Routing MIDI messages in a shared music studio environment

The thesis submitted to Rhodes University in fulfilment of the requirements for the degree of Master of Science.

by

Thabo Mosala

Computer Science Department,
Rhodes University, Grahamstown 6140.

December, 1994

To my mother, 'Makeneuoe and 'Masemetse

ACKNOWLEDGEMENTS

I would like to thank all those people who had, in one way or another, something to do with this thesis, my fellow students and the staff members of the Computer Science department. A special thanks to Sharlotte Jefferay, Dave Sewry and John Ebden for reading my thesis and assisting in its improvement. A more special thanks to my supervisor, Richard Foss, for the effort he made in seeing this thesis to fruition.

Abstract

The Rhodes Computer Music Network (RHOCMN) is a network which allows main studio resources to be shared. RHOCMN is growing into a multi-workstation environment and additional devices are being incorporated into the system. A star configuration is used for transmitting MIDI from a MIDI patch bay to the workstations and MIDI devices. This imposes several disadvantages on the use of the studio, such as wiring problems. In a quest to avoid problems related to MIDI in RHOCMN, the MIDINet system was developed. The idea was to acquire a viable solution to MIDI's main problems which does not involve a redefinition of the MIDI specification.

CONTENTS

Introduction

MIDI is a standard protocol that permits music devices to communicate and share control information. MIDI has opened up a whole new range of possibilities in electronic music and studio production. It is specialized to handle most direct musical communication, yet its generality has allowed it to adapt to applications that were not foreseen when it was originally drafted. It has been adopted in many related industries such as film editing and lighting control.

MIDI was originally intended for connecting two keyboards in order to play one of them and get the sound of both (just as if you had played both keyboards simultaneously). Nowadays, the most common application for MIDI is the recording of music using a sequencer. A sequencer is a MIDI-based recorder or composition program.

MIDI transmits very simple messages, generated by user action on a musical device. When you press a key on a MIDI keyboard, a message is transmitted, specifying which key was pressed, and with what force you struck it. This is coded into three simple numbers, MIDI bytes. MIDI bytes come in two types called *status* and *data*. Status bytes describe the *kind* of information being sent. They inform the other instruments whether this is a key press, a pitch wheel move, or another type of action. It is the first byte sent by a MIDI instrument when an action occurs. On the other hand, data bytes indicate the actual *value* of the event. For instance, if the status byte indicated a key press, then the following data bytes would indicate which key was pressed, and the velocity with which it was pressed. A status byte is followed by one or more data bytes depending on the type of message being sent.

MIDI offers the capability to remotely control and synchronize devices in a music studio. Remote control allows bulky devices to be placed remotely while keeping control central. The flow of control information between these devices provides them with the ability to synchronize to each other.

In order to share control over these music devices, a music network was created at Rhodes University, the Rhodes Computer Music Network (RHOCMN). RHOCMN comprises a server

and several workstation nodes. Each node enables access to a shared studio. The server and the workstations are connected via Ethernet. The studio comprises several MIDI controllable devices. There is a MIDI patch bay and an Audio patch bay. The network was built with an aim to improve the utilization of music resources. The goal of the network is to communicate the media found within the network to and from each of its users. These media are related to the control and performance aspects of composition.

In RHOCMN, most devices are MIDI controlled. This control information needs to be communicated in real-time. Interconnecting all devices fitted with the MIDI interface are MIDI interconnects. Devices are interconnected through a MIDI patch bay in a star configuration. The MIDI patch bay allows the user to control routing of MIDI control data between devices. The patching of MIDI is required in order that MIDI data transmitted from the user reaches those devices in the main studio to which he has access. It also ensures that MIDI data generated by these devices can be transmitted to the user. There are several problems related to MIDI transmission in the network; one of these is low bandwidth. In this thesis it is argued that the Ethernet network can be used to enhance transmission of the control information.

This thesis discusses how network hardware and protocols can be used to enhance control over shared music studio resources. Local Area Networks (LANs) are used in computer systems and feature high-speed, bidirectional protocols that share and distribute large amounts of information to a variety of devices. However, LANs typically do not distribute this information under real time constraints. What is needed is a protocol that gives the highest priority to messages that need to be communicated in real time. There have been several proposals and attempts to utilise LANs in music environments. Our attempt, the MIDINet concept, follows one of the proposals described later.

The MIDINet concept is introduced in this thesis. The necessary hardware and software for the implementation of the MIDINet concept is described as a means to provide the enhanced control. The goal is to utilise cheap and readily available equipment in a quest to solve MIDI transmission problems in RHOCMN. The hardware that was custom-built at Rhodes University is utilised to achieve this goal. It is also indicated how the MIDINet system fits into a shared music studio environment.

To show how the above mentioned goals were achieved, the thesis begins with a description

of the nature of RHOCMN, including both the hardware and software. The general problems related to MIDI in music studios are discussed in more detail. Following that, problems inherent in RHOCMN are addressed. The MIDINet system is proposed as a solution to overcome these MIDI related problems in RHOCMN. Current systems or networks that have already been developed to solve MIDI related problems are evaluated.

The MIDINet system requirements specification is laid out in chapter 3. The requirements specification describes how devices are identified and how to achieve routing between these devices. It outlines the services provided to the user and how the user interacts with the system. The advantages and disadvantages of having such a system are addressed. There are added advantages provided by the system besides solving MIDI transmission problems.

The process conducted for the implementation of the MIDINet system is described in chapter 4. This includes the choice of an operating system, custom-built hardware and the analysis and design of the system. The power of any operating system lies in the services it provides. PC-DOS is compared with PC-Xinu, the operating system used for the MIDINet system. Following that, the hardware that forms an interface between the system and the MIDI devices is described. After laying out the requirements specification of the MIDINet system, a structured analysis and design of the system is presented.

Before and after the system was built, it was analyzed to measure its performance. This analysis is explained in detail in chapter 5. Performance is a key factor in the design and use of computer systems. The goal of the designer is always to achieve the highest performance at a given cost. Performance evaluation is required at every stage in the life cycle of a computer system, including design, use and upgrade. It is required when the designer wants to compare a number of alternative designs and find the best design. The results determined from the analysis are explained; the analysis includes empirical analysis.

From the analysis it was verified that the non-deterministic nature of Ethernet could cause problems in the system. Ethernet is efficient in transmitting large packets periodically, but its efficiency decreases rapidly when the size of the packet decreases and when their frequency increases. The problems are solved with the help of a protocol implemented above Ethernet. The protocol, Netlink, is described and the performance of the system equipped with this protocol is studied.

The conclusion to the thesis indicates the adequacy of the MIDINet system in its attempt at solving the problems related to MIDI in the Rhodes Computer Music Studio. The description centres around the dysfunctions of MIDI and the problems of MIDI in RHOCMN. Possible improvements to the system are also suggested.

Rhodes Computer Music Network

2.1 Introduction

The Rhodes Computer Music Network (RHOCMN) is a network which allows main studio resources to be shared. These resources are often expensive, and must usually be shared by numerous users. The motivation behind RHOCMN is to share music studio resources in an efficient and cost-effective manner. The argument is that computers and networks can be used to effect this sharing. In order to introduce this network, the fundamentals of a single user studio need to be discussed.

This chapter introduces the terminology relating to a typical computer controlled music studio. The chapter gives a general description of the devices found in a single user studio. It describes the way these devices interact. Following this, the concept of a Computer Music Network is introduced. In this network most devices can be controlled via the MIDI protocol. Most devices today incorporate a MIDI interface. However, problems related to MIDI exist in a computer music network. These problems are addressed and previous attempts to solve these problems are discussed.

2.2 Music Studio

Current music studios are single user studios in which a user has control over all devices in the studio. A music studio has several audio devices. The user can either control the devices through direct physical control or by using a controller. A controller is a device used to generate data to feed other devices. Audio devices can be classified into three categories; audio generators, audio modifiers and audio recorders.

2.2.1 Audio Generators

Audio generators are responsible for generating sound. Typical examples of audio generators are synthesizers and samplers. Synthesizers electronically produce audio information in the form of waveforms. Analogue synthesizers use electronic circuitry to allow for synthesis and shaping of sound timbres. This electronic circuitry includes voltage controlled oscillators,

voltage controlled amplifiers, and voltage controlled filters. Digital synthesizers use synthesis algorithms for the production of digital audio waveforms. These audio waveform producing algorithms can create sounds rich in harmonics [De Furia, 1986].

Samplers offer the facilities for playing back already recorded sampled sound at various frequencies. A digital sampler converts the analogue audio signal into a digital representation using an analogue to digital converter, in order to sample sound. The digital information is then stored sequentially within the sampler's memory. This stored information can be reconverted from the digital domain using a digital to analogue converter.

2.2.2 Audio Modifiers

After the sound has been generated from audio generators it can be passed through audio modifiers to be manipulated. Audio modifiers help the user to alter, mix and listen to sounds. They can modify the sound by compressing or limiting the dynamic range, filtering it, or shifting the pitch. Altering sound is achieved through audio effects units. Typical examples of modifications include equalization, echo and reverb, delay, pitch shifting, compression and expansion. These modifications allow the user to create complex sound effects from the sound the audio generators have produced.

After being modified, the sound is usually mixed by an audio mixer. The mixer combines the outputs of microphones and instruments, and routes these to various destinations. Huber provides a short introduction to mixing [Huber, 1991]. The outputs may be sent to a multitrack tape recorder for recording. Audio amplifiers and speakers allow the user to listen to sound being produced either directly from an audio mixer and/or being replayed from an audio recorder.

2.2.3 Audio Recorders

The sound that has been modified, can be recorded with the help of Audio recorders. This recorded sound can be modified, listened to, and later edited. Associated with the recording medium are tracks. Tracks are physical areas on the recording medium where an audio signal is recorded. Multiple sound generators can be recorded simultaneously onto separate tracks of a multitrack tape recorder, or they can be recorded in succession [Ziffer, 1991].

Audio recorders are either analogue or digital in nature. Analogue recorders are slowly being replaced by their digital counterparts. Analogue audio recorders do not have the sound reproduction capabilities of digital audio recorders [Hurtig, 1992]. Furthermore, digital recorders allow for editing and random access to a recorded information. However, Brighton indicates how analogue recording can be improved through carefully planned setups [Brighton, 1993]. Some analogue decks outperform any commercially released medium including compact disc. A further point made by Hurtig is that some analog-to-digital and digital-to-analog converters suffer from phase shift and quantizing noise, making it preferable to have analogue throughout the recording process [Hurtig, 1992]. In digital recording, the medium can either be a hard disk or a magnetic tape. Hard disk based recording systems allow for the editing of the digital audio [Meyer, 1990], whereas tape-based digital systems are used in a manner similar to the traditional analogue systems.

2.2.4 Audio Patch Bays

Audio signals are transferred between these audio devices through interconnects. The data is transferred in two modes; analogue form or digital form, which is becoming more common. The routing of audio signals can be effected with the aid of an audio patch bay.

Audio patch bays allow the user to control the routing of audio signals between audio generators, modifiers and recorders (Figure 2.1). Audio patch bays enable interconnections between the audio devices to be modified. Using audio patch bays, direct connectivity between any audio devices can be achieved. Hence, audio signals need not be routed through the audio mixer in order to travel from one device to another. The operation of manual audio patch bays is described in [Brighton, 1992].

Audio patch bays originally required the user to make patches physically. Currently, audio patch bays whose functionality can be accessed remotely by controllers are available [Oppenheimer, 1991]. This speeds up access to the patch bay.

The possible interconnection between audio devices is shown in figure 2.1. The audio signals created by the audio generators are transferred to the audio mixer where the overall gain and tonal attributes of these audio signals is first altered. A modified form of these audio signals is then routed to the audio effects units and audio recorders. The audio mixer is also fed with

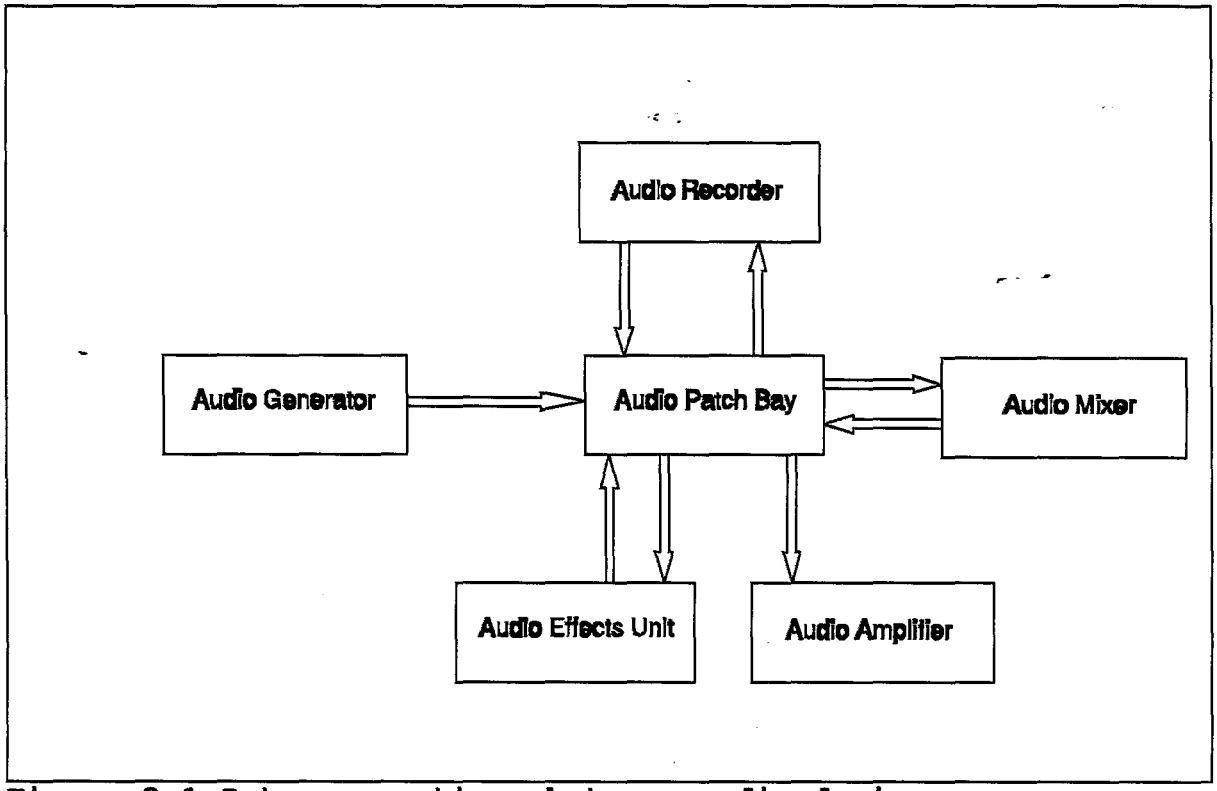


Figure 2.1 Interconnections between audio devices

audio signals returning from audio effects units. Audio signals coming from audio recorders are processed in the same manner as audio signals coming from audio generators.

2.3 Computer Controlled Music Studios

Computers running suitable software introduce the capability to control remotely the devices within the studio. Devices such as audio recorders can be placed remotely while keeping their control central. Audio controllers need to transfer control information to the audio devices under their control. This control information can be communicated via a computer. Changes to the communicated data can be effected within the software run by the computer. In the past, this control over devices was performed via various protocols. Manufacturers of music equipment used proprietary protocols, often above the RS422 or RS232 physical protocols, to effect control over audio devices. These protocols were used to control devices such as multitrack tape decks. Now, MIDI is largely being used to control studio devices. However, some devices such as multitrack tape recorders and mixers still have their own proprietary protocols.

2.3.1 MIDI

MIDI is a standard for interconnecting musical equipment fitted with the necessary electronic hardware [IMA, 1989b]. MIDI provides a way of getting musical devices to talk to one another. It permits devices built by different manufacturers to communicate with one another. Hence, it provides a common ground for communication. MIDI works in the same way as an RS232 modem protocol, but it is optimized for musical data. Modern MIDI record/playback systems can be regarded as a type of multiprocessor network, because an intelligent master (keyboard or computer) controls several devices, each with its own onboard intelligence.

2.3.2 MIDI specification

The MIDI specification comprises both a hardware and a software specification: the hardware consists of a 32 kHz optically isolated serial loop with separate cables for sending and receiving. MIDI uses a five pin plug called a *DIN plug* for connection onto a device. MIDI uses only a single wire to transmit information. The information travels in only one direction over a single MIDI cable. At the sender side, data is converted into serial form by a standard UART (Universal Asynchronous Receiver/Transmitter). A UART at the receiver converts the serial data to parallel form for reception by the MIDI instrument. Each MIDI instrument has three separate connectors: one to send data out (OUT port), one to receive data in (IN port), and another to pass incoming data on through to another MIDI instrument (THRU port).

The software specification provides detailed methods for transmitting note-control data. MIDI messages are divided into several groups. The most common messages are CHANNEL VOICE MESSAGES. Channel voice messages include NOTE-ON and NOTE-OFF, which tell the receiver what note to play, REAL-TIME messages for coordinating events, and CONTROLLER MESSAGES which control the response characteristics of the receiving device. SYSTEM EXCLUSIVE MESSAGES form a looser specification for handling manufacturer-specific interaction between products. A more detailed description of MIDI messages is given in section 4.5.2.4. A useful description of MIDI can be found in [Wilkinson, 1993].

Associated with each channel voice message is a channel number. Channel numbers make it possible for sounds to be layered on top of one another. Different musical parts, each meant to be played on different instruments, can be transmitted together over the same MIDI cable.

Each part has a channel number associated with it and will be played by the instrument set to this channel. In a setup where multiple instruments are connected to a single controller, the controller could select particular synthesizers to activate, by sending channel voice messages with the appropriate channel numbers.

The MIDI standard has been extended to allow for the transmission of timing information over MIDI lines. This extension is known as MIDI Time Code [IMA, 1986] [IMA, 1989b]. It enables devices to synchronize to one another. Synchronization is the process of ensuring that two time-based devices stay in time with one another. Typically, the requirement is to synchronize MIDI sequencers and audio recorders.

2.3.3 MIDI Applications

The original motivation behind MIDI was to allow a performance on one instrument to be played by any other instrument. This capability was extended to the recording of performance information. It has evolved into a communication pipeline that can link all the equipment used in music production. It has found acceptance in many related industries such as lighting control, film editing, and automated audio mixing. It has opened new dimensions for recording, composition, and live performance, because of supporting real-time access to interconnected devices.

2.3.4 Sequencer

MIDI-based recorders or composition programs are known as sequencers. With the aid of sequencers, computers have the ability to intercept and record musical events in the form of MIDI messages. The recorded data can be manipulated in ways that were impossible using standard audio tape recorders. Transposition (pitch shift), copying, and rearranging can be accomplished with software alone. The errors made during the recording process can be corrected without having to do the recording again.

A sequencer records data digitally. It allows MIDI data generated by a controller to be recorded onto separate tracks, analogous to an audio recorder. Once recorded, the MIDI tracks can be 'played back' by the sequencer. The sequencer transmits the recorded MIDI performance data to one or more synthesizers. The synthesizers respond to the MIDI messages as they would in a controller-synthesizer interaction.

In the modern context, a sequencer might be visualized as being much like an audio tape recorder. Facilities such as fast-forward, rewind and tracking can be effected from within the sequencer. Some sequencers are built into keyboard instruments while others come in the form of a separate piece of hardware. Hardware sequencers are dedicated devices that record and play back performance-related MIDI messages. The major difference between an audio tape recorder and a sequencer is that the tape records the sounds of a performance, a sequencer records the events in a performance. A MIDI sequencer not only permits the editing of performance data but it also allows the manipulation of the entire compositions.

2.3.5 MIDI patch bays

It is necessary to mention MIDI patch bays since this work is centred around developing a system that adapts the functionality of a MIDI patch bay. Interconnecting all devices fitted with the MIDI interface are MIDI interconnects. These interconnects allow for the routing of MIDI control data between the devices. These interconnects can be directed to a MIDI patch bay.

MIDI patch bays allow the user to control the routing of MIDI control data between devices. Similar to audio patch bays, MIDI patch bays originally required the user to make the patch physically. MIDI patch bays whose functionality can be accessed by using MIDI are now available [Oppenheimer, 1991].

MIDI patch bays offer features such as MIDI merging, filtering, transformations and transposing. They also can be used as MIDI thru boxes. MIDI merging is a process by which the messages on two or more MIDI links are merged into one. MIDI transformation is the process by which MIDI messages passing through the patch bay are altered in a manner specified by the user. MIDI filtering allows for the removal of MIDI events from a MIDI link. MIDI transposing permits the pitch of MIDI note information to be altered.

2.3.6 Problems with MIDI

There are problems associated with MIDI. Firstly, MIDI is slow, it does not have the bandwidth to handle large complex setups and MIDI cables are restricted to 30 metres in length. However, the problem of bandwidth may be caused by a poor implementation within a particular device [Buxton, 1987]. It may also arise when there are several devices being

controlled by a computer or a sequencer. This could be alleviated by having multiple ports on the sequencer. The amount of data to be communicated to the devices may saturate the channel capacity of MIDI. This problem is further discussed by Dimuzio in his article on real-time control of several devices [Dimuzio, 1990]. The speed problem can occur because of propagation delay, where several devices are 'daisy chained' together. Daisy chaining refers to several devices being connected to one another using the MIDI THRU feature.

Secondly, MIDI ports are unidirectional. If a device has to send and receive MIDI data, then it must have two MIDI cables connected to it: one incoming, another outgoing. The hardware communication links lead to a cluttered hardware configuration.

One of the motivations for the MIDINet system discussed in this thesis is to solve the above mentioned problems.

2.4 The Rhodes Computer Music Network (RHOCMN)

As mentioned in the introduction, the motivation behind RHOCMN is to effect sharing of music studio devices using computers and networks. A network has been constructed to allow the sharing of devices in a studio. The network provides several workstation nodes. Each node enables access to a shared studio, with a view to improving resource utilization [Foss, 1990]. The objective of the music network is to communicate several media found within a music studio to the nodes. The media include MIDI, SMPTE and audio. The information that is sent across relate to both control and performance aspects of composition. A real time response is required in communicating most of the media.

