OPEN SYSTEMS INTERCONNECTION PASSIVE MONITOR

OSI-PM

By

Jeffrey Lo (Kin Hung)

Bachelor Of Computer Science (BCS), Carleton University, 1988

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
(DEPARTMENT OF COMPUTER SCIENCE)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

August 1990

© Jeffrey Lo (Kin Hung), 1990
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date Aug 30, 1990
Abstract

The Open Systems Interconnection Passive Monitor (OSI-PM), which is based on the principles of the OSI-Reference Model (OSI-RM), provides a framework for the development of multi-layer passive monitoring and testing. It adopts the same seven-layer architecture of the OSI-RM and provides the capability of selectively displaying, capturing, and analyzing the protocol events on single or multiple connections for any subset or all of the seven layers. Different from conventional monitors, the OSI-PM is able to detect protocol violation as they occur in addition to the monitoring functions. The current OSI-PM is able to monitor and test up to the transport layer of the OSI-RM. This thesis discusses the design, prototype implementation and testing of the OSI-PM.
# Contents

Abstract ii

Contents iii

List of Figures v

Abbreviations v

Acknowledgement vii

1 Introduction 1
   1.1 A brief introduction to the OSI-RM 2
   1.2 Related Work 3
      1.2.1 MPT386.2 X.25 monitor 3
      1.2.2 Trace Analysis 4
   1.3 Goals of the Thesis 5
   1.4 Outline of the Thesis 6

2 Architecture 7
   2.1 Data Extraction Module (DEM) 9
   2.2 Data Storage Module 9
   2.3 Monitor Module (MM) 10
      2.3.1 Synchronization 12
      2.3.2 Protocol Stack Submodule (PSS) 13
      2.3.3 Data Processing Submodule (DPS) 14

3 Implementation 18
   3.1 The environment 18
      3.1.1 The Synchronous Serial Driver 19
   3.2 Data Extraction Module 20
3.3 Data Storage Module ................................................. 21
3.4 Monitor Module (MM) .................................................. 21
  3.4.1 Data Processing Submodule ..................................... 21
  3.4.2 Protocol Stack Submodule ...................................... 24

4 Testing of the OSI-PM ................................................. 29
  4.1 MPT386.2 X.25 Monitoring ......................................... 29
  4.2 Report Generation .................................................. 33
    4.2.1 Report of Error ............................................... 34
  4.3 Report of Sample Traces .......................................... 35
    4.3.1 Report #1 .................................................... 35
    4.3.2 Report #2 .................................................... 41
    4.3.3 Report #3 .................................................... 42
    4.3.4 Report #4 .................................................... 43
  4.4 Error Report ...................................................... 45
    4.4.1 Error Report #1 ............................................... 45
    4.4.2 Error Report #2 ............................................... 46
    4.4.3 Error Report #3 ............................................... 48

5 Conclusion and Future Work ........................................ 50

A User Manual ........................................................ 55

B Software and Hardware Configuration ............................... 59
  B.1 Software Configuration .......................................... 59
  B.2 Hardware Configuration .......................................... 60

C Guide for adding a new entity .................................... 63
List of Figures

1.1 Test architecture with arbiter .................................................. 4
2.1 OSI-PM ........................................................................ 7
2.2 Structure of the Monitor Module ........................................... 11
3.1 Format of Packed Data ............................................................. 20
4.1 Connecting the DTE, the DCE and the OSI-PM ......................... 30
4.2 Connecting the DTE, the DCE and the MPT386.2 ....................... 31
4.3 Throughput of ean line, Thur., July 27, 1989 ............................. 32
4.4 Report Format ..................................................................... 33
B.1 Configuration of the Cables ....................................................... 61
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Abstract Service Primitive</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DCE</td>
<td>Data Circuit-Terminating Equipment</td>
</tr>
<tr>
<td>DPS</td>
<td>Data Processing Submodule</td>
</tr>
<tr>
<td>DEM</td>
<td>Data Extraction Module</td>
</tr>
<tr>
<td>DSM</td>
<td>Data Storage Module</td>
</tr>
<tr>
<td>DTE</td>
<td>Data Terminal Equipment</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organization</td>
</tr>
<tr>
<td>IUT</td>
<td>Implementation Under Test</td>
</tr>
<tr>
<td>MM</td>
<td>Monitor Module</td>
</tr>
<tr>
<td>MPT</td>
<td>Multi-port Protocol Tester</td>
</tr>
<tr>
<td>OSI-PM</td>
<td>Open Systems Interconnection Passive Monitor</td>
</tr>
<tr>
<td>OSI-RM</td>
<td>Open Systems Interconnection Reference Model</td>
</tr>
<tr>
<td>PDU</td>
<td>Protocol Data Unit</td>
</tr>
<tr>
<td>PSS</td>
<td>Protocol Stack Submodule</td>
</tr>
<tr>
<td>SAP</td>
<td>Service access point</td>
</tr>
</tbody>
</table>
Acknowledgement

I sincerely thank Dr. Samuel T. Chanson, my thesis supervisor, for his invaluable guidance, commitment and supportiveness. Thanks also go to Dr. Son Vuong who acts as the second reader of my thesis.

Thanks to my friends Bernard P. Lee and M. Lau for the many discussions on this work and all the help they gave me.

Thanks to all of my friends in the department.

Last but not the least, my deepest thanks go to my parents for their endless help, support and love without which none of this would have been possible.
Chapter 1

Introduction

The Open Systems Interconnection Passive Monitor (OSI-PM), which is based on the principles of the OSI-Reference Model (OSI-RM), provides a framework for the development of multi-layer passive monitoring and testing. The term passive means the monitor is capable of performing its functions without interfering with the normal operations of the communicating entities. The OSI-PM adopts the same seven-layer architecture of the OSI-RM representing the seven protocol layers from the Physical layer up to the Application layer. It provides the capability of selectively displaying, capturing, and analyzing the protocol events on single or multiple connections for all of the seven layers or any subset. Protocol events include decoded PDUs, internal state changes, protocol violations and timeouts. The major differences of the OSI-PM from conventional monitors are its ability to detect protocol violation as they occur (passive testing) and the modular design that allows easy extension to multi-layer monitoring and testing. The monitor is especially useful in capturing errors which tend to occur from time to time, and monitoring the behavior of two implementations which have been tested separately and have just been put into use. It should be pointed out, however, that unlike active testing, passive testing is unable (nor is it meant) to conclusively determine if a protocol
implementation conforms to the specifications it purports to adhere.

This thesis discusses the design of the OSI-PM and describes a prototype implementation. In the following sections, we present an overview of the OSI-RM, related work and goals of the research. A layout of the rest of the thesis is also given.

1.1 A brief introduction to the OSI-RM

The seven-layer OSI Reference Model developed by the International Standards Organization (ISO) provides a framework for the production of standards for communication and distributed data-processing systems [1].

The idea of layering is that the communications functions are partitioned into a vertical set of layers. Each layer performs a related subset of the functions required to communicate with another system. A lower layer performs more primitive functions and provides services to the next higher layer. Layers are independent with each others so that changes in one layer do not disrupt the other layers.

Each layer is known as the $n^{th}$ layer, the layers below and above being known as the $n-1^{th}$ and $n+1^{th}$ layers respectively. The active elements in each layer are called entities. Two entities communicate with each other through service access points (SAPs). Each entity will only communicate with the entities directly above and below it. The lower layer of two adjacent layers is called the service provider and the upper layer is called the service user. In other words, if the $n^{th}$ layer is the service provider, the $n+1^{th}$ layer is the service user. The service user uses the service provided by its service provider.

Services are specified by a set of abstract service primitives (ASPs) available to a user or other entity to access the service. There are four classes of (ASPs) defined in the OSI-RM: request, indication, response and confirm.
The rules and conventions used for the conversation between layer n on two different machines, i.e. conversation between two peer entities, are collectively known as the layer n protocol. In reality, data are not directly transferred from layer n on one machine to layer n on another machine. Each layer passes the data to the layer immediately below it, until the lowest layer is reached. The lowest layer, known as the physical layer in the OSI-RM is actually a physical medium through which actual communication occurs.

Data exchanged between two layer n entities using a layer n protocol is called Protocol Data Units (PDU). PDUs contain either data or control information necessary for the conversation.

1.2 Related Work

1.2.1 MPT386.2 X.25 monitor

The Multi-port Protocol Tester (MPT386.2) manufactured by IDACOM Electronics, is a new generation of portable protocol tester. It includes most of the capabilities of dedicated protocol test systems and all of the capabilities of commercial portable testers [6]. It has a built-in Forth interpreter by which most of its applications are implemented.

The X.25 monitor [5] is written in Forth and is one of the applications that runs on the MPT386.2. It is able to capture and display the decoded X.25 packets from the physical interface between the DTE\(^1\) and the DCE\(^2\). This information may be printed on a display screen, stored in a capture ram, or recorded on disk. Its trigger and filter functions allow the user to select desired events to report, capture and record. The data transmission throughput between the DCE and the DTE can also be measured.

