptr);

    /* determine the destination */
    /* dont allow this station's address as a possible outcome */
    gen_packet:
    dest_addr = op_dist_outcome (dest_dist_ptr);
    if (dest_addr != -1 && dest_addr == station_addr)
      goto gen_packet;

    /* 26DEC94 & 29JAN94: determine its priority */
    pkt_prio = op_dist_outcome (pkt_priority_ptr);

    /* create a packet to send to mac */
    pkptr = op_pk_create_fmt ("fddi_llc_fr");

    /* assign its overall size. */
    op_pk_total_size_set (pkptr, pklen);

    /* assign the time of creation */
    op_pk_nfd_set (pkptr, "cr_time", op_sim_time ());
```

```
  /* place the destination address into the ICI */
  /* (the protocol_type field will default) */
  op_ici_attr_set (mac_iciptr, "dest_addr", dest_addr);

  /* place the source address into the ICI *//* 17APR94*/
  op_ici_attr_set (mac_iciptr, "src_addr", station_addr);

  /* assign the priority, and requested token class */
  /* also assign the service class; 29JAN94: the fddi_llc_fr */
  /* format is modified to include a "pri" field. */
  if (op_dist_uniform (1.0) <= async_mix)
    {
  op_pk_nfd_set (pkptr, "pri", pkt_prio); /* 29JAN94 */
    op_ici_attr_set (mac_iciptr, "svc_class", FDDI_SVC_ASYNC);
  op_ici_attr_set (mac_iciptr, "pri", pkt_prio); /* 29JAN94 */
    }
  else{
  op_pk_nfd_set (pkptr, "pri", 8); /* 29JAN94 */
    op_ici_attr_set (mac_iciptr, "svc_class", FDDI_SVC_SYNC);
  op_ici_attr_set (mac_iciptr, "pri", 8); /* 29JAN94 */
    }

  /* Request only nonrestricted tokens after transmission */
  op_ici_attr_set (mac_iciptr, "tk_class", FDDI_TK_NONRESTRICTED);

  /* Having determined priority, assign it; 26DEC93 */
  /* op_ici_attr_set (mac_iciptr, "pri", pkt_prio); */

  /* send the packet coupled with the ICI */
  op_ici_install (mac_iciptr);
  /* check if destination address is in the local lan(collection platform)*/
  if(dest_addr <= 9)
    /* if it is, this packet is to send llc_sink directly */
    op_pk_send (pkptr, LLC_SINK_OUT_STREAM); /*18APR94*/
  else
    /* if not, the packet is destined for remote lan (surface stations)*/
    op_pk_send (pkptr, MAC_LAYER_OUT_STREAM);

  /* schedule the next arrival */
  fddi_gen_schedule ();
}
}
```

```
    /** blocking after enter executives of unforced state. **/
    FSM_EXIT (3,sp_fddi_gen)


    /** state (ARRIVAL) exit executives **/
    FSM_STATE_EXIT_UNFORCED (1, state1_exit_exec, "ARRIVAL")
      {
      }


    /** state (ARRIVAL) transition processing **/
    FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
    /*---------------------------------------------------------*/



    }


  FSM_EXIT (0,sp_fddi_gen)
  }



void
sp_fddi_gen_svar (prs_ptr,var_name,var_p_ptr)
  sp_fddi_gen_state*prs_ptr;
  char    *var_name, **var_p_ptr;
  {

  FIN (sp_fddi_gen_svar (prs_ptr))

  *var_p_ptr = VOS_NIL;
  if (Vos_String_Equal ("inter_dist_ptr" , var_name))
    *var_p_ptr = (char *) (&prs_ptr->sv_inter_dist_ptr);
  if (Vos_String_Equal ("len_dist_ptr" , var_name))
    *var_p_ptr = (char *) (&prs_ptr->sv_len_dist_ptr);
  if (Vos_String_Equal ("dest_dist_ptr" , var_name))
    *var_p_ptr = (char *) (&prs_ptr->sv_dest_dist_ptr);
  if (Vos_String_Equal ("pkt_priority_ptr" , var_name))
    *var_p_ptr = (char *) (&prs_ptr->sv_pkt_priority_ptr);
```

```
if (Vos_String_Equal ("mac_objid" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_mac_objid);
if (Vos_String_Equal ("my_id" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_my_id);
if (Vos_String_Equal ("low_dest_addr" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_low_dest_addr);
if (Vos_String_Equal ("high_dest_addr" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_high_dest_addr);
if (Vos_String_Equal ("station_addr" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_station_addr);
if (Vos_String_Equal ("src_addr" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_src_addr);
if (Vos_String_Equal ("low_pkt_priority" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_low_pkt_priority);
if (Vos_String_Equal ("high_pkt_priority" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_high_pkt_priority);
if (Vos_String_Equal ("ppp_pid_h" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_ppp_pid_h);
if (Vos_String_Equal ("ppp_pid_l" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_ppp_pid_l);
if (Vos_String_Equal ("arrival_rate" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_arrival_rate);
if (Vos_String_Equal ("mean_pk_len" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_mean_pk_len);
if (Vos_String_Equal ("async_mix" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_async_mix);
if (Vos_String_Equal ("mac_iciptr" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_mac_iciptr);
if (Vos_String_Equal ("mac_iciptr1" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_mac_iciptr1);
if (Vos_String_Equal ("llc_ici_ptr" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_llc_ici_ptr);
if (Vos_String_Equal ("pkptr1" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_pkptr1);
if (Vos_String_Equal ("ppp_pkptr1" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_ppp_pkptr1);
if (Vos_String_Equal ("ppp_pkptr2" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_ppp_pkptr2);
if (Vos_String_Equal ("hist_len" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_hist_len);
if (Vos_String_Equal ("link_status" , var_name))
   *var_p_ptr = (char *) (&prs_ptr->sv_link_status);
if (Vos_String_Equal ("pkts_in_error" , var_name))
```

```
    *var_p_ptr = (char *) (&prs_ptr->sv_pkts_in_error);
  if (Vos_String_Equal ("old_ratio" , var_name))
    *var_p_ptr = (char *) (&prs_ptr->sv_old_ratio);

  FOUT;
  }



void
sp_fddi_gen_diag ()
  {
  Packet*pkptr;
  int     pklen;
  int     dest_addr;
  int     i,j, restricted;
  int pkt_pric;
  int     num_errors;
  doublenew_ratio, upper_thresh, lower_thresh;
  doubleLQR_trans_delta;

  FIN (sp_fddi_gen_diag ())


  FOUT;
  }



void
sp_fddi_gen_terminate ()
  {
  Packet*pkptr;
  int     pklen;
  int     dest_addr;
  int     i,j, restricted;
  int pkt_prio;
  int     num_errors;
  doublenew_ratio, upper_thresh, lower_thresh;
  doubleLQR_trans_delta;
```

```
    *var_p_ptr = (char *) (&prs_ptr->sv_pkts_in_error);
  if (Vos_String_Equal ("old_ratio" , var_name))
    *var_p_ptr = (char *) (&prs_ptr->sv_old_ratio);

  FOUT;
  }



void
sp_fddi_gen_diag ()
  {
  Packet*pkptr;
  int     pklen;
  int     dest_addr;
  int     i,j, restricted;
  int pkt_prio;
  int     num_errors;
  doublenew_ratio, upper_thresh, lower_thresh;
  doubleLQR_trans_delta;

  FIN (sp_fddi_gen_diag ())


  FOUT;
  }



void
sp_fddi_gen_terminate ()
  {
  Packet*pkptr;
  int     pklen;
  int     dest_addr;
  int     i,j, restricted;
  int pkt_prio;
  int     num_errors;
  doublenew_ratio, upper_thresh, lower_thresh;
  doubleLQR_trans_delta;
```

```
   /* schedule the arrival of next generated packet */
   op_intrpt_schedule_self (op_sim_time () + inter_time, 0);
   }

struct history_element *create_history(size)
   int size;

/* returns a pointer to the first element in a circular linked list */
   {
   struct history_element *p, *new, *start;
 int i;

   for (i=1;i<=size;++i)
      {
      printf ("inside on iteration %d %d\n",i,size);
      new = (struct history_element*)malloc(sizeof(struct history_element));
      if (!new)
         {
         printf("ERROR: MEMORY ALLOCATION");
         exit(1);
         }
      if (i==1)
         {
         p = new;
         start = new;
         }
         else p->next = new; /* stick new element on end of list*/
    new->next = NULL;
      new->num_errs = 0;
    p = new;
 }
   p->next = start;
 return (start);
 }
```

APPENDIX E

RING 1 LLC_SINK MODULE CODE "sp_fddi_sink.pr.c"

```
/* Process model C form file: sp_fddi_sink.pr.c */
/* Portions of this file Copyright (C) MIL 3, Inc. 1992 */



/* OPNET system definitions */
#include <opnet.h>
#include "sp_fddi_sink.pr.h"
FSM_EXT_DECS



/* Header block */
/* Globals */
/* array format installed 20JAN94; positions 0-7 represent the asynch priority levels, PRIORITIES
+ 1 */
/* represents synch traffic, and grand totals are as given in the original. */

#define PRIORITIES 8 /* 20JAN94 */
#define   XMITTER_BUSY 0 /*10MAY94 */

#define   LLC_SOURCE_INPUT_STREAM 1 /*26JUL94*/
#define   MAC_INPUT_STREAM 0

static /* 05FEB94 */
double fddi2_sink_accum_delay = 0.0;
static /* 05FEB94 */
double    fddi2_sink_accum_delay_a[PRIORITIES + 1] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0};
static /* 05FEB94 */
int fddi2_sink_total_pkts = 0;
static /* 05FEB94 */
int       fddi2_sink_total_pkts_a[PRIORITIES + 1] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
static /* 05FEB94 */
double fddi2_sink_total_bits = 0.0;
static /* 05FEB94 */
double    fddi2_sink_total_bits_a[PRIORITIES + 1] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
static /* 05FEB94 */
```

```
double fddi2_sink_peak_delay = 0.0;
static /* 05FEB94 */
double    fddi2_sink_peak_delay_a[PRIORITIES + 2] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
static /* 05FEB94 */
int       fddi2_sink_scalar_write = 0;
static /* 05FEB94 */
int pri2_set = 20; /* 20JAN94 */
double busy = 0.0; /* 10MAY94 */

/* Statistics used for command link:21APR94 */
static
int fddilp2_total_pkts = 0;
static
int       fddilp2_total_pkts_a[PRIORITIES + 1] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
static
double fddilp2_total_bits = 0.0;
static
double    fddilp2_total_bits_a[PRIORITIES + 1] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0. 0.0};

/* Externally defined globals. */
extern doublefddi_t_opr [];

/*12JAN94:attributes from the Environment file */
double Offered_Load; /* 12JAN94 */
double Asynch_Offered_Load; /* 12JAN94 */

/* transition expressions */
#define END_OF_SIM op_intrpt_type() == OPC_INTRPT_ENDSIM



/* State variable definitions */
typedef struct
        {
        FSM_SYS_STATE
        Gshandle                sv_thru2_gshandle;
        Gshandle                sv_m2_delay_gshandle;
        Gshandle                sv_ete2_delay_gshandle;
        Gshandle                sv_thru2_gshandle_a[10];
        Gshandle                sv_m2_delay_gshandle_a[10];
        Gshandle                sv_ete2_delay_gshandle_a[9];
        Gshandle                sv_t2_gshandle;
        Gshandle                sv_t2_gshandle_a[10];
        Objid                   sv_my_id;
        } sp_fddi_sink_state;
```

```
#define pr_state_ptr            ((sp_fddi_sink_state*) SimI_Mod_State_Ptr)
#define thru2_gshandle          pr_state_ptr->sv_thru2_gshandle
#define m2_delay_gshandle       pr_state_ptr->sv_m2_delay_gshandle
#define ete2_delay_gshandle     pr_state_ptr->sv_ete2_delay_gshandle
#define thru2_gshandle_a        pr_state_ptr->sv_thru2_gshandle_a
#define m2_delay_gshandle_a     pr_state_ptr->sv_m2_delay_gshandle_a
#define ete2_delay_gshandle_a   pr_state_ptr->sv_ete2_delay_gshandle_a
#define t2_gshandle             pr_state_ptr->sv_t2_gshandle
#define t2_gshandle_a           pr_state_ptr->sv_t2_gshandle_a
#define my_id                   pr_state_ptr->sv_my_id



/* Process model interrupt handling procedure */



void
sp_fddi_sink ()
  {
  double    delay, creat_time;
  Packet*   pkptr;
  Packet*   ppp_pkptr;
  Packet* pkptr1 ; /*5APR94*/
  int           src_addr, my_addr;
  int           dest_addr;/*14APR94*/
  Ici*      from_mac_ici_ptr;
  double    fddi_sink_trt;
  char      pk_format[10];
  int           input_stream;
  int           fd_index, FDDI_frame_size;


  FSM_ENTER (sp_fddi_sink)

  FSM_BLOCK_SWITCH
    {
    /*--------------------------------------------------------*/
    /** state (DISCARD) enter executives **/
    FSM_STATE_ENTER_UNFORCED (0, state0_enter_exec, "DISCARD")
      {
      /* determine the type of interrupt */
      switch(op_intrpt_type())
        {
        /* check if transmitter is busy */
        case OPC_INTRPT_STAT:
```

```
            {
            busy = op_stat_local_read (XMITTER_BUSY);
            break;
            }
          /* check if a packet has arrived */
          case OPC_INTRPT_STRM:
            {
            /* get the packet*/
            input_stream = op_intrpt_strm();
            pkptr = op_pk_get (input_stream);
            op_pk_format(pkptr,pk_format);

            /* if the packet is a ppp control packet from the llc_source */
            if ((strcmp(pk_format,"ppp")==0) &&(input_stream == LLC_SOURCE_INPUT_-
STREAM))
              {
              /* bypass all this and send pkt out on the command link */
              }
           else
              {
              /* assume this is a FDDI frame */
              from_mac_ici_ptr = op_intrpt_ici ();

              /* 20JAN94: get the packet's priority level, which */
              /* will be used to index arrays of thruput and delay */
              /* computations. */
              /* pri2_set = op_pk_priority_get (pkptr); doesn't work here */
              op_pk_nfd_get (pkptr, "pri", &pri2_set); /* 29JAN94 */

              /* determine the time of creation of the packet */
              op_pk_nfd_get (pkptr, "cr_time", &creat_time);

              /* determine the dest address of the packet */ /*18APR94*/
              op_pk_nfd_get (pkptr, "dest_addr", &dest_addr);

              /* 7APR94:determine id of own processor to use in finding */
              /* station address of the bridge node */
              my_id = op_id_self();

              /* 14APR94 : also get my own address */
              op_ima_obj_attr_get ( my_id, "station_address", &my_addr);

              /* destroy the packet */
              /* op_pk_destroy (pkptr); */
              /* 03FEB94: rather, enqueue the packet. This will be the */
              /* first step toward developing a LAN bridging structure. */
              /* -Nix */
```

```
/* op_subq_pk_insert (pri_set, pkptr, OPC_QPOS_TAIL); */

/* 14APR94: check the frame passed to "llc" is destined for */
/* this station. If it is destroy the packet; if not, allocate the packets */
/* to the command link transmitter since they are destined for the remote lan */
/* -Karayakaylar */
/* determine the packets coming from surface stations, this will */
/* be counted for local traffic */
/* 9(nine) is model specific, this is the "station_number" of */
/* collection platform bridge station */
if((dest_addr == my_addr)&&(src_addr > 9))
  {
  /* add in its size */
  fddi2_sink_total_bits += op_pk_total_size_get (pkptr);
  fddi2_sink_total_bits_a[pri2_set] += op_pk_total_size_get (pkptr); /* 20JAN-
20APR94 */

  /* accumulate delays */
  delay = op_sim_time () - creat_time;
  fddi2_sink_accum_delay += delay;
  fddi2_sink_accum_delay_a[pri2_set] += delay; /* 20JAN-20APR94 */

  /* keep track of peak delay value */
  if (delay > fddi2_sink_peak_delay)
    fddi2_sink_peak_delay = delay;

  /* 20JAN94: keep track by priority levels as well 23JAN-20APR94 */
  if (delay > fddi2_sink_peak_delay_a[pri2_set])
  fddi2_sink_peak_delay_a[pri2_set] = delay;

  op_pk_destroy (pkptr);

  /* increment packet counter; 20JAN94 */
  fddi2_sink_total_pkts++;
  fddi2_sink_total_pkts_a[pri2_set]++;

  /* if a multiple of 25 packets is reached, update stats */
  /* 03FEB94: [0]->[7] represent asynch priorities 1->8, */
  /* respectively; [8] represents synchronous traffic, */
  /* and [9] represents overall asynchronous traffic.-Nix */
  if (fddi2_sink_total_pkts % 25 == 0)
    {
    op_stat_global_write (thru2_gshandle,
    fddi2_sink_total_bits / op_sim_time ());

  op_stat_global_write (thru2_gshandle_a[pri2_set],
fddi2_sink_total_bits_a[0] / op_sim_time());
```

```
        op_stat_global_write (thru2_gshandle_a[0],
    fddi2_sink_total_bits_a[1] / op_sim_time());
        op_stat_global_write (thru2_gshandle_a[1],
    fddi2_sink_total_bits_a[pri2_set] / op_sim_time());
        op_stat_global_write (thru2_gshandle_a[2],
    fddi2_sink_total_bits_a[2] / op_sim_time());
        op_stat_global_write (thru2_gshandle_a[3],
    fddi2_sink_total_bits_a[3] / op_sim_time());
        op_stat_global_write (thru2_gshandle_a[4],
    fddi2_sink_total_bits_a[4] / op_sim_time());
        op_stat_global_write (thru2_gshandle_a[5],
    fddi2_sink_total_bits_a[5] / op_sim_time());
        op_stat_global_write (thru2_gshandle_a[6],
    fddi2_sink_total_bits_a[6] / op_sim_time());
        op_stat_global_write (thru2_gshandle_a[7],
    fddi2_sink_total_bits_a[7] / op_sim_time());
        op_stat_global_write (thru2_gshandle_a[8],
   fddi2_sink_total_bits_a[8] / op_sim_time());

          /* 30JAN94: gather all asynch stats into one overall figure */
        op_stat_global_write (thru2_gshandle_a[9],
    (fddi2_sink_total_bits - fddi2_sink_total_bits_a[8]) /
    op_sim_time());


/* (fddi2_sink_total_bits_a[0] + fddi2_sink_total_bits_a[1] + */
/* fddi2_sink_total_bits_a[2] + fddi2_sink_total_bits_a[3] + */
/* fddi2_sink_total_bits_a[4] + fddi2_sink_total_bits_a[5] + */
/* fddi2_sink_total_bits_a[6] + fddi2_sink_total_bits_a[7]) / */
/* op_sim_time()); */


        op_stat_global_write (m2_delay_gshandle,
          fddi2_sink_accum_delay / fddi2_sink_total_pkts);

        op_stat_global_write (m2_delay_gshandle_a[0],
    fddi2_sink_accum_delay_a[0] / fddi2_sink_total_pkts_a[0]);
        op_stat_global_write (m2_delay_gshandle_a[1],
    fddi2_sink_accum_delay_a[1] / fddi2_sink_total_pkts_a[1]);
        op_stat_global_write (m2_delay_gshandle_a[2],
   fddi2_sink_accum_delay_a[2] / fddi2_sink_total_pkts_a[2]);
          op_stat_global_write (m2_delay_gshandle_a[3],
    fddi2_sink_accum_delay_a[3] / fddi2_sink_total_pkts_a[3]);
          op_stat_global_write (m2_delay_gshandle_a[4],
    fddi2_sink_accum_delay_a[4] / fddi2_sink_total_pkts_a[4]);
        op_stat_global_write (m2_delay_gshandle_a[5],
   fddi2_sink_accum_delay_a[5] / fddi2_sink_total_pkts_a[5]);
```

```
            op_stat_global_write (m2_delay_gshandle_a[6],
        fddi2_sink_accum_delay_a[6] / fddi2_sink_total_pkts_a[6]);
            op_stat_global_write (m2_delay_gshandle_a[7],
        fddi2_sink_accum_delay_a[7] / fddi2_sink_total_pkts_a[7]);
            op_stat_global_write (m2_delay_gshandle_a[8],
        fddi2_sink_accum_delay_a[8] / fddi2_sink_total_pkts_a[8]);

            /* 30JAN94: gather all asynch stats into one figure */
            op_stat_global_write (m2_delay_gshandle_a[9],
        (fddi2_sink_accum_delay - fddi2_sink_accum_delay_a[8]) /
        (fddi2_sink_total_pkts - fddi2_sink_total_pkts_a[8]));


    /* (fddi2_sink_accum_delay_a[0] + fddi2_sink_accum_delay_a[1] + */
    /* fddi2_sink_accum_delay_a[2] + fddi2_sink_accum_delay_a[3] + */
    /* fddi2_sink_accum_delay_a[4] + fddi2_sink_accum_delay_a[5] + */
    /* fddi2_sink_accum_delay_a[6] + fddi2_sink_accum_delay_a[7]) / */
    /* (fddi2_sink_total_pkts_a[0] + fddi2_sink_total_pkts_a[1] + */
    /* fddi2_sink_total_pkts_a[2] + fddi2_sink_total_pkts_a[3] + */
    /* fddi2_sink_total_pkts_a[4] + fddi2_sink_total_pkts_a[5] + */
    /* fddi2_sink_total_pkts_a[6] + fddi2_sink_total_pkts_a[7])); */


            /* also record actual delay values */
            op_stat_global_write (ete2_delay_gshandle, delay);
            op_stat_global_write (ete2_delay_gshandle_a[pri2_set], delay);
            }
          }/*end of if(dest_addr==my_addr)&&(src_addr > 9)statement */

        /* 20APR94: destroy the packets coming from the first lan destined */
        /* for this station.These packets are not counted for local traffic.*/
        else if(dest_addr == my_addr)
          op_pk_destroy(pkptr);


        /* Other frames passed to "llc" should be destined for other lan */
        /* 18APR94 :allocate the packets to transmitter of command link */
        else
          {

          /* add in its size */
          fddilp2_total_bits += op_pk_total_size_get (pkptr);
          fddilp2_total_bits_a[pri2_set] += op_pk_total_size_get (pkptr); /* 20JAN-
20APR94 */

          /* increment packet counter; 20APR94 */
          fddilp2_total_pkts++;
```

```
        fddilp2_total_pkts_a[pri2_set]++;

        /* if a multiple of 25 packets is reached, update stats */
        /* [0]->[7] represent asynch priorities 1->8, */
        /* respectively; [8] represents synchronous traffic, */
        /* and [9] represents overall asynchronous traffic.-Nix */
        if (fddilp2_total_pkts % 25 == 0)
           {
           op_stat_global_write (t2_gshandle,
           fddilp2_total_bits / op_sim_time ());

        op_stat_global_write (t2_gshandle_a[pri2_set],
    fddilp2_total_bits_a[0] / op_sim_time());
        op_stat_global_write (t2_gshandle_a[0],
    fddilp2_total_bits_a[1] / op_sim_time());
           op_stat_global_write (t2_gshandle_a[1],
   fddilp2_total_bits_a[pri2_set] / op_sim_time()):
           op_stat_global_write (t2_gshandle_a[2],
      fddilp2_total_bits_a[2] / op_sim_time());
         op_stat_global_write (t2_gshandle_a[3],
    fddilp2_total_bits_a[3] / op_sim_time());
         op_stat_global_write (t2_gshandle_a[4],
      fddilp2_total_bits_a[4] / op_sim_time());
        op_stat_global_write (t2_gshandle_a[5],
    fddilp2_total_bits_a[5] / op_sim_time());
         op_stat_global_write (t2_gshandle_a[6],
      fddilp2_total_bits_a[6] / op_sim_time());
           op_stat_global_write (t2_gshandle_a[7],
      fddilp2_total_bits_a[7] / op_sim_time());
        op_stat_global_write (t2_gshandle_a[8],
     fddilp2_total_bits_a[8] / op_sim_time());

           /* gather all asynch stats into one overall figure */
        op_stat_global_write (t2_gshandle_a[9],
    (fddilp2_total_bits - fddilp2_total_bits_a[8]) /
    op_sim_time());


/* (fddilp2_total_bits_a[0] + fddilp2_total_bits_a[1] + */
/* fddilp2_total_bits_a[2] + fddilp2_total_bits_a[3] + */
/* fddilp2_total_bits_a[4] + fddilp2_total_bits_a[5] + */
/* fddilp2_total_bits_a[6] + fddilp2_total_bits_a[7]) / */
/* op_sim_time()); */


        }
```

```
            /* 21APR94:allocate packets to the command link transmitter */
            /* altered for ppp 1AUG94 */
            ppp_pkptr = op_pk_create_fmt("ppp_ml");
        op_pk_nfd_set (ppp_pkptr, "pid_h", 0x00);
            op_pk_nfd_set (ppp_pkptr, "pid_l", 0x3d);
            op_pk_nfd_set (ppp_pkptr, "FDDI_frame", pkptr);
            /* put ppp packet on subqueue to be xmitted */
            op_subq_pk_insert(0, ppp_pkptr, OPC_QPOS_TAIL);

            }/* end of else */
          }/* end of else for (if ppp and from LLC_source) */

          /* check if this subqueue is empty and transmitter is not busy */
          if ((!op_subq_empty(0))&&(busy == 0.0))
            {
            /*access the first packet in the subqueue */
            pkptr1 = op_subq_pk_remove (0, OPC_QPOS_HEAD);
          /* forward it to the transmitter of command link */
            op_pk_send (pkptr1, 0);
            }

          break;
        }/* end of case OPC_INTRPT_STRM statement */

      }/* end of switch */









    }


/** blocking after enter executives of unforced state. **/
FSM_EXIT (1,sp_fddi_sink)


/** state (DISCARD) exit executives **/
FSM_STATE_EXIT_UNFORCED (0, state0_exit_exec, "DISCARD")
    {
    }
```

```
/** state (DISCARD) transition processing **/
FSM_INIT_COND (END_OF_SIM)
FSM_DFLT_COND
FSM_TEST_LOGIC ("DISCARD")

FSM_TRANSIT_SWITCH
   {
   FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;)
   FSM_CASE_TRANSIT (1, 0, state0_enter_exec, ;)
   }
/*---------------------------------------------------------*/