The nodes allow the booking of resources such as tape decks and synthesizers. A user is also able to do MIDI patching, and audio patching and mixing of the inputs and outputs from these resources. Mixing can be automated. A MIDI-based sequencer can be used at the workstation. Thus, users can share the resources of a common studio. The workstation node provides mixing and channel selection capabilities, which normally have to be done "hands on" in the studio. The workstation node allows for two-way MIDI communication between any booked music resources.

2.4.1 Hardware layout

The hardware configuration of the initial network is shown in Figure 2.2. Ethernet connects the workstations and the server computers. Booking, patching and tape control information is transmitted over Ethernet from the workstations to the server. The server has direct control over the audio and MIDI patch bays, and over the multitrack tape. Both the audio and MIDI patch bays were custom-built for the network project.

2.4.1 Audio patch bay

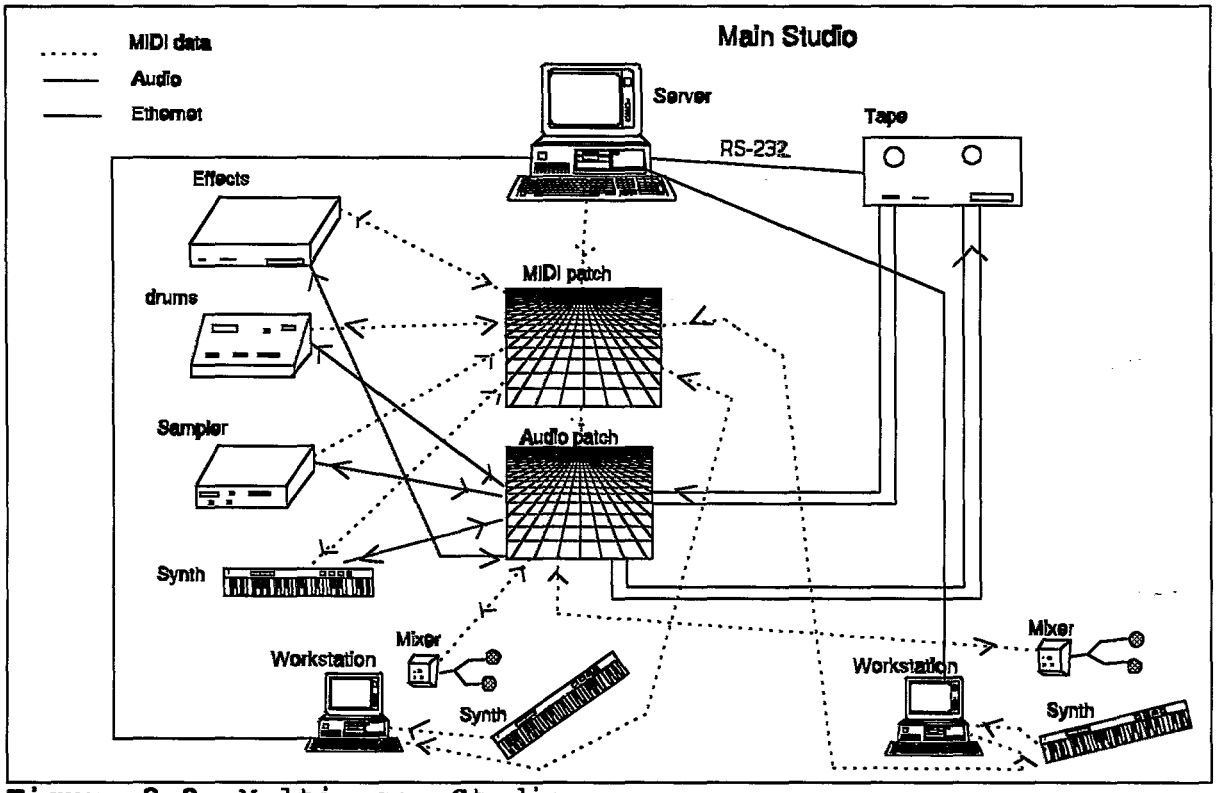


Figure 2.2 Multi-user Studio

The audio patch bay allows the interconnections between the audio devices. It also allows for mixing of signals at patch nodes and volume control at these nodes. It enables the user to control the routing of audio signals between audio generators, modifiers and recorders. An audio patch bay has 16 inputs and 16 outputs. It can be extended to increase the number of In and Outs. It has a pool of nodes assigned to patch points by an embedded processor. The audio patch bay is MIDI controllable, patching being performed by the server.

A star configuration is used for transmitting audio from the audio patch bay to the

workstations. Two lines transmit audio from the audio patch bay to a workstation and from the workstation to a patch bay. Further information on this patch bay can be found in [Wilks, 1994].

2.4.2 MIDI patch bay

The MIDI patch bay is built as a matrix of patch points. Patching within the MIDI patch bay is performed by an on-board processor. The MIDI patch bay allows the user to control the routing of MIDI control data between devices and workstations. It can also be extended to provide more MIDI Ins and Outs.

A star configuration is used for transmitting MIDI from the MIDI patch bay to the workstations and MIDI devices. Two lines transmit MIDI from the MIDI patch bay to a workstation and from the workstation to the patch bay. Further information can be found in [Wilks, 1994].

2.4.3 Workstation MIDI Sequencer and MIDI controller

The workstation can be configured to have the MIDI sequencer software on the same computer as the workstation network software. Alternatively, the MIDI sequencer can be independent of the workstation network computer. This permits any commercial MIDI sequencer to be used at the workstation. This is an advantage to the users as they can use sequencers of their own choice.

Each workstation has its own MIDI controller which allows the user to record MIDI data using the MIDI sequencer in the normal manner. A MIDI controller, within a MIDI based computer music studio, is a device capable of generating MIDI events for the control of devices fitted with the MIDI interface. The MIDI controller may combine a synthesizer to provide the user easy access to voices. These voices can be used for the initial development of compositions. The recorded MIDI data can then be broadcast to all booked devices.

2.5 Problems with MIDI in RHOCMN

RHOCMN is growing into a multi-workstation environment and additional devices are being incorporated into the system. When several devices are being controlled by a computer or a

sequencer, a problem of bandwidth may arise. This is exactly the same problem discussed for the single user studio situation. In the case of the network, only one MIDI link exists to carry data for the control of all the devices booked by a user. The amount of data to be communicated to these devices may saturate the channel capacity of MIDI.

A star configuration is used for transmitting MIDI from the MIDI patch bay to the workstations and MIDI devices. Two lines transmit MIDI from the MIDI patch bay to a workstation and from the workstation to the patch bay. The hardware configuration becomes complex.

MIDI cables are restricted to fifteen metres in length. This imposes constraints on remote access to the studio. Workstations must be within 15 metres of the main studio. Some manufacturers have provided RS422 or fibre optic links which extend the carrying length of MIDI [Lone Wolf, 1989]. They are typically expensive solutions and do not solve the configuration problem.

2.6 Attempts to solve MIDI problems

In an attempt to solve the above mentioned problems, the concept of a MIDINet unit was introduced. A MIDINet unit is an IBM PC with custom-built MIDI interface cards. The MIDINet units are connected via Ethernet. They have In and Out ports to which MIDI devices connect. A MIDINet unit provides an interface between MIDI devices and the network. The MIDI interface is effectively distributed over the network. Messages from any source device(s) on any unit are routed to any destination device(s) on any unit. These routings between devices can be set up via software interfaces on the MIDINet units, or by sending routing messages over the Ethernet according to the MIDINet protocol.

The concept of transmitting MIDI over a network is not new. Buxton in his paper introduces the concept of a *MIDI server* [Buxton, 1987]. The MIDI server is a node on a LAN which serves one or more MIDI devices. The MIDI server is able to communicate with the LAN, and it can also communicate with the MIDI world. The introduction of MIDI servers into a LAN permits the communication of MIDI data across a network.

The problem with Buxton's proposal is that most LANs today have communication protocols that do not support the real time communication of data. However, one company, Lone Wolf,

devised a protocol (The Medialink Protocol) that gives the highest priority to messages which need to be communicated in real time [Westfall, 1989]. The MediaLink protocol is designed for high speed fiber optic networks optimized for real-time applications such as MIDI and SMPTE (Society of Motion Picture and Television Engineers). Analogous to Buxton's MIDI server, medialink uses the MIDITap unit to interface with MIDI devices. It is designed to interconnect large arrays of electronic musical instruments and other devices. The MIDITap is a single rack unit featuring four pairs of MIDI In and Out ports, a pair of fiber optic ports and an RS232/422 computer port [Lone Wolf, 1990] [IMA, 1989a]. MIDI messages can be routed from any MIDI in, on any MIDITap unit, to any MIDI out, on any MIDITap unit. A MIDITap unit provides a bridge between MIDI and MediaLink protocol by accepting MIDI messages sent to any In port in the system, converting them to MediaLink messages.

Unfortunately, MIDITap units could not be used in RHOCMN. This was due to the lack of IBM PC cards which implement the Medialink protocol, and meant that server control over the MIDITap units could not be achieved. This inspired the idea to develop MIDINet units whose control could be effected from the server in RHOCMN.

ArcoNet (Artist's Computer Network) is another proposal for a LAN which provides real time communication of MIDI data [Allik, 1990]. The motivation behind ArcoNet was to provide a standard network for real-time communication between devices used in the performing and recording arts. It was not supposed to be restricted to musical equipment alone. The network requires specialized hardware that forms an interface between the devices and the network medium. There has been no implementation of ArcoNet.

ZIPI is a recent network protocol that was proposed for electronic musical devices [McMillen, 1994]. Its network specification defines a collection of protocols that allow musical instruments, computers and other similar devices to communicate. The architecture provides for a deterministic, low-cost, efficient and low latency network. ZIPI contains several application layer protocols such as the Music Parameter Description Language (MPDL) for musical control and the MIDI protocol. MPDL forms an interface between the controller and other devices. MPDL allows for editing of the control messages. There is a possibility of defining other application layer protocols such as studio automation, lighting and other media. Once again, there is no available implementation of ZIPI which would allow its use in the music network.

Fober has also recently proposed a way to transmit real-time musical data flow on Ethernet and described an Apple-Mac based implementation of it [Fober, 1994]. The implementation is based on the software architecture of MIDIShare. MIDIShare is a real-time multi-tasking MIDI operating system. The motivation behind Fober's proposal is to extend the communication capabilities of musical applications, allowing them to share remote resources (hardware and software). This is achieved through overcoming the non-deterministic nature of Ethernet and improving the efficiency of Ethernet when small size packets are transmitted frequently. The implementation is similar to the MIDINet system in several aspects. The MIDINet system also utilises Ethernet, and implements a protocol above it to overcome its non-deterministic nature. Analogous to MIDIShare, the MIDINet system employs the PC-Xinu multi-tasking operating system. PC-Xinu is modified to include MIDI drivers (refer to section 4.3).

The remainder of this thesis describes the concept and development of a MIDINet unit. Chapter 3 describes the MIDINet system requirements specification and the facilities it provides the user. Chapter 4 gives a detailed description of how the MIDINet system was implemented. Chapter 5 depicts the analysis that was followed to study the performance of the MIDINet system, before and after it was implemented. Chapter 6 describes the Netlink protocol, designed to overcome the nondeterministic nature of Ethernet.

The MIDINet Concept

3.1 Introduction

In the previous chapter, problems related to MIDI in RHOCMN were discussed. It was shown that RHOCMN is growing into a multi-workstation environment and additional devices are being incorporated into the system. There is a single MIDI cable which carries MIDI to the main studio. As a result there are only 16 channels available. The problem of limited bandwidth may arise when several devices are being controlled by a computer. There may be large amounts of data to be communicated that would saturate the capacity of MIDI cables. It was also shown that a star configuration is used for transmitting MIDI from the MIDI patch bay to the workstations and MIDI devices. Each workstation is connected to the MIDI patch bay through two MIDI cables. Consequently, the hardware configuration is complex. A further complicating factor is that MIDI cables are restricted to fifteen metres in length. Remote access to the studio is restricted.

In this chapter the MIDINet concept is described. The goal of a MIDINet system is to overcome the MIDI problems in RHOCMN. The MIDINet system can also be used in a typical single user studio. It allows the user to control the routing of MIDI control data between devices. Features such as MIDI merging, filtering, transformations and transposing can be easily effected within the system. A MIDINet unit can also be used as a MIDI THRU box, passing identical data to multiple devices.

The MIDINet system allows the user to look at the collection of his MIDI devices as a single, unified system. It comprises several units connected via Ethernet.

As was indicated in the previous chapter, there have been several attempts to solve MIDI routing and transmission problems. Some of these attempts are expensive to implement. One of the advantages of the MIDINet approach is that standard off-the-shelf components are used to effect transmission and routing of MIDI.

3.2 What is a MIDINet unit?

A MIDINet unit is an IBM PC configured with two proprietary MIDI cards. Each card has two **Ins** and two **Outs**. A MIDINet unit can be configured with more than two cards. Section 5.4.3 provides a more complete analysis of this problem with guidelines regarding how many cards a unit should be configured with.

Any MIDI-based resources can have their MIDI Ins and Outs connected to MIDINet units as shown in Figure 3.1. A MIDINet unit accepts data from a source device and performs routings. Any device connected to a MIDINet unit can transmit MIDI data to any other device connected to a MIDINet unit.

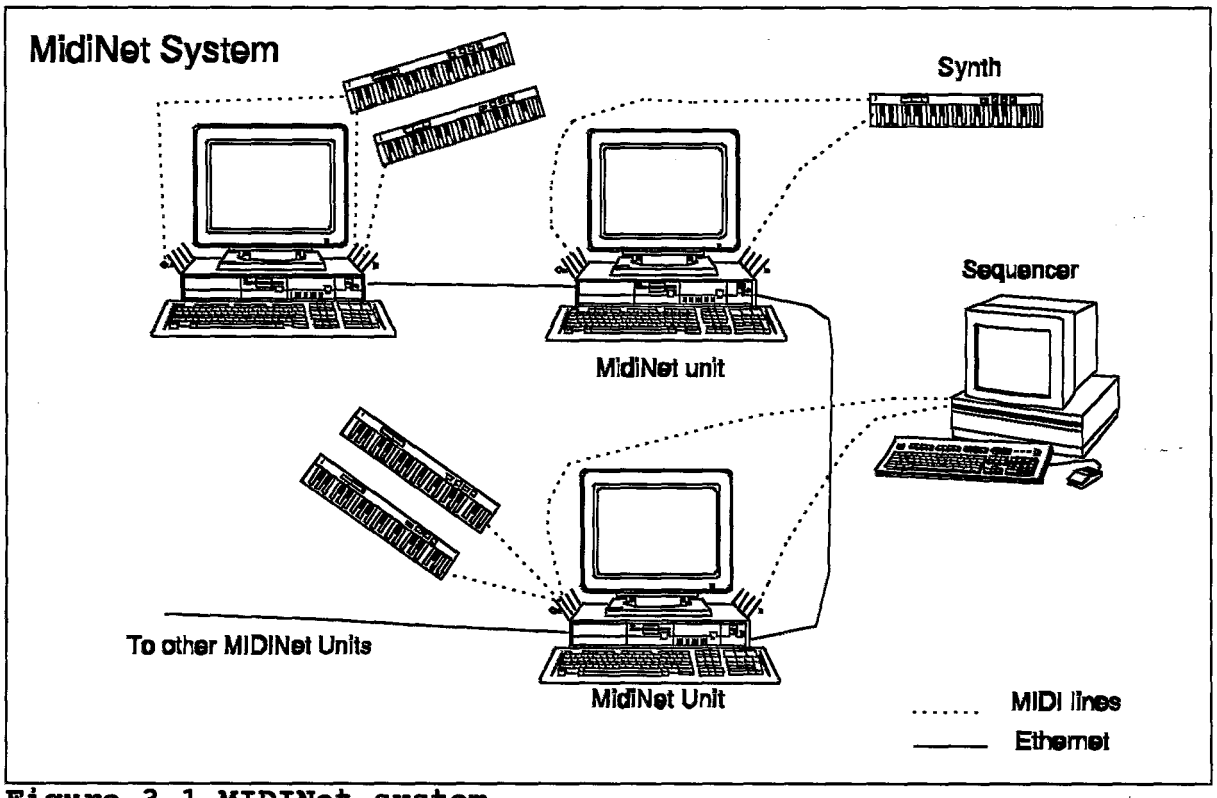


Figure 3.1 MIDINet system

The MIDINet units are connected to each other via Ethernet thus forming a network of units. The MIDINet unit software runs under a modified version of the multitasking Xinu operating system [Comer, 1888b] [Watkins, 1990]. Like any other operating system, PC-Xinu's essence lies in services it provides to user programs. An important feature of PC-Xinu for this project

is its multitasking capabilities. A user can create processes that could run concurrently. This feature is lacking in DOS and was the major motivating factor in choosing PC-Xinu. The PC-Xinu operating system was enhanced with a text-based window manager and will run on IBM PC AT's in 100k of memory [Rehmet, 1992].

Figure 3.2 shows how MIDINet units can be incorporated into RHOCMN. One or more MIDINet units provide the functionality of a MIDI patch bay. All MIDI devices are connected to MIDINet units. Each device is accessible from any workstation. Routing performed through a patch bay can now be done via MIDINet units. Patching capabilities are effected within the MIDINet units. Any device on any MIDINet unit can be 'patched' to any device on the same or another MIDINet unit. Devices can route MIDI data down the Ethernet through MIDINet units.

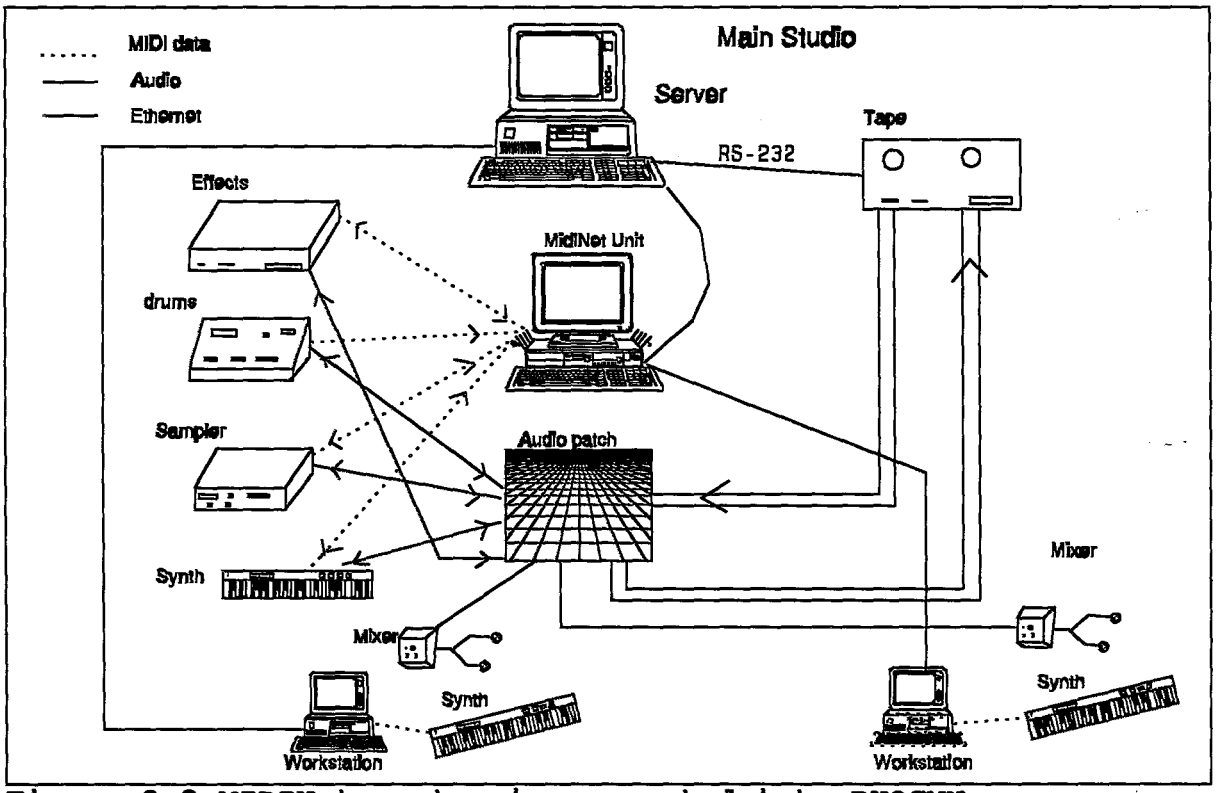


Figure 3.2 MIDINet system incorporated into RHOCMN

3.3 Configuring a MIDINet system.

Each MIDI device is connected to an In or an Out port on a MIDINet unit. These devices need to be identified to the system and configured. The MIDINet system is menu driven and

this identification can be done via a menu. The identification includes a MIDINet ID, a Port ID, a Channel number, and a symbolic name.

The MIDINet ID is a number between 1 and 10 inclusive. There had to be a limit as there is a limit to the number of MIDINet units that can be connected together. The limit is determined from the simulation results. The results show that as more devices are added to the network, the throughput declines and the response time increases (discussed in more detail in chapter 5). The MIDINet ID uniquely identifies a particular MIDINet unit in the whole system. The system has no way of checking if two or more units have the same MIDINet IDs. It is the responsibility of the user who is carrying out the configuration process to maintain uniqueness in identifying the units. Another option was to use network physical addresses as a form of identification, as these addresses are unique in the case of Ethernet. However, it is easier to store and manipulate numbers rather than strings of digits. Hence, numbered IDs were preferred over physical addresses. More importantly, this makes the MIDINet system portable to other network topologies such as token ring.

The Port ID is a number between 1 and 4 inclusive. Each MIDINet unit has 4 Ins and 4 Outs. The manner in which In and Out ports are identified is similar. Both types are identified by numbers in the range of 1 to 4. Differentiation is maintained at a user level. From the menu, a user is able to configure a destination device (any device connected to an Out port) or a source device (any device connected to an In port).