---

\(^1\)the terminal or computer is officially called a DTE (Data Terminal Equipment)

\(^2\)the modem is officially called a DCE (Data Circuit-Terminating Equipment
CHAPTER 1. INTRODUCTION

The main advantage of the X.25 monitor is its portability. However, it is restricted to packet layer monitoring only and it is not easily expandable. It is also not able to perform passive testing.

1.2.2 Trace Analysis

There are two major concerns in conformance testing: test sequence generation and trace analysis [7]. The test systems which correspond to these are the test engine and the arbiter (sometimes called observer or trace analyzer) respectively. The test engine is responsible for determining and sending the test sequence to the IUT (Implementation Under Test), and the arbiter is responsible for checking whether the trace conforms to the protocol specification.

Test sequence generation is an active test method whereas trace analysis is a passive
test method. An arbiter for testing between two OSI Transport Protocol implementations is developed in [7] as shown in Figure 1.1. The arbiter observes the PDUs exchanged between two protocol implementations and detects any inconsistency caused by the IUT(s) not conforming to the protocol specification. The arbiter contains two trace analysis modules, one for each observed IUT. The observed trace of PDUs received from the different IUTs is recorded in a trace file for later processing.

The reference specification of the two trace analysis modules is the same as that for a lower tester in the distributed test architecture. Only the properties visible at the lower interfaces of the IUTs can be checked by the arbiter. Many protocol properties, such as the correct transfer of user data cannot be checked [7].

The arbiter was first specified in Estelle. After this specification was sufficiently completed, it was translated into Pascal and combined with run-time support routines. These routines include an interface to the underlying Network service (provided by an X.25 network) and functions for operator control and copying the trace onto a file.

1.3 Goals of the Thesis

The goal of this thesis is to design and implement an OSI multiple layer passive monitor. The monitor is to include most of the capabilities found in the MPT386.2 X.25 monitor and it is to be expandable up to the application layer. Its architecture should be as close to the architecture of the OSI-RM as possible. It is also to be a seven layer passive tester and each layer is responsible for testing the corresponding layer in the OSI-RM. Like the OSI-RM, layers are independent with each other so that changes in one layer do not require change in the other layers.

The monitor is designed to monitor the conversation between the DCE and the DTE
CHAPTER 1. INTRODUCTION

in real time. The DCE and the DTE will not wait for the monitor to finish processing one PDU before sending the next one; in fact, they are unaware of the existence of the monitor. Therefore, the monitor should be fast enough to catch all interactions between the two terminals without missing any of the packets exchanged.

The design should allow protocol layer(s) to be selected for displaying and testing. Only those desired information will be captured and displayed. Each layer contains a decoder and one (or more) finite state machine(s) (FSM)\(^3\). Testing in each layer can be done by driving the decoded PDU through the FSM(s) of that layer.

1.4 Outline of the Thesis

Following the introduction, Chapter 2 describes the detailed design and architecture of the monitor. Descriptions of the prototype implementation are contained in chapter 3. Chapter 4 discusses the results of using the monitor to monitor a life communication line. Chapter 5 concludes the thesis and outlines some future work.

\(^3\)For protocols that contain sublayers or more than one class (such as X.400 and ISO Transport), more than one FSM will be needed.
Chapter 2

Architecture

The OSI-PM, as shown in Figure 2.1, contains three modules: Data Extraction Module (DEM), Data Storage Module (DSM) and Monitor Module (MM). The DEM extracts data from the physical interface between the DTE and DCE by using a Synchronous Serial (ZSS) interface. Data are sent by the DEM to the DSM for temporary storage until the MM is ready to process it. The amount of code of the DEM and DSM is much smaller.
CHAPTER 2. ARCHITECTURE

than that of the MM, since the functions of the DEM and DSM are simple though the processing overhead is high. The three modules are actually implemented on three different machines so that they can run concurrently in a pipeline fashion to increase packet processing speed.

The whole system was originally implemented on one single machine and it contained only the DEM and MM. The two modules ran in pseudo parallelism as two interleaved concurrent processes. Data extracted by the DEM from the communication line were stored in a buffer and processed by the MM directly from that buffer. However, the system was too slow and could not keep up with a line rate of 4800 b/s of a X.25 line\(^1\).

The MM took most of the CPU cycles so that the ZSS driver could not keep up. This caused some of the X.25 packets to be missed and the testing could not be done correctly. Therefore, the design was changed to build the two modules on two different machines, and they communicated with each other via an Ethernet socket. The speed of the monitor increased and it could handle throughput at a rate between 4800 and 9600 b/s. However, it was still not fast enough for a standard 9600 b/s line. The DEM was found to be the bottleneck since the ZSS driver performs a lot of i/o and interrupt operations and thus requires much CPU processing. The design was changed again by introducing a new DSM which is implemented on a third machine. The DEM is only responsible for extracting data and sending them to the DSM. As a result, the final version of the monitor is capable of handling more than 19200 b/s throughput rate.

The following sections describe the detailed design of each of the three modules.

---
\(^1\)Data flowing through the X.25 line consist not only of X.25 packets but also the higher level PDUs, since the higher level protocol PDUs are embedded in the X.25 packets
2.1 Data Extraction Module (DEM)

The current DEM and the ZSS driver are both developed on a Sun-3/50 workstation. The main function of the DEM is to catch all data flowing along the communication line without missing any of them. Since the driver and the module are running on the same machine, the DEM should be as efficient as possible in order to allow more CPU cycles for the ZSS driver. The original version of the DEM contained code only for reading data from the driver and sending it to the DSM.

Measurement data show that the read and send system calls are quite expensive. In order to minimize the number of these calls, several data frames are packed into a buffer to form a larger data unit before being sent to the DSM. The whole buffer is sent when it is full, and the DEM starts to fill the buffer again.

The number of frames to be packed in a buffer is determined by the size of each individual frame. The frame size of the line being tested varies from 2 bytes (flow control frame) to 256 bytes (information frame). All frames from the line and their lengths are stored in the buffer. The buffer size should be made as large as possible so that more frames can be sent by one call. Since Unix running on an Ethernet is not able to transmit data larger than 2K bytes, a buffer size of 2K bytes is chosen.

2.2 Data Storage Module

A requirement of a passive monitor is to catch all data passing through the communication line and not miss one single packet. Since the MM is the most processing intensive part of the system, a storage module is needed for temporarily storing data when the monitor is too busy. This module is implemented on a Sun-Sparcstation and is connected to the DSM and MM via Ethernet sockets.
The DSM is structured as a circular buffer with data units of 2K bytes each. Data from the DEM are copied directly into a data unit of the buffer without making any modification. The total buffer size should be chosen to ensure that it will not overflow even at peak load. From our measurement results, the buffer is almost always empty since the MM is typically faster than the rate of data extraction. Currently, the buffer consists of 100 data units but a much smaller circular buffer would be sufficient.

There are four different message types from the MM to the DSM concerning the current state of the monitor:

1. **Ready state**: the MM is ready to process the next data unit and the DSM may send data to it.

2. **Stopped state**: the monitor is stopped by the user and therefore all data from the DEM should be discarded.

3. **Restart state**: the monitor is back to its normal working condition after being stopped. The DSM should start buffering data again.

4. **Shut down state**: the monitor is shut down and the DSM should also be shut down.

Note that the DSM will send data to the MM only if a ready message is sent by the MM, thus there is no need to have a MM_not_ready message.

### 2.3 Monitor Module (MM)

The DEM and DSM described previously are responsible for data extraction and storage, but the actual processing of the packets takes place in the MM. The MM is connected to the DSM and sends a ready signal to the DSM when it is ready to process
Figure 2.2: Structure of the Monitor Module
the next data unit. The MM consists of two submodules: data processing submodule (DPS) and protocol stack submodule (PSS) (see figure 2.2). The following subsections describe the two submodules and how the monitor synchronizes its state with the two terminals (the DTE and the DCE).

2.3.1 Synchronization

The monitor passively monitors and tests the two data terminals, and the initial internal states of the two terminals are not known to the monitor at the start of a session. Therefore, it is necessary to synchronize the monitor with the current states of the two terminals to start the analysis. There are two ways to do this. The first way is to wait for a restart request packet from the DTE or a restart indication packet from the DCE. When the two terminals are doing a restart procedure, they will be reset to their ready state (P1), and the state of the monitor can also be set to the ready state.

The second method is to wait until all current calls or connections to be completed and the states of the two terminals return to their ready state. The monitor waits until there are no data flowing between the two terminals for a short period of time, and assumes that all current calls are completed. Our measurement data show that for the X.25 line being tested by the current OSI-PM, the longest idle time between two packets within one call is around 10 to 12 seconds. If the waiting time is longer than 12 seconds, the synchronization procedure should be completed correctly. However, the waiting time should be as short as possible, and finally 15 seconds is chosen based on experimental results. This wait time can be changed by the user (see section 2.2.2.4).
2.3.2 Protocol Stack Submodule (PSS)

The Protocol Stack submodule is constructed as a seven-layer protocol stack. The name of each layer inside the protocol stack is chosen to be the same as the corresponding layer of the OSI-RM for easy reference. For example, layer 2 of the protocol stack submodule is called the data link layer and it performs monitoring and testing of the data link layer of the OSI-RM.