/** state (STATS) enter executives **/
FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "STATS")
   {
   /* At end of simulation, scalar performance statistics */
   /* and input parameters are written out. */
   /* This is for command link throughput :21APR94*/
      op_stat_scalar_write ("CL Throughput (bps), Priority 1",
           fddilp2_total_bits_a[0] / op_sim_time ());

    op_stat_scalar_write ("CL Throughput (bps), Priority 2",
    fddilp2_total_bits_a[1] / op_sim_time ());

    op_stat_scalar_write ("CL Throughput (bps), Priority 3",
    fddilp2_total_bits_a[2] / op_sim_time ());

    op_stat_scalar_write ("CL Throughput (bps), Priority 4",
    fddilp2_total_bits_a[3] / op_sim_time ());

    op_stat_scalar_write ("CL Throughput (bps), Priority 5",
    fddilp2_total_bits_a[4] / op_sim_time ());

    op_stat_scalar_write ("CL Throughput (bps), Priority 6",
    fddilp2_total_bits_a[5] / op_sim_time ());

    op_stat_scalar_write ("CL Throughput (bps), Priority 7",
    fddilp2_total_bits_a[6] / op_sim_time ());

    op_stat_scalar_write ("CL Throughput (bps), Priority 8",
    fddilp2_total_bits_a[7] / op_sim_time ());

    op_stat_scalar_write ("CL Throughput (bps), Asynchronous",
```

```
(fddilp2_total_bits - fddilp2_total_bits_a[8]) / op_sim_time ());


/* (fddilp2_total_bits_a[0] + fddilp2_total_bits_a[1] + */
/* fddilp2_total_bits_a[2] + fddilp2_total_bits_a[3] + */
/* fddilp2_total_bits_a[4] + fddilp2_total_bits_a[5] + */
/* fddilp2_total_bits_a[6] + fddilp2_total_bits_a[7]) / */
/* op_sim_time ()); */



op_stat_scalar_write ("CL Throughput (bps), Synchronous",
fddilp2_total_bits_a[8] / op_sim_time ());

op_stat_scalar_write ("CL Throughput (bps), Total",
fddilp2_total_bits / op_sim_time ());



/* Only one station needs to do this for the second ring(Ring 1)*/
if (!fddi2_sink_scalar_write)
  {
  /* set the scalar write flag */
  fddi2_sink_scalar_write = 1;

  op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 1",
       fddi2_sink_accum_delay_a[0] / fddi2_sink_total_pkts_a[0]);

op_stat_scalar_write ("Mean End-to-End Delay-1(sec.), Priority 2",
fddi2_sink_accum_delay_a[1] / fddi2_sink_total_pkts_a[1]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 3",
fddi2_sink_accum_delay_a[2] / fddi2_sink_total_pkts_a[2]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 4",
fddi2_sink_accum_delay_a[3] / fddi2_sink_total_pkts_a[3]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 5",
fddi2_sink_accum_delay_a[4] / fddi2_sink_total_pkts_a[4]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 6",
fddi2_sink_accum_delay_a[5] / fddi2_sink_total_pkts_a[5]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 7",
fddi2_sink_accum_delay_a[6] / fddi2_sink_total_pkts_a[6]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 8",
```

```
fddi2_sink_accum_delay_a[7] / fddi2_sink_total_pkts_a[7]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Asynchronous",
(fddi2_sink_accum_delay - fddi2_sink_accum_delay_a[8]) /
(fddi2_sink_total_pkts - fddi2_sink_total_pkts_a[8]));

/* (fddi2_sink_accum_delay_a[0] + fddi2_sink_accum_delay_a[1] + */
/* fddi2_sink_accum_delay_a[2] + fddi2_sink_accum_delay_a[3] + */
/* fddi2_sink_accum_delay_a[4] + fddi2_sink_accum_delay_a[5] + */
/* fddi2_sink_accum_delay_a[6] + fddi2_sink_accum_delay_a[7]) / */
/* (fddi2_sink_total_pkts_a[0] + fddi2_sink_total_pkts_a[1] + */
/* fddi2_sink_total_pkts_a[2] + fddi2_sink_total_pkts_a[3] + */
/* fddi2_sink_total_pkts_a[4] + fddi2_sink_total_pkts_a[5] + */
/* fddi2_sink_total_pkts_a[6] + fddi2_sink_total_pkts_a[7])); */


op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Synchronous",
fddi2_sink_accum_delay_a[8] / fddi2_sink_total_pkts_a[8]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Total",
fddi2_sink_accum_delay / fddi2_sink_total_pkts);

  op_stat_scalar_write ("Throughput-1 (bps), Priority 1",
       fddi2_sink_total_bits_a[0] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Priority 2",
fddi2_sink_total_bits_a[1] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Priority 3",
fddi2_sink_total_bits_a[2] / op_sim_time ());

op_stat_scalar_write ("Throughput-1( bps), Priority 4",
fddi2_sink_total_bits_a[3] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Priority 5",
fddi2_sink_total_bits_a[4] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Priority 6",
fddi2_sink_total_bits_a[5] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Priority 7",
fddi2_sink_total_bits_a[6] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Priority 8",
fddi2_sink_total_bits_a[7] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Asynchronous",
```

```
(fddi2_sink_total_bits - fddi2_sink_total_bits_a[8]) / op_sim_time ());


/* (fddi2_sink_total_bits_a[0] + fddi2_sink_total_bits_a[1] + */
/* fddi2_sink_total_bits_a[2] + fddi2_sink_total_bits_a[3] + */
/* fddi2_sink_total_bits_a[4] + fddi2_sink_total_bits_a[5] + */
/* fddi2_sink_total_bits_a[6] + fddi2_sink_total_bits_a[7]) / */
/* op_sim_time ()); */



op_stat_scalar_write ("Throughput-1 (bps), Synchronous",
fddi2_sink_total_bits_a[8] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Total",
fddi2_sink_total_bits / op_sim_time ());


  op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 1",
       fddi2_sink_peak_delay_a[0]);

op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 2",
fddi2_sink_peak_delay_a[1]);

op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 3",
fddi2_sink_peak_delay_a[2]);

op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 4",
fddi2_sink_peak_delay_a[3]);

op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 5",
fddi2_sink_peak_delay_a[4]);

op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 6",
fddi2_sink_peak_delay_a[5]);

op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 7",
fddi2_sink_peak_delay_a[6]);

op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 8",
fddi2_sink_peak_delay_a[7]);

op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Synchronous",
fddi2_sink_peak_delay_a[8]);

op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Overall",
fddi2_sink_peak_delay);
```

```
        /* Write the TTRT value for ring 0. This preserves*/
        /* the old behavior for single-ring simulations.*/
        op_stat_scalar_write ("TTRT (sec.) - Ring 1",
             fddi_t_opr [1]);

      /* 12JAN94: obtain offered load information from the Environment */
      /* file; this will be used to provide abscissa information that */
      /* can be plotted in the Analysis Editor (see "fddi_sink" STATS */
      /* state. To the user: it's your job to keep these current in */
      /* the Environment File. -Nix */
      op_ima_sim_attr_get (OPC_IMA_DOUBLE, "total_offered_load_1", &Offered_Load);
      op_ima_sim_attr_get (OPC_IMA_DOUBLE, "asynch_offered_load_1", &Asynch_Of-
fered_Load);


       /* 12JAN94: write the total offered load for this run */
       op_stat_scalar_write ("Total Offered Load-1 (Mbps)",
       Offered_Load);

       op_stat_scalar_write ("Asynchronous Offered Load-1 (Mbps)",
       Asynch_Offered_Load);
         }
       }


    /** blocking after enter executives of unforced state. **/
    FSM_EXIT (3,sp_fddi_sink)


    /** state (STATS) exit executives **/
    FSM_STATE_EXIT_UNFORCED (1, state1_exit_exec, "STATS")
      {
      }


    /** state (STATS) transition processing **/
    FSM_TRANSIT_MISSING ("STATS")
    /*---------------------------------------------------------*/



    /** state (INIT) enter executives **/
    FSM_STATE_ENTER_FORCED (2, state2_enter_exec, "INIT")
      {
      /* get the gshandles of the global statistic to be obtained */
```

```
/* 20JAN94: set array format */

thru2_gshandle_a[0] = op_stat_global_reg ("pri 1 throughput-1 (bps)");
thru2_gshandle_a[1] = op_stat_global_reg ("pri 2 throughput-1 (bps)");
thru2_gshandle_a[2] = op_stat_global_reg ("pri 3 throughput-1 (bps)");
thru2_gshandle_a[3] = op_stat_global_reg ("pri 4 throughput-1 (bps)");
thru2_gshandle_a[4] = op_stat_global_reg ("pri 5 throughput-1 (bps)");
thru2_gshandle_a[5] = op_stat_global_reg ("pri 6 throughput-1 (bps)");
thru2_gshandle_a[6] = op_stat_global_reg ("pri 7 throughput-1 (bps)");
thru2_gshandle_a[7] = op_stat_global_reg ("pri 8 throughput-1 (bps)");
thru2_gshandle_a[8] = op_stat_global_reg ("synch throughput-1 (bps)");
thru2_gshandle_a[9] = op_stat_global_reg ("async throughput-1 (bps)");
thru2_gshandle = op_stat_global_reg ("total throughput-1 (bps)");

m2_delay_gshandle_a[0] = op_stat_global_reg ("pri 1 mean delay-1 (sec.)");
m2_delay_gshandle_a[1] = op_stat_global_reg ("pri 2 mean delay-1 (sec.)");
m2_delay_gshandle_a[2] = op_stat_global_reg ("pri 3 mean delay-1 (sec.)");
m2_delay_gshandle_a[3] = op_stat_global_reg ("pri 4 mean delay-1 (sec.)");
m2_delay_gshandle_a[4] = op_stat_global_reg ("pri 5 mean delay-1 (sec.)");
m2_delay_gshandle_a[5] = op_stat_global_reg ("pri 6 mean delay-1 (sec.)");
m2_delay_gshandle_a[6] = op_stat_global_reg ("pri 7 mean delay-1 (sec.)");
m2_delay_gshandle_a[7] = op_stat_global_reg ("pri 8 mean delay-1 (sec.)");
m2_delay_gshandle_a[8] = op_stat_global_reg ("synch mean delay-1 (sec.)");
m2_delay_gshandle_a[9] = op_stat_global_reg ("async mean delay-1 (sec.)");
m2_delay_gshandle = op_stat_global_reg ("total mean delay-1 (sec.)");

ete2_delay_gshandle_a[0] = op_stat_global_reg ("pri 1 end-to-end delay-1 (sec.)");
ete2_delay_gshandle_a[1] = op_stat_global_reg ("pri 2 end-to-end delay-1 (sec.)");
ete2_delay_gshandle_a[2] = op_stat_global_reg ("pri 3 end-to-end delay-1 (sec.)");
ete2_delay_gshandle_a[3] = op_stat_global_reg ("pri 4 end-to-end delay-1 (sec.)");
ete2_delay_gshandle_a[4] = op_stat_global_reg ("pri 5 end-to-end delay-1 (sec.)");
ete2_delay_gshandle_a[5] = op_stat_global_reg ("pri 6 end-to-end delay-1 (sec.)");
ete2_delay_gshandle_a[6] = op_stat_global_reg ("pri 7 end-to-end delay-1 (sec.)");
ete2_delay_gshandle_a[7] = op_stat_global_reg ("pri 8 end-to-end delay-1 (sec.)");
ete2_delay_gshandle_a[8] = op_stat_global_reg ("synch end-to-end delay-1 (sec.)");
ete2_delay_gshandle = op_stat_global_reg ("total end-to-end delay-1 (sec.)");

t2_gshandle_a[0] = op_stat_global_reg ("pri 1 CL throughput (bps)");
t2_gshandle_a[1] = op_stat_global_reg ("pri 2 CL throughput (bps)");
t2_gshandle_a[2] = op_stat_global_reg ("pri 3 CL throughput (bps)");
t2_gshandle_a[3] = op_stat_global_reg ("pri 4 CL throughput (bps)");
t2_gshandle_a[4] = op_stat_global_reg ("pri 5 CL throughput (bps)");
t2_gshandle_a[5] = op_stat_global_reg ("pri 6 CL throughput (bps)");
t2_gshandle_a[6] = op_stat_global_reg ("pri 7 CL throughput (bps)");
t2_gshandle_a[7] = op_stat_global_reg ("pri 8 CL throughput (bps)");
t2_gshandle_a[8] = op_stat_global_reg ("synch CL throughput (bps)");
t2_gshandle_a[9] = op_stat_global_reg ("async CL throughput (bps)");
```

```
        t2_gshandle = op_stat_global_reg ("total CL throughput (bps)");


        }


    /** state (INIT) exit executives **/
    FSM_STATE_EX...FORCED (2, state2_exit_exec, "INIT")
        {
        }


    /** state (INIT) transition processing **/
    FSM_INIT_COND (END_OF_SIM)
    FSM_DFLT_COND
    FSM_TEST_LOGIC ("INIT")

    FSM_TRANSIT_SWITCH
        {
        FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;)
        FSM_CASE_TRANSIT (1, 0, state0_enter_exec, ;)
        }
    /*---------------------------------------------------------*/



    }


  FSM_EXIT (2,sp_fddi_sink)
  }



void
sp_fddi_sink_svar (prs_ptr,var_name,var_p_ptr)
  sp_fddi_sink_state*prs_ptr;
  char    *var_name, **var_p_ptr;
  {

  FIN (sp_fddi_sink_svar (prs_ptr))

  *var_p_ptr = VOS_NIL;
  if (Vos_String_Equal ("thru2_gshandle" , var_name))
    *var_p_ptr = (char *) (&prs_ptr->sv_thru2_gshandle);
```

```
    if (Vos_String_Equal ("m2_delay_gshandle" , var_name))
       *var_p_ptr = (char *) (&prs_ptr->sv_m2_delay_gshandle);
    if (Vos_String_Equal ("ete2_delay_gshandle" , var_name))
       *var_p_ptr = (char *) (&prs_ptr->sv_ete2_delay_gshandle);
    if (Vos_String_Equal ("thru2_gshandle_a" , var_name))
       *var_p_ptr = (char *) (prs_ptr->sv_thru2_gshandle_a);
    if (Vos_String_Equal ("m2_delay_gshandle_a" , var_name))
       *var_p_ptr = (char *) (prs_ptr->sv_m2_delay_gshandle_a);
    if (Vos_String_Equal ("ete2_delay_gshandle_a" , var_name))
       *var_p_ptr = (char *) (prs_ptr->sv_ete2_delay_gshandle_a);
    if (Vos_String_Equal ("t2_gshandle" , var_name))
       *var_p_ptr = (char *) (&prs_ptr->sv_t2_gshandle);
    if (Vos_String_Equal ("t2_gshandle_a" , var_name))
       *var_p_ptr = (char *) (prs_ptr->sv_t2_gshandle_a);
    if (Vos_String_Equal ("my_id" , var_name))
       *var_p_ptr = (char *) (&prs_ptr->sv_my_id);

    FOUT;
    }



void
sp_fddi_sink_diag ()
    {
    double delay, creat_time;
    Packet*pkptr;
    Packet*ppp_pkptr;
    Packet* pkptr1 ; /*5APR94*/
    int         src_addr, my_addr;
    int         dest_addr;/*14APR94*/
    Ici*    from_mac_ici_ptr;
    double fddi_sink_trt;
    char    pk_format[10];
    int         input_stream;
    int         fd_index, FDDI_frame_size;

    FIN (sp_fddi_sink_diag ())


    FOUT;
    }
```

```
void
sp_fddi_sink_terminate ()
  {
  double delay, creat_time;
  Packet*pkptr;
  Packet*ppp_pkptr;
  Packet* pkptr1 ; /*5APR94*/
  int       src_addr, my_addr;
  int       dest_addr;/*14APR94*/
  Ici*    from_mac_ici_ptr;
  double fddi_sink_trt;
  char    pk_format[10];
  int       input_stream;
  int       fd_index, FDDI_frame_size;

  FIN (sp_fddi_sink_terminate ())


  FOUT;
  }



Compcode
sp_fddi_sink_init (pr_state_pptr)
  sp_fddi_sink_state**pr_state_pptr;
  {
  static VosT_Cm_Obtypeobtype = OPC_NIL;

  FIN (sp_fddi_sink_init (pr_state_pptr))

  if (obtype == OPC_NIL)
    {
    if (Vos_Catmem_Register ("proc state vars (sp_fddi_sink)",
      sizeof (sp_fddi_sink_state), Vos_Nop, &obtype) == VOSC_FAILURE)
      FRET (OPC_COMPCODE_FAILURE)
    }

  if ((*pr_state_pptr = (sp_fddi_sink_state*) Vos_Catmem_Alloc (obtype, 1)) == OPC_NIL)
    FRET (OPC_COMPCODE_FAILURE)
  else
    {
    (*pr_state_pptr)->current_block = 4;
    FRET (OPC_COMPCODE_SUCCESS)
    }
  }
```

APPENDIX F

ENVIRONMENT FILE EXAMPLE
"run4.ef"

```
# cdl4_lb0jam0.ef
# 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

#*** Attributes related to loading used by "fddi_gen" ***

# 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


# 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" :  250
"*.*.*.mean pk length" : 20000
#"ring0.f9.*.arrival rate": 0
#"ring0.f9.*.mean pk length": 0
#"ring1.f9.*.arrival rate": 0
#"ring1.f9.*.mean pk length": 0

# 7APR94 - S.Karayakaylar
# 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.llc_sink.load balancing algorithm":0

# 15JUN94 - Ike
# determine rate at which link monitoring packets will be sent (secs/pkt)
# and size of pkt (bits)
"top.ring0.f9.llc_sink.link_monitor_trans_rate":0.000729
"top.ring0.f9.llc_sink.monitoring pkt size":5000
# 1 % overhead

# 26JUL94 - Ike
# 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":0.099
"*.*.f9.*.upper hysteresis threshold":0.5
"*.*.f9.*.lower hysteresis threshold":0.3
"*.*.f9.*.history length":50
"top.ring1.f9.*.ecc threshold":0.005
#18AUG94 - Ike
#allow xmission rates to be set at env. file The corresponding xmtr/rcvr
#rates should be the same. (bits/sec)
"ring0.f9.pt_1[0].data rate":1530000
"ring1.f9.pr_1[0].data rate":1530000
"ring0.f9.pt_2[0].data rate": 3060000
"ring1.f9.pr_2[0].data rate": 3060000
"ring0.f9.pt_3[0].data rate": 21420000
"ring1.f9.pr_3[0].data rate": 21420000
"ring0.f9.pt_4[0].data rate": 42840000
"ring1.f9.pr_4[0].data rate": 42840000

# 15APR94 :determine the station address of the bridge node in
# both rings.
"top.ring0.f9.llc_sink.station_address":9
"top.ring1.f9.llc_sink.station_address":19

# 12DEC93: 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" : 0.9
#"top.ring0.f9.llc_src.async_mix" : 1

#*** 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
```

```
"*.*.mac.sync bandwidth" : 0.08955675
#"top.ring0.f9.mac.sync bandwidth": 0.0

# Target Token Rotation Time (one half of maximum synchronous response time)
# (This is commented out for compatibility with the fddi_script, which
# sets T_Req on the simulation command line; remove the comment pound-sign
# below to make this environment file self-sufficient.)

# 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 = 1.97388 ms.
"*.*.mac.T_Req" :.004



# Index of the station which initially launches the token
# 17APR94 : -Karayakaylar
# This index should be greater than the maximum station number
# Bridge stations spawns token for interconnected simulation by default.
"spawn station":20



# 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

# CDL link related attributes -Karayakaylar 7APR94
# The attributes below are specified with respect to the jammer type
# There are two types of jamming models which the CDL is exposed to.
#
# (1) Pulsed jammer (jammer_type = 0)
#(2) Channel-swept jammer (jammer_type = 1)
#
# NOTE:For pulsed jammer init_jam_offset may be zero, whereas a proper
# offset should be specified for channel-swept jammer.
# jam_length, jam_ber, interval_bet_jam_len, ber_bet_jam_len are maximum
# values in the case of pulsed jammer since they are randomized in the
```

```
# error allocation pipeline stage.
# For channel-swept jammer only jam_ber and ber_bet_jam_len attributes are
# maximium values to be randomized.
# return link ls_0 to ls_3
# command link ls_4
*.ls_0.jam_length: 0.05
*.ls_1.jam_length: 0.02
*.ls_2.jam_length: 0.01
*.ls_3.jam_length: 0.09
*.ls_4.jam_length: 0.05
*.ls_0.jam_ber: 2e-3
*.ls_1.jam_ber: 2e-3
*.ls_2.jam_ber: 2e-3
*.ls_3.jam_ber: 2e-3
*.ls_4.jam_ber: 0.0
*.ls_0.interval_bet_jam_len: 0.03
*.ls_1.interval_bet_jam_len: 0.03
*.ls_2.interval_bet_jam_len: 0.03
*.ls_3.interval_bet_jam_len: 0.03
*.ls_4.interval_bet_jam_len: 0.03
*.ls_0.ber_bet_jam_len: 2e-6
*.ls_1.ber_bet_jam_len: 2e-6
*.ls_2.ber_bet_jam_len: 2e-6
*.ls_3.ber_bet_jam_len: 2e-6
*.ls_4.ber_bet_jam_len: 0.0
*.ls_0.init_jam_offset: 0.0
*.ls_1.init_jam_offset: 0.0
*.ls_2.init_jam_offset: 0.0
*.ls_3.init_jam_offset: 0.0
*.ls_4.init_jam_offset: 0.0
*.ls_0.jammer_type: 1
*.ls_1.jammer_type: 1
*.ls_2.jammer_type: 1
*.ls_3.jammer_type: 1
*.ls_4.jammer_type: 0

# 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

#*** Simulation related 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. -Karayakaylar
# error documented on MIL3 bbs - Ike
accelerate_token:0


# Run control attributes
seed: 10
#duration: 10.001
verbose_sim:TRUE
upd_int:.1
#os_file:

# (This is commented out for compatibility with the fddi_script, which
# sets the output vector file on the simulation command line; remove the
# comment pound-sign below to make this environment file self-sufficient.)
#ov_file:


# Opnet Debugger (odb) enabling attribute
debug:FALSE
```

APPENDIX G

OPNET C-SHELL SCRIPT FILE EXAMPLE "opnet_script"

```
#! /bin/csh -f
setenv debug FALSE
cd $HOME/op_models/cdl

fddi_cdl.sim -ov_file tester8_1 -ef run1 \
-probe fddi_cdl_probe_special -duration 10.001

fddi_cdl.sim -ov_file tester8_2 -ef run2 \
-probe fddi_cdl_probe_special -duration 10.001

fddi_cdl.sim -ov_file tester8_3 -ef run3 \
-probe fddi_cdl_probe_special -duration 10.001

fddi_cdl.sim -ov_file tester8_4 -ef run4 \
-probe fddi_cdl_probe_special -duration 10.001
```

APPENDIX H

TCP RING 0 LLC_SRC MODULE CODE "cp_fddi_gen_tcp.pr.c"

```
/* Process model C form file: cp_fddi_gen_tcp.pr.c */
/* Portions of this file Copyright (C) MIL 3, Inc. 1992 */



/* OPNET system definitions */
#include <opnet.h>
#include "cp_fddi_gen_tcp.pr.h"
FSM_EXT_DECS



/* Header block */
#define MAC_LAYER_OUT_STREAM    0
#define LLC_SINK_OUT_STREAM     1 /*18APR94*/

#define RECEIVER_IN_STREAM      0
#define IP_IN_STREAM            1

/* define possible service classes for frames */
#define FDDI_SVC_ASYNC0
#define FDDI_SVC_SYNC           1

/* define token classes */
#define FDDI_TK_NONRESTRICTED   0
#define FDDI_TK_RESTRICTED      1

/*ADDITIONS AFTER HERE 10SEP94 */
/* this can be used if checking to ensure incoming IP dgrams */
#define NET_PROT_IP             0x0800



/* State variable definitions */
typedef struct
```

```
	{
	FSM_SYS_STATE
	Objid		sv_mac_objid;
	Objid		sv_my_id;
	int		sv_station_addr;
	int		sv_src_addr;
	double		sv_arrival_rate;
	double		sv_mean_pk_len;
	double		sv_async_mix;
	Ici*		sv_mac_iciptr1;
	Ici*		sv_llc_iciptr;
	Packet*		sv_pkptr1;
	} cp_fddi_gen_tcp_state;

#define pr_state_ptr		((cp_fddi_gen_tcp_state*) SimI_Mod_State_Ptr)
#define mac_objid		pr_state_ptr->sv_mac_objid
#define my_id			pr_state_ptr->sv_my_id
#define station_addr		pr_state_ptr->sv_station_addr
#define src_addr		pr_state_ptr->sv_src_addr
#define arrival_rate		pr_state_ptr->sv_arrival_rate
#define mean_pk_len		pr_state_ptr->sv_mean_pk_len
#define async_mix		pr_state_ptr->sv_async_mix
#define mac_iciptr1		pr_state_ptr->sv_mac_iciptr1
#define llc_iciptr		pr_state_ptr->sv_llc_iciptr
#define pkptr1			pr_state_ptr->sv_pkptr1



/* Process model interrupt handling procedure */



void
cp_fddi_gen_tcp ()
  {
  Packet *pkptr, *ppp_pkptr, *ip_pkptr, *mac_frame_ptr;
  int	pklen;
  int	dest_addr;
  int	i, restricted;
  int	ppp_pid_h, ppp_pid_l;
  int	status;
  double creation_time;
```

```
Ici     *ip_iciptr;


FSM_ENTER (cp_fddi_gen_tcp)

FSM_BLOCK_SWITCH
  {
  /*---------------------------------------------------------*/
  /** state (INIT) enter executives **/
  FSM_STATE_ENTER_UNFORCED (0, state0_enter_exec, "INIT")
    {
    /* determine id of own processor to use in finding attrs */
    my_id = op_id_self ();

    /* determine object id of connected 'mac' layer process */
    mac_objid = op_topo_assoc (my_id, OPC_TOPO_ASSOC_OUT,
         OPC_OBJMTYPE_MODULE, MAC_LAYER_OUT_STREAM);

    /* determine the address assigned to it */
    /* which is also the address of this station */
    op_ima_obj_attr_get (mac_objid, "station_address", &station_addr);

         /* set up an interface control information (ICI) structure */
         /* to communicate parameters to the mac layer process */
         /* (it is more efficient to set one up now and keep it */
         /* as a state variable than to allocate one on each packet xfer) */
         mac_iciptr1 = op_ici_create ("fddi_mac_req_tcp");
       llc_iciptr = op_ici_create ("fddi_mac_ind_tcp");
    }