The symbolic name is a string of not more than 10 characters. This is a name given to a device by the user who configures it. Symbolic names are assigned by users. Different users may choose to assign different names for the same device.

A channel number is a number between 1 and 16. Channels are used in MIDI to select devices or device parts. The channel number entered when configuring a device must be the same as the receive channel number set on the physical device being configured. Otherwise, this device will not respond to messages sent to it.

A combination of a MIDINet ID, Port ID, and a channel number form a unique identification for a particular device within the system. In the case of multi-part receivers connected to an Out port, each part may be assigned a different channel number. MIDINet IDs and Port IDs for these parts will be the same. The same concept applies to sequencers connected to In

ports, different sequencer output channels are assigned symbolic names. MIDINet IDs and Port IDs will be the same for each output channel. Before a MIDI message is transmitted to an Out port, the channel number is mapped to the channel number set on the receiving device.

M-Id	P-Id	S-Name	C#
#1	#1	Dx 7	6
#1	#2	Casio	2

M-Id : MIDINet Identification number
P-Id : Port identification number
S-Name : Device's symbolic name
C# : Channel number

Figure 3.3 Configuration information for a device

3.3.1 Configuration

The user must configure every device in the entire MIDINet system. The user captures particular information for every device that is connected to any port on any MIDINet unit. The information includes the MIDINet unit ID, Port ID and the device's symbolic name. The device being configured need not be local to the unit being used for configuration (see Figure 3.3).

Four types of devices exist, namely, multi-channel transmitters, single channel transmitters, single-channel receivers and multi-channel receivers. For multi-channel transmitters, which are normally sequencers, channel numbers are assigned symbolic names and are treated as independent devices within the system (fig 3.4). The information related to these devices is sent to other units to update their tables or lists.

Multi channel receivers usually have 'parts' associated with their channels. These parts are

M-Id	P-Id	S-Name	C#
#1	#2	Voyetra 1	1
#1	#2	Voyetra 2	2

M-Id : MIDINet Identification number
P-Id : Port identification number
S-Name : Device's symbolic name
C# : Channel number

Figure 3.4 Configuration Information for a multi-channel transmitter

M-Id	P-Id	S-Name	C#
#1	#2	D110 Part 1	1
#1	#2	D110 Part 2	2

M-Id : MIDINet Identification number
P-Id : Port identification number
S-Name : Device's symbolic name
C# : Channel number

Figure 3.5 Configuration Information for a multi-channel receiver

assigned symbolic names and are treated as independent devices within the system (See Figure 3.5). The MIDINet IDs and Port IDs of the various parts will be the same, since a multi-part device only has a single MIDI in and single MIDI out.

Configuration for single channel transmitters and receivers is the same. They are assigned a

MIDINet ID, Port ID, symbolic name and the channel number each device is set to transmit or receive on. It should be pointed out that sequencers can transmit to multiple ports. A sequencer can be connected to more than one port on a MIDINet system. These ports need not be on the same MIDINet unit.

3.3.2 Connection of devices

After configuring all the devices, a user can establish connections between source and destination devices. A user pulls down a list of source devices, selects a source device, then

Device List

M-Id	P-Id	S-Name	C#
#1	#2	D110 Part 1	1
#1	#3	DX 7	2
#3	#4	Voyetra 3	2
.	.	.	.
.	.	.	.

M-Id : MIDINet Identification number
P-Id : Port identification number
S-Name : Device's symbolic name
C# : Channel number

Figure 3.6 Source/Destination device list

pulls down a list of destination devices and selects a destination device. A typical pull down menu of source devices is shown in figure 3.6. The lists for source and destination devices are similar. Figure 3.7 shows a list of connections.

A connection is a path for MIDI messages between a source(s) and destination(s) devices. The source and the destination devices can be on different MIDINet units. A user can establish new connections or break already existing connections. The user can choose to connect several sources to a single destination (merging) and connect a single source to several destinations.

Connection Table at application level

Source Devices					Destination Devices			
M-ID	**P-ID**	**S-Name**	**C#**		**M-Id**	**P-Id**	**S-Name**	**C#**
#2	#2	Voyetra 1	2	-	#1	#2	Casio	1
#2	#2	Voyetra 2	3	-	#1	#3	DX 110	2

M-Id : MIDINet Identification number
P-Id : Port identification number
S-Name : Device's symbolic name
C# : Channel number

Figure 3.7 Connection Table

3.4 How the MIDINet system operates

At startup, a MIDINet unit (*client*) broadcasts a startup request to other units in the MIDINet system and then waits. If there is no response, it tries three times. If there is still no response after three attempts, then the unit assumes that it is the only unit running. On the other hand, any running unit (*server*) will respond to the request and will send an acknowledge message. At this stage, a communication link has been established between a *client* and a *server*. The *client* will consider the first response and ignore others. The *client* will send a message to the *server* to be loaded with configuration information. This message is sent specifically to the *server*. The *server* will respond by loading any configuration information. If there is no configuration information, that is, no devices have been configured and no connections have been established, then the *server* will ignore the request and the *client* will time out. The user can then configure devices and establish connections on this new unit.

MIDI bytes entering the In port of any unit are parsed. There are two types of MIDI bytes called *status*, and *data* bytes. A status byte describes the nature of the MIDI messages being sent. This is the first byte sent by a MIDI instrument when an action occurs. Data bytes

follow the status bytes and indicate the actual *values* of the event. A status byte is followed by one or more data bytes depending on the type of information being sent. For a more in-depth description of MIDI bytes refer to [Rona, 1987].

The MIDI parser forms messages from these bytes. MIDI messages can be grouped into several groups and subgroups:

1) MIDI channel messages.
    a) MIDI channel voice messages.
    b) MIDI channel mode messages.
2) MIDI system messages.
    a) MIDI system common messages.
    b) MIDI system real time messages.
    c) MIDI exclusive messages.

MIDI channel Messages are the most commonly used messages in the communication standard. The status byte in all MIDI channel messages contains the MIDI channel number. It is this channel number that is extracted from the MIDI message for identifying its source.

The destination for a message is determined from the connections table. The message is identified by the MIDINet ID, port ID and channel number on which the device is transmitting. If the destination device is local, then the message is sent through an appropriate out port. If the destination device(s) is not local, then the message is packed into a packet and broadcast to other MIDINet units. The message is sent with the particulars of the source device: MIDINet ID, Port ID and Channel Number.

When a MIDI message is received by a unit from the network, again the unit determines its destination from the connection table. If the message is destined to a non-local device, then the unit discards the message. This is possible since all units receive the same message.

If a unit determines that the message is destined for one of its local devices, then the message has to go through some processing; the channel number on the MIDI message is mapped to the channel number configured on the destination device. This is the stage where modifications such as filtering and transposition can be effected.

3.5 MIDINet unit in relation to other systems

In order to configure devices connected to MIDINet units and establish connections between these devices, a non-MIDINet unit may be used. A non-MIDINet unit has to send a message that conforms to the MIDINet protocol. This feature enables the MIDINet system to be incorporated into RHOCMN. A user could establish and break connections from a workstation if it had suitable software. However, any foreign device in a MIDINet system must conform completely to the MIDINet protocol. It must update the configuration and connection information correctly.

Permitting foreign devices to update configuration and connection information overcomes the problem with Lone Wolf MIDITap discussed in the previous chapter (section 2.6). Any device configured with an Ethernet card can communicate with the MIDINet system, if it adheres to the MIDINet protocol.

3.6 The advantages of the MIDINet system to the Rhodes Network

One of the reasons why LANs are becoming increasingly popular is that they permit personal workstations to communicate. They permit users to share expensive resources that are typically attached to dedicated processors on a network. The MIDINet system also offers the ability to share MIDI devices among users.

MIDI devices are connected to the network through MIDINet units. The MIDINet unit can replace the conventional computer-MIDI interface. The workstation can be set up to use the LAN interface for transmitting MIDI data. Under this architecture, the MIDI interface is distributed over the network.

3.6.1 MIDI patch bays

In the main studio, a star configuration is currently used for transmitting MIDI from the MIDI patch bay to the workstations and MIDI devices. As previously mentioned, two lines transmit MIDI from the MIDI patch bay to a workstation and from the workstation to the patch bay. The hardware communications links lead to a complex hardware configuration.

The MIDINet unit provides the functionality of the MIDI patch bay as shown in figure 3.2.

Each device is independently connected to a unit. The star configuration is dissolved, which leads to reduced wiring complexity. Routing is performed via the network. If more devices are required, then additional units are incorporated into the system.

Ethernet LANs can extend to 2.5 km. The enhanced system permits users outside the studio to have more distant access to MIDI devices in the main studio.

3.6.2 Channelisation

Most MIDI synthesizers provide a way of selecting the channel numbers on which they will send and receive MIDI data. In RHOCMN, the server sends system exclusive messages to MIDI synthesizers to select the channel number on which they should send and receive MIDI data. This means that manufacturer specific information must be kept in the system. When a new MIDI device is installed in the studio, the server has to be loaded with the device's system exclusive message type for changing channels.

Each device on any MIDINet unit is uniquely identified. It is immaterial whether there is more than one device using the same channel number. Any MIDI message transmitted bears the source device's identification. Hence, the message is routed to connected destination device(s) only, not broadcast as it is in the original system. Channel mapping occurs at the destination. The server is not required to send system exclusive messages to select channel numbers.

3.6.3 Other advantages

In a typical single user studio, the MIDINet system offers a MIDI thru facility, MIDI merging and channelization. Merging is achieved by creating a connection between more than one source device and a single destination. The channelization feature has been described in section 3.6.2 above.

In conclusion, a MIDINet LAN is a high-speed network capable of distributing MIDI information in a complex system. It offers the advantages of bidirectionality and greater bandwidth. A potential problem is the non-deterministic nature of Ethernet. This will be explored in the following chapters.

Implementation of the MIDINet System

4.1 Introduction

This chapter provides details of how the MIDINet system was implemented. In modelling the MIDINet system, the Ward and Mellor Design tools were adopted [Ward, 1988a,b,c]. The Ward and Mellor Design tools are real-time systems design tools. They model the complex event capturing and device control required in a real-time system.

PC-Xinu, the operating system under which MIDINet system was implemented is introduced. PC-Xinu is a small operating system which supports preemptive multitasking. User processes can be prioritized. It supports a wide selection of interprocess communication facilities. It runs on IBM PC or compatible machines and resides on top of MS-DOS.

Serial cards were adapted to provide MIDI interfaces to the MIDINet units (MUART - MIDI/Universal Asynchronous Reciever/Transmitter). These cards were built for the purpose of this project. The cards are built using the INS8250 chip. The INS8250 is a general purpose Universal Asynchronous Receiver/Transmitter. These cards are another factor contributing to the cost effectiveness of the MIDINet units. PC-Xinu was modified to add drivers for the MUART and Ethernet devices.

4.2 Hardware Configuration of a MIDINet system

Each MIDINet unit is an IBM PC configured with two proprietary MIDI cards. Each card has two **Ins** and two **Outs**. Any MIDI-based resources have their MIDI Ins and Outs connected to the MIDINet units. The MIDINet units are connected to each other via Ethernet, to form a network.

4.2.1 The MIDI card (MUART)

As already mentioned, the MUART cards are modified serial cards. The idea was to build non-intelligent and cheap cards. Each card contains two INS8250 chips. The INS8250 is a

general purpose serial I/O chip. The chip is manufactured using NMOS technology and packaged as a 40-pin dual in-line package (DIP) and uses a single +5V power supply. All input and output signals are TTL-compatible [Kane, 1978]. Each chip has a receive and transmit pin. In order to avoid having several cards plugged into a PC, these cards were built in such a way that each would provide two Ins and two Outs, hence, the need for two chips on one card.

4.2.1.1 Interrupts on the PC

Data transfer between the CPU and peripheral devices is accomplished in various ways. Frequently, both software and hardware interrupt techniques are involved. In the IBM PC, an *interrupt controller* chip manages interrupts [Holt, 1985]. This chip receives interrupt requests from peripherals, examines their priorities, and sends an interrupt request signal to the processor. When an interrupt acknowledgement is received from the CPU, the controller transmits a code to the processor via the data bus. This code is used to provide a vector address mapping to a routine in memory. The routine is designed to perform the service required by the device. During initialization, the address of an interrupt service routine should be installed in the vector, so that it can be called when an interrupt occurs.

4.2.1.2 INS8250 Addressable locations

The INS8250 has only a few control signals associated with its serial data input and output. It makes extensive use of internal registers. Most serial I/O devices commonly output a control signal to identify parallel data being transferred from the Transmit buffer to the Transmit register. Another control signal normally identifies data being transferred from the Receive register to the Receive buffer. The INS8250 uses a different mechanism. It uses Status register bit settings to indicate these operations.

The INS8250 has ten addressable locations. Seven of them are read/write locations. The receive buffer and the Interrupt ID register are read only locations, while the Transmit buffer is a write-only location. The device is accessed as eight I/O ports or eight memory locations. The Address Bus can be used in many ways to select INS8250 locations. Read and write control signals on the CPU interface to the INS8250 have been designed to make this device interface easily with any microprocessor. Figure 4.1 shows the ten addressable locations of an INS8250 chip.

```
#define MIDI_BASE_PORT_1      0x3E8

#define TX_DATA_1       (MIDI_BASE_PORT_1 + 0x00) /* Transmit data */
#define RX_DATA_1       (MIDI_BASE_PORT_1 + 0x00) /* Receive data */
#define DIV_LSB_1       (MIDI_BASE_PORT_1 + 0x00) /* Baud Rate Divisor lsb */
#define DIV_MSB_1       (MIDI_BASE_PORT_1 + 0x01) /* Baud Rate Divisor msb */
#define INT_ENABLE_1    (MIDI_BASE_PORT_1 + 0x01) /* Interrupt Enable */
#define INT_IDENT_1     (MIDI_BASE_PORT_1 + 0x02) /* Interrupt ID */
#define LINE_CTL_1      (MIDI_BASE_PORT_1 + 0x03) /* Line Control */
#define MODEM_CTL_1     (MIDI_BASE_PORT_1 + 0x04) /* Modem Control */
#define LINE_STATUS_1   (MIDI_BASE_PORT_1 + 0x05) /* Line Status */
#define MODEM_STATUS_1 (MIDI_BASE_PORT_1 + 0x06) /* Modem Status */
```

Figure 4.1 INS8250 registers and addresses

The Transmit and Receive buffers share a single address. The Transmit buffer is a write-only location and the Receive buffer is a read-only location; therefore, conflicts do not arise. In order to identify Transmit and receive operations, a control code has to be written into the Line Control register.

4.2.1.3 Operations

The INS8250 can be used for transmitting only, receiving only, transmitting and receiving alternatively, or transmitting and receiving in parallel. Before any transmit or receive operation, Interrupt Enable, Line Control, Modem Control, and Divisor registers should be set. The INS8250 has no enable and disable logic for serial data transfer.

The Modem handshaking protocol that may be associated with any data transfer must be handled under program control by sending appropriate data to the Modem Control register, and by reading back data from Modem Status register. Once the required protocol exchange has been established, the program monitoring the INS8250 can attend to other affairs. Separate routines will handle serial data output or serial data input, reading from the Line Status register and the Modem Status register to determine conditions within the INS8250 - and therefore the required response.

If interrupt logic is not used, then the program which monitors the INS8250 must poll the device by reading the contents of the interrupt ID register. If bit 0 of this register is 0, then further service is required; if bit 0 is 1, then no service is required.

4.3 PC-Xinu

PC-Xinu is a multitasking operating system developed initially by Douglas Comer and modified for the IBM PC by Tim Fossum [Comer, 1988]. It is small, and yet relatively complex. It has a mechanism of prioritized processes and has a wide selection of interprocess communication facilities such as semaphores and message passing. Its kernel consists of a set of routines to implement operating system services. A user program can bind to these routines to perform various tasks.

PC-Xinu programs can access the MS-DOS file system, making it easy to exchange data between ordinary MS-DOS programs and PC-XINU programs. New device drivers can easily be added to the operating system for new devices. These facts make PC-Xinu an attractive option for the writing of real-time software programs on the cheap and popular IBM PC computers.

PC-DOS, the standard operating system for IBM PCs does not support multitasking. PC-DOS does not support processes running concurrently or any kind of interprocess communication. The MIDINet system comprises several independent tasks. Some of these tasks are similar. For example, processing MIDI from the In ports. Some tasks generate inputs for others, hence coordination must be maintained between them. Under PC-Xinu, a user can create several processes carrying out similar or different tasks. Processes are not uniquely associated with a piece of code; multiple processes can execute the same code simultaneously [Comer, 1988]. At the heart of the system lies the scheduler and the context switching mechanism. These are responsible for switching the CPU among processes ready to run. Multitasking and the capability to offer coordination between processes made PC-Xinu a suitable choice for this project. However, PC-DOS development tools can be used to create PC-Xinu programs.

4.3.1 Process Management and context switching

The backbone of PC-Xinu process management comprises linked lists and the procedures which manipulate them. These procedures perform such tasks as inserting items into lists, removing them from the lists and allocating new lists.

The process manager often needs to select the process with the highest priority key from a list. The linked list routines must be able to maintain lists of processes that have an associated

priority. **Priority** is an integer value assigned to a process.

This priority scheme is suitable for the implementation of the MIDINet system. Tasks in the MIDINet system can be assigned varying priorities. For instance, tasks carrying out MIDI data reading, processing and writing can be assigned high priorities. A process that is carrying out reading will be assigned the highest priority, since data should not be missed.

Processes use coordination primitives to synchronize their actions and to share resources. Semaphores are a basic coordination technique in PC-Xinu. Under the MIDINet system, semaphores are used to coordinate buffered I/O.

Message passing is a form of inter-process communication in which one process requests the operating system to send data directly to another process. Messages can also provide a means of process coordination, because the receiver can be delayed until it receives a message. Unlike semaphores, message passing need not be synchronized. This way, a process need not know how many messages it will receive, and which process sent the message. A typical example in the MIDINet system is when an interrupt service routine sends a message to a process to inform it that data is available on one of the ports. The process can then retrieve the data.

At any time, the highest priority process eligible for CPU service is executing. For equal priority processes, rescheduling is in a round robin fashion. The processes are kept in various lists depending on their states.

Scheduling and context switching are closely related activities that make concurrent execution possible. Scheduling consists of selecting a process for CPU usage, from the eligible processes. Context Switching consists of stopping one process and starting another.

4.3.2 Memory management

Memory is an important resource, essential for program execution. The operating system keeps track of the location and size of available free space, allocating it on demand and recovering it when a process completes.

PC-Xinu allocates memory dynamically. It keeps track of storage that has been released. At

this level, free space is treated as an exhaustible resource-the system simply passes it out provided requests can be satisfied. The low-level memory manager rejects requests that cannot be satisfied; mechanisms to prevent processes from using all the free memory or to block processes until their requests can be satisfied, do not exist. Processes need space for stacks to hold procedure frames and local variables, and the so-called *heap* space for other dynamically allocated variables. The table below shows the storage layout during execution.

Code	data	heap	stack	free....

The original version of PC-Xinu was written for a small memory model, where the code of the operating system kernel and user program are bundled into a single code segment (64k). In a similar fashion, user and system data are stored in the same data segment. This imposes a severe restriction on the size of any user programs or kernel extensions. This was one of the motivations for altering PC-Xinu for the implementation of the MIDINet system [Rehmet, 1992] [Comer, 1987]. This altered PC-Xinu is discussed in section 4.3.5.

4.3.3 Interrupt processing

Hardware interrupts offer a powerful mechanism that provides support for many of the services the operating system supplies. Without an interrupt mechanism, the operating system could not guarantee that it would ever regain control once it started executing a user process.

In PC-Xinu, BIOS, the basic input/output routines of DOS, are used to process keyboard, timer, and clock interrupts. For MUART and Ethernet interrupts, there is no BIOS handling.

4.3.4 Device drivers

The hardware interface to most devices is relatively crude, requiring complex software to control them. Control is provided by routines called *device drivers*. The operating system should employ a fair and safe policy in making the devices available to processes. The users in their programs refer to the devices by their names, not knowing the machine configuration. The programs access the services provided by these devices by making *system calls*. System calls provide an interface between the operating system and user programs.

In PC-Xinu the device switch table, *devtab*, contains the entire I/O configuration, and is used when the system is compiled. The information stored in this table includes the I/O primitives and interrupt vectors that enable operation of devices. The structure of entries in the *devtab* table is shown in figure 4.2.

```
/*------------------------------------------------------------------------
 * Format of each entry is:
 *
 * device number, device name,
 * init, open, close,
 * read, write, seek,
 * getc, putc, cntl,
 * port addr, device input vector, device output vector,
 * input interrupt routine, output interrupt routine,
 * device i/o block, minor device number
 *------------------------------------------------------------------------
 */
```

Figure 4.2 Structure of entries to the *devtab* table

A driver can be added to the PC-Xinu operating system without any manual changes being made to the existing kernel modules. Only the configuration file, *pcxconf*, needs to be changed [Comer, 1988]. A new system configuration can then be generated by running the *config* utility, in order to build the modules *conf.h* and *conf.c*. These modules contain configuration declarations and the entries to the device switch table, respectively. MUART and Ethernet devices had to be added to the operating system, hence they have entries in the *devtab* table. This is further explored in section 4.4.

4.3.5 Memory resident Xinu

The original PC-Xinu system is structured in such a way that the system kernel and the user program are linked together in one executable file. As a result, a PC-Xinu program must have a copy of the operating system kernel linked to it [Watkins, 1990]. Hence, PC-Xinu programs become unnecessarily large, and development becomes a tedious process. Any changes or updates to the system kernel of the original PC-Xinu system require the relinking of user programs.

These problems formed a motivation to build a memory resident version of PC-Xinu. Under

the resident version, user program binaries can be run without modifications when slight modifications are made to the kernel. Software interrupts are used in order to build an interface between the user program and the PC-Xinu kernel. System calls are uniquely identified by numbers. Via this number, a proper routine in the kernel can be executed.