Like the OSI-RM, the layer boundaries should be chosen to minimize the information flow across the interface between the two layers. Each layer in the protocol stack has exactly one entry point. Information is passed from a lower layer to the layer above it through the upper layer entry point. Information will never flow in the reverse direction since it is a receive-only implementation.

The active elements in each layer are called entities. Protocol entities are analogous to OSI-RM entities. They are basically reference implementation of the supported protocols. The implementation of each of the entities follows the corresponding protocol specification. Each entity contains a decoder and one (or more) finite state machine(s). In order to achieve more efficient execution time, only the DTE is being tested. Therefore, the FSM is built for the DTE only. This is sufficient because if a protocol violation is detected by the DCE FSM, it will also be detected by the DTE FSM. Therefore, testing the DCE is redundant and it will make the implementation more complex and slow down the monitor.

The FSM of each entity is driven by two types of data, the data which are sent and received by the DTE. They can be distinguished by knowing the host which sent the data, and this information is available to the data processing submodule (DPS) and is passed to the protocol stack submodule (PSS) as a parameter. Thus, if the sending host
CHAPTER 2. ARCHITECTURE

is the DCE, the data are received by the DTE, and if the sending host is the DTE, the data are sent by the DTE. When a protocol violation has occurred, the entity which detects the error reports this to the DPS which performs error handling (discussed in section 2.3.3).

Each entity contains its own reset function, filter function and entry point. The addresses of those functions and entry point are known to the monitor from the initialization procedure of that entity. The reset function is used to reset an entity when the monitor needs to restart or when a protocol violation has occurred. The filter function allows the user to add or delete selected protocol data units for displaying and capturing for a particular layer. These functions are only called by the data processing submodule.

2.3.3 Data Processing Submodule (DPS)

The data processing submodule is the main control module inside the MM. It sits between the protocol entities and the DSM. Its functions include interfacing between the MM and the DSM, capturing and displaying decoded PDUs, error handling, and user interface handling.

2.3.3.1 Interfacing between the MM and the DSM

Data are read from the socket connecting the DSM and the MM. As described in section 2.1, frames are packed together in the DEM before being sent to the DSM. Therefore, data units coming from the DSM contain packed data. The MM analyses all the frames inside the packed data and acknowledges to the DSM when it is done.

The length of a frame inside the packed data is available to the MM since the first byte of a frame (which is added by the DEM) contains the length field of the frame. After reading the length of the frame, the DPS extracts the exact number of bytes from the
packed data and passes the frame (and control) to the PSS for further analysis. The entry point of the PSS is the entry point of the data link layer. When control is passed back from the PSS, the DPS starts capturing and displaying the information (see section 2.3.3.2 below). After the whole process is completed, the DPS extracts the next frame from the packed data or sends an acknowledgement to the DSM when all frames from the packed data have been processed.

2.3.3.2 Capturing and Displaying

The monitor extracts data from the physical medium between the DTE and the DCE, so only external events, such as PDUs, can be captured. The DPS maintains a capture buffer for temporary storage of the output data. Like the receive buffer in DSM, the capture buffer is structured as a circular buffer but its size can be changed by the user. Each element of the buffer contains the completely decoded information of a physical packet which contains PDUs from every layer from one of the two terminals. That is to say, the information contains the decoded PDUs from the data link layer up to some higher layer and the information is stored in a human readable form. Information is stored as character strings so that it can be printed on the display screen or saved into a file directly.

This module contains a macro to be used by the entities of the PSS to insert data into the buffer. The organization of each of the elements of the buffer is as follows. Starting from the data link entity (if selected), the DPS stores the output from this entity at the beginning of an empty character string. Any additional data will be appended to the end of the string, until control is passed back from the PSS. The DPS will then add the name of the sending host (DTE or DCE) in front of the character string. The whole string is
then printed on the screen and the pointer advanced to the next element of the buffer.

### 2.3.3.3 Error Handling

One of the important features provided by the ISO-PM is its passive testing capability. The actual testing procedure is done by the PSS, with each entity of the PSS responsible for the corresponding protocol. When a protocol violation occurs, the corresponding entity reports the error, and the whole system is reset. The DPS is responsible for handling the procedures after an error has occurred.

The DPS provides a macro for dealing with error messages. When a protocol violation is detected, the entity which detected the error may call that macro to report an error message to the user. The error message is stored in the capture buffer before being saved in an error record file and displayed on the screen. The DPS keeps track of the history of the conversation between the DCE and the DTE in order to allow the user to examine how the error has occurred. Since all the information has been stored in the capture buffer, the DPS simply dumps the whole capture buffer into an error record file. This file is a regular text file so that it may be printed on the display screen or line printer.

After a protocol violation has occurred, the monitor resets the whole protocol stack, i.e., every entity inside the PSS, to its original state. As described in the previous section, each entity contains a reset function whose address is known to the DPS. To reset the PSS, the DPS simply calls the reset function of each of the entities. The monitor will stop monitoring, and the synchronization procedure has to be restarted if monitoring is to be resumed.
2.3.3.4 User Interface Handling

The monitor contains a simple menu driven user interface. Its functions and their descriptions are as follows:

1. **Display Error Record File**: displays the error record file on the display screen. Since the file is a UNIX text file, the DPS calls the corresponding UNIX command to print it.

2. **Filter**: selects specific PDUs within each layer for display and capture.

3. **Select Layer**: this function is used to add or delete layer(s) for display or capture. This gives the user the flexibility to look at the decoded PDUs for all seven layers or any particular layer(s). This can be done by disabling or enabling the display function(s) of that layer(s).

4. **Disable or Enable Layer**: this function is to allow the user to shut down or turn on the FSM of any particular layer(s). Note that if the user wants to study a specific layer, all the layers beneath it must be enabled.

5. **Change size of Capture RAM**: changes the size of the capture buffer (this affects the amount of history that will be dumped in case of error).

6. **Save Capture RAM to a file**: saves the whole capture buffer into a text file for off line analysis.

7. **Read File**: reads a file.

8. **Change Synchronization Wait Time**: changes the waiting time for the synchronization procedure.
Chapter 3
Implementation

The three modules of the current OSI-PM have been implemented on top of the UNIX operating system in three different Sun-3/50 workstations. They are written in C and the monitor is able to test up to the transport layer of the OSI-RM. This chapter discusses the prototype implementation of the monitor and its environment.

3.1 The environment

There are two existing environments available for developing the OSI-PM - the MPT386.2 and UNIX running on Sun-3 workstations. We chose the second environment for the following reasons:

1. UNIX provides a much better programming, debugging and text processing environment. The development time will be reduced significantly.

2. It contains a huge C - programming library, almost all needed primitive functions are implemented.

3. It is one of the most popular operating systems for minicomputers, workstations, and now PCs.
There is one advantage for building the OSI-PM on the MPT386.2. We do not have to worry about the interface between the monitor and the communication line since there is an existing X.25 monitor built on the MPT386.2. The OSI-PM may use the interface built in the X.25 monitor. If the OSI-PM is developed on the Sun-3 workstation, it requires a synchronous serial driver for the workstation in order to communicate with the X.25 line. Fortunately, we managed to get hold of a driver and modified it for our use.

3.1.1 The Synchronous Serial Driver

The synchronous serial driver (ZSS) contains two layers. The lower layer is used to send and receive raw data through a set of low level interrupt handlers. The higher layer consists of some kernel interface routines which passes data through mbufs using software interrupts. The driver has a socket interface that is accessible to the user programs. When an inet socket is opened with domain AF_CCITT and interface name zssn, an entry in the protocol switch table of the kernel will be created and the socket will be connected to the serial port n. Since the kernel end of the socket interface also uses mbufs to pass data in and out of the sockets, data are passed directly between the serial port and the socket interface so that the user program is able to read data from the serial port.

There are two serial ports installed at the back of a Sun-3 workstation - serial ports A and B. The two serial ports can be used to connect the Sun workstation with the DTE and DCE respectively, so that serial port A is used to collect data from the DTE and serial port B is used to collect data from the DCE. However, a socket of family AF_CCITT

---

1mbufs are standard Unix system memory buffers  
2Unix Internet Domain.
CHAPTER 3. IMPLEMENTATION

Figure 3.1: Format of Packed Data

opens the synchronous driver on both serial ports A and B and the user program is not able to distinguish from which port the data are read. Since only the lower layer of the driver knows where the data come from, we had to modify the lower layer of the driver so that it passes also the port name to the mbufs. In order to minimize the change to the driver, we modified the address field of the data link layer's frame. Since only bit one and bit two are used in the address field, we use bit eight to indicate the port number such that the bit is set to one if the data are from port B, and zero otherwise.

3.2 Data Extraction Module

The implementation of the data extraction module (DEM) is relatively simple since the DEM contains only code for reading, packing and sending data. When a frame is read from one of the serial ports, the frame and its length are inserted into a buffer before sending to the DSM. The buffer is constructed as a one dimension array of size 2K bytes and has the format shown in figure 3.1.

In figure 3.1, the first frame contains 2 bytes of data, the second frame contains 5 bytes of data and so on. The buffer is sent to the DSM when it is full\(^3\) or when the

\(^3\)In fact the buffer is sent if the new frame inserted brings the total information in the buffer above
communication line is idle, and the DEM starts to fill the buffer again.