  /** blocking after enter executives of unforced state. **/
  FSM_EXIT (1,cp_fddi_gen_tcp)


  /** state (INIT) exit executives **/
  FSM_STATE_EXIT_UNFORCED (0, state0_exit_exec, "INIT")
    {
    }


  /** state (INIT) transition processing **/
  FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
```

```
/*--------------------------------------------------------*/


/** state (ARRIVAL) enter executives **/
FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "ARRIVAL")
  {
  /* This station should receive frames from the other lan as long as */
  /* there are frames in the input streams addressed to this lan */
  /*check if the interrupt type is stream interrupt *//*12APR94*/
  if((op_intrpt_type() == OPC_INTRPT_STRM) && (op_intrpt_strm() ==
RECEIVER_IN_STREAM))
    {
    /* if it is, get the packet in the input stream causing interrupt */
    ppp_pkptr = op_pk_get(op_intrpt_strm());
    op_pk_nfd_get (ppp_pkptr, "pid_h",&ppp_pid_h);
    switch (ppp_pid_h)
      {
      case 0x00: /* data frame */
        op_pk_nfd_get (ppp_pkptr, "FDDI_frame", &pkptr1);
        op_pk_destroy (ppp_pkptr);
        /* 11SEP94 get ICI associated with FDDI LLC frame */
    llc_iciptr = op_pk_ici_get (pkptr1);

        /* get the destination address of the frame */
        /* 16APR94 */
        op_ici_attr_get(llc_iciptr, "dest_addr", &dest_addr);
        /* check if this frame destined for the local bridge station */
        if(dest_addr == station_addr)
          {
          /* if it is, send the packet to llc_sink directly */
          /* in order to prevent overhead of mac access */
          op_ici_install (llc_iciptr);
          op_pk_send(pkptr1, LLC_SINK_OUT_STREAM);/*19APR94*/
          }
        else
          /* this packet is to be sent to MAC */
          {
          /* determine the source address of the frame */
          op_ici_attr_get(llc_iciptr, "src_addr", &src_addr);
          /* set up an ICI structure to communicate parameters to */
          /* MAC layer process (initialized in INIT state)*/
          /* place the original source address into the ICI *//* 16APR94 */
```

```
            /* "fddi_mac_req" is modified so that it contains the original */
            /* source address from the remote lan */
            op_ici_attr_set(mac_iciptr1, "src_addr", src_addr);
            /* place the destination address into the ICI */ /*12APR94*/
            op_ici_attr_set(mac_iciptr1, "dest_addr", dest_addr);
            /* assign the service class and requested token class */
            /* At this moment the frames coming from the remote lan are assumed to have*/
            /* the same priority as synchronous frames in order not to accumulate */
            /* packets on the bridge station mac and instead to deliver their destinations */
            /* as soon as possible */
            /*10SEP94: I don't know why Selcuk did this, so I'll leave the values */
            /* but change to conform with the new packet/ICI formats */
            op_ici_attr_set(mac_iciptr1, "svc_class", FDDI_SVC_SYNC);
            op_ici_attr_set(mac_iciptr1, "pri", 8);
            op_ici_attr_set(mac_iciptr1, "tk_class", FDDI_TK_NONRESTRICTED);
            op_ici_attr_get(mac_iciptr1, "cr_time", &creation_time);
            op_ici_attr_set(mac_iciptr1, "cr_time", creation_time);
            /* send the packet coupled with the ICI */
            op_ici_install(mac_iciptr1);
            op_pk_send(pkptr1, MAC_LAYER_OUT_STREAM);
          }
          break;

        case 0xc0: /* ppp control packet - either LQR or monitoring pkt*/
          /* Since the command link is deemed unjammable, this should */
          /* only be for LQR's. Simply delete them for now until a*/
          /* policy is determined on how to handle the LQR's.*/
    op_pk_nfd_get (ppp_pkptr, "LQR_info", &status);
          printf ("LQR received at cp ");
          switch (status)
            {
            case 0: printf ("status GOOD trend DOWN\n");
            case 1: printf ("status GOOD trend UP\n");
            case 2: printf ("status BAD trend DOWN\n");
            case 3: printf ("status BAD trend UP\n");
            }
          op_pk_destroy (ppp_pkptr);
        break;

        default:
          printf ("ERROR: packet rcvd at cp: neither data nor control\n");
          break;
        } /* end switch */
```

```
        } /* end if (op_intrpt_type()==OPC_INTRPT_STRM) && */
        /* (op_intrpt_strm() == RECEIVER_IN_STRM))*/

      /* otherwise, it's an IP dgram. Find destination and send */
      /* ALL PAST THIS POINT: edited 11SEP94 */
      else if ((op_intrpt_type()==OPC_INTRPT_STRM) && (op_intrpt_strm() ==
IP_IN_STREAM))
        {
        /* get IP dgram and associated ICI */
        ip_pkptr = op_pk_get (op_intrpt_strm());
        ip_iciptr = op_intrpt_ici();

        /* put IP dgram in FDDI LLC frame */
        pkptr = op_pk_create_fmt ("fddi_llc_fr_tcp");
        /* Further explanations of this format is in "llc_src_fddi_tcp" ARRIVAL state*/
        op_pk_nfd_set (pkptr, "datagram",ip_pkptr);

        op_ici_attr_get (ip_iciptr, "dest_addr", &dest_addr);
        if(dest_addr > 9)
      {
          /* if it is, this packet is to be sent llc_sink directly */
          /* for xmsn to the remote LAN. The LLC frame must be*/
          /* also have an ICI, as if the frame was coming from the MAC*/
          op_ici_attr_set (llc_iciptr, "dest_addr", dest_addr);
          op_ici_attr_set (llc_iciptr, "src_addr", station_addr);
          op_ici_attr_set (llc_iciptr, "pri", 0);
          op_ici_attr_set (llc_iciptr, "cr_time", op_sim_time());

          /* install LLC ICI and send frame to LLC sink */
          op_ici_install (llc_iciptr);
          op_pk_send (llc_iciptr, LLC_SINK_OUT_STREAM); /*18APR94*/
          }
      else
          /* if not, the packet is destined for local lan, so send LLC*/
          /* frame to MAC */
          {
          /* set up an ICI structure to communicate parameters to */
          /* MAC layer process (initialized in INIT state)*/
          /* place the this station's address into the ICI *//* 16APR94 */
          op_ici_attr_set(mac_iciptr1, "src_addr", station_addr);
          /* place the destination address into the ICI */ /*12APR94*/
          op_ici_attr_set(mac_iciptr1, "dest_addr", dest_addr);
          /* assign the service class and token class IAW RFC 1390*/
```

```
        op_ici_attr_set(mac_iciptr1, "svc_class", FDDI_SVC_ASYNC);
        op_ici_attr_set(mac_iciptr1, "pri", 0);
        op_ici_attr_set(mac_iciptr1, "tk_class", FDDI_TK_NONRESTRICTED);
        op_pk_nfd_get (pkptr1, "cr_time", &creation_time);
        op_ici_attr_set(mac_iciptr1, "cr_time", creation_time);


        /* send the packet coupled with the ICI */
        op_ici_install(mac_iciptr1);
        op_pk_send (pkptr, MAC_LAYER_OUT_STREAM);
    }

      } /* end else (if((op_intrpt_type() == OPC_INTRPT_STRM) && */
      /*(op_intrpt_strm() == IP_IN_STREAM)) */

    }


  /** blocking after enter executives of unforced state. **/
  FSM_EXIT (3,cp_fddi_gen_tcp)


  /** state (ARRIVAL) exit executives **/
  FSM_STATE_EXIT_UNFORCED (1, state1_exit_exec, "ARRIVAL")
    {
    }


  /** state (ARRIVAL) transition processing **/
  FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
  /*---------------------------------------------------------*/



  }


FSM_EXIT (0,cp_fddi_gen_tcp)
}
```

```
void
cp_fddi_gen_tcp_svar (prs_ptr,var_name,var_p_ptr)
   cp_fddi_gen_tcp_state*prs_ptr;
   char    *var_name, **var_p_ptr;
   {

   FIN (cp_fddi_gen_tcp_svar (prs_ptr))

   *var_p_ptr = VOS_NIL;
   if (Vos_String_Equal ("mac_objid" , var_name))
      *var_p_ptr = (char *) (&prs_ptr->sv_mac_objid);
   if (Vos_String_Equal ("my_id" , var_name))
      *var_p_ptr = (char *) (&prs_ptr->sv_my_id);
   if (Vos_String_Equal ("station_addr" , var_name))
      *var_p_ptr = (char *) (&prs_ptr->sv_station_addr);
   if (Vos_String_Equal ("src_addr" , var_name))
      *var_p_ptr = (char *) (&prs_ptr->sv_src_addr);
   if (Vos_String_Equal ("arrival_rate" , var_name))
      *var_p_ptr = (char *) (&prs_ptr->sv_arrival_rate);
   if (Vos_String_Equal ("mean_pk_len" , var_name))
      *var_p_ptr = (char *) (&prs_ptr->sv_mean_pk_len);
   if (Vos_String_Equal ("async_mix" , var_name))
      *var_p_ptr = (char *) (&prs_ptr->sv_async_mix);
   if (Vos_String_Equal ("mac_iciptr1" , var_name))
      *var_p_ptr = (char *) (&prs_ptr->sv_mac_iciptr1);
   if (Vos_String_Equal ("llc_iciptr" , var_name))
      *var_p_ptr = (char *) (&prs_ptr->sv_llc_iciptr);
   if (Vos_String_Equal ("pkptr1" , var_name))
      *var_p_ptr = (char *) (&prs_ptr->sv_pkptr1);

   FOUT;
   }



void
cp_fddi_gen_tcp_diag ()
   {
   Packet *pkptr, *ppp_pkptr, *ip_pkptr, *mac_frame_ptr;
   int      pklen;
   int      dest_addr;
   int      i, restricted;
```

```
    int     ppp_pid_h, ppp_pid_l;
    int     status;
    double creation_time;
    Ici     *ip_iciptr;

    FIN (cp_fddi_gen_tcp_diag ())


    FOUT;
    }



void
cp_fddi_gen_tcp_terminate ()
    {
    Packet *pkptr, *ppp_pkptr, *ip_pkptr, *mac_frame_ptr;
    int     pklen;
    int     dest_addr;
    int     i, restricted;
    int     ppp_pid_h, ppp_pid_l;
    int     status;
    double creation_time;
    Ici     *ip_iciptr;

    FIN (cp_fddi_gen_tcp_terminate ())


    FOUT;
    }



Compcode
cp_fddi_gen_tcp_init (pr_state_pptr)
    cp_fddi_gen_tcp_state**pr_state_pptr;
    {
    static VosT_Cm_Obtypeobtype = OPC_NIL;

    FIN (cp_fddi_gen_tcp_init (pr_state_pptr))

    if (obtype == OPC_NIL)
```

```
  {
  if (Vos_Catmem_Register ("proc state vars (cp_fddi_gen_tcp)",
    sizeof (cp_fddi_gen_tcp_state), Vos_Nop, &obtype) == VOSC_FAILURE)
    FRET (OPC_COMPCODE_FAILURE)
  }

if ((*pr_state_pptr = (cp_fddi_gen_tcp_state*) Vos_Catmem_Alloc (obtype, 1)) == OPC_NIL)
  FRET (OPC_COMPCODE_FAILURE)
else
  {
  (*pr_state_pptr)->current_block = 0;
  FRET (OPC_COMPCODE_SUCCESS)
  }
}
```

APPENDIX I

TCP RING 0 LLC_SINK MODULE CODE "cp_fddi_sink_tcp.pr.c"

```
/* Process model C form file: cp_fddi_sink_tcp.pr.c */
/* Portions of this file Copyright (C) MIL 3, Inc. 1992 */



/* OPNET system definitions */
#include <opnet.h>
#include "cp_fddi_sink_tcp.pr.h"
FSM_EXT_DECS



/* Header block */
/* Globals */
/* array format installed 20JAN94; positions 0-7 represent the asynch priority levels, PRIORITIES
+ 1 */
/* represents synch traffic, and grand totals are as given in the original. */

#define PRIORITIES 8 /* 20JAN94 */
#define     XMITTER_ONE     0 /*10MAY94*/
#define     XMITTER_TWO     1
#define     XMITTER_THREE   2
#define     XMITTER_FOUR    3
#define     IP_OUT_STRM     4/*11SEP94*/

static /* 05FEB94 */
double fddi_sink_accum_delay = 0.0;
static /* 05FEB94 */
double      fddi_sink_accum_delay_a[PRIORITIES + 1] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0};
static /* 05FEB94 */
int fddi_sink_total_pkts = 0;
static /* 05FEB94 */
int             fddi_sink_total_pkts_a[PRIORITIES + 1] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
static /* 05FEB94 */
```

```
double fddi_sink_total_bits = 0.0;
static /* 05FEB94 */
double          fddi_sink_total_bits_a[PRIORITIES + 1] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0};
static /* 05FEB94 */
double fddi_sink_peak_delay = 0.0;
static /* 05FEB94 */
double          fddi_sink_peak_delay_a[PRIORITIES + 2] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0};
static /* 05FEB94 */
int                     fddi_sink_scalar_write = 0;
static /* 05FEB94 */
int pri_set = 20; /* 20JAN94 */
static
int                     subq_index = 0; /* 5APR94 */
static
double          buffer[4]={0.0,0.0,0.0,0.0}; /*10MAY94*/


/* statistics used for CDL throughput */
static /* 20APR94 */
int fddilp1_total_pkts = 0;
static
int                     fddilp1_total_pkts_a[PRIORITIES + 1] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
static
double fddilp1_total_bits = 0.0;
static
double          fddilp1_total_bits_a[PRIORITIES + 1] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};


/* Externally defined globals. */
extern double  fddi_t_opr [];

/*12JAN94:attributes from the Environment file */
double Offered_Load; /* 12JAN94 */
double Asynch_Offered_Load; /* 12JAN94 */

/* transition expressions */
#define END_OF_SIM op_intrpt_type() == OPC_INTRPT_ENDSIM

/* get picture of jamming. this variable allows anyone who*/
/* is writing to the global statistic to ensure it has */
/* been initialized. 20AUG94*/
```

```
int         jammer_stats_init=OPC_FALSE;
Gshandle link_gshandle[4];



/* State variable definitions */
typedef struct
    {
    FSM_SYS_STATE
    Gshandle    sv_thru_gshandle;
    Gshandle    sv_m_delay_gshandle;
    Gshandle    sv_ete_delay_gshandle;
    Gshandle    sv_thru_gshandle_a[10];
    Gshandle    sv_m_delay_gshandle_a[10];
    Gshandle    sv_ete_delay_gshandle_a[9];
    Gshandle    sv_t_gshandle;
    Gshandle    sv_t_gshandle_a[10];
    Objid       sv_my_id;
    int         sv_PPP_seq_number;
    double      sv_time;
    Ici*        sv_from_mac_ici_ptr;
    } cp_fddi_sink_tcp_state;

#define pr_state_ptr            ((cp_fddi_sink_tcp_state*) SimI_Mod_State_Ptr)
#define thru_gshandle           pr_state_ptr->sv_thru_gshandle
#define m_delay_gshandle        pr_state_ptr->sv_m_delay_gshandle
#define ete_delay_gshandle      pr_state_ptr->sv_ete_delay_gshandle
#define thru_gshandle_a         pr_state_ptr->sv_thru_gshandle_a
#define m_delay_gshandle_a      pr_state_ptr->sv_m_delay_gshandle_a
#define ete_delay_gshandle_a    pr_state_ptr->sv_ete_delay_gshandle_a
#define t_gshandle              pr_state_ptr->sv_t_gshandle
#define t_gshandle_a            pr_state_ptr->sv_t_gshandle_a
#define my_id                   pr_state_ptr->sv_my_id
#define PPP_seq_number          pr_state_ptr->sv_PPP_seq_number
#define time                    pr_state_ptr->sv_time
#define from_mac_ici_ptr        pr_state_ptr->sv_from_mac_ici_ptr



/* Process model interrupt handling procedure */
```

```
void
cp_fddi_sink_tcp ()
    {
    double      delay, creat_time;
    Packet*     pkptr;
    Packet* pkptr1 ; /*5APR94*/
    Packet*     ppp_pkptr;/*15JUN94*/
    int             src_addr, my_addr;
    int             dest_addr;/*14APR94*/
    double      fddi_sink_ttrt;
    int         xmit_subq_index;/*5APR94*/
    int             load_balance_code; /*6APR94*/
    int             i,subq_no; /*25APR94*/
    int             index; /*10MAY94 */
    double      link_mon_trans_rate; /*15JUN94*/
    char        str0[512], str1 [512]; /* for diagnostics*/
    int             mon_pkt_size; /*26JUL*/
    double      sim_duration; /*26JUL*/
    Packet*     ip_pkptr; /*11SEP*/


    FSM_ENTER (cp_fddi_sink_tcp)

    FSM_BLOCK_SWITCH
        {
        /*--------------------------------------------------------*/
        /** state (DISCARD) enter executives **/
        FSM_STATE_ENTER_UNFORCED (0, state0_enter_exec, "DISCARD")
            {
            /* determine type of interrupt:10MAY94 */
            switch (op_intrpt_type())
                {
            case OPC_INTRPT_STAT:
            /* the interrupt is caused by the transmitters' status */
                {
                index = cp_intrpt_stat();
                switch(index)
                    {
                    case XMITTER_ONE:
                        {
                        buffer[0] = op_stat_local_read(XMITTER_ONE);
                        break;
```

```
                    }
                case XMITTER_TWO:
                    {
                    buffer[1] = op_stat_local_read(XMITTER_TWO);
                    break;
                    }
                case XMITTER_THREE:
                    {
                    buffer[2] = op_stat_local_read(XMITTER_THREE);
                    break;
                    }
                case XMITTER_FOUR:
                    {
                    buffer[3] = op_stat_local_read(XMITTER_FOUR);
                    break;
                    }
                default:
                    {
                    op_sim_end("*** FDDI-CDL : FATAL ERROR","Unexpected stat
interrupt","","");
                    }
                }
            break;
            }
        case OPC_INTRPT_STRM:
        /* the interrupt is caused by the incoming packets */
            {
        /* get the packet and the interface control info */
        pkptr = op_pk_get (op_intrpt_strm ());
        from_mac_ici_ptr = op_intrpt_ici ();

        /* 20JAN94: get the packet's priority level, which */
        /* will be used to index arrays of thruput and delay */
        /* computations. */
        op_ici_attr_get (from_mac_ici_ptr, "pri", &pri_set); /* 11SEP94 */

        /* determine the time of creation of the packet */
        op_ici_attr_get (from_mac_ici_ptr, "cr_time", &creat_time);

        /* 11SEP94:determine the destination address of the packet */
        op_ici_attr_get (from_mac_ici_ptr, "dest_addr", &dest_addr);

        /* 11SEP94:determine the source address of the packet */
```

```
op_ici_attr_get (from_mac_ici_ptr, "src_addr", &src_addr);

/* 7APR94: determine which load balancing algorithm is in use */
op_ima_obj_attr_get ( my_id, "load balancing algorithm", &load_balance_code );

/* 14APR94 : also get my own address */
op_ima_obj_attr_get ( my_id, "station_address", &my_addr);

/* destroy the packet */
/* op_pk_destroy (pkptr); */
/* 03FEB94: rather, enqueue the packet. This will be the */
/* first step toward developing a LAN bridging structure. */
/* -Nix */
/* op_subq_pk_insert (pri_set, pkptr, OPC_QPOS_TAIL); */

/* 14APR94: check the frame passed to "llc" is destined for */
/* this station. If it is update the local traffic */
/* statistics; if not, allocate the packets */
/* to the transmitters since they are destined for the remote lan */
/* update also incoming return link statistics for the frames */
/* which will be queued in llc_sink to be sent to remote lan.*/
/* -Karayakaylar */

if((dest_addr == my_addr)&&(src_addr < my_addr))
    {
    /* add in its size */
    fddi_sink_total_bits += op_pk_total_size_get (pkptr);
    fddi_sink_total_bits_a[pri_set] += op_pk_total_size_get (pkptr); /* 20JAN-
20APR94 */

    /* accumulate delays */
    delay = op_sim_time () - creat_time;
    fddi_sink_accum_delay += delay;
    fddi_sink_accum_delay_a[pri_set] += delay; /* 20JAN-20APR94 */

    /* keep track of peak delay value */
    if (delay > fddi_sink_peak_delay)
        fddi_sink_peak_delay = delay;

    /* 20JAN94: keep track by priority levels as well 23JAN-20APR94 */
    if (delay > fddi_sink_peak_delay_a[pri_set])
    fddi_sink_peak_delay_a[pri_set] = delay;
```

```
  /* increment packet counter; 20JAN94 */
  fddi_sink_total_pkts++;
  fddi_sink_total_pkts_a[pri_set]++;

  /* if a multiple of 25 packets is reached, update stats */
  /* 03FEB94: [0]->[7] represent asynch priorities 1->8, */
  /* respectively; [8] represents synchronous traffic, */
  /* and [9] represents overall asynchronous traffic.-Nix */
  /*if (fddi_sink_total_pkts % 25 == 0)
      {
      op_stat_global_write (thru_gshandle,
          fddi_sink_total_bits / op_sim_time ());

  op_stat_global_write (thru_gshandle_a[pri_set],
fddi_sink_total_bits_a[0] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[0],
fddi_sink_total_bits_a[1] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[1],
fddi_sink_total_bits_a[pri_set] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[2],
fddi_sink_total_bits_a[2] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[3],
fddi_sink_total_bits_a[3] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[4],
fddi_sink_total_bits_a[4] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[5],
fddi_sink_total_bits_a[5] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[6],
fddi_sink_total_bits_a[6] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[7],
fddi_sink_total_bits_a[7] / op_sim_time());
  op_stat_global_write (thru_gshandle_a[8],
fddi_sink_total_bits_a[8] / op_sim_time());
   */
     /* 30JAN94: gather all asynch stats into one overall figure */
  /*op_stat_global_write (thru_gshandle_a[9],
(fddi_sink_total_bits - fddi_sink_tota'  _a[8]) /
op_sim_time());
*/

/* (fddi_s  _total_bits_a[0] + fddi_sink_total_bits_a[1] + */
/* fddi_s   otal_bits_a[2] + fddi_sink_total_bits_a[3] + */
```

```
/* fddi_sink_total_bits_a[4] + fddi_sink_total_bits_a[5] + */
/* fddi_sink_total_bits_a[6] + fddi_sink_total_bits_a[7]) / */
/* op_sim_time()); */

/*
   op_stat_global_write (m_delay_gshandle,
         fddi_sink_accum_delay / fddi_sink_total_pkts);

   op_stat_global_write (m_delay_gshandle_a[0],
fddi_sink_accum_delay_a[0] / fddi_sink_total_pkts_a[0]);
   op_stat_global_write (m_delay_gshandle_a[1],
fddi_sink_accum_delay_a[1] / fddi_sink_total_pkts_a[1]);
   op_stat_global_write (m_delay_gshandle_a[2],
fddi_sink_accum_delay_a[2] / fddi_sink_total_pkts_a[2]);
   op_stat_global_write (m_delay_gshandle_a[3],
fddi_sink_accum_delay_a[3] / fddi_sink_total_pkts_a[3]);
   op_stat_global_write (m_delay_gshandle_a[4],
fddi_sink_accum_delay_a[4] / fddi_sink_total_pkts_a[4]);
   op_stat_global_write (m_delay_gshandle_a[5],
fddi_sink_accum_delay_a[5] / fddi_sink_total_pkts_a[5]);
   op_stat_global_write (m_delay_gshandle_a[6],
fddi_sink_accum_delay_a[6] / fddi_sink_total_pkts_a[6]);
   op_stat_global_write (m_delay_gshandle_a[7],
fddi_sink_accum_delay_a[7] / fddi_sink_total_pkts_a[7]);
   op_stat_global_write (m_delay_gshandle_a[8],
fddi_sink_accum_delay_a[8] / fddi_sink_total_pkts_a[8]);
*/
      /* 30JAN94: gather all asynch stats into one figure */
   /*op_stat_global_write (m_delay_gshandle_a[9],
(fddi_sink_accum_delay - fddi_sink_accum_delay_a[8]) /
(fddi_sink_total_pkts - fddi_sink_total_pkts_a[8]));
*/

/* (fddi_sink_accum_delay_a[0] + fddi_sink_accum_delay_a[1] + */
/* fddi_sink_accum_delay_a[2] + fddi_sink_accum_delay_a[3] + */
/* fddi_sink_accum_delay_a[4] + fddi_sink_accum_delay_a[5] + */
/* fddi_sink_accum_delay_a[6] + fddi_sink_accum_delay_a[7]) / */
/* (fddi_sink_total_pkts_a[0] + fddi_sink_total_pkts_a[1] + */
/* fddi_sink_total_pkts_a[2] + fddi_sink_total_pkts_a[3] + */
/* fddi_sink_total_pkts_a[4] + fddi_sink_total_pkts_a[5] + */
/* fddi_sink_total_pkts_a[6] + fddi_sink_total_pkts_a[7])); */
```

```
        /* also record actual delay values */
        /*op_stat_global_write (ete_delay_gshandle, delay);
        op_stat_global_write (ete_delay_gshandle_a[pri_set], delay);
        }
        */
    } /* end of if(dest_addr == my_addr)&&(src_addr < my_addr) statement */

/* 20APR94: check the frame passed to "llc" is destined for remote lan */
/* This will allow only the packets to be counted for CDL traffic.*/
/* -Karayakaylar */
else
    {
    /* add in its size */
    fddilp1_total_bits += op_pk_total_size_get (pkptr);
    fddilp1_total_bits_a[pri_set] += op_pk_total_size_get (pkptr); /* 20APR94 */

    /* increment packet counter; 20APR94 */
    fddilp1_total_pkts++;
    fddilp1_total_pkts_a[pri_set]++;