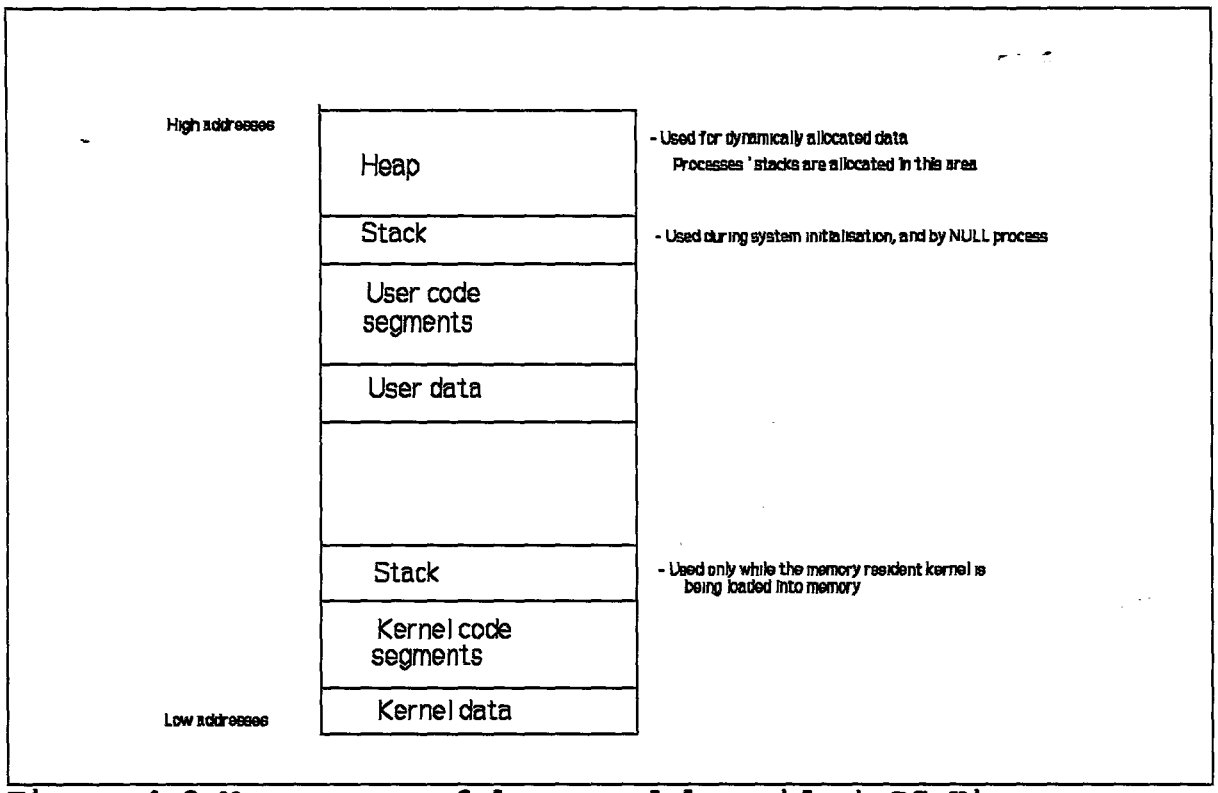


Figure 4.3 Memory map of large model resident PC-Xinu

Figure 4.3 shows a memory map of the memory resident PC-Xinu system, with an application program. The memory resident version offers much more space for code and dynamically allocated data. At initialization, all available memory in the DOS heap is allocated for PC-Xinu to use. User programs become reasonably small and development is more efficient.

4.3.6 Windowing

The memory resident version of PC-Xinu was extended to provide a windowing interface. It was styled after MS Windows [Rehmet, 1992]. A fundamental characteristic of this windowing interface is that it is event driven. This means that when an event such as a mouse button click occurs, a message is sent to an active window or a window currently receiving mouse messages. The windowing system provides facilities such as creating menus and scroll

bars. Other useful features include controls such as push buttons, which allow a user to make simple choices between different options, and text editing keys to edit the text.

The event driven approach provides a close link to the event driven style of modelling used for the MIDINet system. Hence, the MIDINet system could be easily implemented under the windowing system.

4.3.7 System initialization

When the PC-Xinu system starts, initialization routines are called in order to create the PC-Xinu operating environment, before the user program is started. In the memory resident version, at initialization, instead of starting up the initialization code, the memory resident operating system is loaded into memory. Initialization now only takes place when a user program is executed. The original initialization code is called via the interrupt mechanism which is used for the PC-Xinu system calls.

4.4 Added drivers

PC-Xinu device drivers had to be written to provide easy access to MUART and Ethernet cards. Following the discussion in section 4.3.5, the configuration file *pcxconf* was modified to add drivers for two MUART cards and one Ethernet card.

4.4.1 MUART under Xinu

Like other devices, the details behind MIDI devices or MIDI ports are hidden in the *device drivers*. Since two ports are on the same card, they are treated as a single device under PC-Xinu, that is they are referenced via the same routines. They are distinguished by parameters.

MIDI devices do not make use of all the PC-Xinu I/O primitives. Only *read, write, init and close* are used. Each device has an entry in the *device switch table*. Each entry in this table corresponds to a single device; it contains the device name, the addresses of the device driver routines for that device, the device port and vector addresses, and other information used by the drivers. Figure 4.4 shows the two entries in the device switch table for the two MIDI devices.

```
/* MUARTA is muarta on MUART */
37,"muarta",
muartinita,ioerr,muartclosea,
muartreada,muartwritea,ioerr,
ioerr,ioerr,ioerr,
0,MUAVECA,0,
muartiinta,ioerr,
NULLPTR,0,

/* MUARTB is muartb on MUART */
38,"muartb",
muartinitb,ioerr,muartcloseb,
muartreadb,muartwriteb,ioerr,
ioerr,ioerr,ioerr,
0,MUAVECB,0,
muartiintb,ioerr,
NULLPTR,0,
```

Figure 4.4 Maurt entries in the device switch table

Like most device drivers, the MIDI driver routines can be partitioned into two sets; the *upper-half* and the *lower-half*. User processes call upper-half routines to read and write bytes. The two halves of a driver communicate only through the shared buffers. The upper-half routines enqueue requests for data transfer. Lower-half routines transfer data from buffers to devices or from devices to buffers (Figure 4.5).

The MUART device has a control block (*muartblk*). This structure defines the input and output buffers and the semaphores controlling access to the buffers. A buffer is an array of characters with head and tail pointers. The buffers are treated as circular lists.

4.4.1.1 Initializing

At initialization (*muartinit*), the format of the asynchronous data word is specified by writing to the Line Control Register. The format specifies the length of the word and the presence of parity and stop bits. The length of the word is set to 8 bits. There is no stop bit and no parity. The last bit of the Line Control Register is used to determine addressable locations.

The divisor latches are programmed before communications to give the desired bit rate. The baud rate is set to match the rate of MIDI, 31.25 kHz.

The Modem Control Register is set. This register controls the interface with the device

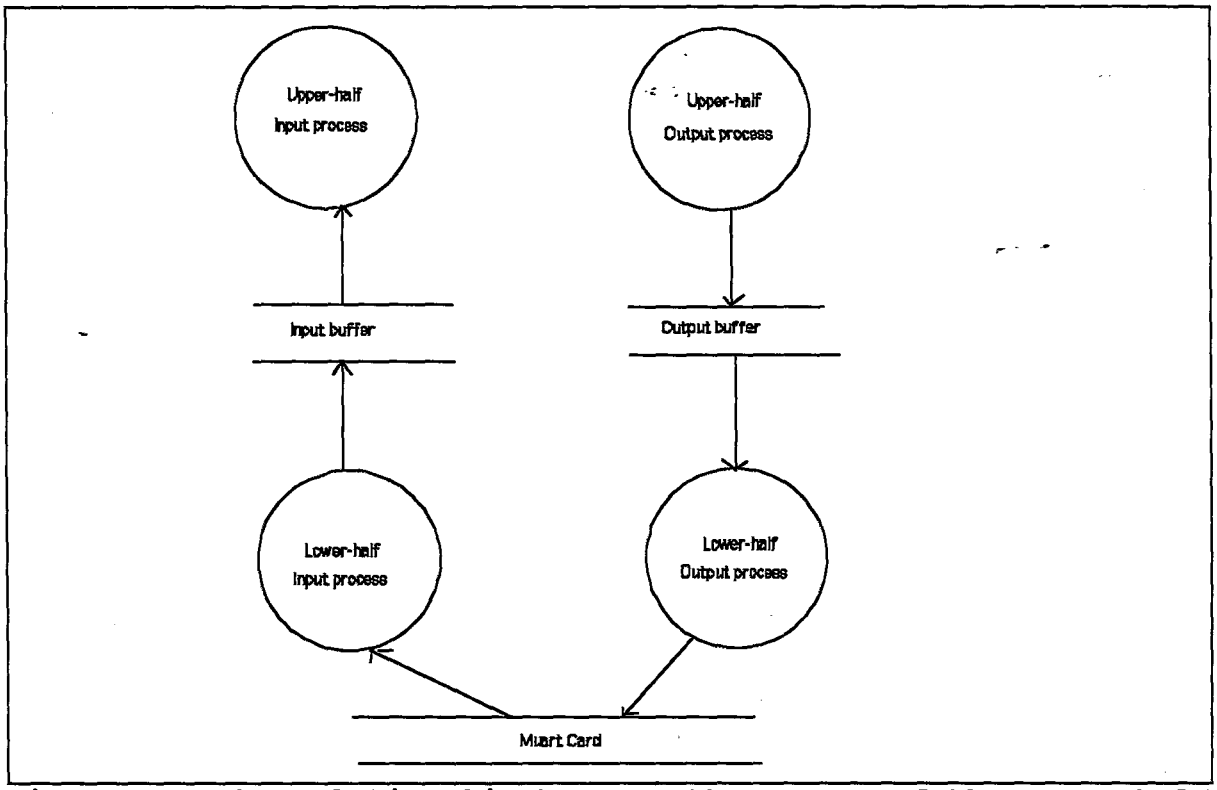


Figure 4.5 The relationship between the Lower and the upper half routines of a device driver

connected to the card. The loopback feature for diagnostic testing is enabled. When activated, the output of the transmitter shift register is fed internally into the receiver shift register, so that a word written to the transmitter buffer register returns immediately via the receiver buffer register. This can be read and compared with what was written, which provides a way of verifying the transmit and receive data paths. This method is used to detect the presence of MUART cards in a MIDINet unit.

Using the MUART's Interrupt Enable register, interrupts are enabled in the MUART card. The four low-order Interrupt Enable register bits are used to enable or disable interrupts by priority levels.

The input and output driver semaphores are created. Pointers to I/O buffers are also initialized.

4.4.1.2 MUART - Interrupts and reading

In PC-Xinu, interrupts are first handled by an interrupt dispatcher, which calls the routine that is designed to perform the services required by the device. The interrupt handling routine (*muartiint)* sends a message to an input process (*muart_iproc),* which first clears interrupts so that data cannot be overwritten. The Line Status register of INS8250 is used to determine the status of the data transfer. Then, the byte is read and interrupts are enabled.

The byte that has been read is pushed into an input buffer. The upper half input routines are signalled to read data.

Since two ports are serviced by the same routines, when an interrupt occurs, there is no way of determining which port caused an interrupt. Hence, each time an interrupt occurs, both ports are polled.

The parameters of the PC-Xinu device read function include a buffer and the number of bytes to be read. However, in the case of the MUART device, bytes are read one at a time. The parameter that indicates the number of bytes to be read, is used to distinguish between two ports since they are on the same card. For example, a read instruction is *read(..,buff, x)*, where x indicates the port number. Hence, the reading routine is passed 1 or 2 to indicate which port to poll.

4.4.1.3 Transmission

At the lowest level, transmission is performed by the routine *muartwrite*. Before data is transmitted, the status is determined using the Line Status register. If the status permits transmission of data, then data is transmitted by writing to the *transmitter buffer register.*

An application writes a byte by specifying *write(..,buff, x)* , where x indicates the port number. To indicate the port to transmit on, the function is passed 1 or 2.

4.4.2 Ethernet

At the lowest level, network communication consists of transferring a copy of data from one machine to another over a network to which both machines directly connect. Data is

transmitted in the form of *packets*, which could be described as "a quantity of data sent across a packet-switched network" [Comer, 1987].

The *packet driver* provides the network interface between the medium and the operating system. It is loaded at the DOS level before the PC-Xinu operating system is run. Packet drivers provide a simple, common programming interface that allows multiple applications to share a network interface at the data link level [FTP Software, 1989]. The packet driver provides calls to initiate access to a specific packet type, to end access to it, or to send a packet. There are packet drivers for specific classes. The class tells what kind of medium the interface supports, such as Ethernet, Token Ring, and so on. Within a class of medium there are particular instances of an interface supporting this class such as 3Com and Western Digital.

Applications which use the packet driver can also run on different network hardware of the same class without being modified; only a new packet driver need be supplied. So, the MIDINet system can run on any PC that has an Ethernet interface and for which there is a packet driver.

Under PC-Xinu, Ethernet device drivers were built to make calls to services provided by the DOS packet driver. A process writes to the Ethernet device interface to send a packet and reads from the Ethernet device interface to receive a packet.

In the case of Ethernet, the destination network address is required when writing data. This makes Ethernet slightly different from other devices. The design followed was to pack the destination address with the data. The aim was to have the higher level software accept and deliver packets already in the proper form expected by the packet driver.

The Ethernet device also has an entry in the *device switch table* of PC-Xinu. The Ethernet device does not make use of all the PC-Xinu I/O primitives. Figure 4.6 shows the Ethernet device entry in the table.

Definitions for Ethernet constants and packet format are given in a file *ether.h* shown in Figure 4.7. The *header* of a packet refers to all components of a packet except the data. The structure *eheader* describes the format of an Ethernet packet in memory. The packet consists of a header followed by data for that packet. The format of structure *epacket* differs from the

```
/* ETH is eth on ETHERNET */
39,"eth",
ethinit,ioerr,ethclose,
ethread,ethwrite,ioerr,
ioerr,ioerr,ioerr,
0,0,0,
ioerr,ioerr,
NULLPTR,0
};
```

Figure 4.6 Ethernet entry in the device switch table

```
/* ether.h */
/* Ethernet definitions and constants */

#define EPADLEN 6       /* number of bytes in physical address */
typedef byte Eaddr[EPADLEN];    /* length of physical address(48 bits) */

struct eheader {        /* format of header in ethernet packet */
       Eaddr e_dest;          /* destination host address */
       Eaddr e_src;           /* source host address */
       byte  e_ptype[2];      /* Ethernet packet type */
};

struct epacket {        /* complete structure of ethernet packet */
       struct eheader ep_hdr; /* packet header */
       char ep_data[EDLEN];   /* data in packet */
};

/* Ethernet control block descriptions */

struct etblk {
       Eaddr etpaddr;   /* Ethernet physical device address */
       byte ptype[2];   /* ethernet packet type */
       byte far *rbuff; /* pointer to packet driver read buffer*/
       int pkt_len;      /* length of the read packet */
       int etr_request;  /* flag to indicate a read request */
       int etrpid;       /* id of process reading from ethernet*/
       int etwpid;       /* id of process writing to ethernet*/
       int etrsem;       /* mutex for reading from ethernet */
       int etwsem;       /* mutex for writing to the ethernet*/
       int pdint;  /* software interrupt for packet driver*/
       Bool packet_in_rbuff;    /* New packet in buffer? */
       class;            /* class, type and number relate to */
       type;             /* hardware interface card */
       number;
       handle;           /* handle onto packet driver */
};
```

Figure 4.7 Contents of *ether.h*

format of packets transmitted on the Ethernet, because the structure contains neither the

Ethernet preamble nor CRC.

Field *e_ptype* encodes the packet type, which specifies to the receiver the purpose and format of the data area. A special comment should be made here about the packet type used. In this work the type ***0x2000*** was adopted for music packets, a packet type not already in use by standard protocols was selected.

Like other devices, Ethernet has a control block defined in the structure *etblk*. Fields in the *etblk* structure define all the data kept by the device driver. There are two semaphores that provide mutual exclusion among processes reading or writing to the Ethernet; fields *etrsem* and *etwsem* contain the ids of these two semaphores. Once a process calls the upper-half read routine, it is blocked until a packet arrives.

4.4.2.1 Initializing

At initialization time (*ethinit*), the initializing routine determines if the DOS packet driver has been installed. If it has been installed, then initialization continues, otherwise it stops. The Ethernet device control block is filled, including creating mutual exclusion semaphores *etrsem* and *etwsem*.

Figure 4.8 shows the communication links between drivers. The class, type and number of the DOS packet driver is determined. The DOS packet driver is passed the address of the receiving routine and the receiving mode is set. In this case mode 3 is used which means receive 'broadcast and packets set for this interface only'. The physical address is also determined.

The input mechanism is initialized; the pointers to the circular buffer are initialized and the input semaphore *isem* is created. The semaphore ensures synchronization between the lower and the upper half input routines.

4.4.2.2 Input routines

The DOS packet driver is passed an address of a routine to call when a packet arrives (*ethdisp*). This routine calls another routine, *ethrcv*. The DOS packet driver makes two calls,

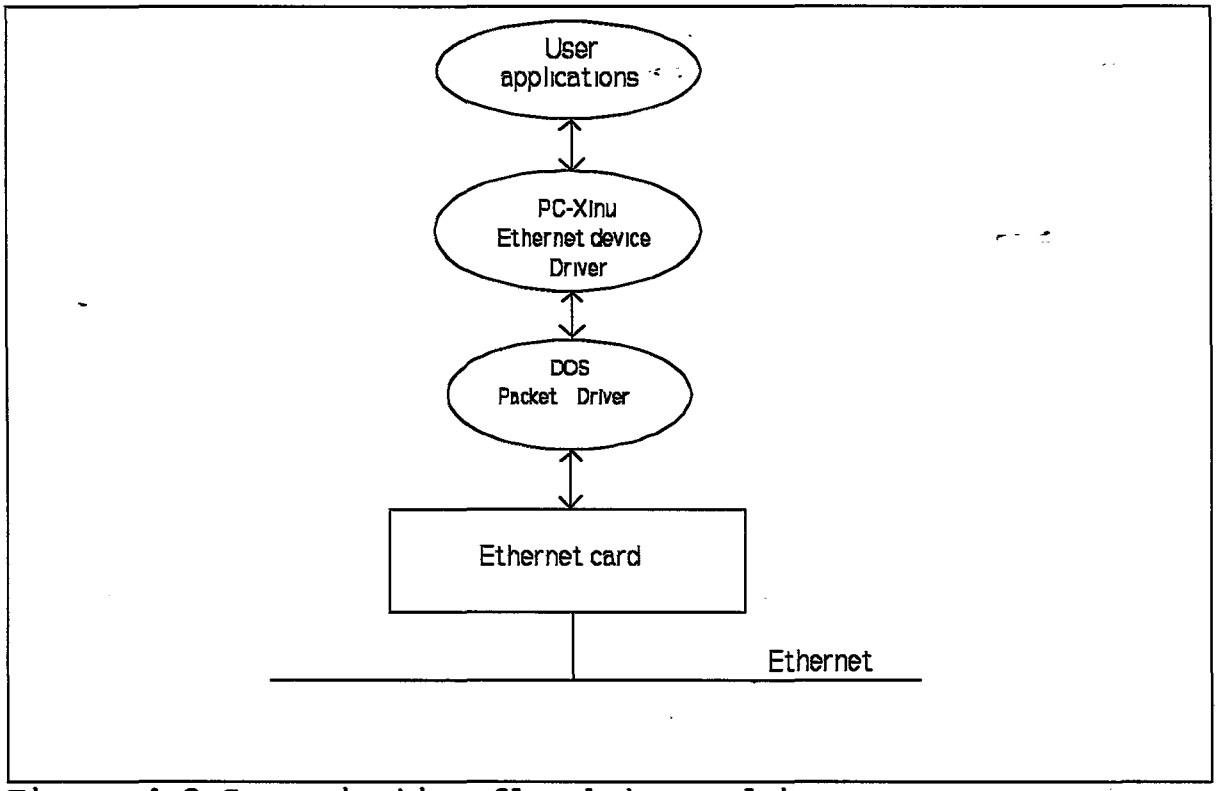


Figure 4.8 Communication flow between drivers

the first to obtain a buffer for copying data into, the second to confirm successful copying. The packet is then stored in a circular buffer by the lower half routine.

The upper half routine, *ethread* waits on two semaphores; one to ensure mutual exclusion if several process are reading from this device, *etrsem*, and the other to ensure synchronization, *isem*.

4.4.2.3 Output routines

At the lowest level there is the *ethwpdrv* routine which interacts with the DOS packet driver. It executes the packet driver's transmit function. This routine is called by the upper half routine, *ethwrite*. The *ethwrite* routine sets up the source address component of the packet header. It also waits on a semaphore to ensure mutual exclusion.

4.4.2.4 Closing the device

When closing the ethernet device the release function is executed. This function breaks the

connection between the DOS packet driver and the lower half receiving routine.

4.5 MIDINet system

In the previous chapter the requirements specification of the MIDINet system was laid out. The next step was to go through a structured analysis and design of the system. Structured design *is the art of designing the components of a system and the interrelationship between those components in the best possible way* [Yourdon, 1979]. The design process determines the major characteristics of the final system, and establishes the upper bounds in performance and quality that the best implementation can achieve.

4.5.1 Analysis and Design of the system

Ward and Mellor design tools were adopted for the modelling of the system [Ward, 1985a,b,c]. They have their roots in the data flow oriented approach to systems analysis proposed by DeMarco [DeMarco, 1979] and structured approach of Yourdon and Constantine [Yourdon, 1979].

A real time system development methodology provides modelling tools to model the control aspects of real time systems. It ensures that the development of a system is well structured and documented. This has the added advantage that the system is more easily understood by its users and those wishing to modify it.