### 3.3 Data Storage Module

As described in section 2.2, the data storage module (DSM) is structured as a circular buffer with data units of 2K bytes each. Each data unit is formatted as follows:

```c
typedef struct RxbufRecord{
    int len;
    byte data[2000];
} RxbufRecord;
```

Each data unit contains the data and its length. When a packed data is received from the DEM, it is stored at the head of the circular buffer. The packed data at the tail of the buffer will be sent to the Monitor Module (MM) when a ready signal is received from the MM.

### 3.4 Monitor Module (MM)

As described in chapter 2, the Monitor Module (MM) contains two submodules: Data Processing Submodule (DPS) and Protocol Stack Submodule (PSS). The following subsections describe the implementations of the two submodules.

#### 3.4.1 Data Processing Submodule

**3.4.1.1 Entity Management**

The entity table is the data structure which connects the DPS with the entities in the PSS. The table records the locations of the reset functions, filter functions and entry

---

1.5K bytes.
points of the PSS entities. It is structured as an array of entity table records indexed by
the name of the entities. There is an entity table record for each protocol entity of the
PSS and this record is formatted as follows:

```c
typedef struct {
    void (*entryPt)();
    void (*resetProc)();
    void (*filterProc)();
}entitiesTableRecord;

entitiesTableRecord entitiesTable[MAX_LAYER+1];
```

- `entryPt` is a pointer to the entry point of the entity.
- `resetProc` is a pointer to the reset function.
- `filterProc` is a pointer to the filter function.

### 3.4.1.2 Capture Buffer Management

The capture buffer is structured as a circular buffer. It is implemented as a circular
link list with elements as follows (the capture buffer is called the script buffer inside the
program):

```c
typedef struct scriptBufStruct{
    char    data[100];
    struct scriptBufStruct *next;
}scriptBufRecord;

scriptBufRecord *scriptHead, *scriptBase;
```

The buffer is created at system initialization time. Data are inserted into the buffer
using the two macros:

- `outdata(X)` : used to insert decoded PDUs into the buffer. \( X \) is the data to be
  inserted.
CHAPTER 3. IMPLEMENTATION

- \textit{errorMsg}(X) : used to insert error message into the buffer in case a protocol violation has occurred.

The DPS contains a set of routines to maintain the buffer at system run time. The names and descriptions of these routines are as follows:

- \textit{flushScriptBuffer}(\textit{outputFile}) : this routine simply copies all data from the circular buffer into the file \textit{outputFile}.

- \textit{resetScriptBuffer}() : resets the buffer so that it contains no data.

- \textit{changeScriptBufSize}() : this routine is used when the user wants to change the buffer size. It destroys the original buffer, frees all the memory space, and creates a new buffer with the new size.

- \textit{saveScriptBuf}() : it calls \textit{flushScriptBuffer} to dump the buffer into a file specified by the user.

3.4.1.3 Data Passing

Data are received from the socket connecting the DSM and MM. The DPS uses the system call \textit{select} to check if there are data waiting. After the data are received from the DSM, the DPS extracts frames from the packed data and determines the host which sent the data. As described in section 3.1.1, bit eight of the address field of a link packet is one if the packet was from the DTE, and zero if it came from the DCE. The DPS passes the frame and whether the host is the DTE or DCE as parameters to the data link entity by the following procedure call:

\texttt{entitiesTable[DATA\_LINK].entryPt();}
The DPS uses DATA\_LINK as an index to locate the entry of the data link entity from the entity table and retrieve the entry point of this entity. It then passes frames to this entity through the entry point. This procedure call is also used by all entities to pass data to the entities above them. For example, the packet entity passes data to the transport entity by calling \textit{entitiesTable[TRANSPORT].entryPt()}.

### 3.4.1.4 Error Handling

The actual testing procedure is done by the PSS with each entity of the PSS testing the corresponding protocol entity of the OSI-RM. When a protocol violation is detected, the entity which detects the error uses the macro \textit{errorMSG} to report the error. The entity then calls the routine \textit{resetAllLayer} in DPS to deal with the error.

This routine resets all entities inside the PSS by calling the reset function of each of the entities. Since the reset functions are accessible through the entity table, the procedure call:

\begin{verbatim}
entitiesTable[DATA\_LINK].resetProc();
\end{verbatim}

calls the reset function of the data link entity (for example) with DATA\_LINK as an index to the entity table.

After resetting the entities, the DPS calls \textit{flushScriptBuffer} to dump the entire capture buffer into the error record file, and \textit{resetScriptBuffer} to reset the buffer. The current time and date are also recorded in the file.

### 3.4.2 Protocol Stack Submodule

The current protocol stack submodule (PSS) contains three protocol entities: data link entity, network entity, and transport entity. The system is capable of testing up to the
transport layer of the OSI-RM. Each entity consists of an entry point (which is actually a procedure call), decode function, reset function, filter function, display function, initialization function and a set of supported routines. It is up to the implementer to design his own set of functions. The following sections give a general overview of the implementation of the PSS.

3.4.2.1 Initialization

The implementer of the entity is responsible to add the initialization routine to the main routine of the DPS in the program named monitor.c. A typical name for the initialization routine is \texttt{XX\_init()} where \texttt{XX} is the short form of the entity name. For example, \texttt{DL\_init()} is the initialization routine for the data link entity and \texttt{PK\_init()} is the initialization routine for the packet (or network) entity.

3.4.2.2 Creation of Entities

A new entity is added to the OSI-PM by executing its initialization routine which calls the \texttt{setEntryPoint} routine. The \texttt{setEntryPoint} routine is one of the supported routines for the DPS. It adds a new entity table record to the entity table.

\begin{verbatim}
void setEntryPoint(layer, entryPoint, resetProc, filterProc);
\end{verbatim}

The entity passes the following parameters to the routine:

- the name of the layer such as \texttt{DATA\_LINK} and \texttt{NETWORK}. The entity table record is referenced by this name.
- the entry point of the entity.
- the addresses of the reset and the filter functions.
3.4.2.3 Decoding and Testing

For each of the current three entities of the PSS, decoding and testing are implemented as one single routine in order to minimize the number of procedure calls. The PDUs are driven through the Finite State Machine (FSM) immediately after they are decoded. Each FSM of the entities is implemented as a cascaded switch statement. The entity performs action that corresponds to the decoded PDU within the current state of the DTE. For example, part of the FSM for the packet entity is as follows:

```c
switch(decoded PDU type) {
    case N_PTI_CALL_REQ:
        switch(DTE current FSM) {
            case CALL_SETUP_STATE:
                switch(DTE current state) {
                    case P1:
                        statement1;
                        break;
                    case P2:
                        statement2;
                        break;
                    case P3:
                        break;
                    case P7:  
                        break;
                }  
            break;
        }
    break;
    case DATA_TRANSFER:
        
    }
break;
    case N_PTI_CALL_CONN:
        
}  
```
CHAPTER 3. IMPLEMENTATION

For the above FSM, if the decoded PDU type is the CALL REQUEST packet and the current FSM used is CALL.SETUP.STATE\(^4\) and the current state of the DTE is the ready state P1, then statement1 will be executed. We assume the reader of this thesis is familiar with the X.25 packet layer protocol, and will not go into details.

The FSMs for the data link entity and the transport entity also follow this format.

3.4.2.4 Reset Function

The reset function is used to reset an entity when the monitor needs to restart or when a protocol violation has occurred. This function simply resets the FSMs of the DTE of a particular entity to their ready state. For example, the reset function for the packet entity resets the current FSM of the DTE to its ready state (P1).

3.4.2.5 Filter and Display Functions

The filter function allows the user to add or delete selected protocol data units of a particular layer for display and capturing. The typical implementation of the filter function is as follows (using the packet layer as an example). A variable \(PKmask[pk-name]\) is defined for each packet of the packet entity where pk-name is a variable representing the name of the packet such as CALL_REQUEST and RESTART_REQUEST. The variable \(PKmask[pk-name]\) is set to a constant \(MASK\) if the user does not want to capture and display the packet named pk-name; it is set to zero otherwise. This variable is used by the display function to determine whether the information should be inserted in the capture buffer or discarded.

The display function of the packet entity is as follows:

\(^4\)There are three different FSMs built in the packet entity, they are RESTART.STATE, CALL.SETUP.STATE and DATA.TRANSFER.


\begin{verbatim}
switch(packet name | PKmask[packet name]){
    case INCOMING_CALL :
        outdata("INCOMING CALL");
        break;
    case CALL_REQUEST :
        outdata("CALL REQUEST");
        break;
    case CALL_CONNECTED :
        outdata("CALL CONNECTED");
        break;
    case CLEAR_REQUEST :
        outdata("CLEAR REQUEST");
        break;
    case DIAGNOSTIC :
        break;
}
\end{verbatim}

If the packet INCOMING_CALL is sent or received by the DTE, it is or-ed with the variable \texttt{PKmask[INCOMING_CALL]} before it is inserted in the capture buffer. Therefore, unless \texttt{PKmask[INCOMING_CALL]} is equal to zero (i.e., the user wants to capture and display this packet) the information "INCOMING CALL" will not be added to the buffer using \texttt{outdata}.