    /* if a multiple of 25 packets is reached, update stats */
    /* [0]->[7] represent asynch priorities 1->8, */
    /* respectively; [8] represents synchronous traffic, */
    /* and [9] represents overall asynchronous traffic.-Nix */
    /*if (fddilp1_total_pkts % 25 == 0)
        {
        op_stat_global_write (t_gshandle,
            fddilp1_total_bits / op_sim_time ());

    op_stat_global_write (t_gshandle_a[pri_set],
 fddilp1_total_bits_a[0] / op_sim_time());
    op_stat_global_write (t_gshandle_a[0],
 fddilp1_total_bits_a[1] / op_sim_time());
    op_stat_global_write (t_gshandle_a[1],
 fddilp1_total_bits_a[pri_set] / op_sim_time());
    op_stat_global_write (t_gshandle_a[2],
 fddilp1_total_bits_a[2] / op_sim_time());
    op_stat_global_write (t_gshandle_a[3],
 fddilp1_total_bits_a[3] / op_sim_time());
    op_stat_global_write (t_gshandle_a[4],
 fddilp1_total_bits_a[4] / op_sim_time());
    op_stat_global_write (t_gshandle_a[5],
```

```
fddilp1_total_bits_a[5] / op_sim_time());
   op_stat_global_write (t_gshandle_a[6],
fddilp1_total_bits_a[6] / op_sim_time());
   op_stat_global_write (t_gshandle_a[7],
fddilp1_total_bits_a[7] / op_sim_time());
   op_stat_global_write (t_gshandle_a[8],
fddilp1_total_bits_a[8] / op_sim_time());
    */
       /* gather all asynch stats into one overall figure */
   /*op_stat_global_write (t_gshandle_a[9],
(fddilp1_total_bits - fddilp1_total_bits_a[8]) /
op_sim_time());

*/
/* (fddilp1_total_bits_a[0] + fddilp1_total_bits_a[1] + */
/* fddilp1_total_bits_a[2] + fddilp1_total_bits_a[3] + */
/* fddilp1_total_bits_a[4] + fddilp1_total_bits_a[5] + */
/* fddilp1_total_bits_a[6] + fddilp1_total_bits_a[7]) / */
/* op_sim_time()); */

       /* }if (fddilp1_total_pkts % 25 == 0) */

   } /* end of else if(dest_addr == my_addr)&&(src_addr < my_addr) statement */

/* Frames coming to cp_sink only go to this station or CDL 11SEP94*/
if (dest_addr == my_addr)
   {
   /* send packet to IP layer */
op_pk_nfd_get (pkptr, "datagram", &ip_pkptr);
   op_pk_destroy (pkptr);
   op_pk_send (ip_pkptr, IP_OUT_STRM);
   }
else /* send to the CDL*/
   {
   /* 11SEP94: first step is to take LLC ICI that was sent with*/
   /* frame and attach directly to LLC frame instead of event*/
   /* This is necessary because LLC frame doesn't have all req'd info */
   op_pk_ici_set(pkptr, from_mac_ici_ptr);

   /*15JUN94 : before allocating, encapsulate the FDDI frame into a PPP packet */
   ppp_pkptr = op_pk_create_fmt ("ppp_ml");
   op_pk_nfd_set (ppp_pkptr, "seq_number", PPP_seq_number++);
   op_pk_nfd_set (ppp_pkptr, "FDDI_frame", pkptr);
```

```
                /* 14APR94 :allocate the packets to llc_sink subqueues */

                /* 6APR94 -Karayakayiar*/
                /* check if load balancing algorithm is circular */
                /* zero(0) is the circular load balancing code */
                if (load_balance_code == 0)
                    {
                /* 5APR94 */
                    /* Apply load balancing to insert the packets in the */
                    /* subqueues, in a circular order */
                    subq_no = subq_index % 4;
                    op_subq_pk_insert(subq_no, ppp_pkptr, OPC_QPOS_TAIL); /*15JUN
altered to send ppp*/
                subq_index++;
                }

                 /* 25APR94 */
                /* check if load balancing algorithm is empty allocation */
                /* one(1) is the empty allocation load balancing code */
                if (load_balance_code == 1)
                    {
                    /* Apply load balancing to insert the packets in the */
                    /* subqueues by choosing the subqueue which has the maximum current */
                    /* number of free packet slots */
                    subq_no = op_subq_index_map(OPC_QSEL_MAX_FREE_PKSIZE);
                    op_subq_pk_insert(subq_no, ppp_pkptr, OPC_QPOS_TAIL); /*15JUN
altered to send ppp*/
                }

                }/*else if(dest_addr = my_addr) statement */

                break;
            }/* end of case OPC_INTRPT_STRM statement */

            case OPC_INTRPT_SELF: /*27JUL94*/
            /* if it is a self-interrupt, it is time to send a monitoring packet. Send one. */
                {
                ppp_pkptr = op_pk_create_fmt("ppp");
                op_pk_nfd_set (ppp_pkptr,"pid_h",0xc0);
                op_pk_nfd_set (ppp_pkptr,"pid_l",0x21);
                op_ima_obj_attr_get (my_id,"monitoring pkt size", &mon_pkt_size);
                op_pk_bulk_size_set (ppp_pkptr, mon_pkt_size);
```

```
if (load_balance_code == 1)
        {
        subq_no = op_subq_index_map(OPC_QSEL_MAX_FREE_PKSIZE);
        op_subq_pk_insert (subq_no,ppp_pkptr,OPC_QPOS_TAIL);
        }
    else if (load_balance_code == 0)
        {
        subq_no = subq_index % 4;
        op_subq_pk_insert (subq_no,ppp_pkptr,OPC_QPOS_TAIL);
        subq_index++;
        }
    break;
    }

}/* end of switch */


        /* check if each subqueue is not empty and transmitter is not busy */
        /* irregardless if an interrupt is received or not JAUG94*/
    for (xmit_subq_index = 0;xmit_subq_index <=3; ++xmit_subq_index)
    {
        if ((!op_subq_empty(xmit_subq_index))&&(buffer[xmit_subq_index] ==
0.0))
            {
            /*access the first packet in the subqueue */
            pkptr1 = op_subq_pk_remove (xmit_subq_index, OPC_QPOS_HEAD);
        /* forward it to the destination xmitter */
            /* associated with the subqueue index */
            op_pk_send (pkptr1, xmit_subq_index );
            /*if (op_sim_debug()==OPC_TRUE)
            printf ("packet sent to cp xmtr from subqueue %d\n",xmit_subq_index);*/
            }
    }
    }


/** blocking after enter executives of unforced state. **/
FSM_EXIT (1,cp_fddi_sink_tcp)


/** state (DISCARD) exit executives **/
FSM_STATE_EXIT_UNFORCED (0, state0_exit_exec, "DISCARD")
    {
```

```
    }


/** state (DISCARD) transition processing **/
FSM_INIT_COND (END_OF_SIM)
FSM_DFLT_COND
FSM_TEST_LOGIC ("DISCARD")

FSM_TRANSIT_SWITCH
    {
    FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;)
    FSM_CASE_TRANSIT (1, 0, state0_enter_exec, ;)
    }
/*---------------------------------------------------------*/



/** state (STATS) enter executives **/
FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "STATS")
    {
    /* At end of simulation, scalar performance statistics */
    /* and input parameters are written out. */

       op_stat_scalar_write ("RL Throughput (bps), Priority 1",
            fddilp1_total_bits_a[0] / op_sim_time ()); /*20APR94*/

    op_stat_scalar_write ("RL Throughput (bps), Priority 2",
    fddilp1_total_bits_a[1] / op_sim_time ());

    op_stat_scalar_write ("RL Throughput (bps), Priority 3",
    fddilp1_total_bits_a[2] / op_sim_time ());

    op_stat_scalar_write ("RL Throughput (bps), Priority 4",
    fddilp1_total_bits_a[3] / op_sim_time ());

    op_stat_scalar_write ("RL Throughput (bps), Priority 5",
    fddilp1_total_bits_a[4] / op_sim_time ());

    op_stat_scalar_write ("RL Throughput (bps), Priority 6",
    fddilp1_total_bits_a[5] / op_sim_time ());

    op_stat_scalar_write ("RL Throughput (bps), Priority 7",
    fddilp1_total_bits_a[6] / op_sim_time ());
```

```
op_stat_scalar_write ("RL Throughput (bps), Priority 8",
fddilp1_total_bits_a[7] / op_sim_time ());

op_stat_scalar_write ("RL Throughput (bps), Asynchronous",
(fddilp1_total_bits - fddilp1_total_bits_a[8]) / op_sim_time ());


/* (fddilp1_total_bits_a[0] + fddilp1_total_bits_a[1] + */
/* fddilp1_total_bits_a[2] + fddilp1_total_bits_a[3] + */
/* fddilp1_total_bits_a[4] + fddilp1_total_bits_a[5] + */
/* fddilp1_total_bits_a[6] + fddilp1_total_bits_a[7]) / */
/* op_sim_time ()); */


op_stat_scalar_write ("RL Throughput (bps), Synchronous",
fddilp1_total_bits_a[8] / op_sim_time ());

op_stat_scalar_write ("RL Throughput (bps), Total",
fddilp1_total_bits / op_sim_time ()); /*20APR94*/


/* Only one station needs to do this */
if (!fddi_sink_scalar_write)
   {
   /* set the scalar write flag */
   fddi_sink_scalar_write = 1;

   op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 1",
         fddi_sink_accum_delay_a[0] / fddi_sink_total_pkts_a[0]);

op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 2",
fddi_sink_accum_delay_a[1] / fddi_sink_total_pkts_a[1]);

op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 3",
fddi_sink_accum_delay_a[2] / fddi_sink_total_pkts_a[2]);

op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 4",
fddi_sink_accum_delay_a[3] / fddi_sink_total_pkts_a[3]);

op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 5",
```

```
fddi_sink_accum_delay_a[4] / fddi_sink_total_pkts_a[4]);

op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 6",
fddi_sink_accum_delay_a[5] / fddi_sink_total_pkts_a[5]);

op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 7",
fddi_sink_accum_delay_a[6] / fddi_sink_total_pkts_a[6]);

op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Priority 8",
fddi_sink_accum_delay_a[7] / fddi_sink_total_pkts_a[7]);

op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Asynchronous",
(fddi_sink_accum_delay - fddi_sink_accum_delay_a[8]) /
(fddi_sink_total_pkts - fddi_sink_total_pkts_a[8]));

/* (fddi_sink_accum_delay_a[0] + fddi_sink_accum_delay_a[1] + */
/* fddi_sink_accum_delay_a[2] + fddi_sink_accum_delay_a[3] + */
/* fddi_sink_accum_delay_a[4] + fddi_sink_accum_delay_a[5] + */
/* fddi_sink_accum_delay_a[6] + fddi_sink_accum_delay_a[7]) / */
/* (fddi_sink_total_pkts_a[0] + fddi_sink_total_pkts_a[1] + */
/* fddi_sink_total_pkts_a[2] + fddi_sink_total_pkts_a[3] + */
/* fddi_sink_total_pkts_a[4] + fddi_sink_total_pkts_a[5] + */
/* fddi_sink_total_pkts_a[6] + fddi_sink_total_pkts_a[7])); */


op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Synchronous",
fddi_sink_accum_delay_a[8] / fddi_sink_total_pkts_a[8]);

op_stat_scalar_write ("Mean End-to-End Delay-0 (sec.), Total",
fddi_sink_accum_delay / fddi_sink_total_pkts);

   op_stat_scalar_write ("Throughput-0 (bps), Priority 1",
         fddi_sink_total_bits_a[0] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Priority 2",
fddi_sink_total_bits_a[1] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Priority 3",
fddi_sink_total_bits_a[2] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Priority 4",
fddi_sink_total_bits_a[3] / op_sim_time ());
```

```
op_stat_scalar_write ("Throughput-0 (bps), Priority 5",
fddi_sink_total_bits_a[4] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Priority 6",
fddi_sink_total_bits_a[5] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Priority 7",
fddi_sink_total_bits_a[6] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Priority 8",
fddi_sink_total_bits_a[7] / op_sim_time ()),

op_stat_scalar_write ("Throughput-0 (bps), Asynchronous",
(fddi_sink_total_bits - fddi_sink_total_bits_a[8]) / op_sim_time ());


/* (fddi_sink_total_bits_a[0] + fddi_sink_total_bits_a[1] + */
/* fddi_sink_total_bits_a[2] + fddi_sink_total_bits_a[3] + */
/* fddi_sink_total_bits_a[4] + fddi_sink_total_bits_a[5] + */
/* fddi_sink_total_bits_a[6] + fddi_sink_total_bits_a[7]) / */
/* op_sim_time ()); */



op_stat_scalar_write ("Throughput-0 (bps), Synchronous",
fddi_sink_total_bits_a[8] / op_sim_time ());

op_stat_scalar_write ("Throughput-0 (bps), Total",
fddi_sink_total_bits / op_sim_time ());


   op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 1",
         fddi_sink_peak_delay_a[0]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 2",
fddi_sink_peak_delay_a[1]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 3",
fddi_sink_peak_delay_a[2]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 4",
fddi_sink_peak_delay_a[3]);
```

```
op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 5",
fddi_sink_peak_delay_a[4]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 6",
fddi_sink_peak_delay_a[5]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 7",
fddi_sink_peak_delay_a[6]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Priority 8",
fddi_sink_peak_delay_a[7]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Synchronous",
fddi_sink_peak_delay_a[8]);

op_stat_scalar_write ("Peak End-to-End Delay-0 (sec.), Overall",
fddi_sink_peak_delay);


   /* Write the TTRT value for ring 0. This preserves*/
   /* the old behavior for single-ring simulations.*/
   op_stat_scalar_write ("TTRT (sec.) - Ring 0",
         fddi_t_opr [0]);

/* 12JAN94: obtain offered load information from the Environment */
/* file; this will be used to provide abscissa information that */
/* can be plotted in the Analysis Editor (see "fddi_sink" STATS */
/* state. To the user: it's your job to keep these current in */
/* the Environment File. -Nix */
op_ima_sim_attr_get (OPC_IMA_DOUBLE, "total_offered_load_0",
&Offered_Load);
op_ima_sim_attr_get (OPC_IMA_DOUBLE, "asynch_offered_load_0",
&Asynch_Offered_Load);


/* 12JAN94: write the total offered load for this run */
op_stat_scalar_write ("Total Offered Load-0 (Mbps)",
Offered_Load);

op_stat_scalar_write ("Asynchronous Offered Load-0 (Mbps)",
Asynch_Offered_Load);
   }
}
```

```
/** blocking after enter executives of unforced state. **/
FSM_EXIT (3,cp_fddi_sink_tcp)


/** state (STATS) exit executives **/
FSM_STATE_EXIT_UNFORCED (1, state1_exit_exec, "STATS")
    {
    }


/** state (STATS) transition processing **/
FSM_TRANSIT_MISSING ("STATS")
/*---------------------------------------------------------*/



/** state (INIT) enter executives **/
FSM_STATE_ENTER_FORCED (2, state2_enter_exec, "INIT")
    {
    /* get the gshandles of the global statistic to be obtained */
    /* 20JAN94: set array format */
    /*
    thru_gshandle_a[0] = op_stat_global_reg ("pri 1 throughput-0 (bps)");
    thru_gshandle_a[1] = op_stat_global_reg ("pri 2 throughput-0 (bps)");
    thru_gshandle_a[2] = op_stat_global_reg ("pri 3 throughput-0 (bps)");
    thru_gshandle_a[3] = op_stat_global_reg ("pri 4 throughput-0 (bps)");
    thru_gshandle_a[4] = op_stat_global_reg ("pri 5 throughput-0 (bps)");
    thru_gshandle_a[5] = op_stat_global_reg ("pri 6 throughput-0 (bps)");
    thru_gshandle_a[6] = op_stat_global_reg ("pri 7 throughput-0 (bps)");
    thru_gshandle_a[7] = op_stat_global_reg ("pri 8 throughput-0 (bps)");
    thru_gshandle_a[8] = op_stat_global_reg ("synch throughput-0 (bps)");
    thru_gshandle_a[9] = op_stat_global_reg ("async throughput-0 (bps)");
    thru_gshandle = op_stat_global_reg ("total throughput-0 (bps)");

    m_delay_gshandle_a[0] = op_stat_global_reg ("pri 1 mean delay-0 (sec.)");
    m_delay_gshandle_a[1] = op_stat_global_reg ("pri 2 mean delay-0 (sec.)");
    m_delay_gshandle_a[2] = op_stat_global_reg ("pri 3 mean delay-0 (sec.)");
    m_delay_gshandle_a[3] = op_stat_global_reg ("pri 4 mean delay-0 (sec.)");
    m_delay_gshandle_a[4] = op_stat_global_reg ("pri 5 mean delay-0 (sec.)");
    m_delay_gshandle_a[5] = op_stat_global_reg ("pri 6 mean delay-0 (sec.)");
    m_delay_gshandle_a[6] = op_stat_global_reg ("pri 7 mean delay-0 (sec.)");
```

```
m_delay_gshandle_a[7] = op_stat_global_reg ("pri 8 mean delay-0 (sec.)");
m_delay_gshandle_a[8] = op_stat_global_reg ("synch mean delay-0 (sec.)");
m_delay_gshandle_a[9] = op_stat_global_reg ("async mean delay-0 (sec.)");
m_delay_gshandle = op_stat_global_reg ("total mean delay-0 (sec.)");

ete_delay_gshandle_a[0] = op_stat_global_reg ("pri 1 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[1] = op_stat_global_reg ("pri 2 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[2] = op_stat_global_reg ("pri 3 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[3] = op_stat_global_reg ("pri 4 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[4] = op_stat_global_reg ("pri 5 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[5] = op_stat_global_reg ("pri 6 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[6] = op_stat_global_reg ("pri 7 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[7] = op_stat_global_reg ("pri 8 end-to-end delay-0 (sec.)");
ete_delay_gshandle_a[8] = op_stat_global_reg ("synch end-to-end delay-0 (sec.)");
ete_delay_gshandle = op_stat_global_reg ("total end-to-end delay-0 (sec.)");

t_gshandle_a[0] = op_stat_global_reg ("pri 1 RL throughput (bps)");
t_gshandle_a[1] = op_stat_global_reg ("pri 2 RL throughput (bps)");
t_gshandle_a[2] = op_stat_global_reg ("pri 3 RL throughput (bps)");
t_gshandle_a[3] = op_stat_global_reg ("pri 4 RL throughput (bps)");
t_gshandle_a[4] = op_stat_global_reg ("pri 5 RL throughput (bps)");
t_gshandle_a[5] = op_stat_global_reg ("pri 6 RL throughput (bps)");
t_gshandle_a[6] = op_stat_global_reg ("pri 7 RL throughput (bps)");
t_gshandle_a[7] = op_stat_global_reg ("pri 8 RL throughput (bps)");
t_gshandle_a[8] = op_stat_global_reg ("synch RL throughput (bps)");
t_gshandle_a[9] = op_stat_global_reg ("async RL throughput (bps)");
t_gshandle = op_stat_global_reg ("total RL throughput (bps)");
*/
link_gshandle[0] = op_stat_global_reg ("Link 0 jamming");
link_gshandle[1] = op_stat_global_reg ("Link 1 jamming");
link_gshandle[2] = op_stat_global_reg ("Link 2 jamming");
link_gshandle[3] = op_stat_global_reg ("Link 3 jamming");
jammer_stats_init = OPC_TRUE;

subq_no = 0;

/* 7APR94:determine id of own processor to use in finding */
/* load balancing attribute and station address of the bridge node */
my_id = op_id_self();

/* 15JUN94: get rate for link monitoring packet transmission */
op_ima_obj_attr_get (my_id,"link_monitor_trans_rate", &link_mon_trans_rate );
```

```
    /* 26JUL94: get duration of the simulation */
    op_ima_sim_attr_get (OPC_IMA_DOUBLE, "duration", &sim_duration);

    /* 26JUL94:generate a self interrupt at each time that a monitoring packet*/
    /* is to be sent. Since these packets are to be sent at a fixed terate (per */
    /* second), the only way to guarantee packet generation in this event */
    /* driven simulation is to create an interrupt at the appropriate sim */
    /* times. These interrupts will cause a monitoring packet to be sent */
    /* through the discard state.*/

    for (time= op_sim_time(); time<=sim_duration; time+=link_mon_trans_rate)
        {
        op_intrpt_schedule_self (time, 0xc021);
     }

    /*11SEP94: create an ICI to attach to all LLC frames transiting */
    /*the CDL. Though the LLC frames will be encapsulated in PPP, */
    /*the ICI will be associated with the LLC frame and will remain */
    /*intact until gotten before the LLC frame is sent to the MAC */
    from_mac_ici_ptr = op_ici_create ("fddi_mac_ind_ctp");
    }


/** state (INIT) enter executives **/
FSM_STATE_EXIT_FORCED (2, state2_exit_exec, "INIT")
    {
    }


/** state (INIT) transition processing **/
FSM_INIT_COND (END_OF_SIM)
FSM_DFLT_COND
FSM_TEST_LOGIC ("INIT")

FSM_TRANSIT_SWITCH
    {
    FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;)
    FSM_CASE_TRANSIT (1, 0, state0_enter_exec, ;)
    }
/*---------------------------------------------------------*/
```

```
        }


    FSM_EXIT (2,cp_fddi_sink_tcp)
    }



void
cp_fddi_sink_tcp_svar (prs_ptr,var_name,var_p_ptr)
    cp_fddi_sink_tcp_state*prs_ptr;
    char            *var_name, **var_p_ptr;
    {

    FIN (cp_fddi_sink_tcp_svar (prs_ptr))

    *var_p_ptr = VOS_NIL;
    if (Vos_String_Equal ("thru_gshandle" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_thru_gshandle);
    if (Vos_String_Equal ("m_delay_gshandle" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_m_delay_gshandle);
    if (Vos_String_Equal ("ete_delay_gshandle" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_ete_delay_gshandle);
    if (Vos_String_Equal ("thru_gshandle_a" , var_name))
        *var_p_ptr = (char *) (prs_ptr->sv_thru_gshandle_a);
    if (Vos_String_Equal ("m_delay_gshandle_a" , var_name))
        *var_p_ptr = (char *) (prs_ptr->sv_m_delay_gshandle_a);
    if (Vos_String_Equal ("ete_delay_gshandle_a" , var_name))
        *var_p_ptr = (char *) (prs_ptr->sv_ete_delay_gshandle_a);
    if (Vos_String_Equal ("t_gshandle" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_t_gshandle);
    if (Vos_String_Equal ("t_gshandle_a" , var_name))
        *var_p_ptr = (char *) (prs_ptr->sv_t_gshandle_a);
    if (Vos_String_Equal ("my_id" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_my_id);
    if (Vos_String_Equal ("PPP_seq_number" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_PPP_seq_number);
    if (Vos_String_Equal ("time" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_time);
    if (Vos_String_Equal ("from_mac_ici_ptr" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_from_mac_ici_ptr);
```

```
    FOUT;
    }



void
cp_fddi_sink_tcp_diag ()
    {
    double      delay, creat_time;
    Packet*     pkptr;
    Packet* pkptr1 ; /*5APR94*/
    Packet*     ppp_pkptr;/*15JUN94*/
    int             src_addr, my_addr;
    int             dest_addr;/*14APR94*/
    double      fddi_sink_ttrt;
    int         xmit_subq_index;/*5APR94*/
    int             load_balance_code; /*6APR94*/
    int             i,subq_no; /*25APR94*/
    int             index; /*10MAY94 */
    double      link_mon_trans_rate; /*15JUN94*/
    char        str0[512], str1 [512]; /* for diagnostics*/
    int             mon_pkt_size; /*26JUL*/
    double      sim_duration; /*26JUL*/
    Packet*     ip_pkptr; /*11SEP*/

    FIN (cp_fddi_sink_tcp_diag ())


    /* find out why straight line syndrome */

        op_prg_odb_print_major ("----------DEBUGGING for straight line---------
",OPC_NIL);
        sprintf (str0,"count of packets : (%d)",subq_index);
     sprintf (str1,"subqueue no : (%d)",subq_no);
        op_prg_odb_print_minor (str0,str1, OPC_NIL);


    FOUT;
    }
```

```
void
cp_fddi_sink_tcp_terminate ()
    {
    double      delay, creat_time;
    Packet*     pkptr;
    Packet* pkptr1 ; /*5APR94*/
    Packet*     ppp_pkptr;/*15JUN94*/
    int             src_addr, my_addr;
    int             dest_addr;/*14APR94*/
    double      fddi_sink_ttrt;
    int         xmit_subq_index;/*5APR94*/
    int             load_balance_code; /*6APR94*/
    int             i,subq_no; /*25APR94*/
    int             index; /*10MAY94 */
    double      link_mon_trans_rate; /*15JUN94*/
    char        str0[512], str1 [512]; /* for diagnostics*/
    int             mon_pkt_size; /*26JUL*/
    double      sim_duration; /*26JUL*/
    Packet*     ip_pkptr; /*11SEP*/

    FIN (cp_fddi_sink_tcp_terminate ())


    FOUT;
    }



Compcode
cp_fddi_sink_tcp_init (pr_state_pptr)
    cp_fddi_sink_tcp_state**pr_state_pptr;
    {
    static VosT_Cm_Obtypeobtype = OPC_NIL;

    FIN (cp_fddi_sink_tcp_init (pr_state_pptr))

    if (obtype == OPC_NIL)
        {
        if (Vos_Catmem_Register ("proc state vars (cp_fddi_sink_tcp)",
            sizeof (cp_fddi_sink_tcp_state), Vos_Nop, &obtype) == VOSC_FAILURE)
            FRET (OPC_COMPCODE_FAILURE)
        }
```

```
    if ((*pr_state_pptr = (cp_fddi_sink_tcp_state*) Vos_Catmem_Alloc (obtype, 1)) ==
OPC_NIL)
        FRET (OPC_COMPCODE_FAILURE)
    else
        {
        (*pr_state_pptr)->current_block = 4;
        FRET (OPC_COMPCODE_SUCCESS)
        }
    }
```

APPENDIX J

TCP RING 0 MAC MODULE CODE
"cp_fddi_mac_tcp.pr.c"

Unchanged portions of this file have been deleted for brevity.