The Ward and Mellor development methodology uses symbols for modelling the system. It also separates the essence of the system from its implementation. The essential model describes what the system must do in response to events in its environment. This model is independent of the technology employed. The description of a particular technology used in realizing the system is called the implementation model. The implementation model is an elaboration of the essential model.

4.5.1.1 The Essential Model

The essential model consists of two parts, the environmental model and the behavioral model. The environmental model describes the boundary between the system and its environment, the interfaces between the two, and the events that occur within the environment to which the

system must respond. The behavioral model describes how the system is required to behave in response to events. These models are accompanied by the data dictionary. The data dictionary gives the descriptions of objects, data flows and the content of these flows (see appendix 1).

4.5.1.2 The Environmental Model

The environmental model describes the *context* of the system, that is the interfaces between the system and its environment. The Context schema describes the boundary between a system and its environment, and the interfaces between the two [Ward, 1985b].

The context schema for the MIDINet system is shown in Figure 4.9. The notation used is as follows :

- Each box represents a terminator; something outside the system boundary with which the system interacts.
- A circle denotes a transform. In the context schema a single transform denotes the entire system and its activities.
- Inputs and outputs are shown as labelled arrows (flows). A time-continuous flow is shown with a double-headed arrow, a time-discrete flow is shown with a single-headed arrow. A broken line with a single headed arrow denotes flows which have no content, and are either signals or commands. The direction of the arrow indicates the direction of causality.

The main feature of the environmental model is the specification of the events to which a system must respond. These external events occur in the system's environment, at a specific time, and elicit a preplanned response from the system. The external events that occur in the MIDINet system's environment are given in appendix 1.

The environmental model further describes the information model of a system. Ward and Mellor's description of the information model or data schema for a system is not particularly thorough. The description does not reflect the importance of the information model in the system analysis process. The information model is the foundation upon which the rest of the system is built. It is the most fundamental and stable part of a system. Consequently, Shlear and Mellor [Shlear, 1988] tools were adopted in the discussion of information modelling.

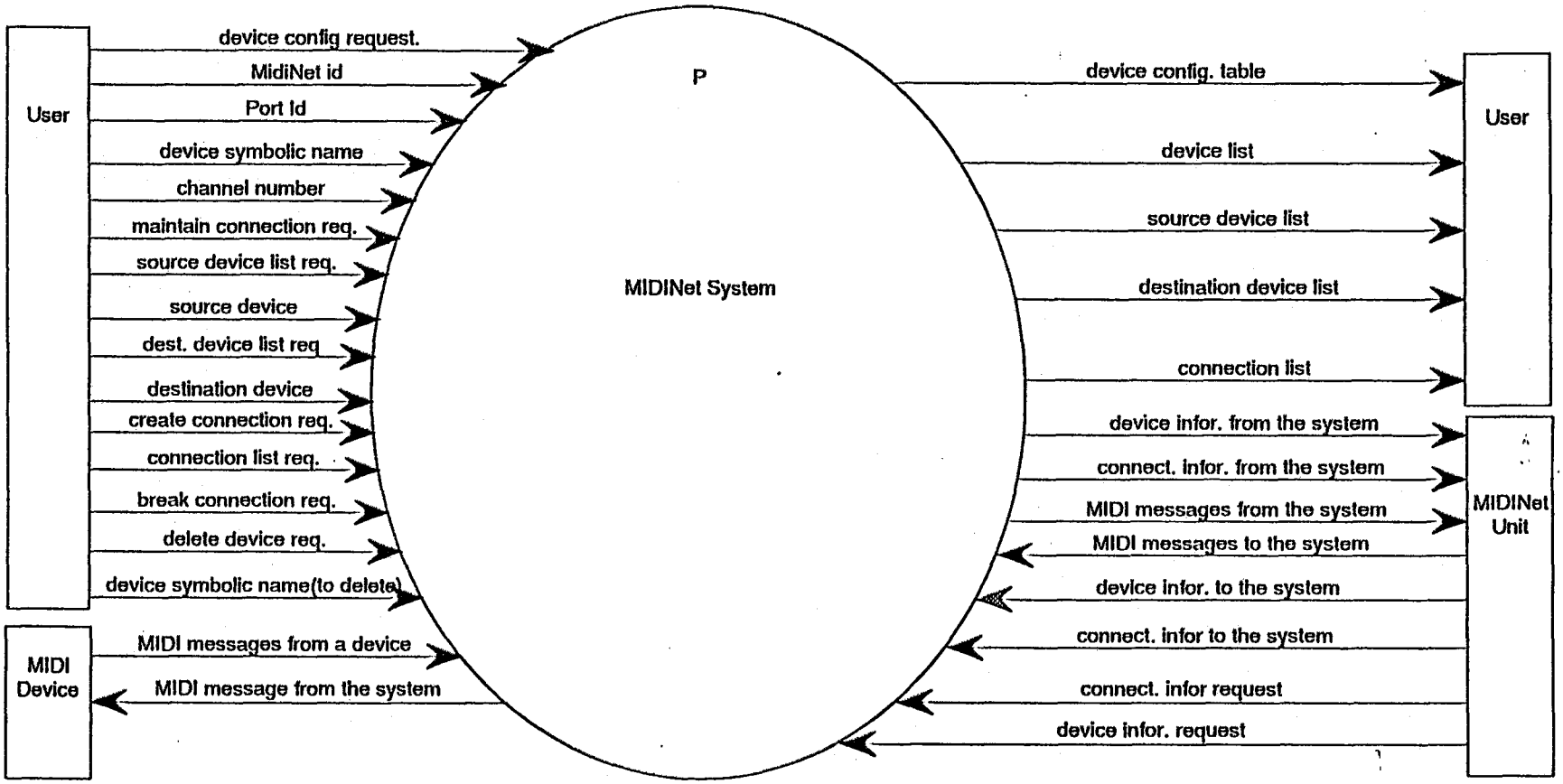


Figure 4.9 Context Schema for the MIDINet system

Shlear and Mellor's approach is to name the objects within the system, list their attributes, and indicate the relation between these objects. Refer to [Shlear, 1988] for a formal description of an object and attributes.

Figure 4.10 shows part of the information model of the MIDINet system. The whole information model is shown in appendix 1. A box shows an object with its attributes. Attributes preceded by an asterix form an identifier to that object. An identifier is a set of one or more attributes which uniquely distinguishes each instance of an object. Some attributes are referential attributes, they are denoted by an *(R)* at the end. Referential attributes capture information which ties an instance of one object to an instance of another object. The arrows indicate the relationship between the objects. Relationships involving only two objects can be classified into three fundamental forms; one-to-one, one-to-many, and many-to-many. Refer to [Shlear, 1988] for a formal description of these relationships.

4.5.1.3 The Behavioral Model

The context schema with the external event list and the information model constitute a detailed model of a system's environment. This does not contain a detailed description of a system's behaviour. The next step is to describe the response of the system to each event in the external event list. This is the task of a behavioral model.

The list of events and the response of the system to the events is shown in appendix 1. The description of the response must include at least:

- the most common or most likely response to the event.
- alternate responses and the conditions under which they are made.
- conditions under which the system will not respond to the event [Ward, 1985b, p41].

Most of the events in the MIDINet system are associated with the arrival of discrete data, and the system responds in a similar way for all possible values. Such responses are best modelled using data transforms. These data transforms make up what is known as the transformation schema [Ward, 1985b, p46]. The transformation schema models a system as an active entity - as a network of activities that accept and produce data and control messages. The transforms receive and produce data and event flows. Figure 4.11a shows part of the MIDINet unit transformation schema. It shows connection related flows. The complete transformation

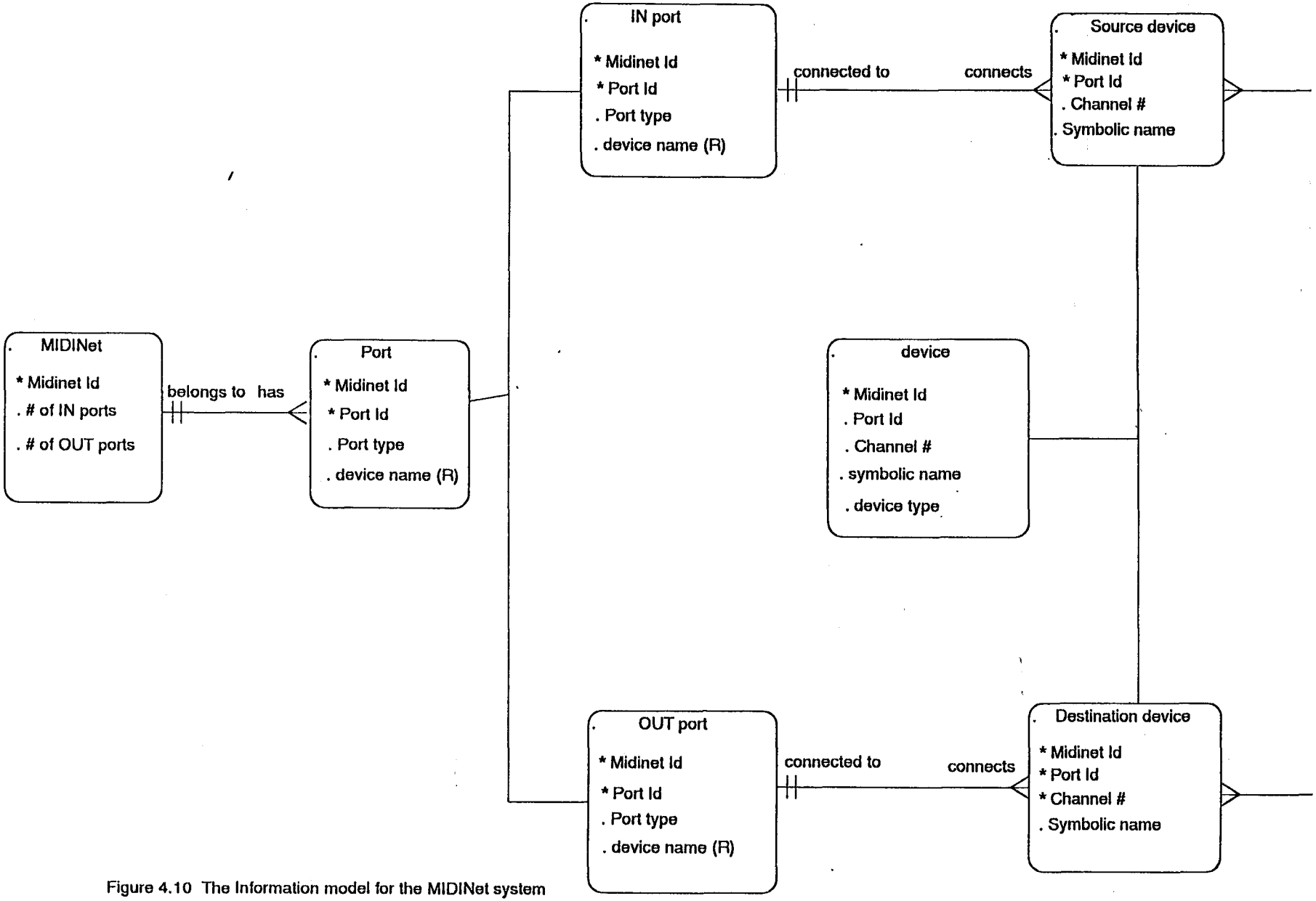


Figure 4.10 The Information model for the MIDINet system

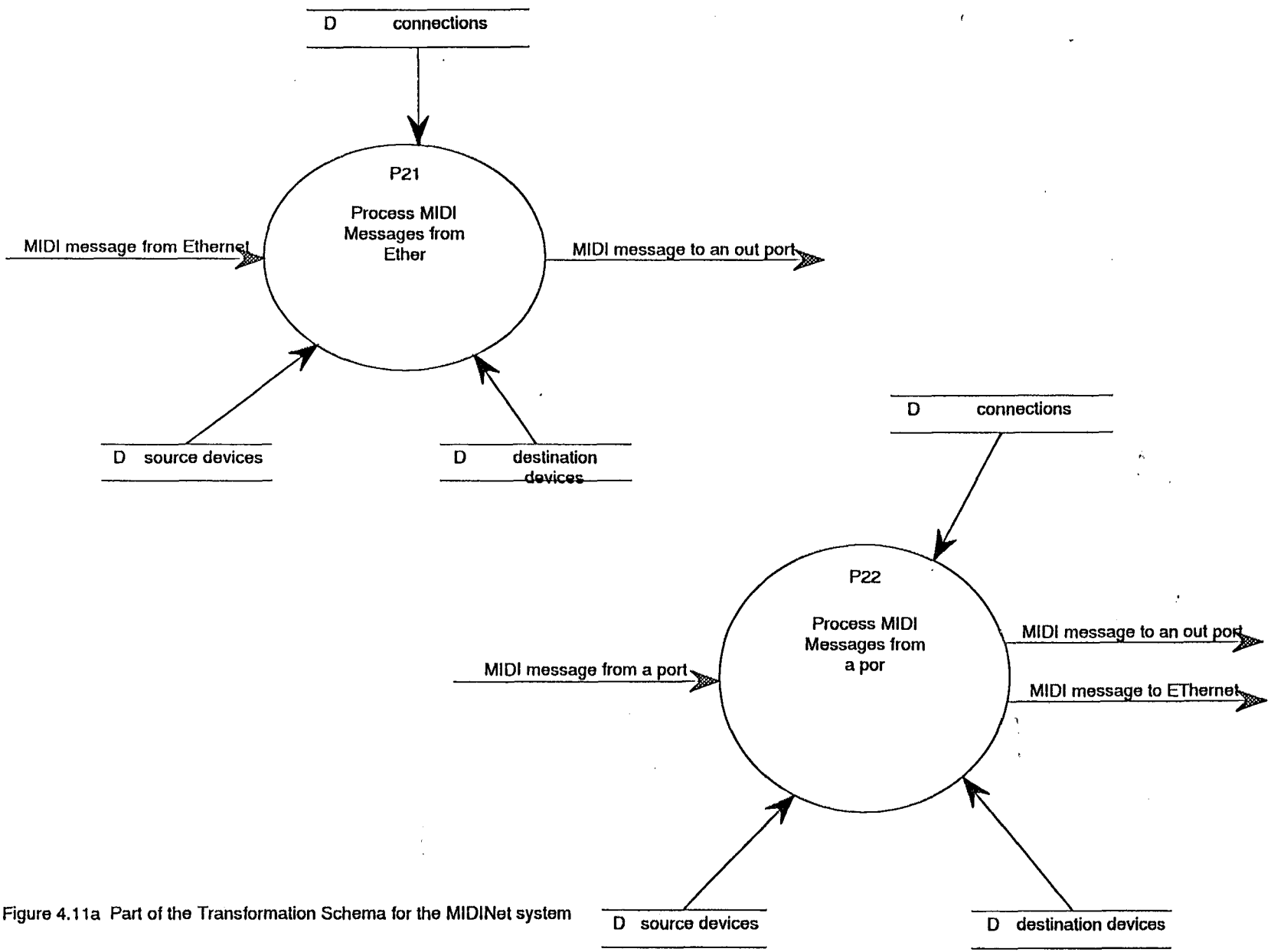


Figure 4.11a Part of the Transformation Schema for the MIDINet system

schema is shown in appendix 1.

The notation is similar to the notation used in presenting the context schema except for the addition of stores. The store notation is used to represent an item or a set that is operated on by a group of transformations but whose basic character remains unchanged. The flow connecting the store and the transformation is not labelled, it represents the availability of the store to the transformation.

The approach followed to create the transformations was to independently transform each flow on the context schema. At this stage the transformation schema has each transformation at approximately the same level of detail and with minimum interfaces between the transformations. Minimizing interfaces is important because the understandability of a section of the model is proportional to its independence from other sections. This transformation schema contains many transformations and requires levelling to attain a verifiable and reviewable form. Hence, the process of *upward* levelling was conducted.

Upward levelling is the grouping of event responses and associated flows and stores into larger units. Similar to event responses, the upper-level groupings, must continue to reflect the structure of the environment in which the system will function. Ward and Mellor suggest three criteria that could be followed to do upward-levelling: response-related groupings, terminator-related groupings, and nameability of groupings [Ward, 1985b p66]. The Response-related grouping criterion was followed. The basic structure of the essential model is based on responses, hence it helps to maintain this perspective when choosing upper-level groupings. For example, in the MIDINet system, MIDI data is processed in a much similar way despite its source. Consequently, parts of the transformation schema that transform MIDI related inputs can be grouped together. Figure 4.11b shows upward levelling of MIDI related transforms shown in figure 4.11a. Refer to Appendix 1 to get a clear picture of upward levelling.

4.5.1.4 Implementation Model

The essential model can be considered as an idealized model of the system which must now be reorganized to fit into the constraints of the implementation technology. The implementation model is derived by doing a top down allocation of the essential model to the implementation technology. The top down allocation process must be guided by heuristics.

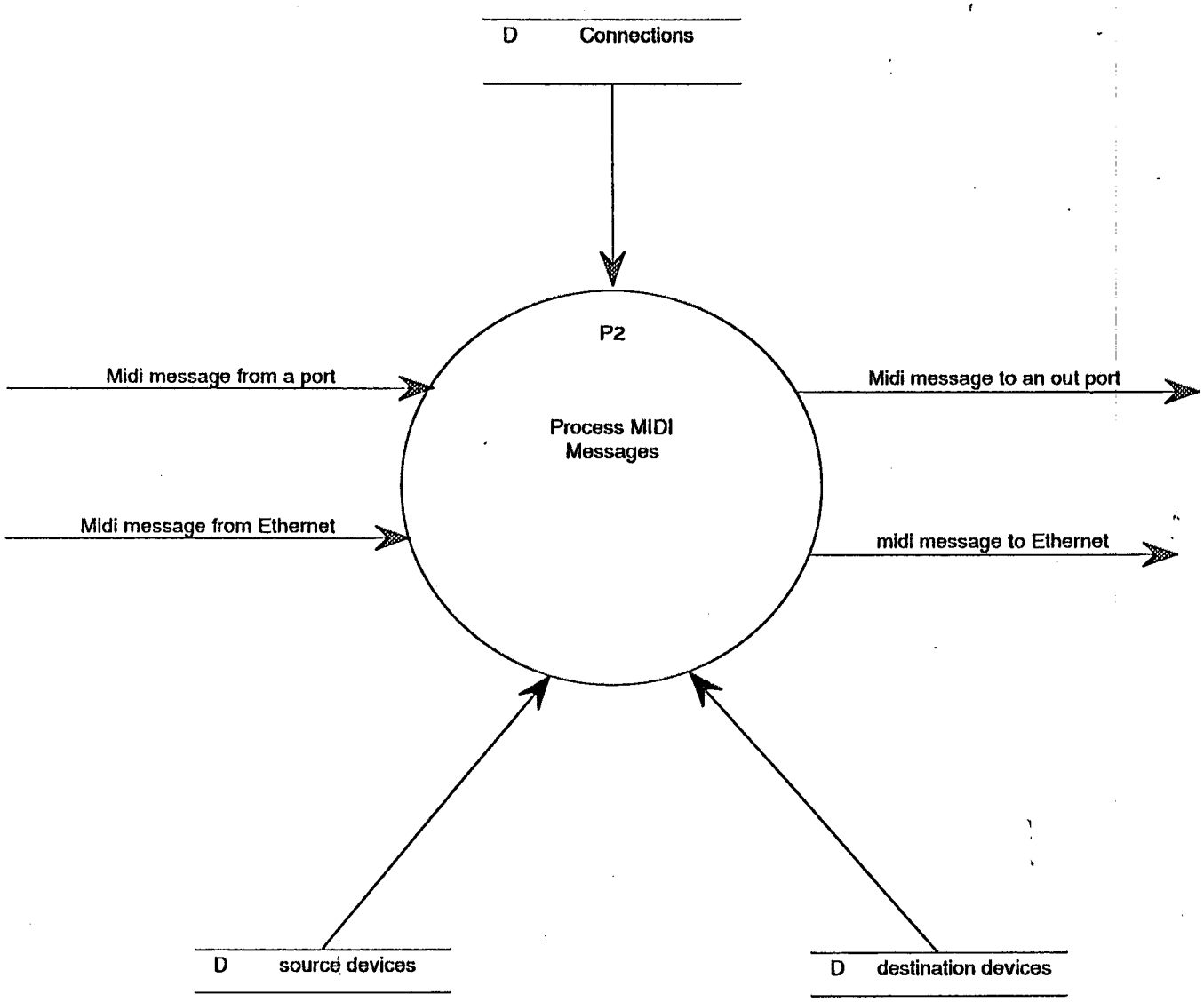


Figure 4.11b Upward levelling of MIDI related transforms

The main heuristic for this allocation is that it should cause minimal distortion to the essential model [Ward, 1985c, p3]. The MIDINet system is allocated to a single processor. A processor is a person or machine capable of performing instructions and storing data. The entire essential model is allocated to this single processor, a computer. Therefore, this allocation causes no distortion to the essential model.

4.5.1.4.1 Task Modelling

The processor is required to perform several tasks. A task is a set of instructions that can be started, stopped and interrupted by the processor. The MIDINet system has the following main tasks: *Read MIDI data, Process MIDI data, Send out MIDI data, Process configuration information, and Manage devices and Connections.* The upper level transforms abtained from performing upward leveling were assigned to these different tasks. In the case of MIDI related tasks, several tasks had to be derived in order to assign highest priority for the task that performs reading of MIDI data, so that the data is not lost. MIDI data is read from the ports or from the network and queued in buffers. It is then processed. After being processed it is queued in buffers waiting to be transmitted to its destination.

4.5.1.4.2 Models of interfaces

The allocation of transforms to processors and tasks tends to fragment the transforms and tends to introduce additional inter-task data flows and stores. Potential problems exist involving data exchange, stored data access, and control synchronization. The problems can be detected by modelling the interfaces between the tasks [Ward, 1985c, p52]. In the case of MIDINet systems, the important aspect is synchronization of access to stored data such as MIDI data. Since the data items (MIDI messages) are organized into a queue, the consumer task takes a unit of data previously placed by the supplier. Therefore, synchronization between the consumer and the supplier tasks must be maintained. This synchronization is modeled as control transformation when modelling the interfaces. Figure 4.12 shows the modeled interfaces related to the acceptance and processing of data from MIDI ports. The remaining interface related control transformations can be found in appendix 2.