\subsection*{3.4.2.6 Error Handling Function}

The error handling function of an entity is called when a protocol violation is detected by the entity. It constructs the error messages to be reported to the user and calls the \texttt{resetAllLayer} routine to instruct the DPS to reset the monitor.
Chapter 4

Testing of the OSI-PM

The OSI-PM was tested by using it to monitor the traffic to and from the ean\(^1\) gateway. The OSI-PM was connected between the DTE and the DCE as shown in figure 4.1. As described in section 3.1.1, the DTE, which represents the host (the ean gateway), was connected to serial port B and the DCE, which represents the outside world, was connected to serial port A at the back of the Sun-3 workstation. All seven layers of the OSI-RM were implemented on the host.

The following sections describe the result of monitoring the ean gateway using the MPT386.2 X.25 monitor, report generation for the OSI-PM, and some sample traces and error reports generated by the OSI-PM.

4.1 MPT386.2 X.25 Monitoring

The ean gateway was monitored by the MPT386.2 X.25 monitor before the OSI-PM was designed and implemented. The reasons for monitoring the line were as follows:

- to find out the bit patterns of the data traffic of a live communication line,

\(^1\)ean is one of the major electronic mailing systems used in the University of British Columbia.
CHAPTER 4. TESTING OF THE OSI-PM

Figure 4.1: Connecting the DTE, the DCE and the OSI-PM

- to find out the peak data rate of the communication line,

- to understand how a passive monitor works in order to design the OSI-PM.

The X.25 monitor was connected to the ean gateway, using a Y cable\(^2\), following the connection shown in figure 4.2.

As mentioned in Section 1.1.2, the monitor is restricted to packet layer monitoring only and it is not able to perform passive testing. Some sample traces were collected and they were useful for designing the output format of the OSI-PM. The following example is a short report produced by the X.25 monitor:

\[
\begin{align*}
\text{DCE} & \quad 01 \quad \text{SABM} \\
\text{DTE} & \quad 01 \quad \text{UA} \\
\text{DTE} & \quad 03 \quad \text{I} \quad 0 \quad \text{RESTART INDICATION} \\
\text{DCE} & \quad 03 \quad \text{RR}
\end{align*}
\]

\(^2\)A Y cable is a standard RS-232C cable which has three heads. It is used to connect three components together.
After monitoring the line for a few days, the peak data rate of the line was found to be around 7100 b/s. Therefore, the OSI-PM must be designed to handle more than that peak data rate. The throughput measurement of the ean line on Thursday, July 27, 1989, which contains the peak data rate, is shown in figure 4.3.

The remainder of this chapter discusses report and error generations of the OSI-PM.
Figure 4.3: Throughput of ean line, Thur., July 27, 1989
4.2 Report Generation

Each entity of the OSI-PM is responsible for reporting its decoded PDUs and internal state changes. It is up to the implementer of that entity to design the format of the report. However, a general format is used for all the entities of the current implementation of the OSI-PM. The format of the report of each entity is as follows: Four spaces are skipped at the beginning of the report line followed by a two-character entity name, followed by the information to be displayed. Figure 4.4 shows the report format of each of the entities.

If there is an internal state change to be reported, the name of the state is displayed after seven empty spaces. *(Note, only the state of each entity of the DTE is reported, see section 2.3.2.)* The reason for skipping four spaces before the entity name is to allow space to display the name of the host (DTE or DCE) which sent the data.
CHAPTER 4. TESTING OF THE OSI-PM

An example of the report generated by the OSI-PM is as follows:

DTE DL I NR=3 NS=4
PK LCN=7 CALL_REQUEST
   DTE CALL REQUEST (P2)
DCE DL RR NR=5
DCE DL I NR=5 NS=3
PK LCN=7 CALL_CONNECTED
   FLOW CONTROL READY (D1)
DTE DL I NR=4 NS=5
PK LCN=7 DATA PR=0 PS=0
   TP CONNECTION REQUEST
       WFCC
DCE DL I NR=7 NS=6
PK LCN=7 DATA PR=1 PS=0
   TP CONNECTION CONFIRM
       OPEN

In the above example, the DTE establishes a call in logical channel number seven (LCN=7) at the packet layer level. It sends a CALL REQUEST packet and changes its state to DTE CALL REQUEST (P2). The DCE replies with a CALL CONNECTED packet. The state of the DTE is changed to FLOW CONTROL READY (D1) after it has received the packet. After the network connection is established, the transport layer of the DTE sends a CONNECTION REQUEST using the connection and changes its state to WFCC (wait for connection confirm). Finally, the connection is completed after the transport entity of the DTE has received a CONNECTION CONFIRM from the DCE, and it changes its state to OPEN.

4.2.1 Report of Error

An error report is generated when a protocol violation occurred in a particular entity. It reports the name of the entity, the cause of the error, and the time and date the error occurred. The following is an example of an error reported by the packet entity:
Packet layer error for DTE, unexpected packet sent.....
Packet DATA is sent in state DTE CALL REQUEST (P2), LCN = 10...
Error occurred on Thursday July 5, 90, time 10:52:44

The packet entity of the OSI-PM detected an unexpected packet sent by the DTE at the packet layer of the communication line. The unexpected packet was a DATA packet which was sent in the state DTE CALL REQUEST (P2) in logical channel number 10 on Thursday July 5, 90, at 10:52 am. Note that in state DTE CALL REQUEST (P2), the DTE should be waiting for a CALL CONNECTED packet from the DCE (as in the previous example). A DATA packet should not have been sent before the connection is established.

4.3 Report of Sample Traces

The ean gateway was monitored by the OSI-PM for several days. Traces of the traffic to and from the gateway were collected, and sample reports produced by the OSI-PM are shown in the following sections. Detailed comments of the traces are given in report #1 only since the other reports are similar. Note that the states shown in the report are the states for the DTE (refer to section 2.3.2).

4.3.1 Report #1

This report contains data of all three entities - data link entity, packet entity and transport entity - of the current OSI-PM. Four calls are recorded in the following report. Two calls were requested by the DCE on logical channel number 1, and two calls were requested by the DTE on logical channel number 10. Both calls from the DTE were rejected by the DCE.

DCE DL I NR=1 NS=6 * The DCE sent an INCOMING CALL
CHAPTER 4. TESTING OF THE OSI-PM

PK LCN=1 INCOMING_CALL
DCE INCOMING CALL (P3)

* packet at logical channel number (LCN) 1, the DTE changed to state DCE INCOMING CALL (P3) after the packet was received.

DTE DL I NR=7 NS=1
PK LCN=1 CALL_ACCEPTED
FLOW CONTROL READY (D1)

* The DTE replied with a CALL ACCEPTED packet and changed its state to FLOW CONTROL READY (D1).
* The connection between the DTE and the DCE was established at the packet layer level. The two terminals were ready to exchange data using this packet connection.

DCE DL RR NR=2
DCE DL I NR=2 NS=7
PK LCN=1 DATA PR=0 PS=0
TP CONNECTION REQUEST
WFTRESP

* The transport entity of the DCE sent a CONNECTION REQUEST to the DTE using the network connection. The TPDU was embedded in the DATA packet of packet layer. The transport entity of the DTE changed its state to WFTRESP.

DTE DL I NR=0 NS=2
PK LCN=1 RR PR=1

* The DTE replied with a RR packet at the packet layer level.

DTE DL I NR=0 NS=3
PK LCN=1 DATA PR=1 PS=0
TP CONNECTION CONFIRM OPEN

* The DTE replied with a CONNECTION CONFIRM at the transport layer level, and changed its state to OPEN. The transport connection was established and the two terminals were able to transmit data using this connection.

DCE DL RR NR=3
DCE DL RR NR=4
DCE DL I NR=4 NS=0
PK LCN=1 RR PR=1
CHAPTER 4. TESTING OF THE OSI-PM

DCE DL I NR=4 NS=1
   PK LCN=1 DATA PR=1 PS=1
   TP DATA
   * This was the first DATA TPDU
   * sent by the DCE.

DTE DL I NR=2 NS=4
   PK LCN=1 RR PR=2
DCE DL RR NR=5
DTE DL I NR=2 NS=5
   PK LCN=1 DATA PR=2 PS=1
   TP DATA

DTE DL I NR=2 NS=6
   PK LCN=10 CALL_REQUEST
   DTE CALL REQUEST (P2)
   * The DTE wanted to establish
   * another network connection at
   * LCN #10, in addition to the
   * connection at LCN #1.

DCE DL RR NR=6
DCE DL RR NR=7
DCE DL I NR=7 NS=2
   PK LCN=1 RR PR=2
DTE DL RR NR=3

DCE DL I NR=7 NS=3
   PK LCN=10 CLEAR_INDICATION
   DCE CLEAR INDICATION (P7)
   * However, the DCE rejected the
   * call by returning a CLEAR
   * INDICATION packet. After
   * receiving the packet, the DTE
   * changed its state to DCE CLEAR
   * INDICATION (P7).