```
/* Process model C form file: cp_fddi_mac_tcp.pr.c */
/* Portions of this file Copyright (C) MIL 3, Inc. 1992 */



/* OPNET system definitions */
#include <opnet.h>
#include "cp_fddi_mac_tcp.pr.h"
FSM_EXT_DECS



/* Header block */
/* Define a timer structure used to implement */
/* the TRT and THT timers. The primitives defined to */
/* operate on these timers can be found in the */
/* function block of this process model. */
typedef struct
    {
    int             enabled;
    double          start_time;
    double          accum;
    double          target_accum;
    } FddiT_Timer;


/* Declare certain primitives dealing with timers */
double          fddi_timer_remaining ();
FddiT_Timer* fddi_timer_create ();
double          fddi_timer_value ();

/* Scratch strings for trace statements */
char            str0 [512], str1 [512],
```

```
/* define constants particular to this implementation */
#define FDDI_MAX_STATIONS                512

/* define possible values for the frame control field */
#define FDDI_FC_FRAME                    0
#define FDDI_FC_TOKEN                    1

/* define possible service classes for frames */
#define FDDI_SVC_ASYNC                   0
#define FDDI_SVC_SYNC                    1

/* define input stream indices */
#define FDDI_LLC_STRM_IN                 1
#define FDDI_PHY_STRM_IN                 0

/* define output stream indices */
#define FDDI_LLC_STRM_OUT                1
#define FDDI_PHY_STRM_OUT                0

/* define token classes */
#define FDDI_TK_NONRESTRICTED            0
#define FDDI_TK_RESTRICTED               1

/* Ring Constants */
#define FDDI_TX_RATE                     1.0e+08
#define FDDI_SA_SCAN_TIME                28.0e-08

/* Token transmission time: based on 6 symbols plus 16 symbols of preamble */
#define FDDIC_TOKEN_TX_TIME              88.0e-08


/* Codes used to differentiate remote interrupts */
#define FDDIC_TRT_EXPIRE                 0
#define FDDIC_TK_INJECT                  1


/* Define symbolic expressions used on transition */
/* conditions and in executive statements. */
#define TRT_EXPIRE                       \
    (op_intrpt_type () == OPC_INTRPT_REMOTE && op_intrpt_code () ==
FDDIC_TRT_EXPIRE)
```

```
#define TK_RECEIVED\
    phy_arrival && \
    frame_control == FDDI_FC_TOKEN

#define RC_FRAME    \
    phy_arrival && \
    frame_control == FDDI_FC_FRAME

#define FRAME_ARRIVAL\
    op_intrpt_type () == OPC_INTRPT_STRM &&\
    op_intrpt_strm () == FDDI_LLC_STRM_IN

#define STRIPmy_address == src_addr

/* Define the maximum value for ring_id. This is the*/
/* maximum number of FDDI rings that can exist in a*/
/* simulation. Note that if this number is changed,*/
/* the initialization for fddi_claim_start below must*/
/* also be modified accordingly.*/
#define FDDI_MAX_RING_ID          8

/* Declare the operative TTRT value 'T_Opr' which is the final*/
/* negotiated value of TTRT. This value is shared by all stations*/
/* on a ring so that all agree on its value.*/
double    fddi_t_opr [FDDI_MAX_RING_ID];
#define   Fddi_T_Opr   (fddi_t_opr [ring_id])

/* This flag indicates that the negotiation for the final TTRT*/
/* has not yet begun. It is statically initialized here, and*/
/* is reset by the first station which modifies T_Opr.*/
/* Initialize to 1 for all rings.*/
static
int             fddi_claim_start [FDDI_MAX_RING_ID] = {1,1,1,1,1,1,1,1};
#define   Fddi_Claim_Start(fddi_claim_start [ring_id])


/* Declare station latency parameters. */
/* These are true globals, so they do not need to be arrays. */
double    Fddi_St_Latency;
double    Fddi_Prop_Delay;

/* Declare globals for Token Acceleration Mechanism.*/
/* Hop delay and token acceleration are true globals.*/
```

```
double   Fddi_Tk_Hop_Delay;
static
int              Fddi_Tk_Accelerate = 1;

/* These are actually values shared by all nodes on a ring,*/
/* so they must be defined as arrays.*/
double   fddi_tk_block_base_time [FDDI_MAX_RING_ID];
#define  Fddi_Tk_Block_Base_Time(fddi_tk_block_base_time [ring_id])

int              fddi_tk_block_base_station [FDDI_MAX_RING_ID];
#define  Fddi_Tk_Block_Base_Station(fddi_tk_block_base_station [ring_id])

int              fddi_tk_blocked [FDDI_MAX_RING_ID];
#define  Fddi_Tk_Blocked(fddi_tk_blocked [ring_id])

int              fddi_num_stations [FDDI_MAX_RING_ID];
#define  Fddi_Num_Stations(fddi_num_stations [ring_id])

int              fddi_num_registered [FDDI_MAX_RING_ID];
#define  Fddi_Num_Registered(fddi_num_registered [ring_id])

Objid    fddi_address_table [FDDI_MAX_RING_ID][FDDI_MAX_STATIONS];
#define  Fddi_Address_Table(fddi_address_table [ring_id])

/* Below is part of the OPBUG 2081 patch, FB ended here, before. -Nix */

/* Event handles for the TRT are maintained at a global level to */
/* allow token acceleration mechanism to adjust these as necessary */
/* when blocking and reinjecting the token. TRT_handle simply */
/* represents the TRT for the local MAC*/
Evhandle         fddi_trt_handle [FDDI_MAX_RING_ID][FDDI_MAX_STATIONS];
#define  Fddi_Trt_Handle(fddi_trt_handle [ring_id])
#define TRT_handle       Fddi_Trt_Handle [my_address]

/* Similarly, the TRT data structure is maintained on a global level. */
FddiT_Timer*fddi_trt [FDDI_MAX_RING_ID] [FDDI_MAX_STATIONS];
#define  Fddi_Trt        (fddi_trt [ring_id])
#define  TRT             Fddi_Trt [my_address]

/* Registers to record the expiration time of each TRT when token is blocked. */
double   fddi_trt_exp_time [FDDI_MAX_RING_ID] [FDDI_MAX_STATIONS];
#define Fddi_Trt_Exp_Time(fddi_trt_exp_time [ring_id])
```

```
/* the 'Late_Ct' flag is declared on a global level so that it can be */
/* set at the tim ewhere the token is injected back into the ring. */
int             fddi_late_ct [FDDI_MAX_RING_ID] [FDDI_MAX_STATIONS];
#define   Fddi_Late_Ct (fddi_late_ct [ring_id])
#define Late_Ct         Fddi_Late_Ct [my_address]


/* Convenient macro for setting TRT for a given station and absolute time. */
#define TRT_SET(station_id,abs_time)\
fddi_timer_set (Fddi_Trt [station_id], abs_time - op_sim_time());\
Fddi_Trt_Handle [station_id] = op_intrpt_schedule_remote (abs_time,\
        FDDIC_TRT_EXPIRE, Fddi_Address_Table [station_id]);



/* State variable definitions */
typedef struct
    {
    FSM_SYS_STATE
    int         sv_ring_id;
    FddiT_Timer* sv_THT;
    double      sv_T_Req;
    double      sv_T_Pri [8];
    Objid       sv_my_objid;
    int         sv_spawn_token;
    int         sv_my_address;
    int         sv_orig_src_addr;
    Packet*     sv_tk_pkptr;
    double      sv_sync_bandwidth;
    double      sv_sync_pc;
    int         sv_restricted;
    int         sv_res_peer;
    int         sv_tk_registered;
    Ici*        sv_to_llc_ici_ptr;
    int         sv_tk_trace_on;
    } cp_fddi_mac_tcp_state;

#define pr_state_ptr            ((cp_fddi_mac_tcp_state*) Siml_Mod_State_Ptr)
#define ring_id                 pr_state_ptr->sv_ring_id
#define THT                     pr_state_ptr->sv_THT
#define T_Req                   pr_state_ptr->sv_T_Req
#define T_Pri                   pr_state_ptr->sv_T_Pri
#define my_objid                pr_state_ptr->sv_my_objid
```

```
#define spawn_token             pr_state_ptr->sv_spawn_token
#define my_address              pr_state_ptr->sv_my_address
#define orig_src_addr           pr_state_ptr->sv_orig_src_addr
#define tk_pkptr                pr_state_ptr->sv_tk_pkptr
#define sync_bandwidth          pr_state_ptr->sv_sync_bandwidth
#define sync_pc                 pr_state_ptr->sv_sync_pc
#define restricted              pr_state_ptr->sv_restricted
#define res_peer                pr_state_ptr->sv_res_peer
#define tk_registered           pr_state_ptr->sv_tk_registered
#define to_llc_ici_ptr          pr_state_ptr->sv_to_llc_ici_ptr
#define tk_trace_on             pr_state_ptr->sv_tk_trace_on



/* Process model interrupt handling procedure */


void
cp_fddi_mac_tcp ()
    {
    /* Packets and ICI's */
    Packet*     mac_frame_ptr;
    Packet*     pdu_ptr;
    Packet*     pkptr;
    Packet*     data_pkptr;
    Ici*        ici_ptr;

    /* Packet Fields and Attributes */
    int             req_pri, svc_class, req_tk_class;
    int             frame_control, src_addr, dest_addr;
    int             pk_len, pri_level;

    /* Token - Related */
    int             tk_usable, res_station, tk_class;
    int             current_tk_class;
    double      accum_sync;

    /* Timer - Related */
    double      tx_time, timer_remaining, accum_bandwidth;
    double      tht_value;
```

```
/* Miscellaneous */
int             i;
int             spawn_station, phy_arrival;
char        error_string [512];
int             num_frames_sent, num_bits_sent;

/* 26DEC93: loop management variables, used in RCV_TK */
/* and ENCAP states. -Nix */
int NUM_PRIOS;
int punt;
int q_check;

/*********************************************************/
/* 11SEP94: added creation_time */
double creation_time;


FSM_ENTER (cp_fddi_mac_tcp)

FSM_BLOCK_SWITCH
    {
    /*--------------------------------------------------------*/
    /** state (INIT) enter executives **/
    FSM_STATE_ENTER_FORCED (0, state0_enter_exec, "INIT")
        {
        /* Obtain the station's address . This is an attribute */
        /* of this process. Addressing is simplified by */
        /* simply using integers, and only one mode. */
        /* This mode is 16 bit addressing unless the */
        /* packet format 'fddi_mac_fr' is modified. */
        my_objid = op_id_self(); /* 29DEC93 */
        op_ima_obj_attr_get (my_objid, "station_address", &my_address);

        /* Register the station's object id in a global table. */
        /* This table is used by the mechanism which improves */
        /* simulation efficiency by 'jumping over' idle periods */
        /* rather than circulating an unusable token. */
        fddi_station_register (my_address, my_objid);


        /* Obtain the station latency for tokens and frames. */
        /* Default value is set at 100 nanoseconds. */
        Fddi_St_Latency = 100.0e-09;
```

```
op_ima_sim_attr_get (OPC_IMA_DOUBLE, "station_latency",
&Fddi_St_Latency);

/* Obtain the propagation delay separating stations. */
/* This value is given in seconds with default value 3.3 microseconds. */
Fddi_Prop_Delay = 3.3e-06;
op_ima_sim_attr_get (OPC_IMA_DOUBLE, "prop_delay", &Fddi_Prop_Delay);

/* Derive the Delay for a 'hop' of a freely circulating packet. */
Fddi_Tk_Hop_Delay = Fddi_Prop_Delay + Fddi_St_Latency;


/* The T_Pri [] state variable array supports priority */
/* assignments on a station by station basis by */
/* establishing a correspondence between integer priority */
/* levels assigned to frames and the maximum values of the */
/* token holding timer (THT) which would allow packets to be */
/* sent. Eight levels are supported here, but this can easily */
/* be changed by redimensioning the priority array. */
/* By default all levels are identical here, allowing */
/* any frame to make use of the token, so that in fact */
/* priority levels are not used in the default case. */

/* 01JAN94: (8-i) is a quick attempt to impart different weighting */
/* scales on each priority level, and is not necessarily realistic.-Nix */
/* Be aware of integer-double arithmetic conflicts ie, 1/8 = 0. -Nix */

op_ima_obj_attr_get(my_objid, "T_Req", &T_Req);
for (i = 0; i < 8; i++)
    {
 T_Pri[i] = ((double)(i + 1.0)/8.0) * Fddi_T_Opr;
 /* printf("MAC INIT: T_Pri[%d] is %lf; */
 /* Fddi_T_Opr is %lf\n", i, T_Pri[i], Fddi_T_Opr); */
    }


/* Create the token holding timer (THT) used to restrict the */
/* asynchronous bandwidth consumption of the station */
THT = fddi_timer_create ();

/* Create the token rotation timer (TRT) used to measure the */
/* rotations of the token, detect late tokens and initialize */
/* the THT timer before asynchronous transmissions. */
```

```
TRT = fddi_timer_create ();

/* Set the TRT timer to expire in one TTRT */
TRT_SET (my_address, op_sim_time () + Fddi_T_Opr);

/* Initialize the Late_Ct variable which keeps track. */
/* of the number of TRT expirations. */
Late_Ct = 0;

/* initially the ring operates in nonrestricted mode */
restricted = 0;


/* Create an Interface Control Information structure */
/* to use when delivering received frames to the LLC. */
/*****************************************************************/
/*11SEP94: changed to new format, adding pri and cr_time*/
/*****************************************************************/
to_llc_ici_ptr = op_ici_create ("fddi_mac_ind_tcp");

/* The 'tk_registered' variable indicates if the station */
/* has registered its intent to use the token. */
tk_registered = 0;


/* Determine if the model is to make use of the token */
/* 'acceleration' mechanism. If not, every passing of the */
/* token will be explicityly modeled, leading to large */
/* number of events being scheduled when the ring is idle */
/* (i.e, no stations have data to send). */
op_ima_sim_attr_get (OPC_IMA_INTEGER, "accelerate_token",
    &Fddi_Tk_Accelerate);


/* Obtain the synchronous bandwidth assigned */
/* to this station. It is expressed as a */
/* percentage of TTRT, and then converted to seconds */
op_ima_obj_attr_get (my_objid, "sync bandwidth", &sync_pc);
sync_bandwidth = sync_pc * Fddi_T_Opr;


/* Only one station in the ring is selected to */
/* introduce the first token. Test if this station is it. */
```

```
            /* If so, set the 'spawn_token' flag. */
            /* op_ima_sim_attr_get (OPC_IMA_INTEGER, "spawn station",
&spawn_station); */
            /* spawn_token = (spawn_station == my_address); */
            /* If the station is to spawn the token, create */
            /* the packet which represents the token. */
            /* 14APR94 :the bridges will spawn token in both rings */
            /* -Karayakaylar */
            spawn_token =1;
            if (spawn_token)
               {
               tk_pkptr = op_pk_create_fmt ("fddi_mac_tk");

               /* assign its frame control field */
               op_pk_nfd_set (tk_pkptr, "fc", FDDI_FC_TOKEN);

               /* the first token issued is non-restricted */
               op_pk_nfd_set (tk_pkptr, "class", FDDI_TK_NONRESTRICTED);

               /* The transition will be made into the ISSU_TK */
               /* state where the tk_usable variable is used. */
               /* In case any data has been generated, piset */
               /* this variable to one. */
               tk_usable = 1;
               }


            /* When sending packets the variable accum_bandwidth is */
            /* used as a scheduling base. Init this value to zero. */
            /* This statement is required in case this is the spawning */
            /* station, and the next state entered is ISSUE_TK */
            accum_bandwidth = 0.0;









            }
```

```
/** state (INIT) exit executives **/
FSM_STATE_EXIT_FORCED (0, state0_exit_exec, "INIT")
        {
        }


/** state (INIT) transition processing **/
FSM_INIT_COND (spawn_token)
FSM_DFLT_COND
FSM_TEST_LOGIC ("INIT")

FSM_TRANSIT_SWITCH
        {
        FSM_CASE_TRANSIT (0, 2, state2_enter_exec, ;)
        FSM_CASE_TRANSIT (1, 1, state1_enter_exec, ;)
        }
/*---------------------------------------------------------*/


/** state (FR_REPEAT) enter executives **/
FSM_STATE_ENTER_FORCED (6, state6_enter_exec, "FR_REPEAT")
        {
        /* Extract the destination address of the frame. */
        op_pk_nfd_get (pkptr, "dest_addr", &dest_addr);

        /* If the frame is for this station, make a copy */
        /* of the frame's data field and forward it to */
        /* the higher layer. */
        /* 14APR94 : In order to send the frames which are */
        /* addressed to the remote lan, check the address database */
        /* of remote lan. Frames addressed to the remote lan shouldn't */
        /* be repeated in the local ring -- This is a simple forwarding */
        /* decision algorithm, one of the bridge's function */
        /* - Karayakaylar */
        if((dest_addr == my_address)||(dest_addr > my_address))
            {
            /* record total size of the frame (including data) */
            pk_len = op_pk_total_size_get (pkptr);

            /* decapsulate the data contents of the frame */
        /* 29JAN94: a new field, "pri", has been added to */
        /* the fddi_llc_fr packet format in the Parameters */
        /* Editor, so that output statistics can be */
```

```
/* generated by class and priority. -Nix */
   op_pk_nfd_get (pkptr, "info", &data_pkptr);
   op_pk_nfd_get (pkptr, "pri", &pri_level);

   /* The source and destination address are placed in the */
   /* LLC's ICI before delivering the frame's contents. */
   op_ici_attr_set (to_llc_ici_ptr, "src_addr", src_addr);
   op_ici_attr_set (to_llc_ici_ptr, "dest_addr", dest_addr);
   /*****************************************************************/
   /*11SEP94: added pri and cr_time to fddi_mac_ind ici***/
   /* both are needed for data collection***/
/*****************************************************************/
   op_ici_attr_set (to_llc_ici_ptr, "pri", pri_level);
   op_pk_nfd_get (pkptr, "cr_time", &creation_time);
   op_ici_attr_set (to_llc_ici_ptr, "cr_time", creation_time);

   op_ici_install (to_llc_ici_ptr);

   /* Because, as noted in the FR_RCV state, only the */
   /* frame's leading edge has arrived at this time, the */
   /* complete frame can only be delivered to the higher */
   /* layer after the frame's transmission delay has elapsed. */
   /* (since decapsulation of the frame data contents has occured. */
   /* the original MAC frame length is used to calculate delay) */
   tx_time = (double) pk_len / FDDI_TX_RATE;
   op_pk_send_delayed (data_pkptr, FDDI_LLC_STRM_OUT, tx_time);

   /* Note that the standard specifies that the original */
   /* frame should be passed along until the originating station */
   /* receives it, at which point it is stripped from the ring  */
   /* However, in the simulation model, there is no interest */
   /* in letting the frame continue past its destination unless */
   /* group addresses are used, so that the same frame could be */
   /* destined for several stations. Here the frame is stripped */
   /* for efficiency as it reaches the destination; if the model */
   /* is modified to include group addresses, this should be changed */
   /* so that the frame is copied and the original repeated. */
   /* Logic is already present for stripping the frame at the origin. */
   op_pk_destroy (pkptr);
   }
/* 14APR94 . the frames belong to this ring should be repeated. */
/* Thus, local traffic is constrained.-- This is filtering decision */
/* One of the bridge's function - Karayakaylar */
```

```
    else{
        /* Repeat the original frame on the ring and account for */
        /* the latency through the station and the propagation delay */
        /* for a single hop. */
        /* (Only the originating station can strip the frame). */
        op_pk_send_delayed (pkptr, FDDI_PHY_STRM_OUT,
            Fddi_St_Latency + Fddi_Prop_Delay);
        }
    }


/** state (FR_REPEAT) exit executives **/
FSM_STATE_EXIT_FORCED (6, state6_exit_exec, "FR_REPEAT")
    {
    }


/** state (FR_REPEAT) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
/*---------------------------------------------------------*/


/** state (ENCAP) enter executives **/
FSM_STATE_ENTER_FORCED (8, state8_enter_exec, "ENCAP")
    {
    /* A frame has arrived from a higher layer; place it in 'pdu_ptr'. */
    pdu_ptr = op_pk_get (op_intrpt_strm ());

    /* Also get the interface control information */
    /* associated with the new frame. */
    ici_ptr = op_intrpt_ici ();
    if (ici_ptr == OPC_NIL)
        {
        sprintf (error_string, "Simulation aborted; error in object (%d)",
            op_id_self ());
        op_sim_end (error_string, "fddi_mac: required ICI not received", " ", " ");
        }

    /* Extract the requested service class */
    /* (e.g, synchronous or asynchronous). */
    if (op_ici_attr_exists (ici_ptr, "svc_class"))
        op_ici_attr_get (ici_ptr, "svc_class", &svc_class);
    else svc_class = FDDI_SVC_ASYNC;
```

```
/* Extract the destination address. */
op_ici_attr_get (ici_ptr, "dest_addr", &dest_addr);

/* Extract the original source address from ICI :16APR94 */
op_ici_attr_get (ici_ptr, "src_addr", &orig_src_addr);

/* If the frame is asynchronous, the priority and */
/* requested token class parameter may be specified. */
if (svc_class == FDDI_SVC_ASYNC)
    {
    /* Extract the requested priority level. */
    if (op_ici_attr_exists (ici_ptr, "pri"))
        op_ici_attr_get (ici_ptr, "pri", &req_pri);
    else req_pri = 0;

    /* Extract the token class (restricted or non-restricted). */
    if (op_ici_attr_exists (ici_ptr, "tk_class"))
        op_ici_attr_get (ici_ptr, "tk_class", &req_tk_class);
    else req_tk_class = FDDI_TK_NONRESTRICTED;
    }

/* Check for the default ICI values;if they are not present */
/* compose the frame:21APR94 */
if( dest_addr != orig_src_addr){

/* Compose a mac frame from all these elements. */
/*****************************************************************/
/* 11SEP94: mac frame format is changed to fddi_mac_fr_tcp***/
/*****************************************************************/
mac_frame_ptr = op_pk_create_fmt ("fddi_mac_fr_tcp");
op_pk_nfd_set (mac_frame_ptr, "svc_class", svc_class);
op_pk_nfd_set (mac_frame_ptr, "dest_addr", dest_addr);
/*op_pk_nfd_set (mac_frame_ptr, "src_addr", my_address);*/
/* here original source address should be kept in mac frame :16APR94*/
op_pk_nfd_set (mac_frame_ptr, "src_addr", orig_src_addr);
op_pk_nfd_set (mac_frame_ptr, "info", pdu_ptr);

/*****************************************************************/
/* 11SEP94: cr_time is added to the MAC frame (0bits) because it*/
/* is needed for calculations, but taken out of the LLC frame  */
/*****************************************************************/
op_ici_attr_get (ici_ptr, "cr_time", &creation_time);
```

```
op_pk_nfd_set (mac_frame_ptr, "cr_time", creation_time);

printf("\ndest_addr = %5d\n",dest_addr);
printf("orig_src_addr= %5d\n",orig_src_addr);

if (svc_class == FDDI_SVC_ASYNC)
    {
    op_pk_nfd_set (mac_frame_ptr, "tk_class", req_tk_class);
    op_pk_nfd_set (mac_frame_ptr, "pri", req_pri);
    }

/* 04JAN94: if the frame is synchronous, assign it a separate */
/* priority so that it may be assigned its own subqueue, and */
/* thereby be assigned its own probe for monitoring. -Nix */
if (svc_class == FDDI_SVC_SYNC)
 {
 op_pk_nfd_set (mac_frame_ptr, "pri", 8);
 }

/* Assign the frame control field, which in the model */
/* is used to distinguish between tokens and ordinary */
/* frames on the ring. */
op_pk_nfd_set (mac_frame_ptr, "fc", FDDI_FC_FRAME);


/* Enqueue the frame at the tail of the queue. */
/* 27DEC93: at the tail of the prioritized queue. */
/* 04JAN94: must distinguish between synch & asynch. */
if (svc_class == FDDI_SVC_ASYNC)
 {
 op_subq_pk_insert (req_pri, mac_frame_ptr, OPC_QPOS_TAIL);
 }
if (svc_class == FDDI_SVC_SYNC)
 {
 op_subq_pk_insert (8, mac_frame_ptr, OPC_QPOS_TAIL);
 }


/* if this station has not yet registered its intent to */
/* use the token, it may do so now since it has data to send */
if (!tk_registered)
    {
    fddi_tk_register ();
```

```
            tk_registered = 1;
            }

        } /* end of if(dest_addr != orig_src_addr) statement */




        }


/** state (ENCAP) exit executives **/
FSM_STATE_EXIT_FORCED (8, state8_exit_exec, "ENCAP")
        {
        }


/** state (ENCAP) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
/*---------------------------------------------------------*/
```

APPENDIX K

TCP RING 1 LLC_SRC MODULE CODE "sp_fddi_gen_tcp.pr.c"

```
/* Process model C form file: sp_fddi_gen_tcp.pr.c */
/* Portions of this file Copyright (C) MIL 3, Inc. 1992 */



/* OPNET system definitions */
#include <opnet.h>
#include "sp_fddi_gen_tcp.pr.h"
FSM_EXT_DECS



/* Header block */
#define MAC_LAYER_OUT_STREAM    0
#define LLC_SINK_OUT_STREAM     1 /*18APR94*/

#define IP_IN_STRM              4 /*11SEP94*/

/* define possible service classes for frames */
#define FDDI_SVC_ASYNC          0
#define FDDI_SVC_SYNC           1

/* define token classes */
#define FDDI_TK_NONRESTRICTED   0
#define FDDI_TK_RESTRICTED      1

/* define output statistics */
#define RATIO_OUTSTAT           0
#define LINK_STATUS_OUTSTAT 1

/* link_status constants 30 JUL94*/
#define GOOD        0
#define BAD         2

/* history trend constants 30JUL94*/
```

```
#define DOWN            0
#define UP              1

/* redefine absolute value function for floating pt values 28JUL94*/
#define abs(i) (i)<0 ? -(i) : (i)

struct history_element
/* format for linked list of history values 26JUL94*/
{
    int num_errs;
    struct history_element *next;
} *history;

/* function declaration 28JUL94*/
struct history_element *create_history();



/* State variable definitions */
typedef struct
    {
    FSM_SYS_STATE
    Objid       sv_mac_objid;
    Objid       sv_my_id;
    int         sv_station_addr;
    int         sv_src_addr;
    int         sv_ppp_pid_h;
    int         sv_ppp_pid_l;
    Ici*        sv_mac_iciptr;
    Ici*        sv_llc_iciptr;
    Packet*     sv_pkptr1;
    Packet*     sv_ppp_pkptr1;
    Packet*     sv_ppp_pkptr2;
    int         sv_hist_len;
    int         sv_link_status;
    int         sv_pkts_in_error;
    double      sv_old_ratio;
    } sp_fddi_gen_tcp_state;

#define pr_state_ptr            ((sp_fddi_gen_tcp_state*) SimI_Mod_State_Ptr)
#define mac_objid               pr_state_ptr->sv_mac_objid
#define my_id                   pr_state_ptr->sv_my_id
#define station_addr            pr_state_ptr->sv_station_addr
```