4.5.1.4.3 Structure Charts

A task is composed of modules. A module is a set of instructions that is activated by the

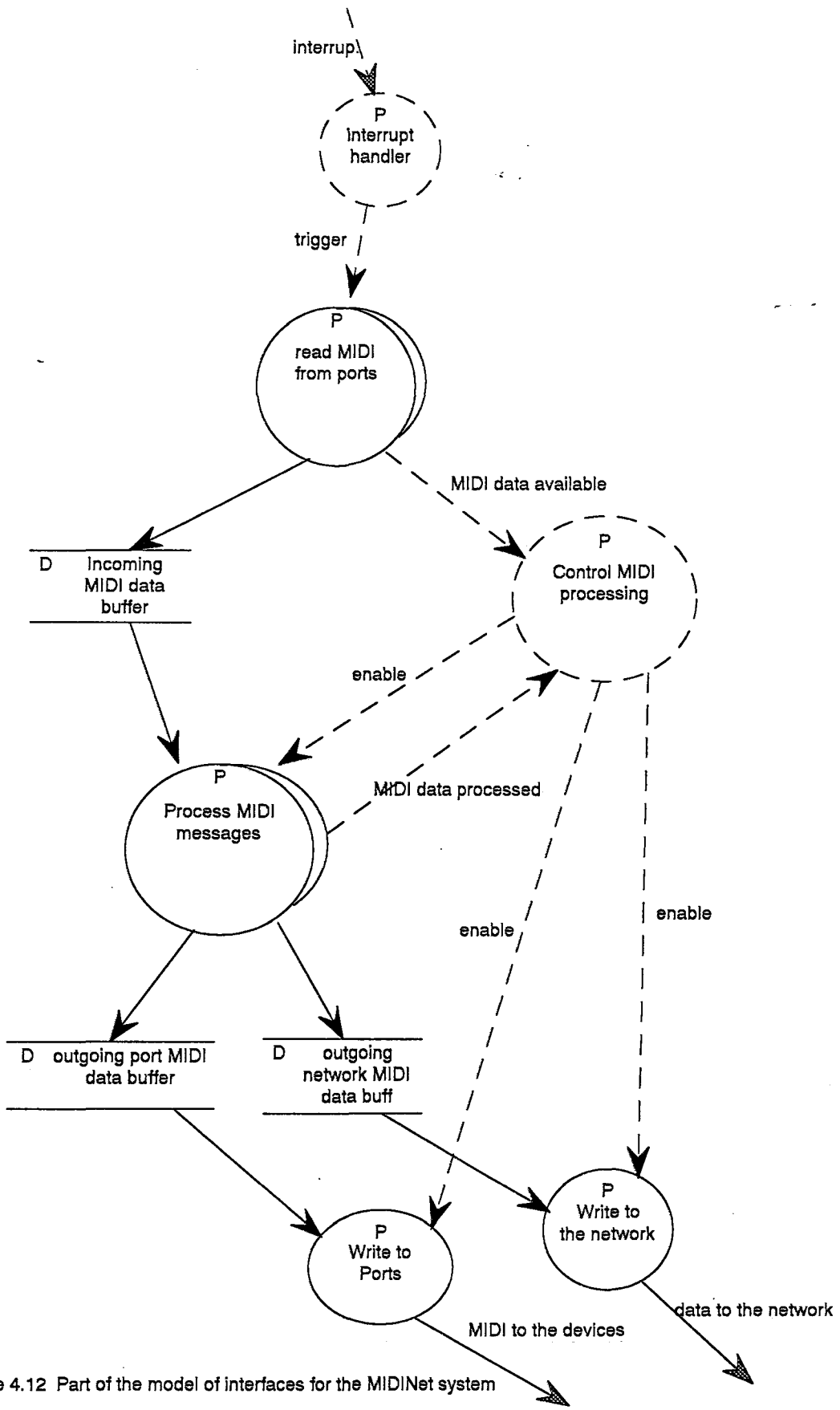


Figure 4.12 Part of the model of interfaces for the MIDINet system

control logic within a task. Only one of a task's modules may be active at a time, which precludes concurrency [Ward, 1985c, p79]. A structure chart shows the modules that comprise the task and it describes their relationships. A module is modeled as a named box with the name of the module placed in the box. A call is an activation of a module. The shared data area between modules is modelled as a named box drawn below, and connecting, each module that shares this data. A couple represents an item of data passed between modules when a call is executed. It is modelled as an arrow with a circle at its end. An open circle denotes processed
data and a filled circle denotes control information that controls the logic of the receiver. An example of a structure chart for the *manage devices* task is given in Figure 4.13. The complete structure charts can be found in appendix 2.

4.5.2 Software for the MIDINet System

The software required to run the MIDINet system was written in C. The tasks modelled in the implementation model are written as processes under the PC-Xinu operating system.

This section describes how configuring devices and establishing connections were implemented. The initialization procedure is also described. The steps that were followed in processing MIDI data are discussed.

4.5.2.1 Configuring devices

To configure a device, the user has to select an appropriate option from the menu. The user provides the MIDInet ID, Port ID, the channel number and the symbolic name for the device being configured. For a device to work both as a source and a destination, it needs to be configured *twice.* More than one device can be connected to a single port. Hence, these devices will bear the same MIDINet ID and Port ID, only channel numbers must differ. A user should be careful not to have more than one device on the same port with the same channel numbers, to maintain distinction between devices.

This configuration process can be performed on any MIDINet unit in the system. A device being configured may be connected to any MIDINet unit. As the devices are being configured, this configuration information is distributed to other MIDINet units in the system. Each MIDINet unit maintains a similar list of devices.

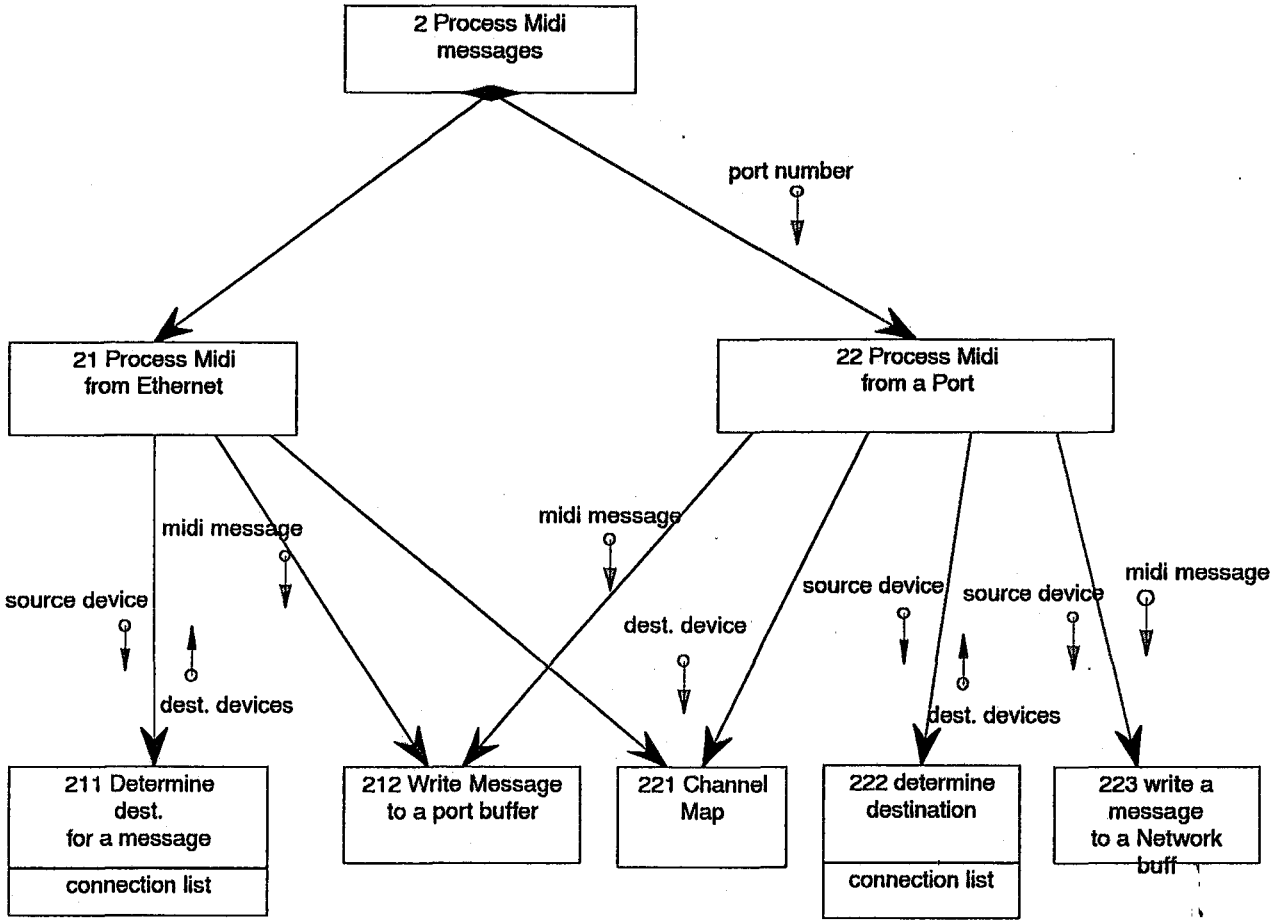


Figure 4.13 The Structure charts for the task Process Midi messages

To delete a device from the system, a user selects an appropriate option from the menu. A list of devices is then provided to the user. Again, a message is sent to other MIDINet units so that they can delete the same device from their lists. The system makes sure that a user does not delete a device involved in a connection.

4.5.2.2 Connections

After devices have been configured, connections between these devices can be established. The power of the system lies in the ability to connect any device to any other device. A single source can be connected to several destinations and several sources can be connected to a single destination.

To establish a connection, a user has to select from two separate lists, a source device and a destination device list. Connections are stored in a table within the system. Similar to the configuration process, connections can be established from any MIDINet unit in the system. The connection information is also distributed to other MIDINet units on the system. Each MIDINet unit in the system maintains its own connection list.

To break a connection, a user has to select a break option from the menu. A list of connections is then provided. An appropriate message is transmitted to other MIDINet units to update connection information.

4.5.2.3 MIDI

MIDI is read from four different ports and from the network. MIDI is read from the ports as a stream of bytes. On the other hand, MIDI from the network is transmitted in the form of MIDI messages. These MIDI messages vary in length.

MIDI from the ports is stored in a circular buffer and awaits processing. MIDI data received from each port and MIDI received from the network are stored in separate buffers. The processing that takes place at this stage is mainly to determine the destination of this information. The destination could be any device connected to any MIDINet unit in the system.

After MIDI data has been processed, it is stored in buffers awaiting to be transmitted, either

to a local port or sent over the network.

4.5.2.4 MIDI processing

A program designed to process MIDI data in real time can be structured around the following steps :-

1. *Get a byte from a MIDI port*
2. *Parse the byte*
3. *Process the byte*
4. *Send the processed byte to the MIDI Out port*

The first and the last steps are simple enough, each requiring a single input/output statement. The second and third steps are more complex and are discussed below.

4.5.2.4.1 MIDI parsing

MIDI processing would be more simple if all messages were sent using the normal transmission format, with each MIDI message sent in its entirety, starting with the status byte, followed by all associated data bytes in their proper order. If this general rule was followed by all devices, the MIDI data stream would be smooth and predictable, making it easy to parse. However, There are exceptions to the general rule : *Running status* format transmission and prioritized *System Real-time* messages.

Running status format is a time-saving method for transmitting long strings of *Channel* messages having the same status bytes. In this case the status byte is redundant and is not transmitted. Real-time messages can be inserted anywhere within a string of Running status bytes.

System Real-time message have a priority over other types of MIDI messages. These messages are used to synchronize devices in a MIDI system. The reason why they need to be inserted anywhere within MIDI data stream is to maintain the most accurate synchronization possible.

4.5.2.4.2 Guidelines for parsing the MIDI data stream

A MIDI program must be able to parse messages received in normal and Running Status formats, with prioritized transmission of System Real-Time messages. The following guidelines taken from [De Furia, 1989] can be followed in creating a parsing routine :-

- Every incoming byte should be identified as either a MIDI status byte or a data byte.

- Use a status buffer to keep the value of the last received Channel voice or Channel Mode status byte, with the following exceptions :

  The buffer should be cleared when System Common messages are received, and at start-up.

  The buffer should be left unaltered when System Real-time messages are received.

- The parsing routine should use the Status buffer to call appropriate message processing procedures.

- The program should ignore incoming data bytes when the status buffer value is zero.

- A counter should be used to keep track of the order of received bytes. It should be cleared whenever a new status byte is received. It should be incremented by one each time a subsequent data byte is received, and set to 1 when the final byte of the message is received. The counter should be left unaltered when a System Real-time status byte is received.

- Processing routines for each recognized message can use the counter value to determine which byte of a message they are currently processing.

Refer to appendix 3 for a table of types of MIDI bytes. The MIDINet system follows these

guidelines to parse MIDI bytes received from the ports. The parsing routine forms MIDI messages. Complete MIDI messages are processed to determine their destinations, then either sent over the network or out on a MIDI port.

4.5.2.4.3 MIDI Mapping

MIDI-Mapping refers to the reassignment of the value of a byte in a MIDI message. The following are some ways to use mapping algorithms :-

- Reassign program numbers
- Reassign channel numbers
- Reassign note number to create transposition
- Scale velocity data to any curve
- Scale controller data to any curve

Channel reassign mapping has been used in the MIDINet system. Before a message is sent out on a MIDI port, its channel number has to be mapped to the channel number set on the destination device (the device connected to this particular out port, refer to section 4.5.2.4.4). Other forms of mapping can be implemented easily by incorporating relevant procedures and flags.

4.5.2.5 Routing MIDI

After a MIDI message has been read from an *In port buffer*, the *status byte* is used to determine the channel number an instrument is transmitting on. The information relating to the source device is now complete. The MIDInet ID, the Port ID and the channel number are known.

The next step is to determine the destination(s) of this MIDI message. The source device is compared with all the source devices within the established connections. The destination devices, if any, are stored in a list, the *destination device list.*

After the *destination device list* is constructed, each device is processed separately to determine if it is a local device, that is if its MIDINet ID matches the ID of the unit that is doing the processing. If it is a local device, then the MIDI message is written to an *Out port*

buffer, waiting to be transmitted to an *Out port*. If the MIDInet IDs do not match, then the message is packed into a packet to be transmitted over the network.

The MIDI message sent over the network is tagged with source device information. The reason is to avoid transmitting the same message more than once. This can occur if the connection is between a single source device and several destination devices. These destination devices may all be non-local. Transmitting the same message more than once will use network bandwidth and make the system inefficient.

Before a message is sent to an *Out port*, channel mapping has to be effected. This is to change the channel the MIDI message originally contained, to the channel number the destination device is set on. The status byte of the first message is stored. Successive messages are sent without the status byte, if it is the same as the stored status byte, in order to achieve running status format with its speed advantages.

MIDI data from Ethernet is processed in a similar manner. Recall, each MIDI message is tagged with source device information. All units that receive the message perform similar processing. They determine the destinations from the common connection table, and send the message to the appropriate ports. There is a slight difference concerning the messages received from the network. If the receiving unit determines that the destination device is not local, then the message is discarded. This only applies to messages received from the network.

4.5.3 Initialization

Each MIDINet unit reads from the file *midiid.txt* to determine its identification number. If this file does not exist, the system is halted with an appropriate message.

A MIDINet unit sends a request to all other units, if any, that are already operating, to be loaded with the configuration data. This refers to the information related to already configured devices and established connection data (described in section 3.4).

The main task of the initialization procedure is to start all the necessary processes for the system to perform. There are three basic types of processes, reading, processing and writing processes. This applies to all four ports and Ethernet. It is at initialization time that all buffers are initialised.

4.6 MIDINet protocol

The MIDInet system was built based on layers. The idea came from the *International Standards Organization (ISO) model of Open System Interconnection (OSI)* [Tanenbaum, 1988] [Halsall, 1988]. Although it was designed to provide a conceptual model and not an implementation guide, the ISO layering scheme has been the basis for several protocol implementations. The OSI model has seven layers. The principles that were applied to arrive at the seven layers are as follows :

1. A layer should be created where a different level of abstraction is required.

2. Each layer should perform a well defined task.

3. The task of each layer should be selected with the aim of defining internationally standardized protocols.

4. The boundaries between the layers should be made in such a way that information flowing between the layers is minimized.

4.6.1 MIDINet system Layer Model

The traditional 7 layer model defined by OSI was created mainly for wide area communication through public and switched telephone networks. The MIDINet system is based on a three layer system which has been optimized for high performance real-time communication.

At the top there is the **Application layer** through which the user interacts with the system. For example, the user is provided with a menu system. The options include configuring devices and establishing connections.

It is in this layer where lists of devices and connections are maintained. Addition and deletion of devices is effected. MIDI data is read, processed, and routed to various destinations. Errors that occur when configuring devices or establishing connections are detected here.

In this layer, data is optimized for storage and manipulation. Data is typically stored in a

structured format. Any mapping of data from its pure format to human representational format is performed here.

Any modifications that need to be effected on incoming MIDI data are done in this layer. These modifications include filtering and mapping. Routing of MIDI is also performed here.

Below the application layer is the **MIDINet layer.** This layer performs the routing of the messages originating in the application layer. A complete description of the MIDINet layer with its associated protocol, is given in section 4.6.2.

The bottom layer is the **transport layer.** This layer describes the transport medium, Ethernet. Ethernet is the technology employed within RHOCMN and was adopted for this project. This layer specifies procedures used to transfer data from one unit to another. This is not to be confused with the OSI transport layer.

An attempt was made to specify the MIDINet system layer model in terms of the OSI reference model. The various protocol standards for LANs, which deal with the physical and data link layer in the context of the OSI Reference model, are those defined in IEEE Standard 802 [Halsall, 1988]. The MIDINet system utilises one of the IEEE standards, IEEE 802.3 (CSMA/CD), hence a relation between the two models can be made. Following design directions for ArcoNet, a comparison between the two models was made [Allik, 1990]. The directions specify the functions of each layer of the OSI Reference model for the implementation of ArcoNet. The directions were followed because ArcoNet is proposed for a MIDI related network similar to the MIDINet system.

Briefly, the ArcoNet proposal states that the low-level protocols implement error detection and recovery. This should be performed in real-time. Different levels of transport service are required, in order that, the data which needs to be communicated in real-time may not be delayed. Polling and acknowledgement messages are also defined. The presentation level protocols specify a common format for signalling events across the network. The application level protocols specify the network management software. The software performs tasks such as network configuration, testing, detection of station failures and synchronization of real-time signals over the network.

In relation to the OSI Reference Model, the MIDINet system transport layer corresponds with the

physical and data link layers of the OSI Reference Model. The MIDINet system utilises Ethernet cable. It also utilises the IEEE 802.3 (CSMA/CD) standard. The physical, Data link and the network layers form what is known as the **subnet** [Tanenbaum 1988]. The network layer is concerned with controlling the operation of the subnet. A key design issue is determining how packets are routed from source to destination. However, in broadcast networks (such as Ethernet), the routing is simple, so the network layer is often thin or even nonexistent [Tanenbaum, 1988]. Hence, in the MIDINet system, the network layer does not exist.

The application layer of the MIDINet system corresponds with the application layers of the OSI Reference model. There is no correspondence between the MIDINet layer and the other upper layers of the OSI reference model. We followed the guidelines proposed for the implementation of ArcoNet in relation to what should be the function of each layer [Allik, 1990] and the function of each layer as specified in the OSI Reference model. It was difficult to create distinct layers corresponding to the upper layers of the OSI Reference model. It was decided that there are no well defined tasks of the MIDINet layer that could be assigned to different OSI layers.

The OSI model has been a basis for several network systems. However, there are systems that do not conform to the OSI model. Lack of functions in the MIDINet system that correspond to the upper layers of the OSI reference model does not make the MIDINet system an unstructured system. An example of such a system is the Transport Control Protocol/Internet Protocol (TCP/IP). TCP/IP is not part of OSI, and only corresponds approximately to layer 3 and 4 of the OSI model [Judge, 1988]. The TCP/IP protocol software is organized into four conceptual layers that build on a fifth layer of hardware [Comer, 1988a].

4.6.2 MIDINet messages

The MIDINet messages are encapsulated in an Ethernet packet. Below is the format of an Ethernet packet and the sizes of its parts in bytes.

Preamble	Source	Destination	Type	Data	CRC
8	6	6	2	46 - 1500	4

A MIDINet packet format has two parts: the code and the data parts. The data could be information related to device configuration, connections and MIDI. The format is shown below and the size of each part in bytes. The data part can be as large as the minimum length allowed by an Ethernet packet as shown above. The aim is to maintaim minimum size packets within the system to minimize transmission delays.

Code	Data
3	0 - 1497

```
/* device codes */

#define ADDCODES    "CDA"
#define ADDCODED    "CDB"
#define DELCODES    "CDC"
#define DELCODED    "CDD"

/* midi code */

#define MIDICODE   "MMM"


/* connections codes */

#define ESTCODE     "CLE"
#define BRECODE     "CLB"


/* initialising codes  */

#define WHO     "WHO"
#define LOAD    "LOA"
#define NOTIFY   "III"
```

Figure 4.14 Codes used in the MIDINet layer to distinguish between messages

Figure 4.14 shows a list of codes used to identify messages. The codes go into the code part of the MIDINet packet. Only three types of packets exist, packets containing MIDI, packets containing device/connection data, and packets containing control and initilization messages. If there are any other packets received that do not bear any of the defined codes, they are ignored.

The first letter of the code is used to distinguish between device/connection data, MIDI data

and initialization data. The second letter of the message helps distinguish between device and connections data. Figure 4.15 shows the layout of packets.