DTE DL I NR=4 NS=7
   PK LCN=10 CLEAR_CONFIRMATION
   READY (P1)
   * The DTE replied to the DCE using
   * a CLEAR CONFIRMATION packet and
   * changed its state to READY (P1).

DCE DL RR NR=0
DCE DL I NR=0 NS=4
   PK LCN=1 DATA PR=2 PS=2
   TP DATA
   * The two terminals were still
   * exchanging data using the
   * previous connection.
DTE DL I NR=5 NS=0
PK LCN=1 RR PR=3
DCE DL RR NR=1
DCE DL I NR=1 NS=5
PK LCN=1 DATA PR=2 PS=3
DTE DL I NR=6 NS=1
PK LCN=1 RR PR=4
DCE DL I NR=1 NS=6
PK LCN=1 DATA PR=2 PS=4
DTE DL I NR=7 NS=2
PK LCN=1 RR PR=5

DCE DL I NR=1 NS=6
PK LCN=1 DATA PR=5 PS=1
TP DATA
DTE DL I NR=7 NS=2
PK LCN=1 RR PR=2
DCE DL RR NR=2
DTE DL I NR=7 NS=3
PK LCN=1 DATA PR=2 PS=5
TP DATA
DCE DL RR NR=3
DCE DL RR NR=4
DCE DL I NR=4 NS=7
PK LCN=1 RR PR=6

DCE DL I NR=4 NS=0
PK LCN=1 CLEAR_INDICATION
DCE CLEAR INDICATION (P7) * The DCE sent a CLEAR INDICATION
* to complete the call at LCN #1.
DTE DL I NR=1 NS=4
PK LCN=1 CLEAR_CONFIRMATION
READY (P1) * The DTE changed its state to
* DCE CLEAR INDICATION (P7), and
* replied with a CLEAR
* CONFIRMATION packet. The state
* of the DTE is changed to
* READY (P1) after sending the
CHAPTER 4. TESTING OF THE OSI-PM

* packet.

The DTE and the DCE tried to set up calls at the same time. The DTE used LCN #10 and the DCE used LCN #1.

* The network connection at LCN #1 was established after the DTE sent a CALL ACCEPTED packet.

DCE DL RR NR=5
DTE DL I NR=1 NS=5
PK LCN=10 CALL_REQUEST
DTE CALL REQUEST (P2)

DCE DL RNR
DCE DL I NR=6 NS=1
PK LCN=1 INCOMING_CALL
DCE INCOMING CALL (P3)

DTE DL I NR=2 NS=6
PK LCN=1 CALL_ACCEPTED
FLOW CONTROL READY (D1)

DCE DL RR NR=7
DCE DL I NR=7 NS=2
PK LCN=1 DATA PR=0 PS=0
TP CONNECTION REQUEST WFTRESP
DTE DL I NR=3 NS=7
PK LCN=1 RR PR=1
DTE DL I NR=3 NS=0
PK LCN=1 DATA PR=1 PS=0

DCE DL I NR=1 NS=4
PK LCN=1 DATA PR=1 PS=1
TP DATA
DTE DL I NR=5 NS=1
PK LCN=1 RR PR=2
DCE DL RR NR=2
DTE DL I NR=5 NS=2
PK LCN=1 DATA PR=2 PS=1
TP DATA
DTE DL I NR=5 NS=3
PK LCN=1 CLEAR_REQUEST
DTE CLEAR REQUEST (P6)
The DTE sent a CLEAR REQUEST packet to clear the call at LCN #10 since the DCE did not reply to this call.
CHAPTER 4. TESTING OF THE OSI-PM

4.3.2 Report #2

One of the features provided by the OSI-PM is that it gives the user flexibility to display and capture the decoded PDUs for all existing layers or any particular subset of the layer(s). For the following report, the data link entity is not selected by the user. Therefore, only the data from the packet and transport entities are reported.

DCE PK LCN=1 INCOMING_CALL
DCE INCOMING CALL (P3)
DTE PK LCN=1 CALL_ACCEPTED
FLOW CONTROL READY (D1)
DCE PK LCN=1 DATA PR=0 PS=0
TP CONNECTION REQUEST
WFTRESP
DTE PK LCN=1 RR PR=1
DTE PK LCN=1 DATA PR=1 PS=0
TP CONNECTION CONFIRM
OPEN
DCE PK LCN=1 RR PR=1
DCE PK LCN=1 DATA PR=1 PS=1
TP DATA
DTE PK LCN=1 DATA PR=2 PS=0
TP DATA
DTE PK LCN=1 RR PR=2
TP DATA
DCE PK LCN=1 DATA PR=1 PS=1

DTE PK LCN=1 DATA PR=2 PS=1
TP DATA
DCE PK LCN=1 DATA PR=2 PS=2
TP DATA
DCE PK LCN=1 RR PR=2
DTE PK LCN=1 CLEAR_REQUEST
DTE CLEAR REQUEST (P6)
DCE PK LCN=1 CLEAR_CONFIRMATION READY (P1)
DTE PK LCN=10 CALL_REQUEST
DTE CALL REQUEST (P2)
DCE PK LCN=10 CALL_CONNECTED
FLOW CONTROL READY (D1)
DTE PK LCN=10 DATA PR=0 PS=0
TP CONNECTION REQUEST
WFCC
DCE PK LCN=10 RR PR=1
DCE PK LCN=10 DATA PR=1 PS=0
TP CONNECTION CONFIRM
OPEN
DTE PK LCN=10 RR PR=1
DTE PK LCN=10 DATA PR=1 PS=1
TP DATA
DCE PK LCN=10 DATA PR=0 PS=2
TP DATA
DTE PK LCN=10 RR PR=3
DTE PK LCN=10 DATA PR=3 PS=0
TP DATA
DCE PK LCN=10 RR PR=1
DCE PK LCN=10 DATA PR=1 PS=3
TP DATA
CHAPTER 4. TESTING OF THE OSI-PM

4.3.3 Report #3

Another feature provided by the OSI-PM is the filter function which allows the user to select specific PDUs within each layer for display and capturing. For the following report, the data link entity is not selected by the user and, in addition, the RR packet of packet layer is filtered out by the filter function.

DTE PK LCN=10 CALL_REQUEST
DTE CALL REQUEST (P2)
DCE PK LCN=10 CALL_CONNECTED
FLOW CONTROL READY (D1)
DTE PK LCN=10 DATA PR=0 PS=0
TP CONNECTION REQUEST
WFCC
DCE PK LCN=10 DATA PR=1 PS=0
TP CONNECTION CONFIRM
OPEN
### CHAPTER 4. TESTING OF THE OSI-PM

<table>
<thead>
<tr>
<th>DCE PK</th>
<th>LCN=10 DATA</th>
<th>PR=2 PS=5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TP DATA</td>
<td></td>
</tr>
<tr>
<td>DTE PK</td>
<td>LCN=9 DATA</td>
<td>PR=1 PS=1</td>
</tr>
<tr>
<td></td>
<td>TP DATA</td>
<td></td>
</tr>
<tr>
<td>DTE PK</td>
<td>LCN=10 CLEAR_REQUEST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTE CLEAR REQUEST (P6)</td>
<td></td>
</tr>
<tr>
<td>DCE PK</td>
<td>LCN=10 CLEAR_CONFIRMATION</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTE CLEAR REQUEST (P1)</td>
<td></td>
</tr>
<tr>
<td>DCE PK</td>
<td>LCN=9 DATA</td>
<td>PR=2 PS=1</td>
</tr>
<tr>
<td></td>
<td>TP DATA</td>
<td></td>
</tr>
<tr>
<td>DTE PK</td>
<td>LCN=9 DATA</td>
<td>PR=2 PS=2</td>
</tr>
<tr>
<td></td>
<td>TP DATA</td>
<td></td>
</tr>
<tr>
<td>DTE PK</td>
<td>LCN=9 DATA</td>
<td>PR=2 PS=3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCE PK</td>
<td>LCN=9 DATA</td>
<td>PR=2 PS=4</td>
</tr>
<tr>
<td></td>
<td>TP DATA</td>
<td></td>
</tr>
<tr>
<td>DTE PK</td>
<td>LCN=9 DATA</td>
<td>PR=5 PS=2</td>
</tr>
<tr>
<td></td>
<td>TP DATA</td>
<td></td>
</tr>
<tr>
<td>DTE PK</td>
<td>LCN=9 DATA</td>
<td>PR=5 PS=3</td>
</tr>
<tr>
<td></td>
<td>TP DATA</td>
<td></td>
</tr>
<tr>
<td>DCE PK</td>
<td>LCN=9 CLEAR_REQUEST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTE CLEAR REQUEST (P6)</td>
<td></td>
</tr>
<tr>
<td>DCE PK</td>
<td>LCN=9 CLEAR_CONFIRMATION</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTE CLEAR REQUEST (P1)</td>
<td></td>
</tr>
<tr>
<td>DTE PK</td>
<td>LCN=10 CALL_REQUEST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTE CALL REQUEST (P2)</td>
<td></td>
</tr>
<tr>
<td>DTE PK</td>
<td>LCN=9 CALL_REQUEST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTE CALL REQUEST (P2)</td>
<td></td>
</tr>
<tr>
<td>DCE PK</td>
<td>LCN=9 CALL_CONNECTED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FLOW CONTROL READY (D1)</td>
<td></td>
</tr>
<tr>
<td>DTE PK</td>
<td>LCN=9 CLEAR_REQUEST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTE CLEAR REQUEST (P6)</td>
<td></td>
</tr>
<tr>
<td>DCE PK</td>
<td>LCN=9 CLEAR_CONFIRMATION</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTE CLEAR REQUEST (P1)</td>
<td></td>
</tr>
<tr>
<td>DTE PK</td>
<td>LCN=10 CALL_REQUEST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTE CALL REQUEST (P2)</td>
<td></td>
</tr>
<tr>
<td>DCE PK</td>
<td>LCN=10 CALL_CONNECTED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FLOW CONTROL READY (D1)</td>
<td></td>
</tr>
<tr>
<td>DTE PK</td>
<td>LCN=10 CLEAR_REQUEST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTE CLEAR REQUEST (P6)</td>
<td></td>
</tr>
<tr>
<td>DCE PK</td>
<td>LCN=9 CALL_REQUEST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTE CALL REQUEST (P2)</td>
<td></td>
</tr>
<tr>
<td>DCE PK</td>
<td>LCN=9 CALL_CONNECTED</td>
<td></td>
</tr>
</tbody>
</table>