```
#define src_addr                pr_state_ptr->sv_src_addr
#define ppp_pid_h               pr_state_ptr->sv_ppp_pid_h
#define ppp_pid_l               pr_state_ptr->sv_ppp_pid_l
#define mac_iciptr              pr_state_ptr->sv_mac_iciptr
#define llc_iciptr              pr_state_ptr->sv_llc_iciptr
#define pkptr1                  pr_state_ptr->sv_pkptr1
#define ppp_pkptr1              pr_state_ptr->sv_ppp_pkptr1
#define ppp_pkptr2              pr_state_ptr->sv_ppp_pkptr2
#define hist_len                pr_state_ptr->sv_hist_len
#define link_status             pr_state_ptr->sv_link_status
#define pkts_in_error           pr_state_ptr->sv_pkts_in_error
#define old_ratio               pr_state_ptr->sv_old_ratio



/* Process model interrupt handling procedure */


void
sp_fddi_gen_tcp ()
    {
    Packet    *pkptr, *ip_pkptr;
    int           pklen;
    int           dest_addr;
    int           i,j, restricted;
    int pkt_prio;
    int           num_errors;
    double    new_ratio, upper_thresh, lower_thresh;
    double    LQR_trans_delta;
    double    creation_time;
    Ici           *ip_iciptr;


    FSM_ENTER (sp_fddi_gen_tcp)

    FSM_BLOCK_SWITCH
        {
        /*---------------------------------------------------------*/
        /** state (INIT) enter executives **/
        FSM_STATE_ENTER_UNFORCED (0, state0_enter_exec, "INIT")
            {
```

```
/* determine id of own processor to use in finding attrs */
my_id = op_id_self ();

/* determine object id of connected 'mac' layer process */
mac_objid = op_topo_assoc (my_id, OPC_TOPO_ASSOC_OUT,
        OPC_OBJMTYPE_MODULE, MAC_LAYER_OUT_STREAM);

/* determine the address assigned to it */
/* which is also the address of this station */
op_ima_obj_attr_get (mac_objid, "station_address", &station_addr);



/* set up history array (dynamically allocated) which will maintain */
/* number of errors in each monitoring packet rcvd. The number of */
/* values (length of the array) saved will be determined by an */
/* environment attribute. 21JUL94 */
op_ima_obj_attr_get (my_id, "history length", &hist_len);
history = create_history(hist_len);
printf ("HISTORY\n");
if (op_sim_debug() == OPC_TRUE)
    {
    for (i=1;i<=hist_len;++i)
        {
     printf ("%d ",history->num_errs);
        history = history->next;
        }
    printf ("\n");
 }

/*11SEP94 set up ICI structure */
mac_iciptr = op_ici_create ("fddi_mac_req_tcp");
llc_iciptr = op_ici_create ("fddi_mac_ind_tcp");
}


/** blocking after enter executives of unforced state. **/
FSM_EXIT (1,sp_fddi_gen_tcp)


/** state (INIT) exit executives **/
FSM_STATE_EXIT_UNFORCED (0, state0_exit_exec, "INIT")
    {
```

```
        }


        /** state (INIT) transition processing **/
        FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
        /*---------------------------------------------------------*/



        /** state (ARRIVAL) enter executives **/
        FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "ARRIVAL")
            {
            /* This station should receive frames from the other lan as long as */
            /* there are frames in the input streams addressed to this lan */
            /*check if the interrupt type is stream interrupt and from the CDL*//*12APR94*/
            if((op_intrpt_type() == OPC_INTRPT_STRM)&& (op_intrpt_strm()!=
IP_IN_STRM))
                {
                /* if it is, get the packet in the input stream causing interrupt */
                /* modified for PPP 12JUL94 */
                ppp_pkptr1 = op_pk_get(op_intrpt_strm());
                /* determine type of PPP packet 12JUL94 */
                op_pk_nfd_get(ppp_pkptr1,"pid_h", &ppp_pid_h);
                /* case on pid_h: 00 - data, 0xc0 - control 13JUL*/
              switch (ppp_pid_h)
                    {
                    case 0x00: /* if data, strip header and send as FDDI frame */
                        /* strip off PPP header 12JUL94*/
                        if (op_sim_debug()==OPC_TRUE)
                        {
                        printf ("pkt rcvd at sp: data\n");
                        num_errors
=op_td_get_int(ppp_pkptr1,OPC_TDA_PT_NUM_ERRORS);
                        printf ("number of errors in data - %d\n",num_errors);
                        }
                        op_pk_nfd_get(ppp_pkptr1,"FDDI_frame",&pkptr1);
                        op_pk_destroy(ppp_pkptr1);
                        /* get ICI associated with FDDI LLC frame 11SEP94*/
                        llc_iciptr = op_pk_ici_get (pkptr1);
                        /* get the destination address of the frame: 11SEP94 */
                        op_ici_attr_get(llc_iciptr, "dest_addr", &dest_addr);
                        /* check if this frame is for the remote bridge station(bridge in surface
lan) */
```

```
                    if(dest_addr == station_addr)
                    {
                    /* if it is, send the packet to llc_sink directly */
                    /* in order to prevent overhead of mac access */
                    op_ici_install (llc_iciptr);
                    op_pk_send(pkptr1, LLC_SINK_OUT_STREAM);/*19APR94*/
                    }
                    else
                    /* this packet is to send to mac */
                    {
                    /* determine the source address of the frame */
                    op_ici_attr_get(llc_iciptr, "src_addr", &src_addr);
                    /* set up an ICI structure to communicate parameters to */
                    /* MAC layer process (initialized in INIT state)*/
                    /* place the original source address into the ICI *//* 16APR94 */
                    /* "fddi_mac_req" is modified so that it contains the original */
                    /* source address from the local lan(collection platform) */
                    op_ici_attr_set(mac_iciptr, "src_addr", src_addr);
                    /* place the destination address into the ICI */ /*12APR94*/
                    op_ici_attr_set(mac_iciptr, "dest_addr", dest_addr);
                    /* assign the service class and requested token class */
                    /* At this moment the frames coming from the remote lan are assumed */
                    /* to have the same priority as synchronous frames in order not to
accumulate */
                    /* packets on the bridge station mac and instead to deliver their
destinations */
                    /* as soon as possible */
                    /* 10SEP94: I don't know why Selcuk did this, so I'll leave the values*/
                    /* but change to conform with the new packet/ICI formats */
                    op_ici_attr_set(mac_iciptr, "svc_class", FDDI_SVC_SYNC);
                    op_ici_attr_set(mac_iciptr, "pri", 8);
                    op_ici_attr_set(mac_iciptr, "tk_class", FDDI_TK_NONRESTRICTED);

                    op_ici_attr_get(llc_iciptr, "cr_time", &creation_time);
                    op_ici_attr_set(mac_iciptr, "cr_time", creation_time);
                    /* send the packet coupled with the ICI */
                    op_ici_install(mac_iciptr);
                    op_pk_send(pkptr1, MAC_LAYER_OUT_STREAM);
                    }
                    break;
                  case 0xc0: /* either monitoring packet or LQR */
                    op_pk_nfd_get (ppp_pkptr1,"pid_1",&ppp_pid_1);
                    switch (ppp_pid_1)
```

```
                        {
                        case 0x21: /*monitoring packet*/
                        printf ("MONITORING PACKET RECEIVED\n");
                        num_errors
=op_td_get_int(ppp_pkptr1,OPC_TDA_PT_NUM_ERRORS);
                        printf (" NUMBER OF ERRORS -->%d\n",num_errors);
                        if ((num_errors != 0) && (history->num_errs==0))
                        pkts_in_error++;
                        else if ((num_errors == 0) && (history->num_errs != 0))
                        pkts_in_error--;
                        printf (" TOTAL NUMBER OF PACKETS IN ERROR --
>%d\n",pkts_in_error);

                        history->num_errs = num_errors;
                        history = history->next;

                        if (op_sim_debug()==OPC_TRUE)
                        /* print out history values 30JUL94 */
                        {
                        for (i=1;i<=hist_len;++i)
                        {
                        printf ("%d ",history->num_errs);
                        history = history->next;
                        }
                        printf ("\n");
                }

                        op_pk_destroy (ppp_pkptr1);

                        new_ratio = (double)pkts_in_error/(double)hist_len;
                        /* outstat(0) will be a record of the ratio 8AUG94*/
                        op_stat_local_write (RATIO_OUTSTAT, new_ratio);
                        op_ima_obj_atr_get (my_id, "LQR transmission delta",
&LQR_trans_delta);
                        if (op_sim_debug()==OPC_TRUE)
                        printf ("LQR delta - %f new_ratio - %f old_ratio -
%f\n",LQR_trans_delta,new_ratio,old_ratio);
                        /* 8AUG94 This condition allows for tolerance in the ratio
calculation(division) */
                        if (abs(new_ratio - old_ratio) >= LQR_trans_delta)
                        {
                        ppp_pkptr2 = op_pk_create_fmt ("ppp");
                        op_pk_nfd_set (ppp_pkptr2, "pid_h", 0xc0);
```

```
                    op_pk_nfd_set (ppp_pkptr2, "pid_l", 0x25);

                    op_ima_obj_attr_get (my_id, "upper hysteresis threshold",
&upper_thresh),
                    op_ima_obj_attr_get (my_id, "lower hysteresis threshold",
&lower_thresh);

                    /* status = GOOD 0 or BAD 2 trend = UP 1 or DOWN 0 2AUG94*/
               if (new_ratio >= upper_thresh)
                    link_status = BAD;
                    else if (new_ratio <= lower_thresh)
                    link_status = GOOD;
                    else link_status = link_status - (link_status % 2);/* remove trend value*/

                    /* outstat(1) will monitor link status 0-GOOD 1-BAD 8AUG94*/
                    op_stat_local_write (LINK_STATUS_OUTSTAT, (double) (link_status/
2));

                    /* trend will change each time */
                    if (new_ratio > old_ratio) /*up-trend */
                    link_status += UP;
                    else link_status += DOWN;



                    op_pk_nfd_set (ppp_pkptr2, "LQR_info", link_status);
                    if (op_sim_debug()==OPC_TRUE)
                    {
                    printf ("resulting LQR pkt\n");
                    op_pk_print(ppp_pkptr2);
                    }
                    old_ratio = new_ratio;
                    op_pk_send (ppp_pkptr2, LLC_SINK_OUT_STREAM);
                    }
                    if (op_sim_debug()==OPC_TRUE)
                    printf (" Latest ratio: %f\n",new_ratio);
                    break;

                    case 0x25:/*LQR from cp - not implemented yet*/
                    break;
                    default:
                    printf ("ERROR: UNKNOWN PPP PACKET -
pid_l=0x%x\n",ppp_pid_l);
```

```
                        op_pk_destroy (ppp_pkptr1);
                        break;
                        }
                    break;
                      default:
                        printf ("ERROR: UNKNOWN PPP PACKET -
pid_h=0x%x\n",ppp_pid_h);
                        op_pk_destroy (ppp_pkptr1);
                      break;
                      }

                    }
                else /* it should be an IP dgram */
                  if((op_intrpt_type() == OPC_INTRPT_STRM)&& (op_intrpt_strm()==
IP_IN_STRM))
                      {
                    /* get IP dgram and associated ICI */
                      ip_pkptr = op_pk_get (IP_IN_STRM);
                      ip_iciptr = op_intrpt_ici();

                      /* create an LLC frame to send to MAC */
                      /* ICI initialized in INIT state */
                      pkptr = op_pk_create_fmt ("fddi_llc_fr_tcp");
                    op_pk_nfd_set (pkptr, "datagram", ip_pkptr);

                      op_ici_attr_get (ip_iciptr,"dest_addr", &dest_addr);
                      if (dest_addr <= 9) /* destined for other LAN*/
                        {
                        op_ici_attr_set (llc_iciptr,"dest_addr", dest_addr);
                        op_ici_attr_set (llc_iciptr,"src_addr", station_addr);
                        op_ici_attr_set (llc_iciptr,"pri", 0);
                        op_ici_attr_set (llc_iciptr,"cr_time", op_sim_time());

                        /*install LLC ICI and send frame to LLC sink */
                        op_ici_install (llc_iciptr);
                        op_pk_send (pkptr,LLC_SINK_OUT_STREAM);
                        }
                      else /* destined for this LAN, so send frame with MAC ICI*/
                        {
                        /* place the destination address into the ICI */
                        op_ici_attr_set (mac_iciptr, "dest_addr", dest_addr);
                        /* place the source address into the ICI *//* 17APR94*/
                        op_ici_attr_set (mac_iciptr, "src_addr", station_addr);
```

```
            /* assign svc_class, token class, and pri IAW RFC 1390 */
                    op_ici_attr_set (mac_iciptr, "svc_class", FDDI_SVC_ASYNC);
                op_ici_attr_set (mac_iciptr, "pri",0);
                    /* Request only nonrestricted tokens after transmission */
                    op_ici_attr_set (mac_iciptr, "tk_class", FDDI_TK_NONRESTRICTED);
                    op_ici_attr_set (mac_iciptr, "cr_time", op_sim_time());

            op_ici_install (mac_iciptr);
                    op_pk_send (pkptr, MAC_LAYER_OUT_STREAM);
            }
                }
            }


        /** blocking after enter executives of unforced state. **/
        FSM_EXIT (3,sp_fddi_gen_tcp)


        /** state (ARRIVAL) exit executives **/
        FSM_STATE_EXIT_UNFORCED (1, state1_exit_exec, "ARRIVAL")
            {
            }


        /** state (ARRIVAL) transition processing **/
        FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
        /*---------------------------------------------------------*/



        }


    FSM_EXIT (0,sp_fddi_gen_tcp)
    }



void
sp_fddi_gen_tcp_svar (prs_ptr,var_name,var_p_ptr)
    sp_fddi_gen_tcp_state*prs_ptr;
    char        *var_name, **var_p_ptr;
```

```
{

FIN (sp_fddi_gen_tcp_svar (prs_ptr))

*var_p_ptr = VOS_NIL;
if (Vos_String_Equal ("mac_objid" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_mac_objid);
if (Vos_String_Equal ("my_id" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_my_id);
if (Vos_String_Equal ("station_addr" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_station_addr);
if (Vos_String_Equal ("src_addr" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_src_addr);
if (Vos_String_Equal ("ppp_pid_h" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_ppp_pid_h);
if (Vos_String_Equal ("ppp_pid_l" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_ppp_pid_l);
if (Vos_String_Equal ("mac_iciptr" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_mac_iciptr);
if (Vos_String_Equal ("llc_iciptr" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_llc_iciptr);
if (Vos_String_Equal ("pkptr1" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_pkptr1);
if (Vos_String_Equal ("ppp_pkptr1" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_ppp_pkptr1);
if (Vos_String_Equal ("ppp_pkptr2" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_ppp_pkptr2);
if (Vos_String_Equal ("hist_len" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_hist_len);
if (Vos_String_Equal ("link_status" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_link_status);
if (Vos_String_Equal ("pkts_in_error" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_pkts_in_error);
if (Vos_String_Equal ("old_ratio" , var_name))
     *var_p_ptr = (char *) (&prs_ptr->sv_old_ratio);

FOUT;
}


void
```

```
sp_fddi_gen_tcp_diag ()
    {
    Packet    *pkptr, *ip_pkptr;
    int           pklen;
    int           dest_addr;
    int           i,j, restricted;
    int pkt_prio;
    int           num_errors;
    double    new_ratio, upper_thresh, lower_thresh;
    double    LQR_trans_delta;
    double    creation_time;
    Ici           *ip_iciptr;

    FIN (sp_fddi_gen_tcp_diag ())


    FOUT;
    }



void
sp_fddi_gen_tcp_terminate ()
    {
    Packet    *pkptr, *ip_pkptr;
    int           pklen;
    int           dest_addr;
    int           i,j, restricted;
    int pkt_prio;
    int           num_errors;
    double    new_ratio, upper_thresh, lower_thresh;
    double    LQR_trans_delta;
    double    creation_time;
    Ici           *ip_iciptr;

    FIN (sp_fddi_gen_tcp_terminate ())


    FOUT;
    }
```

```
Compcode
sp_fddi_gen_tcp_init (pr_state_pptr)
    sp_fddi_gen_tcp_state**pr_state_pptr;
    {
    static VosT_Cm_Obtypeobtype = OPC_NIL;

    FIN (sp_fddi_gen_tcp_init (pr_state_pptr))

    if (obtype == OPC_NIL)
        {
        if (Vos_Catmem_Register ("proc state vars (sp_fddi_gen_tcp)",
            sizeof (sp_fddi_gen_tcp_state), Vos_Nop, &obtype) == VOSC_FAILURE)
            FRET (OPC_COMPCODE_FAILURE)
        }

    if ((*pr_state_pptr = (sp_fddi_gen_tcp_state*) Vos_Catmem_Alloc (obtype, 1)) ==
OPC_NIL)
        FRET (OPC_COMPCODE_FAILURE)
    else
        {
        (*pr_state_pptr)->current_block = 0;
        FRET (OPC_COMPCODE_SUCCESS)
        }
    }



    struct history_element *create_history(size)
        int size;

    /* returns a pointer to the first element in a circular linked list */
        {
        struct history_element *p, *new, *start;
     int i;

        for (i=1;i<=size;++i)
            {
            printf ("inside on iteration %d %d\n",i,size);
            new = (struct history_element*)malloc(sizeof(struct history_element));
            if (!new)
               {
               printf("ERROR: MEMORY ALLOCATION");
```

```
            exit(1);
            }
        if (i==1)
            {
            p = new;
            start = new;
            }
            else p->next = new; /* stick new element on end of list*/
    new->next = NULL;
        new->num_errs = 0;
    p = new;
}
   p->next = start;
return (start);
}
```

APPENDIX L

TCP RING 1 LLC_SINK MODULE CODE "sp_fddi_sink_tcp.pr.c"

```
/* Process model C form file: sp_fddi_sink.pr.c */
/* Portions of this file Copyright (C) MIL 3, Inc. 1992 */



/* OPNET system definitions */
#include <opnet.h>
#include "sp_fddi_sink.pr.h"
FSM_EXT_DECS



/* Header block */
/* Globals */
/* array format installed 20JAN94; positions 0-7 represent the asynch priority levels, PRIORITIES
+ 1 */
/* represents synch traffic, and grand totals are as given in the original. */

#define PRIORITIES 8 /* 20JAN94 */
#define         XMITTER_BUSY            0 /*10MAY94 */

#define         LLC_SOURCE_INPUT_STREAM  1 /*26JUL94*/
#define         MAC_INPUT_STREAM                0

static /* 05FEB94 */
double fddi2_sink_accum_delay = 0.0;
static /* 05FEB94 */
double          fddi2_sink_accum_delay_a[PRIORITIES + 1] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0};
static /* 05FEB94 */
int fddi2_sink_total_pkts = 0;
static /* 05FEB94 */
int                     fddi2_sink_total_pkts_a[PRIORITIES + 1] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
static /* 05FEB94 */
double fddi2_sink_total_bits = 0.0;
```

```
static /* 05FEB94 */
double          fddi2_sink_total_bits_a[PRIORITIES + 1] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0};
static /* 05FEB94 */
double fddi2_sink_peak_delay = 0.0;
static /* 05FEB94 */
double          fddi2_sink_peak_delay_a[PRIORITIES + 2] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0};
static /* 05FEB94 */
int                     fddi2_sink_scalar_write = 0;
static /* 05FEB94 */
int pri2_set = 20; /* 20JAN94 */
double busy = 0.0; /* 10MAY94 */

/* Statistics used for command link:21APR94 */
static
int fddilp2_total_pkts = 0;
static
int                     fddilp2_total_pkts_a[PRIORITIES + 1] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
static
double fddilp2_total_bits = 0.0;
static
double          fddilp2_total_bits_a[PRIORITIES + 1] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

/* Externally defined globals. */
extern doublefddi_t_opr [];

/*12JAN94:attributes from the Environment file */
double Offered_Load; /* 12JAN94 */
double Asynch_Offered_Load; /* 12JAN94 */

/* transition expressions */
#define END_OF_SIM op_intrpt_type() == OPC_INTRPT_ENDSIM



/* State variable definitions */
typedef struct
    {
    FSM_SYS_STATE
    Gshandle    sv_thru2_gshandle;
    Gshandle    sv_m2_delay_gshandle;
```

```
    Gshandle    sv_ete2_del y_gshandle;
    Gshandle    sv_thru2_gshandle_a[10];
    Gshandle    sv_m2_delay_gshandle_a[10];
    Gshandle    sv_ete2_delay_gshandle_a[9];
    Gshandle    sv_t2_gshandle;
    Gshandle    sv_t2_gshandle_a[10];
    Objid       sv_my_id;
    } sp_fddi_sink_state;

#define pr_state_ptr            ((sp_fddi_sink_state*) Siml_Mod_State_Ptr)
#define thru2_gshandle          pr_state_ptr->sv_thru2_gshandle
#define m2_delay_gshandle       pr_state_ptr->sv_m2_delay_gshandle
#define ete2_delay_gshandle     pr_state_ptr->sv_ete2_delay_gshandle
#define thru2_gshandle_a        pr_state_ptr->sv_thru2_gshandle_a
#define m2_delay_gshandle_a     pr_state_ptr->sv_m2_delay_gshandle_a
#define ete2_delay_gshandle_a   pr_state_ptr->sv_ete2_delay_gshandle_a
#define t2_gshandle             pr_state_ptr->sv_t2_gshandle
#define t2_gshandle_a           pr_state_ptr->sv_t2_gshandle_a
#define my_id                   pr_state_ptr->sv_my_id



/* Process model interrupt handling procedure */


void
sp_fddi_sink ()
    {
    double      delay, creat_time;
    Packet*     pkptr;
    Packet*     ppp_pkptr;
    Packet*     pkptr1 ; /*5APR94*/
    int         src_addr, my_addr;
    int         dest_addr;/*14APR94*/
    Ici*        from_mac_ici_ptr;
    double      fddi_sink_rtt;
    char        pk_format[10];
    int         input_stream;
    int         fd_index, FDDI_frame_size;
```

```
FSM_ENTER (sp_fddi_sink)

FSM_BLOCK_SWITCH
    {
    /*---------------------------------------------------------*/
    /** state (DISCARD) enter executives **/
    FSM_STATE_ENTER_UNFORCED (0, state0_enter_exec, "DISCARD")
        {
        /* determine the type of interrupt */
        switch(op_intrpt_type())
            {
            /* check if transmitter is busy */
            case OPC_INTRPT_STAT:
                {
                busy = op_stat_local_read (XMITTER_BUSY);
                break;
                }
            /* check if a packet has arrived */
            case OPC_INTRPT_STRM:
                {
                /* get the packet*/
                input_stream = op_intrpt_strm();
                pkptr = op_pk_get (input_stream);
                op_pk_format(pkptr,pk_format);

                /* if the packet is a ppp control packet from the llc_source */
                if ((strcmp(pk_format,"ppp")==0) &&(input_stream ==
LLC_SOURCE_INPUT_STREAM))
                    {
                    /* bypass all this and send pkt out on the command link */
                    }
                else
                    {
                    /* assume this is a FDDI frame */
                    from_mac_ici_ptr = op_intrpt_ici ();

                    /* 20JAN94: get the packet's priority level, which */
                    /* will be used to index arrays of thruput and delay */
                    /* computations. */
                    /* pri2_set = op_pk_priority_get (pkptr); doesn't work here */
                    op_pk_nfd_get (pkptr, "pri", &pri2_set); /* 29JAN94 */

                    /* determine the time of creation of the packet */
```

```
op_pk_nfd_get (pkptr, "cr_time", &creat_time);

/* determine the dest address of the packet */ /*18APR94*/
op_pk_nfd_get (pkptr, "dest_addr", &dest_addr);

/* 7APR94:determine id of own processor to use in finding */
/* station address of the bridge node */
my_id = op_id_self();

/* 14APR94 : also get my own address */
op_ima_obj_attr_get ( my_id, "station_address", &my_addr);

/* destroy the packet */
/* op_pk_destroy (pkptr); */
/* 03FEB94: rather, enqueue the packet. This will be the */
/* first step toward developing a LAN bridging structure. */
/* -Nix */
/* op_subq_pk_insert (pri_set, pkptr, OPC_QPOS_TAIL); */

/* 14APR94: check the frame passed to "llc" is destined for */
/* this station. If it is destroy the packet; if not, allocate the packets */
/* to the command link transmitter since they are destined for the remote
lan */
/* -Karayakaylar */
/* determine the packets coming from surface stations, this will */
/* be counted for local traffic */
/* 9(nine) is model specific, this is the "station_number" of */
/* collection platform bridge station */
if((dest_addr == my_addr)&&(src_addr > 9))
{
/* add in its size */
fddi2_sink_total_bits += op_pk_total_size_get (pkptr);
fddi2_sink_total_bits_a[pri2_set] += op_pk_total_size_get (pkptr); /*
20JAN-20APR94 */

/* accumulate delays */
delay = op_sim_time () - creat_time;
fddi2_sink_accum_delay += delay;
fddi2_sink_accum_delay_a[pri2_set] += delay; /* 20JAN-20APR94 */

/* keep track of peak delay value */
if (delay > fddi2_sink_peak_delay)
fddi2_sink_peak_delay = delay;
```

```
/* 20JAN94. keep track by priority levels as well 23JAN-20APR94 */
if (delay > fddi2_sink_peak_delay_a[pri2_set])
fddi2_sink_peak_delay_a[pri2_set] = delay;

op_pk_destroy (pkptr);

/* increment packet counter; 20JAN94 */
fddi2_sink_total_pkts++;
fddi2_sink_total_pkts_a[pri2_set]++;