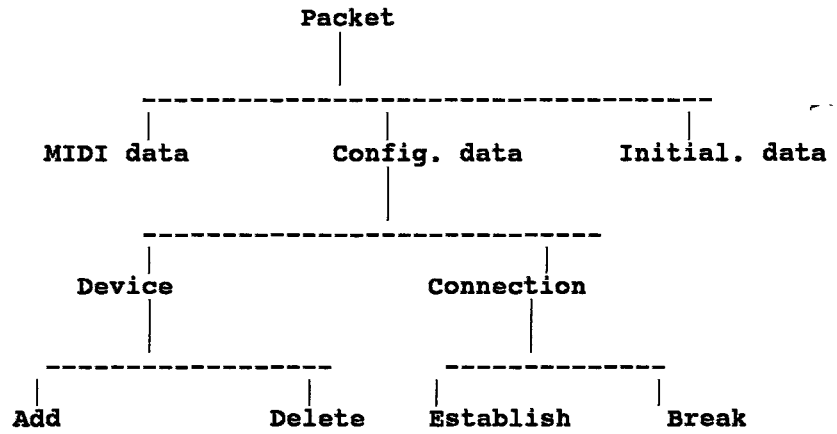


Figure 4.15 MIDINet protocol packet layout

These codes are described below :

C - Configuration

DA - **D**evice to be **A**dded to the list

DR - **D**evice to be **R**emoved from the list

LE - **L**ink to be **E**stablished

LB - **L**ink to be **B**roken

MMM - MIDI carrying message/packet

WHO - The message sent by a MIDINet unit which has just started up to know which other units are already running.

III - Message sent by the already running unit to the requesting unit

LOA - The message sent to request configuration information if a unit has just started up.

In the data part of the MIDINet packet, there could be three types of data, device, connection,

and MIDI data. Device data consists of a MIDINet ID (1 byte), Port ID (1 byte), Channel number (1 byte) and Symbolic name (10 bytes). Connection data consists of source device data similar to device data, a separator (1 byte) and destination device data, also similar to device data. MIDI data consists of source device data and a standard MIDI message. In the case of long *system exclusive messages*, the messages are broken down into several smaller messages that would each fit in a packet (40 bytes).

Any messages sent to a MIDINet unit from a non-MIDINet unit should conform to the specifications above. Otherwise the messages will be discarded. However, a non-MIDINet system should not send initializing messages which ask to be loaded with configuration data unless it has been setup to store and maintain configuration information.

In conclusion, a MIDINet system can be extended in several ways, more especially in the area of MIDI processing. Functions such as MIDI filtering and further MIDI mapping can be implemented. These extensions have been broadly discussed in chapter 7.

For any system to interconnect with the MIDINet system, it must implement the MIDINet protocol. The MIDINet system can run smoothly with any system that utilises Ethernet as its transport medium. However, the non-deterministic nature of Ethernet must be borne in mind. At relatively low traffic the system would perform well. This problem of performance will be dealt with in the next chapter.

Performance Evaluation

5.1 Introduction

Performance evaluation is an essential activity in computer system design. Any system being designed must satisfy certain preassigned performance requirements. Designers use evaluation procedures to construct models of systems which meet these specifications. This enables the designer to visualize a system that is not yet built. The models used are typically mathematical equations, relationships, or observations known about the system.

Performance is a key factor in the design and use of computer systems. The goal of the designer is always to achieve the highest performance at a given cost. The designers should be able to state the performance requirements of their systems. They should be able to compare different alternatives to find the one that best meets their requirements.

The purpose of this chapter is to describe the process conducted to evaluate the performance of the MIDINet system.

5.2 Problem statement

Like any other system, the MIDINet system has constraints and limitations. If two notes are played more than 8 msec apart, they can be individually distiguished [IMA, 1989b]. Hence, the MIDINet system should cause minimal delays between MIDI messages. This point is further addressed by Loy [Loy, 1985] and Moore [Moore, 1988]. They both state that in some musical situations, small delays (few milliseconds) between events are important. These delays between successive events can be readily distinguished. The amount of delay between notes becomes an important determinant of the identity of the sound produced. For example, MIDI can be transmitted at one command per 1 millisecond (3 bytes per command). If several keys (say ten) are played simultaneously on a keyboard, it would take at least 10 msec to transmit this information as the information is transmitted serially. To the receiver it does not appear as if the keys were played simultaneously. Any further delays imposed on this data by any processing device would cause the sound produced to be different from what was

originally intended.

In the MIDINet system, complete messages are processed rather than single bytes. It takes up to 1 millisecond to accumulated each successive message. Within this time the system should have processed the previous message and be ready to transmit it, in order to maintain this rate. Processing time of MIDI messages should be targeted at this value.

5.2.1 Goals

The first step in any performance evaluation exercise is to state the goals of the study and define what constitutes the system by delineating the system boundaries. Each model must be developed with a particular goal in mind. The metrics, workloads, and methodology, all depend upon the goals. Setting goals is not a trivial exercise. Since most performance problems are vague when first presented, understanding the problem sufficiently to write a set of goals is difficult. Once the problem is clear and the goals have been written down, finding the solution is often easier.

In the MIDINet system, the main issue of concern is for a MIDI message to experience as little as possible delay within the system to minimize transmission delays. The process includes reading the MIDI message from a port, parsing the message, determining its destination and either writing it to an out port or transmitting onto the network.

The parameters to be used in the performance evaluation are:

- number of stations: 8-10 workstations.
- number of ports: *4 In and 4 Out* ports.
- message rate : 1000 messages per second. (average length of a message is 3 bytes).
- network capacity: 10 Mbits per second.

After listing the parameters, the next step is to outline a few assumptions. In the worst case, all the 4 In ports of a MIDINet are receiving data. Suppose moreover that this MIDI data has to take the longest route; that is, the data is destined to a device connected to another

MIDINet unit different from the one receiving data. If the data has to be transmitted over the network, how long is the data going to be delayed? If the data has to be buffered, what size buffers ought to be employed?

5.2.2 Services and outcomes

Every system provides a set of services. It helps to create a list of services and possible outcomes as they are useful in selecting the right metrics and workloads. The MIDINet system offers the following services:

- route MIDI messages from one port to another.
- configure devices and establish connections.
- list devices and connections.
- delete devices and break connections.

Configuring devices and establishing connections may be overlooked, since these activities do not occur regularly. The main concern is MIDI data received from the In ports.

5.2.3 Selecting metrics

Selecting metrics constitutes part of the problem of performance analysis. What is the criterion to follow in order to measure performance? In general, the metrics are related to the speed, accuracy, and availability of services. Networks are normally measured by their throughput and reliability; how fast does the system send packets? What is the probability that a packet will get lost or corrupted? All these questions apply to the MIDINet system. When transmitting MIDI data over the network, what sort of delay is imposed by the network?

5.3 Selecting Techniques and Metrics

Three techniques for performance evaluation exist - analytical modelling, simulation, and measurement. Measurement is used mainly if a similar system already exists. When building a new system, simulation and analytic modelling are the best techniques to evaluate

performance.

Analytic models provide a quick and relatively easy means to assess the limitations of a system. This is true if the analysis is limited to elements that can be described by analytical means. In this case of the MIDINet system, analytic models are ideal to determine factors such as how large the buffers should be. What is the probability that the buffer is empty? MIDI data is read from the ports and then buffered. Separate buffers are maintained for different ports. It is important to determine the size limits these buffers should adhere to.

Simulation provides us with the means to model a system on any level of detail. Simulation may be used in a variety of instances because of its flexibility. Since the MIDINet system involves networking, throughput and delay are of great importance. Simulation is ideal to answer throughput and delay questions.

Sometimes it helps to use two or more techniques simultaneously, as one technique may be used to validate the results produced from the other. Analytical modelling and simulation were used for this project.

5.4 Analytic Modelling

Analytical implementations of models rely on the ability of the modeller to describe a model in mathematical terms. For example, if a system can be defined as a collection of queues with service and wait times, queuing analysis can be applied.

5.4.1 Queuing analysis

Queuing theory is a branch of probability theory which involves the mathematical study of queues. The formation of queues is a common phenomenon that occurs whenever requests for a service facility exceed the current capacity of the facility.

The prime motivation for performing queuing analysis is to assess local system behaviour under a variety of assumptions, initial conditions, and operational scenarios. The general process of analytical modelling involves mapping the behaviour of a complex system onto a relatively simple system, solving the simpler system for the measures of interest, and extrapolating the results back to the complex system. The different components of system

behaviour are represented as processes that have calculable statistics. The assumption is that the system to be studied can be adequately represented by a queuing system.

The MIDINet system can be represented by a queuing system. MIDI data is received from the In ports. This MIDI data has to be queued in the buffers and await service.

5.4.2 Basic structure of a queuing system

The basic behaviour assumed by most queuing systems can be modelled as a server with a sequence of customers. If the arriving customer finds the server busy, it joins the queue associated with that server and waits. In such systems, customers arrive at some rate, queue for the service on a first-come-first-served basis, receive service, and exit the system. This kind of model, with jobs entering and leaving the system, is called an "open" model [Fortier, 1990]. On the other hand, in a "closed" model, the number of jobs remains constant; jobs that have been serviced join the queue. The MIDINet system is an example of an "open" model. MIDI messages are regarded as customers and the MIDINet units are servers. MIDI messages are initially queued, later processed, and finally routed to various destinations.

5.4.3 Rules for all queues

Some key variables used in the analysis of single queues and inter-relationships are now introduced [Jain, 1991] (Note-E denotes an expected value).

- τ interarrival time, that is, the time between two successive arrivals.
- λ mean arrival rate = $1/E[\tau]$: the arrival rate of MIDI messages.
- s service time per job: in the MIDINet system, this is the service time per MIDI message.
- μ mean service rate per server, = $1/E[s]$: mean service rate per MIDINet unit.
- n number of jobs (MIDI messages) in the system. This is also called **queue length**. This includes jobs currently receiving service and those waiting in the queue.

n_q number of jobs waiting to receive service. This is always less than n, since it does not include the jobs currently receiving service: in the MIDINet system - the number of MIDI messages in the buffers waiting to be processed.

n_s number of jobs receiving service. There can only be one MIDI message being processed at a particular point in time.

r response time or the time in the system. This includes both the time waiting for service and the time receiving service. In the MIDINet system this is the time it takes a MIDI message from when it is read until it is transmitted to an Out port.

The simplest and most useful model of arrival pattern is the completely random arrival process, usually called the Poisson arrival process [Jain, 1991]. The service rate parameter is defined in a way similar to the arrival rate. This rate is also an average rate that defines how many customers are processed per unit time when the server is busy. The service demands of the customers are identically distributed with a common distribution, the service distribution. In more complicated cases, the customers may be classified into several different types, each with its own distribution of service demand. In the MIDINet system, all MIDI messages go through similar processing, which makes the analysis simple.

In a queuing system, the number of jobs can grow continuously and becomes infinite. In this case the system is said to be unstable. For the system to become stable, the mean arrival rate should be less than the mean service rate.

5.4.4 Kendall Notation

A queue is specified by its arrival process, service mechanism, and queue discipline, hence a notation was developed to succinctly define a queue. The notation is called the *Kendall notation* [Claiborne, 1990]. It is expressed as

$$A/B/C/K/m/z$$

where:

A Arrival process

B Service process

C Number of servers. The simplest case is a single server. A MIDINet unit is regarded as a server and only a single server exists.

K Maximum capacity. This is the number of customers allowed to wait in a queue for service. The simplest case is when **K** = ∞. The MIDINet system assumes finite buffers.

m Population of customers. The simplest case is when **m** = ∞. If **m** is finite, the arrival rate may decrease when a large percentage of customers join the queue. As in the case of **K**, when **m** = ∞ , it is generally omitted from the notation. In the MIDINet system, the arrival rate of MIDI messages is not dependent on the number of messages already in the system. The number of MIDI messages received varies randomly.

z Service discipline. The principal service disciplines are **FIFO** and **FILO**. If **z** is omitted, FIFO is assumed. MIDI messages are serviced using the **FIFO** method.

5.4.5 The M/M/1 System

The **M/M/1** queuing system is characterized by a Poisson arrival process and exponential service time distributions, with one server, and a FIFO queue ordering discipline. There are no buffer size limitations in this system. Despite its simplicity, the **M/M/1** queue provides an accurate representation of most of the processes associated with data communications. In addition to the general rules for all queues, the **M/M/1** system has the rules shown in Box 1 [Jain, 1991].

In the MIDINet system, in the worst case, MIDI bytes can arrive at the rate of **3000 bytes/sec**. On average, this results in **1000 messages/sec**. Therefore, the rate of four ports will be **4000 messages/sec**. Although MIDI messages received from different ports are stored

1. ***Parameters:***
   λ = ***arrival rate in jobs per unit time***
   μ = ***service rate in jobs per unit time***
2. ***Traffic intensity:*** $\rho = \lambda/\mu$
3. ***Stability condition: Traffic intensity*** ρ ***must be less than*** 1.∞
4. ***Probability of zero jobs in the system:*** $p_0=1-\rho$
5. ***Probability of n jobs in the system:*** $p_n=(1-\rho)\rho^n$, $n=0,1,...,\infty$
6. ***Mean number of jobs in the system:*** $E[n]=\rho/(1-\rho)$
7. ***Variance of number of jobs in the system:*** $Var[n]=\rho/(1-\rho)^2$
8. ***Probability of k jobs in the queue***

$$P(n_q=k)=\begin{bmatrix}1-\rho^2, & k=0\\ (1-\rho)\rho^{k+1}, & k>0\end{bmatrix}$$

9. ***Mean number of jobs in the queue:*** $E[n_q]=\rho^2/(1-\rho)$
10. ***Variance of number of jobs in the queue:***

$$Var[n_q]=\rho^2(1+\rho-\rho^2)/(1-\rho)^2$$

11. ***Mean response time:*** $E[r]=(1/\mu)/(1-\rho)$

Box 1 Specific rules for M/M/1 queue

separately, the system can be mapped to a simple system. The assumption is made that the MIDI messages are queued in one buffer since they are serviced by a single processor.

The time it takes a MIDI message to be transmitted from the source to its destination must be targeted at **1 ms** or less. It was explained in the previous sections that it takes roughly **1 ms** to accumulate a MIDI message from a port. So, for the system to read successive messages without losing any, it must take **1 ms** to process a message so as to read the next message. Thus, the time it takes a message to be read, parsed, transmitted over the network, and to arrive at its destination must be **1 ms** at most. The question is, how long should a message reside within a MIDINet unit before it is transmitted over the network or written to an Out port?

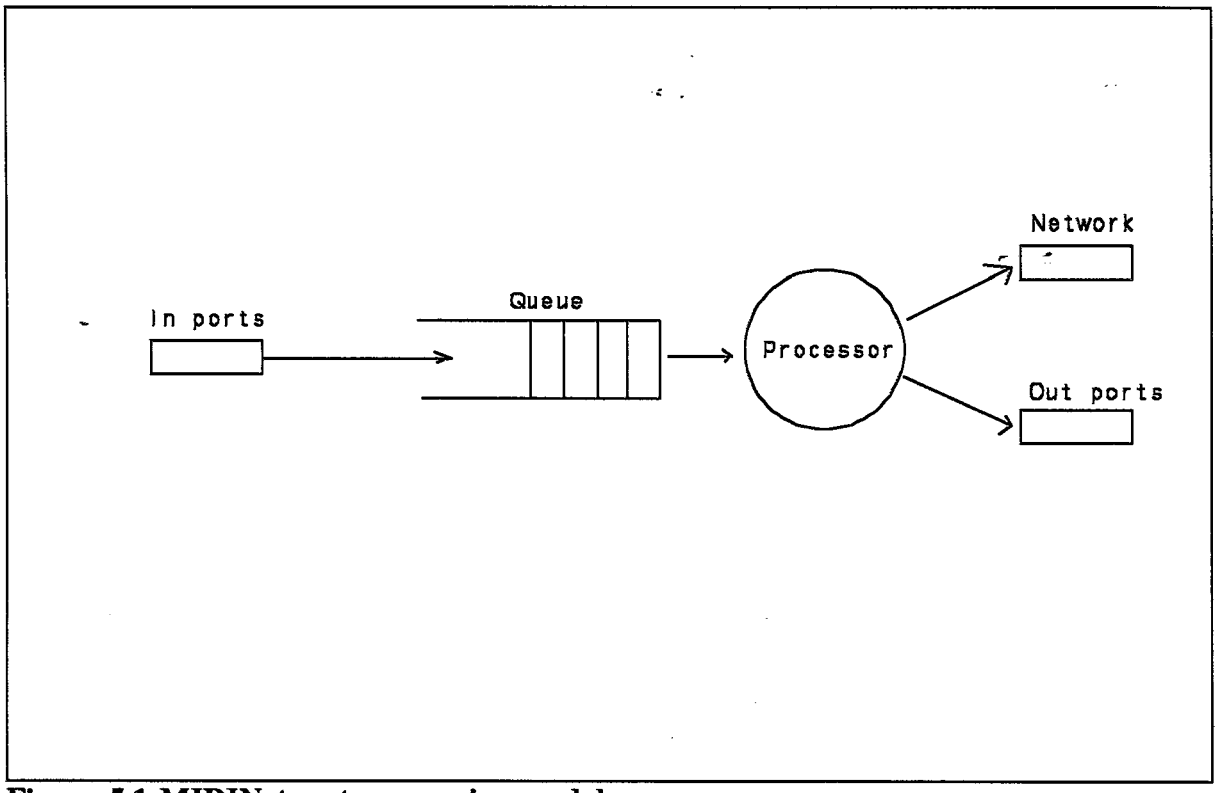


Figure 5.1 MIDINet system queuing model

In the MIDINet system, it is assumed MIDI messages arrive according to a Poisson distribution. MIDI messages are received from four ports. Figure 5.1 shows the queuing model for the MIDINet system. For the sake of the analysis it is assumed that these messages are merged to queue for a single processor. Merging of n Poisson streams with mean rate λ_i results in a Poisson stream with mean rate λ [Jain, 1991]. Using **M/M/1**,

Arrival rate λ = 4000 messages/sec
Service rate μ = ?

Stability condition: $\rho < 1$ i.e

$\lambda/\mu < 1$
$4000/\mu < 1$
$\mu > 4000$

The service rate must be higher than 4000 messages/sec. However, if it takes the system the maximum amount of time to process a message, that is **1 ms** then :-

Service rate μ = 1/0.001 = 1000 messages/sec

therefore λ/μ = 4000/1000 = 4 > 1 ; hence, the system becomes unstable. Under this condition, the buffer will overflow and the messages will be lost.

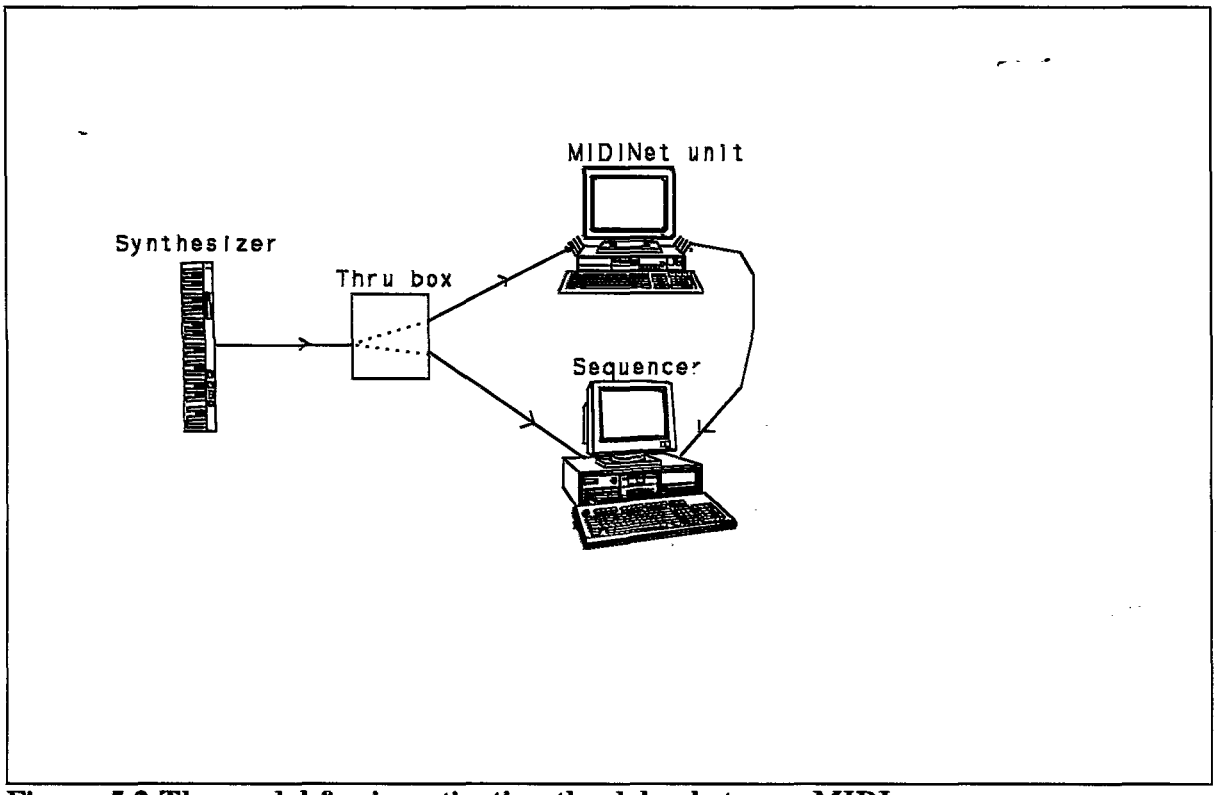


Figure 5.2 The model for investigating the delay between MIDI messages

An experiment was conducted in which MIDI messages were fed to a sequencer from a synthesizer. The model is shown in figure 5.2. Using a MIDI thru box, MIDI messages were sent directly to the sequencer, whilst the same messages were transmitted via a MIDINet unit to the sequencer. The MIDI messages emanating from the MIDINet unit and those sent directly were recorded on separate tracks on the sequencer. The notes on one track were transposed to change their pitch and the tracks were subsequently merged so as to be compared. The difference between the times at which similar messages were recorded on the sequencer were recorded. The maximum delay was found to be *2.7ms*.