#### 4.3.4 Report #4

The following report contains data only for call setup and clear for the packet and transport entities.

| DTE PK | LCN=10 CALL_REQUEST |
|        | DTE CALL REQUEST (P2) |
| DTE PK | LCN=9 CALL_REQUEST |
|        | DTE CALL REQUEST (P2) |
| DCE PK | LCN=9 CALL_CONNECTED |
|        | FLOW CONTROL READY (D1) |
| DTE PK | CONNECTION REQUEST |
|        | WFCC |
| DCE PK | CONNECTION CONFIRM |
|        | OPEN |
| DCE PK | LCN=10 CALL_CONNECTED |
|        | FLOW CONTROL READY (D1) |
| DTE PK | CONNECTION REQUEST |
|        | WFCC |
| DCE PK | CONNECTION CONFIRM |
|        | OPEN |
| DTE PK | LCN=9 CLEAR_REQUEST |
|        | DTE CLEAR REQUEST (P6) |
| DCE PK | LCN=9 CLEAR_CONFIRMATION |
|        | DTE CLEAR REQUEST (P1) |
| DTE PK | LCN=10 CLEAR_REQUEST |
|        | DTE CLEAR REQUEST (P6) |
| DTE PK | LCN=9 CALL_REQUEST |
|        | DTE CALL REQUEST (P2) |
| DCE PK | LCN=9 CALL_CONNECTED |


FLOW CONTROL READY (D1)
DTE TP CONNECTION REQUEST
WFCC
DCE TP CONNECTION CONFIRM
OPEN
DCE PK LCN=10 CALL_CONNECTED
FLOW CONTROL READY (D1)
DTE TP CONNECTION REQUEST
WFCC
DCE TP CONNECTION CONFIRM
OPEN
DTE PK LCN=9 CLEAR_REQUEST
DTE CLEAR REQUEST (P6)
DCE PK LCN=9 CLEAR_CONFIRMATION
READY (P1)
DTE PK LCN=10 CLEAR_REQUEST
DTE CLEAR REQUEST (P6)
DCE PK LCN=10 CLEAR_CONFIRMATION
READY (P1)
DTE PK LCN=10 CALL_REQUEST
DTE CALL REQUEST (P2)
DTE PK LCN=9 CALL_REQUEST
DTE CALL REQUEST (P2)
OPEN
DCE PK LCN=10 CALL_CONNECTED
FLOW CONTROL READY (D1)
DTE TP CONNECTION REQUEST
WFCC
DCE TP CONNECTION CONFIRM
OPEN
4.4 Error Report

An error report is generated when a protocol violation occurred in a particular entity. It reports the name of the entity, the cause of the error, and the time and date the error occurred. The ean line was tested for a few days using the OSI-PM. However, no error was detected by the system. In fact, the ean software has been running for a few years and it is a reliable system. Therefore, we generated some artificial errors in the data extracted from the ean line in order to test the OSI-PM. The errors were generated in the Data Storage Module (DSM). After receiving data from the Data Extraction Module (DEM), the DSM randomly changed some bits of the data before sending them to the Monitor Module (MM). The following sections show the errors that were detected and reported by the system.

4.4.1 Error Report #1

```
DCE DL I  NR=2   NS=7
  PK LCN=8 RR PR=5
DCE DL I  NR=2   NS=0
  PK LCN=8 RR PR=6
DTE DL I  NR=1   NS=2
  PK LCN=10 CALL_REQUEST
    DTE CALL REQUEST (P2) /* CALL REQUEST sent,*/
  /* the state of DTE is*/
  /* changed to DTE CALL*/
DCE DL RR NR=3
DCE DL I  NR=3   NS=1
  PK LCN=8 DATA PR=6 PS=2
    TP DATA /* REQUEST (P2).*/
```

DCE DL I  NR=1   NS=7
Packet layer error for DTE, unexpected packet sent.....
Packet DATA is sent in state DTE CALL REQUEST (P2), LCN = 10...
Error occurred on Thursday July 5, 90, time 10:52:44

The above behavior constituted an error since an unexpected packet DATA was sent in state DTE CALL REQUEST (P2) on logical channel number 10. The system records a trace of the history before the error which may be analyzed to determine the cause of the error. In this case, CALL REQUEST was sent by the DTE in the beginning of the trace, but there was no CALL CONNECT found. A DATA packet should not have been sent before the connection was established.

4.4.2 Error Report #2

DTE DL I NR=2 NS=5
PK LCN=8 RR PR=3
DCE DL RR NR=6
DTE DL I NR=2 NS=6
PK LCN=8 DATA PR=3 PS=5
CHAPTER 4. TESTING OF THE OSI-PM

Packet layer error for DTE, unexpected packet sent.....
Packet DATA is sent in state READY (P1), LCN = 128...
Error occurred on Thursday July 5, 90, time 11:2:40

The system detected the error that an unexpected packet DATA was sent in state READY (P1) at logical channel number 128. This error might be the result of a transmission error or might be caused by noise on the line which changed the logical channel number to 128.
4.4.3 Error Report #3

DCE DL I NR=7 NS=4
   PK LCN=9 CALL_CONNECTED
       FLOW CONTROL READY (D1)
DTE DL I NR=5 NS=7
   PK LCN=9 DATA PR=0 PS=0
       TP CONNECTION REQUEST
       WFCC
DCE DL RR NR=0
DCE DL I NR=0 NS=5
   PK LCN=9 RR PR=1
DCE DL I NR=0 NS=6
   PK LCN=9 DATA PR=1 PS=0
       TP CONNECTION CONFIRM
       OPEN
DTE DL I NR=7 NS=0
   PK LCN=9 RR PR=1

DTE DL I NR=4 NS=6
   PK LCN=9 DATA PR=2 PS=2
       TP DATA
DCE DL RR NR=7
DCE DL I NR=7 NS=4
   PK LCN=1 DATA PR=2 PS=0
       TP DATA

Transport layer error for DTE, invalid TPDU sent
DATA ACKNOWLEDGMENT is sent in state CLOSED, class 0.
Error occurred on Thursday July 5, 90, time 16:8:15
The above represented an error behavior detected by the transport entity of the OSI-PM since an invalid TPDU - DATA ACKNOWLEDGMENT was sent by the DTE in class 0. The DATA ACKNOWLEDGMENT is not defined in class 0.
Chapter 5

Conclusion and Future Work

We have described the design and a prototype implementation of the Open Systems Interconnection Passive Monitor (OSI-PM). The system is based on the principles of the OSI - Reference Model (OSI-RM) and it provides a framework for the development of multi-layer passive monitoring and testing. The OSI-PM adopts the same seven-layer architecture of the OSI-RM representing the seven protocol layers. The features provided by the OSI-PM include:

- detects protocol violation as they occur in addition to the monitoring functions.

- provides the capability of selectively displaying, capturing, and analyzing the protocol events.

- provides flexibility to display and capture the decoded PDUs for all existing layers or any particular subset.

- the filter function allows the user to select specific PDUs within each layer.

- layers are independent of each other so that changes in one layer do not require change in the other layers.
Detailed discussions of the architecture and implementation of the OSI-PM have been presented. The OSI-PM contains three modules: Data Extraction Module (DEM), Data Storage Module (DSM) and Monitor Module (MM). The Monitor Module in turn contains two submodules: Data Processing Submodule (DPS) and Protocol Stack Submodule (PSS). The three main modules are implemented on three different machines so that they can run concurrently in a pipeline fashion to increase packet processing speed. The modules of the current OSI-PM are implemented on top of the UNIX operating system in three different Sun workstations, and currently the system is capable of monitoring and testing the data link layer, the network layer and the transport layer of the OSI-RM.

We have used the OSI-PM to monitor the traffic to and from the can gateway. Some reports generated by the system including sample traces and error reports have also been presented.

Processing speed is one of the most important factors to be considered in developing a passive monitor and tester. Future work of the OSI-PM involves mostly techniques to improve the processing speed of the system further.