/* if a multiple of 25 packets is reached, update stats */
/* 03FEB94: [0]->[7] represent asynch priorities 1->8, */
/* respectively; [8] represents synchronous traffic, */
/* and [9] represents overall asynchronous traffic.-Nix */
if (fddi2_sink_total_pkts % 25 == 0)
{
op_stat_global_write (thru2_gshandle,
fddi2_sink_total_bits / op_sim_time ());

 op_stat_global_write (thru2_gshandle_a[pri2_set],
fddi2_sink_total_bits_a[0] / op_sim_time());
 op_stat_global_write (thru2_gshandle_a[0],
fddi2_sink_total_bits_a[1] / op_sim_time());
 op_stat_global_write (thru2_gshandle_a[1],
fddi2_sink_total_bits_a[pri2_set] / op_sim_time());
 op_stat_global_write (thru2_gshandle_a[2],
fddi2_sink_total_bits_a[2] / op_sim_time());
 op_stat_global_write (thru2_gshandle_a[3],
fddi2_sink_total_bits_a[3] / op_sim_time());
 op_stat_global_write (thru2_gshandle_a[4],
fddi2_sink_total_bits_a[4] / op_sim_time());
 op_stat_global_write (thru2_gshandle_a[5],
fddi2_sink_total_bits_a[5] / op_sim_time());
 op_stat_global_write (thru2_gshandle_a[6],
fddi2_sink_total_bits_a[6] / op_sim_time());
 op_stat_global_write (thru2_gshandle_a[7],
fddi2_sink_total_bits_a[7] / op_sim_time());
 op_stat_global_write (thru2_gshandle_a[8],
fddi2_sink_total_bits_a[8] / op_sim_time());

/* 30JAN94: gather all asynch stats into one overall figure */
op_stat_global_write (thru2_gshandle_a[9],
```

```
    (fddi2_sink_total_bits - fddi2_sink_total_bits_a[8]) /
    op_sim_time());


/* (fddi2_sink_total_bits_a[0] + fddi2_sink_total_bits_a[1] + */
/* fddi2_sink_total_bits_a[2] + fddi2_sink_total_bits_a[3] + */
/* fddi2_sink_total_bits_a[4] + fddi2_sink_total_bits_a[5] + */
/* fddi2_sink_total_bits_a[6] + fddi2_sink_total_bits_a[7]) / */
/* op_sim_time()); */


        op_stat_global_write (m2_delay_gshandle,
       fddi2_sink_accum_delay / fddi2_sink_total_pkts);

        op_stat_global_write (m2_delay_gshandle_a[0],
     fddi2_sink_accum_delay_a[0] / fddi2_sink_total_pkts_a[0]);
       op_stat_global_write (m2_delay_gshandle_a[1],
     fddi2_sink_accum_delay_a[1] / fddi2_sink_total_pkts_a[1]);
       op_stat_global_write (m2_delay_gshandle_a[2],
    fddi2_sink_accum_delay_a[2] / fddi2_sink_total_pkts_a[2]);
       op_stat_global_write (m2_delay_gshandle_a[3],
     fddi2_sink_accum_delay_a[3] / fddi2_sink_total_pkts_a[3]);
        op_stat_global_write (m2_delay_gshandle_a[4],
     fddi2_sink_accum_delay_a[4] / fddi2_sink_total_pkts_a[4]);
       op_stat_global_write (m2_delay_gshandle_a[5],
    fddi2_sink_accum_delay_a[5] / fddi2_sink_total_pkts_a[5]);
        op_stat_global_write (m2_delay_gshandle_a[6],
     fddi2_sink_accum_delay_a[6] / fddi2_sink_total_pkts_a[6]);
       op_stat_global_write (m2_delay_gshandle_a[7],
     fddi2_sink_accum_delay_a[7] / fddi2_sink_total_pkts_a[7]);
       op_stat_global_write (m2_delay_gshandle_a[8],
    fddi2_sink_accum_delay_a[8] / fddi2_sink_total_pkts_a[8]);

       /* 30JAN94: gather all asynch stats into one figure */
        op_stat_global_write (m2_delay_gshandle_a[9],
     (fddi2_sink_accum_delay - fddi2_sink_accum_delay_a[8]) /
     (fddi2_sink_total_pkts - fddi2_sink_total_pkts_a[8]));


/* (fddi2_sink_accum_delay_a[0] + fddi2_sink_accum_delay_a[1] + */
/* fddi2_sink_accum_delay_a[2] + fddi2_sink_accum_delay_a[3] + */
/* fddi2_sink_accum_delay_a[4] + fddi2_sink_accum_delay_a[5] + */
/* fddi2_sink_accum_delay_a[6] + fddi2_sink_accum_delay_a[7]) / */
```

```
/* (fddi2_sink_total_pkts_a[0] + fddi2_sink_total_pkts_a[1] + */
/* fddi2_sink_total_pkts_a[2] + fddi2_sink_total_pkts_a[3] + */
/* fddi2_sink_total_pkts_a[4] + fddi2_sink_total_pkts_a[5] + */
/* fddi2_sink_total_pkts_a[6] + fddi2_sink_total_pkts_a[7])); */


    /* also record actual delay values */
    op_stat_global_write (ete2_delay_gshandle, delay);
    op_stat_global_write (ete2_delay_gshandle_a[pri2_set], delay);
    }
    }/*end of if(dest_addr==my_addr)&&(src_addr > 9)statement */

    /* 20APR94: destroy the packets coming from the first lan destined */
    /* for this station.These packets are not counted for local traffic.*/
    else if(dest_addr == my_addr)
    op_pk_destroy(pkptr);


    /* Other frames passed to "llc" should be destined for other lan */
    /* 18APR94 :allocate the packets to transmitter of command link */
    else
    {

    /* add in its size */
    fddilp2_total_bits += op_pk_total_size_get (pkptr);
    fddilp2_total_bits_a[pri2_set] += op_pk_total_size_get (pkptr); /*
20JAN-20APR94 */

    /* increment packet counter; 20APR94 */
    fddilp2_total_pkts++;
    fddilp2_total_pkts_a[pri2_set]++;

    /* if a multiple of 25 packets is reached, update stats */
    /* [0]->[7] represent asynch priorities 1->8, */
    /* respectively; [8] represents synchronous traffic, */
    /* and [9] represents overall asynchronous traffic.-Nix */
    if (fddilp2_total_pkts % 25 == 0)
    {
    op_stat_global_write (t2_gshandle,
    fddilp2_total_bits / op_sim_time ());

     op_stat_global_write (t2_gshandle_a[pri2_set],
   fddilp2_total_bits_a[0] / op_sim_time());
```

```
        op_stat_global_write (t2_gshandle_a[0],
    fddilp2_total_bits_a[1] / op_sim_time());
        op_stat_global_write (t2_gshandle_a[1],
   fddilp2_total_bits_a[pri2_set] / op_sim_time());
        op_stat_global_write (t2_gshandle_a[2],
         fddilp2_total_bits_a[2] / op_sim_time());
         op_stat_global_write (t2_gshandle_a[3],
    fddilp2_total_bits_a[3] / op_sim_time());
         op_stat_global_write (t2_gshandle_a[4],
         fddilp2_total_bits_a[4] / op_sim_time());
         op_stat_global_write (t2_gshandle_a[5],
    fddilp2_total_bits_a[5] / op_sim_time());
         op_stat_global_write (t2_gshandle_a[6],
         fddilp2_total_bits_a[6] / op_sim_time());
          op_stat_global_write (t2_gshandle_a[7],
         fddilp2_total_bits_a[7] / op_sim_time());
         op_stat_global_write (t2_gshandle_a[8],
        fddilp2_total_bits_a[8] / op_sim_time());

        /* gather all asynch stats into one overall figure */
        op_stat_global_write (t2_gshandle_a[9],
    (fddilp2_total_bits - fddilp2_total_bits_a[8]) /
    op_sim_time());


/* (fddilp2_total_bits_a[0] + fddilp2_total_bits_a[1] + */
/* fddilp2_total_bits_a[2] + fddilp2_total_bits_a[3] + */
/* fddilp2_total_bits_a[4] + fddilp2_total_bits_a[5] + */
/* fddilp2_total_bits_a[6] + fddilp2_total_bits_a[7]) / */
/* op_sim_time()); */


        }

        /* 21APR94:allocate packets to the command link transmitter */
        /* altered for ppp 1AUG94 */
        ppp_pkptr = op_pk_create_fmt("ppp_ml");
op_pk_nfd_set (ppp_pkptr, "pid_h", 0x00);
        op_pk_nfd_set (ppp_pkptr, "pid_l", 0x3d);
        op_pk_nfd_set (ppp_pkptr, "FDDI_frame", pkptr);
        /* put ppp packet on subqueue to be xmitted */
        op_subq_pk_insert(0, ppp_pkptr, OPC_QPOS_TAIL);
```

```
            }/* end of else */
            }/* end of else for (if ppp and from LLC_source) */

            /* check if this subqueue is empty and transmitter is not busy */
            if ((!op_subq_empty(0))&&(busy == 0.0))
            {
            /*access the first packet in the subqueue */
            pkptr1 = op_subq_pk_remove (0, OPC_QPOS_HEAD);
            /* forward it to the transmitter of command link */
            op_pk_send (pkptr1, 0);
            }

            break;
          }/* end of case OPC_INTRPT_STRM statement */

        }/* end of switch */









    }


/** blocking after enter executives of unforced state. **/
FSM_EXIT (1,sp_fddi_sink)


/** state (DISCARD) exit executives **/
FSM_STATE_EXIT_UNFORCED (0, state0_exit_exec, "DISCARD")
    {
    }


/** state (DISCARD) transition processing **/
FSM_INIT_COND (END_OF_SIM)
FSM_DFLT_COND
```

```
FSM_TEST_LOGIC ("DISCARD")

FSM_TRANSIT_SWITCH
     {
     FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;)
     FSM_CASE_TRANSIT (1, 0, state0_enter_exec, ;)
     }
/*--------------------------------------------------------*/



/** state (STATS) enter executives **/
FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "STATS")
     {
     /* At end of simulation, scalar performance statistics */
     /* and input parameters are written out. */
     /* This is for command link throughput :21APR94*/
        op_stat_scalar_write ("CL Throughput (bps), Priority 1",
              fddilp2_total_bits_a[0] / op_sim_time ());

      op_stat_scalar_write ("CL Throughput (bps), Priority 2",
      fddilp2_total_bits_a[1] / op_sim_time ());

      op_stat_scalar_write ("CL Throughput (bps), Priority 3",
      fddilp2_total_bits_a[2] / op_sim_time ());

      op_stat_scalar_write ("CL Throughput (bps), Priority 4",
      fddilp2_total_bits_a[3] / op_sim_time ());

      op_stat_scalar_write ("CL Throughput (bps), Priority 5",
      fddilp2_total_bits_a[4] / op_sim_time ());

      op_stat_scalar_write ("CL Throughput (bps), Priority 6",
      fddilp2_total_bits_a[5] / op_sim_time ());

      op_stat_scalar_write ("CL Throughput (bps), Priority 7",
      fddilp2_total_bits_a[6] / op_sim_time ());

      op_stat_scalar_write ("CL Throughput (bps), Priority 8",
      fddilp2_total_bits_a[7] / op_sim_time ());

      op_stat_scalar_write ("CL Throughput (bps), Asynchronous",
      (fddilp2_total_bits - fddilp2_total_bits_a[8]) / op_sim_time ());
```

```
/* (fddilp2_total_bits_a[0] + fddilp2_total_bits_a[1] + */
/* fddilp2_total_bits_a[2] + fddilp2_total_bits_a[3] + */
/* fddilp2_total_bits_a[4] + fddilp2_total_bits_a[5] + */
/* fddilp2_total_bits_a[6] + fddilp2_total_bits_a[7]) / */
/* op_sim_time ()); */


op_stat_scalar_write ("CL Throughput (bps), Synchronous",
fddilp2_total_bits_a[8] / op_sim_time ());

op_stat_scalar_write ("CL Throughput (bps), Total",
fddilp2_total_bits / op_sim_time ());


/* Only one station needs to do this for the second ring(Ring 1)*/
if (!fddi2_sink_scalar_write)
   {
   /* set the scalar write flag */
   fddi2_sink_scalar_write = 1;

   op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 1",
         fddi2_sink_accum_delay_a[0] / fddi2_sink_total_pkts_a[0]);

op_stat_scalar_write ("Mean End-to-End Delay-1(sec.), Priority 2",
fddi2_sink_accum_delay_a[1] / fddi2_sink_total_pkts_a[1]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 3",
fddi2_sink_accum_delay_a[2] / fddi2_sink_total_pkts_a[2]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 4",
fddi2_sink_accum_delay_a[3] / fddi2_sink_total_pkts_a[3]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 5",
fddi2_sink_accum_delay_a[4] / fddi2_sink_total_pkts_a[4]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 6",
fddi2_sink_accum_delay_a[5] / fddi2_sink_total_pkts_a[5]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 7",
```

```
fddi2_sink_accum_delay_a[6] / fddi2_sink_total_pkts_a[6]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Priority 8",
fddi2_sink_accum_delay_a[7] / fddi2_sink_total_pkts_a[7]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Asynchronous",
(fddi2_sink_accum_delay - fddi2_sink_accum_delay_a[8]) /
(fddi2_sink_total_pkts - fddi2_sink_total_pkts_a[8]));

/* (fddi2_sink_accum_delay_a[0] + fddi2_sink_accum_delay_a[1] + */
/* fddi2_sink_accum_delay_a[2] + fddi2_sink_accum_delay_a[3] + */
/* fddi2_sink_accum_delay_a[4] + fddi2_sink_accum_delay_a[5] + */
/* fddi2_sink_accum_delay_a[6] + fddi2_sink_accum_delay_a[7]) / */
/* (fddi2_sink_total_pkts_a[0] + fddi2_sink_total_pkts_a[1] + */
/* fddi2_sink_total_pkts_a[2] + fddi2_sink_total_pkts_a[3] + */
/* fddi2_sink_total_pkts_a[4] + fddi2_sink_total_pkts_a[5] + */
/* fddi2_sink_total_pkts_a[6] + fddi2_sink_total_pkts_a[7])); */


op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Synchronous",
fddi2_sink_accum_delay_a[8] / fddi2_sink_total_pkts_a[8]);

op_stat_scalar_write ("Mean End-to-End Delay-1 (sec.), Total",
fddi2_sink_accum_delay / fddi2_sink_total_pkts);

   op_stat_scalar_write ("Throughput-1 (bps), Priority 1",
         fddi2_sink_total_bits_a[0] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Priority 2",
fddi2_sink_total_bits_a[1] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Priority 3",
fddi2_sink_total_bits_a[2] / op_sim_time ());

op_stat_scalar_write ("Throughput-1( bps), Priority 4",
fddi2_sink_total_bits_a[3] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Priority 5",
fddi2_sink_total_bits_a[4] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Priority 6",
fddi2_sink_total_bits_a[5] / op_sim_time ());
```

```
op_stat_scalar_write ("Throughput-1 (bps), Priority 7",
fddi2_sink_total_bits_a[6] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Priority 8",
fddi2_sink_total_bits_a[7] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Asynchronous",
(fddi2_sink_total_bits - fddi2_sink_total_bits_a[8]) / op_sim_time ());


/* (fddi2_sink_total_bits_a[0] + fddi2_sink_total_bits_a[1] + */
/* fddi2_sink_total_bits_a[2] + fddi2_sink_total_bits_a[3] + */
/* fddi2_sink_total_bits_a[4] + fddi2_sink_total_bits_a[5] + */
/* fddi2_sink_total_bits_a[6] + fddi2_sink_total_bits_a[7]) / */
/* op_sim_time ()); */



op_stat_scalar_write ("Throughput-1 (bps), Synchronous",
fddi2_sink_total_bits_a[8] / op_sim_time ());

op_stat_scalar_write ("Throughput-1 (bps), Total",
fddi2_sink_total_bits / op_sim_time ());


   op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 1",
         fddi2_sink_peak_delay_a[0]);

op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 2",
fddi2_sink_peak_delay_a[1]);

op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 3",
fddi2_sink_peak_delay_a[2]);

op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 4",
fddi2_sink_peak_delay_a[3]);

op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 5",
fddi2_sink_peak_delay_a[4]);

op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 6",
fddi2_sink_peak_delay_a[5]);
```

```
            op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 7",
            fddi2_sink_peak_delay_a[6]);

            op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Priority 8",
            fddi2_sink_peak_delay_a[7]);

            op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Synchronous",
            fddi2_sink_peak_delay_a[8]);

            op_stat_scalar_write ("Peak End-to-End Delay-1 (sec.), Overall",
            fddi2_sink_peak_delay);


              /* Write the TTRT value for ring 0. This preserves*/
              /* the old behavior for single-ring simulations.*/
              op_stat_scalar_write ("TTRT (sec.) - Ring 1",
                    fddi_t_opr [1]);

            /* 12JAN94: obtain offered load information from the Environment */
            /* file; this will be used to provide abscissa information that */
            /* can be plotted in the Analysis Editor (see "fddi_sink" STATS */
            /* state. To the user: it's your job to keep these current in */
            /* the Environment File. -Nix */
            op_ima_sim_attr_get (OPC_IMA_DOUBLE, "total_offered_load_1",
&Offered_Load);
            op_ima_sim_attr_get (OPC_IMA_DOUBLE, "asynch_offered_load_1",
&Asynch_Offered_Load);


            /* 12JAN94: write the total offered load for this run */
            op_stat_scalar_write ("Total Offered Load-1 (Mbps)",
            Offered_Load);

            op_stat_scalar_write ("Asynchronous Offered Load-1 (Mbps)",
            Asynch_Offered_Load);
              }
            }


        /** blocking after enter executives of unforced state. **/
        FSM_EXIT (3,sp_fddi_sink)
```

```
/** state (STATS) exit executives **/
FSM_STATE_EXIT_UNFORCED (1, state1_exit_exec, "STATS")
	{
	}


/** state (STATS) transition processing **/
FSM_TRANSIT_MISSING ("STATS")
/*---------------------------------------------------------*/



/** state (INIT) enter executives **/
FSM_STATE_ENTER_FORCED (2, state2_enter_exec, "INIT")
	{
	/* get the gshandles of the global statistic to be obtained */
	/* 20JAN94: set array format */

	thru2_gshandle_a[0] = op_stat_global_reg ("pri 1 throughput-1 (bps)");
	thru2_gshandle_a[1] = op_stat_global_reg ("pri 2 throughput-1 (bps)");
	thru2_gshandle_a[2] = op_stat_global_reg ("pri 3 throughput-1 (bps)");
	thru2_gshandle_a[3] = op_stat_global_reg ("pri 4 throughput-1 (bps)");
	thru2_gshandle_a[4] = op_stat_global_reg ("pri 5 throughput-1 (bps)");
	thru2_gshandle_a[5] = op_stat_global_reg ("pri 6 throughput-1 (bps)");
	thru2_gshandle_a[6] = op_stat_global_reg ("pri 7 throughput-1 (bps)");
	thru2_gshandle_a[7] = op_stat_global_reg ("pri 8 throughput-1 (bps)");
	thru2_gshandle_a[8] = op_stat_global_reg ("synch throughput-1 (bps)");
	thru2_gshandle_a[9] = op_stat_global_reg ("async throughput-1 (bps)");
	thru2_gshandle = op_stat_global_reg ("total throughput-1 (bps)");

	m2_delay_gshandle_a[0] = op_stat_global_reg ("pri 1 mean delay-1 (sec.)");
	m2_delay_gshandle_a[1] = op_stat_global_reg ("pri 2 mean delay-1 (sec.)");
	m2_delay_gshandle_a[2] = op_stat_global_reg ("pri 3 mean delay-1 (sec.)");
	m2_delay_gshandle_a[3] = op_stat_global_reg ("pri 4 mean delay-1 (sec.)");
	m2_delay_gshandle_a[4] = op_stat_global_reg ("pri 5 mean delay-1 (sec.)");
	m2_delay_gshandle_a[5] = op_stat_global_reg ("pri 6 mean delay-1 (sec.)");
	m2_delay_gshandle_a[6] = op_stat_global_reg ("pri 7 mean delay-1 (sec.)");
	m2_delay_gshandle_a[7] = op_stat_global_reg ("pri 8 mean delay-1 (sec.)");
	m2_delay_gshandle_a[8] = op_stat_global_reg ("synch mean delay-1 (sec.)");
	m2_delay_gshandle_a[9] = op_stat_global_reg ("async mean delay-1 (sec.)");
	m2_delay_gshandle = op_stat_global_reg ("total mean delay-1 (sec.)");

	ete2_delay_gshandle_a[0] = op_stat_global_reg ("pri 1 end-to-end delay-1 (sec.)");
```

```
        ete2_delay_gshandle_a[1] = op_stat_global_reg ("pri 2 end-to-end delay-1 (sec.)");
        ete2_delay_gshandle_a[2] = op_stat_global_reg ("pri 3 end-to-end delay-1 (sec.)");
        ete2_delay_gshandle_a[3] = op_stat_global_reg ("pri 4 end-to-end delay-1 (sec.)");
        ete2_delay_gshandle_a[4] = op_stat_global_reg ("pri 5 end-to-end delay-1 (sec.)");
        ete2_delay_gshandle_a[5] = op_stat_global_reg ("pri 6 end-to-end delay-1 (sec.)");
        ete2_delay_gshandle_a[6] = op_stat_global_reg ("pri 7 end-to-end delay-1 (sec.)");
        ete2_delay_gshandle_a[7] = op_stat_global_reg ("pri 8 end-to-end delay-1 (sec.)");
        ete2_delay_gshandle_a[8] = op_stat_global_reg ("synch end-to-end delay-1
(sec.)");
        ete2_delay_gshandle = op_stat_global_reg ("total end-to-end delay-1 (sec.)");

        t2_gshandle_a[0] = op_stat_global_reg ("pri 1 CL throughput (bps)");
        t2_gshandle_a[1] = op_stat_global_reg ("pri 2 CL throughput (bps)");
        t2_gshandle_a[2] = op_stat_global_reg ("pri 3 CL throughput (bps)");
        t2_gshandle_a[3] = op_stat_global_reg ("pri 4 CL throughput (bps)");
        t2_gshandle_a[4] = op_stat_global_reg ("pri 5 CL throughput (bps)");
        t2_gshandle_a[5] = op_stat_global_reg ("pri 6 CL throughput (bps)");
        t2_gshandle_a[6] = op_stat_global_reg ("pri 7 CL throughput (bps)");
        t2_gshandle_a[7] = op_stat_global_reg ("pri 8 CL throughput (bps)");
        t2_gshandle_a[8] = op_stat_global_reg ("synch CL throughput (bps)");
        t2_gshandle_a[9] = op_stat_global_reg ("async CL throughput (bps)");
        t2_gshandle = op_stat_global_reg ("total CL throughput (bps)");



        }


/** state (INIT) exit executives **/
FSM_STATE_EXIT_FORCED (2, state2_exit_exec, "INIT")
        {
        }


/** state (INIT) transition processing **/
FSM_INIT_COND (END_OF_SIM)
FSM_DFLT_COND
FSM_TEST_LOGIC ("INIT")

FSM_TRANSIT_SWITCH
        {
        FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;)
        FSM_CASE_TRANSIT (1, 0, state0_enter_exec, ;)
```

```
            }
        /*--------------------------------------------------------*/



        }


    FSM_EXIT (2,sp_fddi_sink)
    }



void
sp_fddi_sink_svar (prs_ptr,var_name,var_p_ptr)
    sp_fddi_sink_state*prs_ptr;
    char        *var_name, **var_p_ptr;
    {

    FIN (sp_fddi_sink_svar (prs_ptr))

    *var_p_ptr = VOS_NIL;
    if (Vos_String_Equal ("thru2_gshandle" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_thru2_gshandle);
    if (Vos_String_Equal ("m2_delay_gshandle" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_m2_delay_gshandle);
    if (Vos_String_Equal ("ete2_delay_gshandle" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_ete2_delay_gshandle);
    if (Vos_String_Equal ("thru2_gshandle_a" , var_name))
        *var_p_ptr = (char *) (prs_ptr->sv_thru2_gshandle_a);
    if (Vos_String_Equal ("m2_delay_gshandle_a" , var_name))
        *var_p_ptr = (char *) (prs_ptr->sv_m2_delay_gshandle_a);
    if (Vos_String_Equal ("ete2_delay_gshandle_a" , var_name))
        *var_p_ptr = (char *) (prs_ptr->sv_ete2_delay_gshandle_a);
    if (Vos_String_Equal ("t2_gshandle" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_t2_gshandle);
    if (Vos_String_Equal ("t2_gshandle_a" , var_name))
        *var_p_ptr = (char *) (prs_ptr->sv_t2_gshandle_a);
    if (Vos_String_Equal ("my_id" , var_name))
        *var_p_ptr = (char *) (&prs_ptr->sv_my_id);

    FOUT;
```

```
    }



void
sp_fddi_sink_diag ()
    {
    double      delay, creat_time;
    Packet*     pkptr;
    Packet*     ppp_pkptr;
    Packet* pkptr1 ; /*5APR94*/
    int             src_addr, my_addr;
    int             dest_addr;/*14APR94*/
    Ici*        from_mac_ici_ptr;
    double      fddi_sink_ttrt;
    char        pk_format[10];
    int             input_stream;
    int             fd_index, FDDI_frame_size;

    FIN (sp_fddi_sink_diag ())


    FOUT;
    }



void
sp_fddi_sink_terminate ()
    {
    double      delay, creat_time;
    Packet*     pkptr;
    Packet*     ppp_pkptr;
    Packet* pkptr1 ; /*5APR94*/
    int             src_addr, my_addr;
    int             dest_addr;/*14APR94*/
    Ici*        from_mac_ici_ptr;
    double      fddi_sink_ttrt;
    char        pk_format[10];
    int             input_stream;
    int             fd_index, FDDI_frame_size;
```

```
    FIN (sp_fddi_sink_terminate ())


    FOUT;
    }



Compcode
sp_fddi_sink_init (pr_state_pptr)
    sp_fddi_sink_state**pr_state_pptr;
    {
    static VosT_Cm_Obtypeobtype = OPC_NIL;

    FIN (sp_fddi_sink_init (pr_state_pptr))

    if (obtype == OPC_NIL)
        {
        if (Vos_Catmem_Register ("proc state vars (sp_fddi_sink)",
            sizeof (sp_fddi_sink_state), Vos_Nop, &obtype) == VOSC_FAILURE)
            FRET (OPC_COMPCODE_FAILURE)
        }

    if ((*pr_state_pptr = (sp_fddi_sink_state*) Vos_Catmem_Alloc (obtype, 1)) == OPC_NIL)
        FRET (OPC_COMPCODE_FAILURE)
    else
        {
        (*pr_state_pptr)->current_block = 4;
        FRET (OPC_COMPCODE_SUCCESS)
        }
    }
```

APPENDIX M

TCP RING 1 MAC MODULE CODE "sp_fddi_mac_tcp.pr.c"

Unchanged portions of this code have been deleted for brevity.

```
/* Process model C form file: sp_fddi_mac_tcp.pr.c */
/* Portions of this file Copyright (C) MIL 3, Inc. 1992 */



/* OPNET system definitions */
#include <opnet.h>
#include "sp_fddi_mac_tcp.pr.h"
FSM_EXT_DECS



/* Header block */
/* Define a timer structure used to implement */
/* the TRT and THT timers. The primitives defined to */
/* operate on these timers can be found in the */
/* function block of this process model. */
typedef struct
    {
    int             enabled;
    double          start_time;
    double          accum;
    double          target_accum;
    } FddiT_Timer;


/* Declare certain primitives dealing with timers */
double          fddi_timer_remaining ();
FddiT_Timer* fddi_timer_create ();
double          fddi_timer_value ();

/* Scratch strings for trace statements */
char            str0 [512], str1 [512];
```

```
/* define constants particular to this implementation */
#define FDDI_MAX_STATIONS       512

/* define possible values for the frame control field */
#define FDDI_FC_FRAME           0
#define FDDI_FC_TOKEN           1

/* define possible service classes for frames */
#define FDDI_SVC_ASYNC          0
#define FDDI_SVC_SYNC           1

/* define input stream indices */
#define FDDI_LLC_STRM_IN        1
#define FDDI_PHY_STRM_IN        0

/* define output stream indices */
#define FDDI_LLC_STRM_OUT       1
#define FDDI_PHY_STRM_OUT       0

/* define token classes */
#define FDDI_TK_NONRESTRICTED 0
#define FDDI_TK_RESTRICTED      1

/* Ring Constants */
#define FDDI_TX_RATE            1.0e+08
#define FDDI_SA_SCAN_TIME       28.0e-08

/* Token transmission time: based on 6 symbols plus 16 symbols of preamble */
#define FDDIC_TOKEN_TX_TIME     88.0e-08


/* Codes used to differentiate remote interrupts */
#define FDDIC_TRT_EXPIRE        0
#define FDDIC_TK_INJECT         1


/* Define symbolic expressions used on transition */
/* conditions and in executive statements. */
#define TRT_EXPIRE  \
    (op_intrpt_type () == OPC_INTRPT_REMOTE && op_intrpt_code () ==
FDDIC_TRT_EXPIRE)
```

```
#define TK_RECEIVED\
    phy_arrival && \
    frame_control == FDDI_FC_TOKEN

#define RC_FRAME    \
    phy_arrival && \
    frame_control == FDDI_FC_FRAME

#define FRAME_ARRIVAL\
    op_intrpt_type () == OPC_INTRPT_STRM &&\
    op_intrpt_strm () == FDDI_LLC_STRM_IN

#define STRIPmy_address == src_addr

/* Define the maximum value for ring_id. This is the*/
/* maximum number of FDDI rings that can exist in a*/
/* simulation. Note that if this number is changed,*/
/* the initialization for fddi_claim_start below must*/
/* also be modified accordingly.*/
#define FDDI_MAX_RING_ID   8

/* Declare the operative TTRT value 'T_Opr' which is the final*/
/* negotiated value of TTRT. This value is shared by all stations*/
/* on a ring so that all agree on its value.*/
double    fddi_t_opr [FDDI_MAX_RING_ID];
#define   Fddi_T_Opr   (fddi_t_opr [ring_id])

/* This flag indicates that the negotiation for the final TTRT*/
/* has not yet begun. It is statically initialized here, and*/
/* is reset by the first station which modifies T_Opr.*/
/* Initialize to 1 for all rings.*/
static
int             fddi_claim_start [FDDI_MAX_RING_ID] = {1,1,1,1,1,1,1,1};
#define         Fddi_Claim_Start(fddi_claim_start [ring_id])


/* Declare station latency parameters. */
/* These are true globals, so they do not need to be arrays. */
double    Fddi_St_Latency;
double    Fddi_Prop_Delay;

/* Declare globals for Token Acceleration Mechanism.*/
/* Hop delay and token acceleration are true globals.*/
```

```
double    Fddi_Tk_Hop_Delay;
static
int             Fddi_Tk_Accelerate = 1;

/* These are actually values shared by all nodes on a ring,*/
/* so they must be defined as arrays.*/
double    fddi_tk_block_base_time [FDDI_MAX_RING_ID];
#define   Fddi_Tk_Block_Base_Time(fddi_tk_block_base_time [ring_id])

int             fddi_tk_block_base_station [FDDI_MAX_RING_ID];
#define   Fddi_Tk_Block_Base_Station(fddi_tk_block_base_station [ring_id])

int             fddi_tk_blocked [FDDI_MAX_RING_ID];
#define   Fddi_Tk_Blocked(fddi_tk_blocked [ring_id])

int             fddi_num_stations [FDDI_MAX_RING_ID];
#define   Fddi_Num_Stations(fddi_num_stations [ring_id])

int             fddi_num_registered [FDDI_MAX_RING_ID];
#define   Fddi_Num_Registered(fddi_num_registered [ring_id])

Objid     fddi_address_table [FDDI_MAX_RING_ID][FDDI_MAX_STATIONS];
#define   Fddi_Address_Table(fddi_address_table [ring_id])

/* Below is part of the OPBUG 2081 patch; FB ended here, before. -Nix */

/* Event handles for the TRT are maintained at a global level to */
/* allow token acceleration mechanism to adjust these as necessary */
/* when blocking and reinjecting the token. TRT_handle simply */
/* represents the TRT for the local MAC*/
Evhandlefddi_trt_handle [FDDI_MAX_RING_ID][FDDI_MAX_STATIONS];
#define   Fddi_Trt_Handle(fddi_trt_handle [ring_id])
#define TRT_handle      Fddi_Trt_Handle [my_address]

/* Similarly, the TRT data structure is maintained on a global level. */
FddiT_Timer*fddi_trt [FDDI_MAX_RING_ID] [FDDI_MAX_STATIONS];
#define   Fddi_Trt        (fddi_trt [ring_id])
#define   TRT             Fddi_Trt [my_address]

/* Registers to record the expiration time of each TRT when token is blocked. */
double    fddi_trt_exp_time [FDDI_MAX_RING_ID] [FDDI_MAX_STATIONS];
#define Fddi_Trt_Exp_Time(fddi_trt_exp_time [ring_id])
```

```
/* the 'Late_Ct' flag is declared on a global level so that it can be */
/* set at the tim ewhere the token is injected back into the ring. */
int             fddi_late_ct [FDDI_MAX_RING_ID] [FDDI_MAX_STATIONS];
#define   Fddi_Late_Ct (fddi_late_ct [ring_id])
#define   Late_Ct       Fddi_Late_Ct [my_address]


/* Convenient macro for setting TRT for a given station and absolute time. */
#define TRT_SET(station_id,abs_time)\
fddi_timer_set (Fddi_Trt [station_id], abs_time - op_sim_time());\
Fddi_Trt_Handle [station_id] = op_intrpt_schedule_remote (abs_time,\
        FDDIC_TRT_EXPIRE, Fddi_Address_Table [station_id]);




/* State variable definitions */
typedef struct
    {
    FSM_SYS_STATE
    int         sv_ring_id;
    FddiT_Timer* sv_THT;
    double      sv_T_Req;
    double      sv_T_Pri [8];
    Objid       sv_my_objid;
    int         sv_spawn_token;
    int         sv_my_address;
    int         sv_orig_src_addr;
    Packet*     sv_tk_pkptr;
    double      sv_sync_bandwidth;
    double      sv_sync_pc;
    int         sv_restricted;
    int         sv_res_peer;
    int         sv_tk_registered;
    Ici*        sv_to_llc_ici_ptr;
    int         sv_tk_trace_on;
    } sp_fddi_mac_tcp_state;

#define pr_state_ptr            ((sp_fddi_mac_tcp_state*) SimI_Mod_State_Ptr)
#define ring_id                 pr_state_ptr->sv_ring_id
```

```
#define THT                     pr_state_ptr->sv_THT
#define T_Req                   pr_state_ptr->sv_T_Req
#define T_Pri                   pr_state_ptr->sv_T_Pri
#define my_objid                pr_state_ptr->sv_my_objid
#define spawn_token             pr_state_ptr->sv_spawn_token
#define my_address              pr_state_ptr->sv_my_address
#define orig_src_addr           pr_state_ptr->sv_orig_src_addr
#define tk_pkptr                pr_state_ptr->sv_tk_pkptr
#define sync_bandwidth          pr_state_ptr->sv_sync_bandwidth
#define sync_pc                 pr_state_ptr->sv_sync_pc
#define restricted              pr_state_ptr->sv_restricted
#define res_peer                pr_state_ptr->sv_res_peer
#define tk_registered           pr_state_ptr->sv_tk_registered
#define to_llc_ici_ptr          pr_state_ptr->sv_to_llc_ici_ptr
#define tk_trace_on             pr_state_ptr->sv_tk_trace_on



/* Process model interrupt handling procedure */


void
sp_fddi_mac_tcp ()
    {
    /* Packets and ICI's */
    Packet*     mac_frame_ptr;
    Packet*     pdu_ptr;
    Packet*     pkptr;
    Packet*     data_pkptr;
    Ici*        ici_ptr;

    /* Packet Fields and Attributes */
    int             req_pri, svc_class, req_tk_class;
    int             frame_control, src_addr, dest_addr;
    int             pk_len, pri_level;

    /* Token - Related */
    int             tk_usable, res_station, tk_class;
    int             current_tk_class;
    double      accum_sync;
```

```
/* Timer - Related */
double      tx_time, timer_remaining, accum_bandwidth;
double      tht_value;

/* Miscellaneous */
int             i;
int             spawn_station, phy_arrival;
char        error_string [512];
int             num_frames_sent, num_bits_sent;

/* 26DEC93: loop management variables, used in RCV_TK */
/* and ENCAP states. -Nix */
int NUM_PRIOS;
int punt;
int q_check;

/*********************************************************/
/* 11SEP94: added creation_time */
double creation_time;



FSM_ENTER (sp_fddi_mac_tcp)

FSM_BLOCK_SWITCH
    {
    /*--------------------------------------------------------*/
    /** state (INIT) enter executives **/
    FSM_STATE_ENTER_FORCED (0, state0_enter_exec, "INIT")
        {
        /* Obtain the station's address . This is an attribute */
        /* of this process. Addressing is simplified by */
        /* simply using integers, and only one mode. */
        /* This mode is 16 bit addressing unless the */
        /* packet format 'fddi_mac_fr' is modified. */
        my_objid = op_id_self(); /* 29DEC93 */
        op_ima_obj_attr_get (my_objid, "station_address", &my_address);

        /* Register the station's object id in a global table. */
        /* This table is used by the mechanism which improves */
        /* simulation efficiency by 'jumping over' idle periods */
        /* rather than circulating an unusable token. */
        fddi_station_register (my_address, my_objid);
```

```
/* Obtain the station latency for tokens and frames. */
/* Default value is set at 100 nanoseconds. */
Fddi_St_Latency = 100.0e-09;
op_ima_sim_attr_get (OPC_IMA_DOUBLE, "station_latency",
&Fddi_St_Latency);

/* Obtain the propagation delay separating stations. */
/* This value is given in seconds with default value 3.3 microseconds. */
Fddi_Prop_Delay = 3.3e-06;
op_ima_sim_attr_get (OPC_IMA_DOUBLE, "prop_delay", &Fddi_Prop_Delay);

/* Derive the Delay for a 'hop' of a freely circulating packet. */
Fddi_Tk_Hop_Delay = Fddi_Prop_Delay + Fddi_St_Latency;


/* The T_Pri [] state variable array supports priority */
/* assignments on a station by station basis by */
/* establishing a correspondence between integer priority */
/* levels assigned to frames and the maximum values of the */
/* token holding timer (THT) which would allow packets to be */
/* sent. Eight levels are supported here, but this can easily */
/* be changed by redimensioning the priority array. */
/* By default all levels are identical here, allowing */
/* any frame to make use of the token, so that in fact */
/* priority levels are not used in the default case. */

/* 01JAN94: (8-i) is a quick attempt to impart different weighting */
/* scales on each priority level, and is not necessarily realistic.-Nix */
/* Be aware of integer-double arithmetic conflicts ie, 1/8 = 0. -Nix */

op_ima_obj_attr_get(my_objid, "T_Req", &T_Req);
for (i = 0; i < 8; i++)
    {
 T_Pri[i] = ((double)(i + 1.0)/8.0) * Fddi_T_Opr;
 /* printf("MAC INIT: T_Pri[%d] is %lf; */
 /* Fddi_T_Opr is %lf\n", i, T_Pri[i], Fddi_T_Opr); */
    }


/* Create the token holding timer (THT) used to restrict the */
/* asynchronous bandwidth consumption of the station */
```

```
THT = fddi_timer_create ();

/* Create the token rotation timer (TRT) used to measure the */
/* rotations of the token, detect late tokens and initialize */
/* the THT timer before asynchronous tranmsmissions. */
TRT = fddi_timer_create ();

/* Set the TRT timer to expire in one TTRT */
TRT_SET (my_address, op_sim_time () + Fddi_T_Opr);

/* Initialize the Late_Ct variable which keeps track. */
/* of the number of TRT expirations. */
Late_Ct = 0;

/* initially the ring operates in nonrestricted mode */
restricted = 0;


/* Create an Interface Control Information structure */
/* to use when delivering received frames to the LLC. */
/*****************************************************************/
/*11SEP94: changed to new format, adding pri and cr_time**/
/*****************************************************************/
to_llc_ici_ptr = op_ici_create ("fddi_mac_ind_tcp");

/* The 'tk_registered' variable indicates if the station */
/* has registered its intent to use the token. */
tk_registered = 0;


/* Determine if the model is to make use of the token */
/* 'acceleration' mechanism. If not, every passing of the */
/* token will be explicityly modeled, leading to large */
/* number of events being scheduled when the ring is idle */
/* (i.e, no stations have data to send). */
op_ima_sim_attr_get (OPC_IMA_INTEGER, "accelerate_token",
    &Fddi_Tk_Accelerate);


/* Obtain the synchronous bandwidth assigned */
/* to this station. It is expressed as a */
/* percentage of TTRT, and then converted to seconds */
op_ima_obj_attr_get (my_objid, "sync bandwidth", &sync_pc);
```

```
sync_bandwidth = sync_pc * Fddi_T_Opr;


/* Only one station in the ring is selected to */
/* introduce the first token. Test if this station is it. */
/* If so, set the 'spawn_token' flag. */
/* op_ima_sim_attr_get (OPC_IMA_INTEGER, "spawn station",
&spawn_station); */
/* spawn_token = (spawn_station == my_address); */
/* If the station is to spawn the token, create */
/* the packet which represents the token. */
/* 14APR94 :the bridges will spawn token in both rings */
/* -Karayakaylar */
spawn_token =1;
if (spawn_token)
   {
   tk_pkptr = op_pk_create_fmt ("fddi_mac_tk");

   /* assign its frame control field */
   op_pk_nfd_set (tk_pkptr, "fc", FDDI_FC_TOKEN);

   /* the first token issued is non-restricted */
   op_pk_nfd_set (tk_pkptr, "class", FDDI_TK_NONRESTRICTED);

   /* The transition will be made into the ISSU_TK */
   /* state where the tk_usable variable is used. */
   /* In case any data has been generated, preset */
   /* this variable to one. */
   tk_usable = 1;
   }


/* When sending packets the variable accum_bandwidth is */
/* used as a scheduling base. Init this value to zero. */
/* This statement is required in case this is the spawning */
/* station, and the next state entered is ISSUE_TK */
accum_bandwidth = 0.0;
```

```
    }


/** state (INIT) exit executives **/
FSM_STATE_EXIT_FORCED (0, state0_exit_exec, "INIT")
    {
    }


/** state (INIT) transition processing **/
FSM_INIT_COND (spawn_token)
FSM_DFLT_COND
FSM_TEST_LOGIC ("INIT")

FSM_TRANSIT_SWITCH
    {
    FSM_CASE_TRANSIT (0, 2, state2_enter_exec, ;)
    FSM_CASE_TRANSIT (1, 1, state1_enter_exec, ;)
    }
/*---------------------------------------------------------*/



/** state (FR_REPEAT) enter executives **/
FSM_STATE_ENTER_FORCED (6, state6_enter_exec, "FR_REPEAT")
    {
    /* Extract the destination address of the frame. */
    op_pk_nfd_get (pkptr, "dest_addr", &dest_addr);

    /* If the frame is for this station, make a copy */
    /* of the frame's data field and forward it to */
    /* the higher layer. */
    /* 14APR94 : In order to send the frames which are */
    /* addressed to the remote lan, check the address database */
    /* of remote lan. Frames addressed to the remote lan shouldn't */
    /* be repeated in the local ring -- This is a simple forwarding */
    /* decision algorithm, one of the bridge's function */
    /* - Karayakaylar */
    if((dest_addr == my_address)||(dest_addr <= 9))
       {
```

```
    /* record total size of the frame (including data) */
    pk_len = op_pk_total_size_get (pkptr);

    /* decapsulate the data contents of the frame */
/* 29JAN94: a new field, "pri", has been added to */
/* the fddi_llc_fr packet format in the Parameters */
/* Editor, so that output statistics can be */
/* generated by class and priority. -Nix */
    op_pk_nfd_get (pkptr, "info", &data_pkptr);
    op_pk_nfd_get (pkptr, "pri", &pri_level);

    /* The source and destination address are placed in the */
    /* LLC's ICI before delivering the frame's contents. */
    op_ici_attr_set (to_llc_ici_ptr, "src_addr", src_addr);
    op_ici_attr_set (to_llc_ici_ptr, "dest_addr", dest_addr);
    /*************************************************************/
    /*11SEP94: added pri and cr_time to fddi_mac_ind ici***/
    /* both are needed for data collection***/
/*************************************************************/
    op_ici_attr_set (to_llc_ici_ptr, "pri", pri_level);
    op_pk_nfd_get (pkptr, "cr_time", &creation_time);
    op_ici_attr_set (to_llc_ici_ptr, "cr_time", creation_time);

    op_ici_install (to_llc_ici_ptr);

    /* Because, as noted in the FR_RCV state, only the */
    /* frame's leading edge has arrived at this time, the */
    /* complete frame can only be delivered to the higher */
    /* layer after the frame's transmission delay has elapsed. */
    /* (since decapsulation of the frame data contents has occured, */
    /* the original MAC frame length is used to calculate delay) */
    tx_time = (double) pk_len / FDDI_TX_RATE;
    op_pk_send_delayed (data_pkptr, FDDI_LLC_STRM_OUT, tx_time);

    /* Note that the standard specifies that the original */
    /* frame should be passed along until the originating station */
    /* receives it, at which point it is stripped from the ring. */
    /* However, in the simulation model, there is no interest */
    /* in letting the frame continue past its destination unless */
    /* group addresses are used, so that the same frame could be */
    /* destined for several stations. Here the frame is stripped */
    /* for efficiency as it reaches the destination; if the model */
    /* is modified to include group addresses. this should be changed */
```

```
        /* so that the frame is copied and the original repeated. */
        /* Logic is already present for stripping the frame at the origin. */
        op_pk_destroy (pkptr);
        }
    /* 14APR94 : the frames belong to this ring should be repeated. */
    /* Thus, local traffic is constrained.-- This is filtering decision */
    /* One of the bridge's function - Karayakaylar */
    else{
        /* Repeat the original frame on the ring and account for */
        /* the latency through the station and the propagation delay */
        /* for a single hop. */
        /* (Only the originating station can strip the frame). */
        op_pk_send_delayed (pkptr, FDDI_PHY_STRM_OUT,
            Fddi_St_Latency + Fddi_Prop_Delay);
        }
    }


/** state (FR_REPEAT) exit executives **/
FSM_STATE_EXIT_FORCED (6, state6_exit_exec, "FR_REPEAT")
    {
    }


/** state (FR_REPEAT) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
/*---------------------------------------------------------*/



/** state (ENCAP) enter executives **/
FSM_STATE_ENTER_FORCED (8, state8_enter_exec, "ENCAP")
    {
    /* A frame has arrived from a higher layer; place it in 'pdu_ptr'. */
    pdu_ptr = op_pk_get (op_intrpt_strm ());

    /* Also get the interface control information */
    /* associated with the new frame. */
    ici_ptr = op_intrpt_ici ();
    if (ici_ptr == OPC_NIL)
        {
        sprintf (error_string, "Simulation aborted; error in object (%d)",
            op_id_self ());
```

```
    op_sim_end (error_string, "fddi_mac: required ICI not received", " ", " ");
    }

/* Extract the requested service class */
/* (e.g, synchronous or asynchronous). */
if (op_ici_attr_exists (ici_ptr, "svc_class"))
    op_ici_attr_get (ici_ptr, "svc_class", &svc_class);
else svc_class = FDDI_SVC_ASYNC;

/* Extract the destination address. */
op_ici_attr_get (ici_ptr, "dest_addr", &dest_addr);

/* Extract the original source address from ICI :16APR94 */
op_ici_attr_get (ici_ptr, "src_addr", &orig_src_addr);


/* If the frame is asynchronous, the priority and */
/* requested token class parameter may be specified. */
if (svc_class == FDDI_SVC_ASYNC)
    {
    /* Extract the requested priority level. */
    if (op_ici_attr_exists (ici_ptr, "pri"))
        op_ici_attr_get (ici_ptr, "pri", &req_pri);
    else req_pri = 0;

    /* Extract the token class (restrictred or non-restricted). */
    if (op_ici_attr_exists (ici_ptr, "tk_class"))
        op_ici_attr_get (ici_ptr, "tk_class", &req_tk_class);
    else req_tk_class = FDDI_TK_NONRESTRICTED;
    }

/* Check for the default ICI values; if they are not present */
/* compose the frame :21APR94*/
if(dest_addr != orig_src_addr){

/* Compose a mac frame from all these elements. */
/*****************************************************************/
/* 11SEP94: mac frame format is changed to fddi_mac_fr_tcp***/
/*****************************************************************/
mac_frame_ptr = op_pk_create_fmt ("fddi_mac_fr_tcp");
op_pk_nfd_set (mac_frame_ptr, "svc_class", svc_class);
op_pk_nfd_set (mac_frame_ptr, "dest_addr", dest_addr);
/*op_pk_nfd_set (mac_frame_ptr, "src_addr", my_address);*/
```

```
/* here original source address should be kept in mac frame :16APR94*/
op_pk_nfd_set (mac_frame_ptr, "src_addr", orig_src_addr);
op_pk_nfd_set (mac_frame_ptr, "info", pdu_ptr);

/*****************************************************************/
/* 11SEP94: cr_time is added to the MAC frame (0bits) because it*/
/* is needed for calculations, but taken out of the LLC frame  */
/*****************************************************************/
op_ici_attr_get (ici_ptr, "cr_time", &creation_time);
op_pk_nfd_set (mac_frame_ptr, "cr_time", creation_time);

printf("\n******dest_addr = %5d\n",dest_addr);
printf("********orig_src_addr= %5d\n",orig_src_addr);

if (svc_class == FDDI_SVC_ASYNC)
    {
    op_pk_nfd_set (mac_frame_ptr, "tk_class", req_tk_class);
    op_pk_nfd_set (mac_frame_ptr, "pri", req_pri);
    }

/* 04JAN94: if the frame is synchronous, assign it a separate */
/* priority so that it may be assigned its own subqueue, and */
/* thereby be assigned its own probe for monitoring. -Nix */
if (svc_class == FDDI_SVC_SYNC)
 {
 op_pk_nfd_set (mac_frame_ptr, "pri", 8);
 }

/* Assign the frame control field, which in the model */
/* is used to distinguish between tokens and ordinary */
/* frames on the ring. */
op_pk_nfd_set (mac_frame_ptr, "fc", FDDI_FC_FRAME);


/* Enqueue the frame at the tail of the queue. */
/* 27DEC93: at the tail of the prioritized queue. */
/* 04JAN94. must distinguish between synch & asynch. */
if (svc_class == FDDI_SVC_ASYNC)
 {
 op_subq_pk_insert (req_pri, mac_frame_ptr, OPC_QPOS_TAIL);
 }
if (svc_class == FDDI_SVC_SYNC)
 {
```

```
        op_subq_pk_insert (8, mac_frame_ptr, OPC_QPOS_TAIL);
        }


      /* if this station has not yet registered its intent to */
      /* use the token, it may do so now since it has data to send */
      if (!tk_registered)
          {
          fddi_tk_register ();
          tk_registered = 1;
          }

      } /* end of if(dest_addr != orig_src_addr) statement */




      }


/** state (ENCAP) exit executives **/
FSM_STATE_EXIT_FORCED (8, state8_exit_exec, "ENCAP")
      {
      }


/** state (ENCAP) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;)
/*---------------------------------------------------------*/
```

LIST OF REFERENCES

1. Defense Support Project Office, *CDL System Description Document for Common Data Link (CDL)*, 1993.

2. Naval Postgraduate School Technical Report Number NPS-EC-94-008, *Interfacing Remote Platforms Using the Common Data Link - Requirements and Architectural Alternatives*, by Shridhar B. Shukla and Samir G. Kelekar, 19 August 1994.

3. Baker, F., and Bowen, R.,eds., "PPP Bridging Control Protocol (BCP)," RFC 1638, Advanced Computer Communications and IBM Corporation, June 1994.

4. Saltzer, J. H., Reed, D. P., and Clark, D. D., "End-to-End Arguments in System Design," *ACM Transactions on Computer Systems*, v 2, no. 4, pp 277-288, November 1994.

5. Simpson, W., ed., "The Point-to-Point Protocol (PPP)," RFC 1548, Daydreamer, December 1993.

6. Simpson, W., ed., "PPP in HDLC Framing," RFC 1549, Daydreamer, December 1993.

7. Sklower, K., Lloyd, B., and Carr, D., " The PPP Multilink Protocol (MP)," work in progress.

8. Simpson, W., "PPP Link Quality Monitoring," RFC 1333, Daydreamer, May 1992.

9. Nix, E. E., *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.

10. Karayakaylar, Selcuk, *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.

11. Katz, D., "Transmission of IP and ARP over FDDI Networks," RFC 1390, Cisco Systems, Inc., January 1993.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Dudley Knox Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
4.	Professor Shridhar Shukla, Code EC/Sh Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	2
5.	Professor Gilbert Lundy, Code CS/Ln Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5118	2
6.	Professor Paul Moose, Code EC/Me Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
7.	Mr. Mark Russon, UNISYS Mail Station F2-G14 640 North 2200 West Salt Lake City, UT 84116-2988	1

8. CDL Program Manager
Defense Support Project Office
Washington D.C. 20330-1000 1