1. The Data Extraction Module was found to be the bottleneck since the ZSS driver performs a lot of i/o and interrupt operations and thus requires much CPU processing. We may use a faster ZSS driver or a faster Sun workstation, such as the Sparc-station, in order to increase the processing speed of the module, so that the system is able to handle much higher data rate.

2. The other way to increase the processing speed is to implement the seven entities into seven different machines and connect them via Ethernet sockets or other network so that all the seven entities can run concurrently in a pipeline fashion.
3. If the speed of the system is increased, more information carried by the PDUs can be reported.

4. The current system uses a menu driven user interface. A more user friendly interface may be developed for the system, such as a window based user interface.

5. Develop the other protocol entities of the OSI-RM.
Bibliography


Appendix A

User Manual

The current OSI-PM contains a menu driven user interface. The menu can be invoked by typing a Control-c at run time. The system stops processing and displays the following menu:

MENU
1. Display Error Record File.
2. Filter
3. Select Layer
4. Disable or Enable Layer
5. Change size of capture RAM
6. Save capture RAM to a file
7. Read file
8. Change synchronization wait time
9. Quit
Enter command or 'q' to go back to monitor:

Descriptions of each of the items of the menu are given in the following sections.

A.1 Display Error Record File

The system maintains an error record file for recording all protocol violations of the communication line which is being tested. This command is used to display the file on the display screen.
A.2 Filter

This command allows the user to select specific PDUs within each layer for display and capture. When this command is selected, the system prompts in which layer the user wants to do the filter function using the following menu:

Which Layer?

2: Data Link
3: Network
4: Transport

Enter number :

If, for example, the network layer is selected, an additional menu for the network layer will be shown on the screen as follows:

1: + INCOMING_CALL
2: + CALL_REQUEST
3: + CALL_CONNECTED
4: + CALL_ACCEPTED
5: + CLEAR_REQUEST
6: + CLEAR_INDICATION
7: + CLEAR_CONFIRMATION
8: - DATA
9: - RECEIVER_READY
10: - RECEIVER_NOT_READY
11: - REJECT
12: + INTERRUPT_REQUEST
13: + INTERRUPT_INDICATION
14: + INTERRUPT_CONFIRMATION
15: + RESET_REQUEST
16: + RESET_INDICATION
17: + RESET_CONFIRMATION
18: + RESTART_REQUEST
19: + RESTART_INDICATION
This menu shows all the packet names of the network layer. The sign '+' in front of a packet name means the packet is selected for display and capture, and the sign '-' means the packet is not selected. In the above example, the DATA packet and all the control flow packets are not selected.

A.3 Select Layer

This function is used to add or delete layer(s) for display or capture. This gives the user flexibility to look at the decoded PDUs for all layers or any particular subset. The following menu is displayed when this function is selected.

2: - Data Link
3: + Network
4: + Transport

Enter the layer number that you want to add or delete, or 'q' to quit:

For the above example, data link layer is not selected. Therefore, only the information from the network and the transport layers are displayed and captured.

A.4 Disable or Enable Layer

This function allows the user to shut down or turn on the FSM of any particular layer(s). The following example shows that the session, presentation and application layers are disable.

2: + Data Link
3: + Network
4: + Transport
5: - Session
6: - Presentation
7: - Application
Enter the layer number that you want to disable or enable, or 'q' to quit:

A.5 Change size of capture RAM

This function allows the user to change the size of the capture RAM. The size of the capture RAM is the amount of information to be saved into the error record file in case of error.

A.6 Save capture RAM to a file

The user may use this command to dump the information which is saved in the capture RAM into a file for off-line or on-line processing. This command will prompt the user for the name of the file.

A.7 Read File

This command allows the user to play back the information from a file.

A.8 Change synchronization wait time

This command allows the user to change the waiting time for the synchronization procedure (refer to section 2.3.1).
Appendix B

Software and Hardware Configuration

B.1 Software Configuration

As described in section 3.1.1, the OSI-PM uses a Synchronous Serial Driver (ZSS) to communicate with a X.25 communication line. Therefore, the driver must be installed so that the system can get to work. The procedures for installing the ZSS driver to a Sun 3/50 kernel are shown as follows:

1. Copy files `zs_sdlc.c`, `zs_proto.c`, `zss_interface.c`, and `zss_interface.h` to directory `/sys/sundev`.
   Copy file `CSSD50ZSS` to directory `/sys/sun3/conf`.
   Copy file `uipc_domain.c` to directory `/sys/os`.

2. Add the following two entries to the file `/sys/conf.common/files.cmnn`

   ```
   sundev/zs_sdlc.c          optional ZSS
   sundev/zss_interface.c    optional ZSS
   ```
and delete all entries starting with *netccitt*.

3. Change to directory /sys/sun3/conf and run `config CS_SD50ZSS`. The directory ../CS_SD50ZSS will be made if it doesn’t exist.

4. Change to directory ../CS_SD50ZSS and type `make` to make the new system.

5. The new kernel is stored in file *vmunix*. The user may copy it to the root directory (`cp vmunix /vmunix`), stop the current system (`/etc/halt`), and boot the new system (`b vmunix`).

Note that the user may refer to the UNIX manual for `config` for more details of configuring a kernel.

**B.2 Hardware Configuration**

After the driver is installed to the kernel, it is necessary to connect the Sun-3/50 workstation to the two terminals, the DTE and the DCE. The Sun-3/50 workstation is connected to the DTE and the DCE using a Y cable. In addition to the Y cable, the workstation requires a special cable to connect the Y cable to the two serial ports (see figure B.1). As described in section 3.1.1, the DTE is connected to the serial port B and the DCE is connected to the serial port A at the back of the workstation. Since the DCE uses the receive pin (pin #3) to send data, we may connect pin #3 of the Y cable to the receive pin (pin #3) of the serial port A. The DTE uses the send pin (pin #2) to send data and we may connect pin #2 of the Y cable to the receive pin of the serial port B. Besides the receive pin, pin #7 (ground) and pin #17 (receiver clock) are also connected from the Y cable to the serial ports as shown in figure B.1.
APPENDIX B. SOFTWARE AND HARDWARE CONFIGURATION

Figure B.1: Configuration of the Cables
To connect the Sun-3 workstation to the DTE and the DCE, we first connect the DTE and the DCE using any two heads of the Y cable, and connect the third head of the Y cable to the special cable. Finally, we connect the special cable to the serial ports of the Sun-3 workstation.
Appendix C

Guide for adding a new entity

One of the capabilities provided by the OSI-PM is that entities (protocol layers) can easily be added and modified. A typical entity of the current OSI-PM contains the following supported routines: \texttt{XX\_monitor()}, \texttt{XX\_decode()}, \texttt{XX\_init()}, \texttt{XX\_reset()}, \texttt{XX\_display()} and \texttt{XX\_filter()}. XX is the short form of the entity name, such as DL, PK and TP. To add a new entity to the system, all the above routines of that entity should be implemented. The detailed implementation of these routines except the \texttt{XX\_monitor()} have been given in section 3.4.2.

As described in section 2.3.2, each entity contains a decoder and one (or more) finite state machine(s). The decoder and the FSM(s) of each entity are implemented in \texttt{XX\_decode()} (see section 3.4.2.3), and they follow the corresponding protocol specifications.

As described in section 3.4.2.1 and section 3.4.2.2, the initialization routine \texttt{XX\_init()} of each entity calls the \texttt{setEntryPoint} routine to register the entity to the OSI-PM so that the OSI-PM knows the existence of that entity. \texttt{setEntryPoint} adds a new entity table record to the entity table.

The following sections discuss the implementation of \texttt{XX\_monitor} and the procedure
for adding a new entity to the system.

C.1 XX_monitor(host, len, cid, data)

Some important global variables used in the program are described below. (Note, the word LAYER refers to the name of the layer such as DATA_LINK, NETWORK and TRANSPORT)

- **monRunning**: set to FALSE if a protocol violation is detected.
- **selected[LAYER]**: set to TRUE if the layer named LAYER is selected by the user for display and capture.
- **exist[LAYER]**: set to TRUE if the layer named LAYER is implemented and the user wants to monitor and test it.

`XX_monitor()` is the entry point of entity XX. Parameters passed to this routine include the name of the host which sent the data, the length of the data, the network connection id and the data. This routine does the following:

1. Calls `XX_decode()` to decode the data and test the decoded PDU.

2. Calls `XX_display()` to display the decoded PDU if no protocol violation is detected, i.e. `monRunning` is set to TRUE.

3. If the PDU type of the decoded PDU is DATA and `exist[LAYER]` is set to TRUE (LAYER is the layer which is immediately above XX), it passes data to the entity of layer LAYER by calling `entitiesTable[LAYER].entryPt()` (see section 3.4.1.1 and 3.4.1.3 for more details).
C.2 procedures for adding a new entity

The procedures for adding a new entity are as follows:

1. Implement all the supported routines described above for the new entity.

2. Add the initialization routine $XX\_init()$ to the main routine of the program named $monitor.c$. $XX\_init()$ contains code to register the entity (see section 3.4.2.2).

3. Compile the new entity with the system and test the new entity.