It has already been stated that it takes **1ms** to accumulate a MIDI message. It also takes **1ms** to transmit it, hence , the system takes **0.7ms** to process a message. This value was verified by carrying out an experiment where MIDI messages were read from a port and the time it

takes to execute processing instructions was recorded. The time was found to be in the region of 0.7ms.

Using the **M/M/1** technique again :

Arrival rate λ = 4000 messages/sec
Service rate μ = 1/0.0007 = 1428 messages/sec

Stability condition: $\rho < 1$ i.e
$\lambda/\mu < 1$

but 4000/1428 $\not<$ 1
under these conditions the system becomes unstable.

However, in a music setup, MIDI messages do not arrive continuously. 4000 messages/sec is the worst case for a MIDI setup. An experiment was conducted in the Rhodes Computer Music Network (RHOCMN), to determine the frequency of messages produced by a sequencer playing a song with several tracks. With the help of a MIDI THRU box, these messages were fed to all four ports. The worst case recorded, was 125 messages/sec.

Using the **M/M/1** technique again:
Arrival rate λ = 125 messages/sec
Service rate μ = 1/0.0007 = 1428 messages/sec

Stability condition: $\rho < 1$ i.e
$\lambda/\mu < 1$

and 125/1428 = 0.09 < 1
hence the system is stable.

The probability of zero messages in the system:

$$p_0 = 1 - \rho = 1 - 0.09 = 0.91$$

The probability of zero messages in the system is not 1, hence, it is reasonable to assume that

there will be a few messages waiting for service.

The mean number of messages in the system:

$$E[n] = \rho/(1 - \rho)$$
$$= 0.09/0.91$$
$$= 0.1$$

The mean time which messages spend in the system is equivalent to the response time of the system:

$$E[r] = (1/\mu)/(1 - \rho)$$
$$= 0.0007/0.91$$
$$= 0.77 \text{ milliseconds}$$

This response time is below the target, **1ms**. The analysis carried out above depicts a stable system. It points to a system with reasonable size buffers. However, the sizes of the buffers are a major concern in the MIDINet system where memory is a vital asset. In the next section, this point is addressed in more detail.

5.4.6 The M/M/1/K System

The realistic variation on the basic **M/M/1** system is a system with a finite queue size. In this system, once the queue is full, new arrivals are lost. In addition to the general rules of all queues, the **M/M/1/K** system has the rules shown in Box 2 [Jain, 1991].

In the MIDINet system, another major concern is the length of message buffers. The investigation can be conducted by the use of **M/M/1/K** technique.

Arrival rate λ = 125 messages/sec
Service rate μ = 1/0.0007 = 1428 messages/sec
K = assume maximum buffer size of 10.

There is no particular reason why K is assumed to be 10; this is for investigation purposes.

Traffic intensity ρ = 0.09

The probability of zero messages in the system:

$$\begin{aligned} p_0 &= (1 - \rho)/(1 - \rho^{K+1}) \\ &= 0.91/(1 - 0.09^{11}) \\ &= 0.91/0.9999 \\ &\approx 0.91 \end{aligned}$$

The probability of buffer overflow = P_n (where n = more than 10 messages in the system)

$$\begin{aligned} p_n &= ((1 - \rho)/(1 - \rho^{K+1}))\rho^{10} \\ &= (0.91/0.999) * 3.5 * 10^{-11} \\ &= 3.17 \text{ X } 10^{-11} \\ &\approx 31 \text{ messages per 100 billion messages.} \end{aligned}$$

To minimize the number of messages lost or to reduce the probability of losing messages, the size of the buffer has to be increased. Taking into consideration that messages from different ports are stored in different buffers, and it is assumed that each buffer can store a maximum of 10 messages, then, for statistical purposes, a buffer with 40 spaces can be assumed.

The probability of buffer overflow = P (n = more than 40 message in the system)

$$\begin{aligned} p_n &= ((1 - \rho)/(1 - \rho^{K+1}))\rho^{40} \\ &= (0.91/0.999) * 1.48 * 10^{-42} \\ &= 1.3 \text{ X } 10^{-42} \end{aligned}$$

The probability is extremely low, in which case few, if any, messages would get lost.

Lastly, the mean number of jobs in the system :

$$\begin{aligned} E[n] &= \rho/(1 - \rho) - ((K + 1)\rho^{K+1})/(1 - \rho^{K+1}) \\ &= 0.09/0.91 - 5.5 * 10^{-42}/0.9999 \\ &= 0.098 \end{aligned}$$

From the above analysis it is clear that the system will rarely overflow.

1. ***Parameters:***

   λ = *arrival rate in jobs per unit time*
   μ = *service rate in jobs per unit time*
   K = *buffer size*

2. ***Traffic intensity:*** $\rho=\lambda/\mu$

3. ***The system is always stable:*** $\rho<\infty$

4. ***Probability of zero jobs in the system***

$$P_0 = \begin{cases} \dfrac{1-\rho}{1-\rho^{K+1}}, & \rho \neq 1 \\ \dfrac{1}{K+1}, & \rho = 1 \end{cases}$$

5. ***Probability of n jobs in the system:***

$$P_n = \begin{cases} \dfrac{1-\rho}{1-\rho^{K+1}}\rho^n, & \rho \neq 1 \quad 0 \le n \le K \\ \dfrac{1}{K+1}, & \rho = 1 \quad 0 \le n \le K \\ 0 & n > K \end{cases}$$

6. ***Mean number of jobs in the system:***

$$E[n]=\frac{\rho}{1-\rho}-\frac{(K+1)\rho^{K+1}}{1-\rho^{K+1}}$$

7. ***Mean number of jobs in the queue:***

$$E[n_q]=\frac{\rho}{1-\rho}-\rho\frac{1+K\rho^K}{1-\rho^{K+1}}$$

Box 2 Specific rules for M/M/1/K queue

At this stage, a consideration is made whether the MIDINet system can be extended to have more than two Ins and two Outs? If an additional card is implemented in a MIDINet unit, how would this affect its performance?

On average each port generates **31 message/sec** (125/4). An additional card adds two Ins; therefore the arrival rate becomes **186 message/sec** (31x6 for six ports). The service rate of 1428 messages/sec is assumed.

Using the **M/M/1** technique:

Arrival rate λ = 186 messages/sec

Service rate μ = 1/0.0007 = 1428 messages/sec

Stability condition: $\rho < 1$ i.e

$$\lambda/\mu < 1$$

and $186/1428 = 0.13 < 1$

hence the system is stable.

The response time of the system is then :

$$E[r] = (1/\mu)/(1 - \rho)$$
$$= 0.0007/0.87$$
$$= 0.8 \text{ milliseconds (previously 0.77ms)}$$

An additional card will impose further delays in processing of data. However, the response time is still within the time limit of **1ms.**

Taking the buffer length into consideration, the **M/M/1/K** technique is applied. Assuming a maximum of 10 messages in a buffer (K = 10) then the probability of zero messages in the system:

$$p_0 = (1 - \rho)/(1 - \rho^{K+1})$$
$$= 0.87/(1 - 0.87^{11})$$
$$= 0.87/0.9999$$
$$\approx 0.87$$

The probability of zero message in the system is reduced from **0.91** that was obtained previously (section 5.4.5). More messages will be queued for service. The question is will the buffers ever overflow?

The probability of buffer overflow = P_n (n = more than 10 messages in the system)

$$p_n = ((1 - \rho)/(1 - \rho^{K+1}))\rho^{10}$$
$$= (0.87/0.999) * 1.3*10^{-9}$$
$$= 1.2 * 10^{-9}$$
$$\approx 12 \text{ messages per billion messages.}$$

Increasing the buffer size to 40, the probability of buffer overflow = P_n (K = more than 40 messages in the system)

$$p_n = ((1 - \rho)/(1 - \rho^{K+1}))\rho^{40}$$
$$= (0.87/0.999) * 3.6 * 10^{-36}$$
$$= 3.1 * 10^{-36}$$
extremely few messages

The analysis shows that it is still safe to have an additional MUART card installed into a MIDINet system. This will make the MIDINet system an even more powerful system.

5.5 Simulation

Simulations take a model or description of a system and, based on this, perform experiments which enable the analyst to determine the behaviour of a system. Simulation provides a means to visualize a system that is not yet built, to analyze a system to determine critical elements, and to act as design assessor in order to evaluate proposals. It also can forecast possible issues in future developments or additions to a system [Fortier, 1989].

However, in an attempt to simulate, there are hidden problems [Schoemaker, 1978]. An erroneous model might be designed by the analyst. A false model can mean unintentionally false projection of the object system onto the target system being simulated. The model can refer to a wrong implementation, which leads to false results, hence an invalid interpretation of results. Another problem can be the use of inappropriate tools. Selecting an appropriate language is probably the most important step in the process of developing a simulation model. Using a general purpose language can mean laborious progress of the modelling process, while the use of a special purpose language enables a better view of the modelling environment.

5.5.1 NETWORK II.5

It has been pointed out that one of the dangers of simulation is the selection of wrong tools. An incorrect decision during this step may lead to long development times, incomplete studies, and failures. For the purpose of this work, NETWORK II.5 was employed [CACI, 1992]. One of the major reasons for making this choice was its widespread use within the Computer Science Department at Rhodes University.

NETWORK II.5 is a design tool which takes a user-specified computer system description and provides measures of hardware utilization, software execution, and conflicts. It is intended for the user who designs or specifies computer systems and local area networks. It is used both to evaluate the ability of a proposed system configuration to meet the required workload, and to evaluate competing designs [Mosala, 1992]. NETWORK II.5 is designed to model a wide variety of computer architectures, from a single processor to a complex system of processors and storage devices connected in a network. It is extremely flexible, allowing the portions of a computer system of special interest to be modelled on a detailed level, while the rest of the system is modelled on a coarser level.

5.5.2 Performance

Local Area Network simulation provides us with a means to model a local area network to any level of detail we deem necessary. In networks, the characteristics of delay and throughput are studied. The throughput of a LAN refers to the amount of information it can carry per unit time. Throughput is measured as a function of the total load offered to the network. Delay refers to the time that a station having a message to send must wait before it is allowed to access the medium. In measuring performance, it is also usual to evaluate delay as a function of the total traffic load being offered to the network by the attached stations.

In the MIDINet system, MIDI messages are transmitted over the network. It is important to investigate the delay the network imposes on these messages.

5.5.2.1 Response time

In a real-time system, it is always important to monitor the response time. It has already been

stated that there is a **1ms** upper limit on the processing time of MIDI messages. It is necessary to analyze the response time under different conditions.

The response time is the length of time it takes to process an input message and send it out. It consists of many parts (queuing, service time) and is thus difficult to determine before the real system exists. Rough estimates can be determined using mathematical analysis, but the system can be better studied using a simulation model. It is possible to determine the length of time it takes for a MIDI message to be transmitted from its source over the network to its destination.

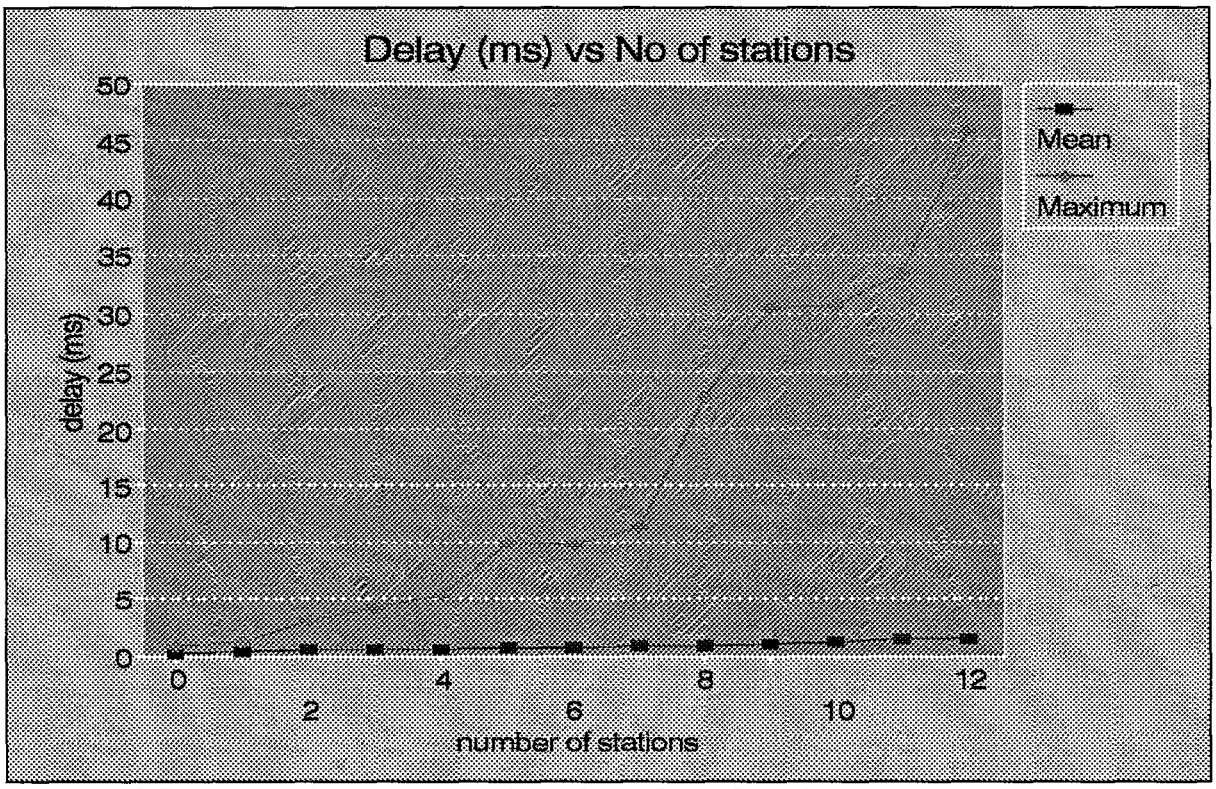


Figure 5.3 Response time as a function of number of stations

A simulation model was built where two stations are communicating over the network and other stations are generating traffic on the network. Up to 15 stations were used in the simulation model. Minimun size packets allowed by Ethernet were utilised. The main idea behind the model was to investigate the non-deterministic nature of Ethernet. The service time at the communicating stations was set to a very low value, since only the queng and trasmission times were being inverstigated (on the report in appendix 5, it is the value of MESSAGE DELIVER TIME). After a station was added, the response time between the two

communication stations was recorded. Figure 5.3 shows a response time as a function of number of stations. The response time increases with the number of stations. This is due to traffic from the other stations. In this experiment, the two communicating stations model two MIDINet units sending each other MIDI data.

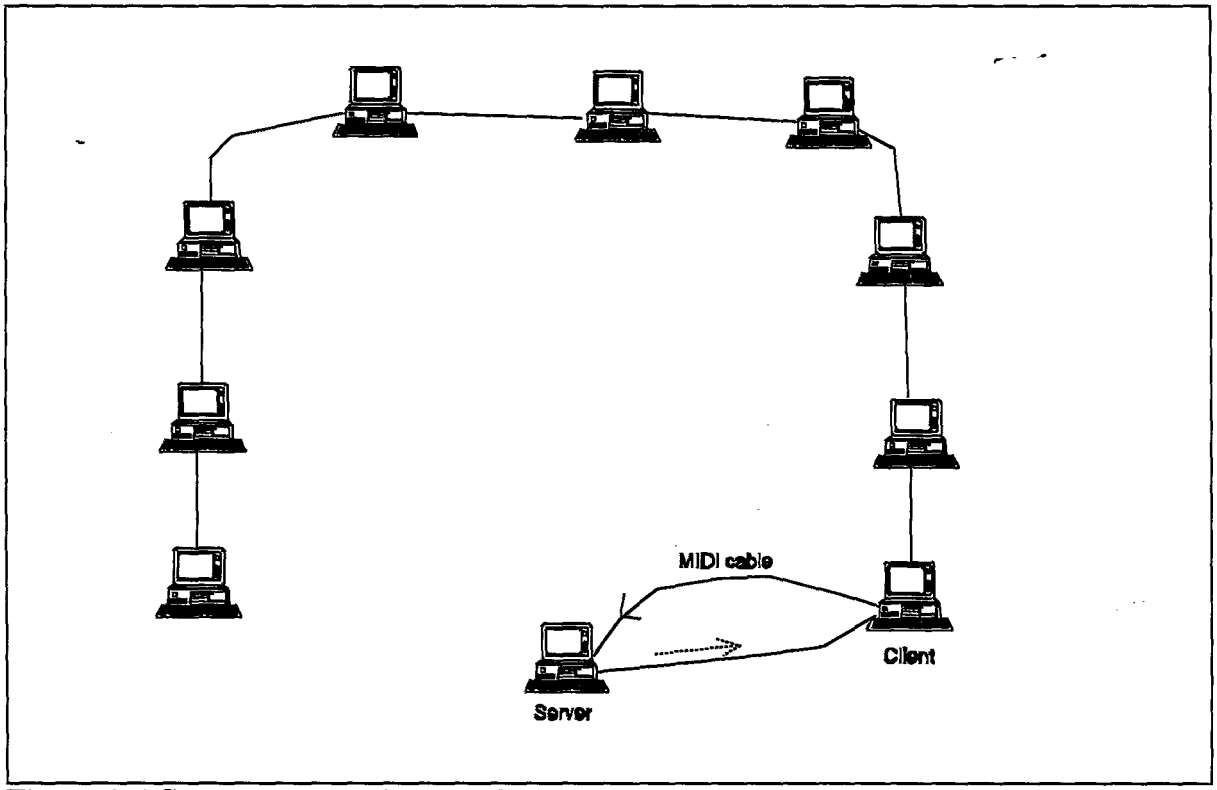


Figure 5.4 Setup for investigating the response time on the network

5.5.2.2 Packet size

Packet size is one of the factors affecting performance on the network. Increasing the size of the packets leads to longer propagation delays. On the other hand, shorter packets impose overheads, since each packet carries a fixed-length header (under Ethernet). Longer packets reduce overhead and allow higher throughput at the cost of increased delays [Kuo, 1981]

In the MIDINet system, short packets are inevitable. The longest message is a system exclusive message. This message has to be broken down to form several messages that could be packed in a shortest standard Ethernet packet. Using the smallest packets, throughput is low, but with the advantage of reduced delays. The MIDINet system as a real-time application uses shorter packets to achieve low delays.

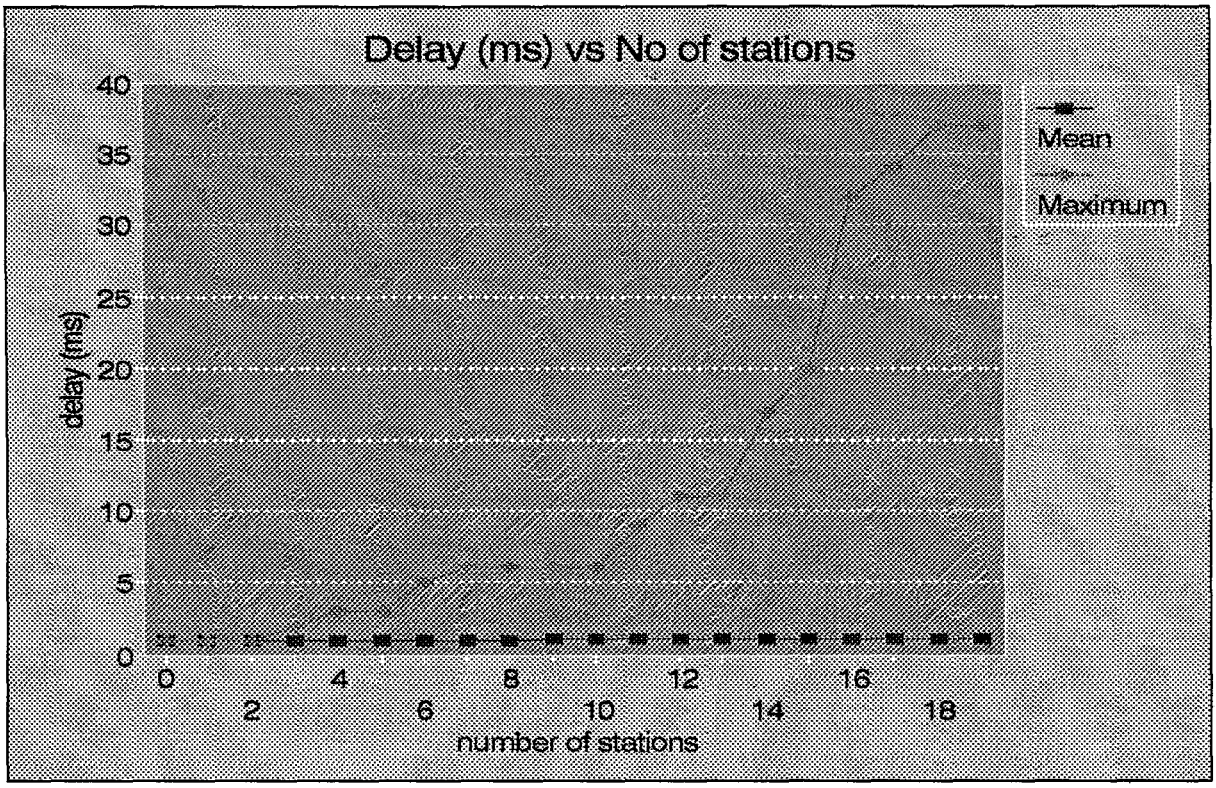


Figure 5.5 Response time as a function of number of stations (Stations sending data continuously)

5.6 Operational analysis

Operational analysis is concerned with extracting information from a working system that is used to develop projections about the system's future operations. The method of operational analysis can be used to derive meaningful information that can be used together with analytical modelling and simulation techniques. It deals with the measurement and evaluation of an actual system in operation. Several experiments were conducted to determine the behaviour of the MIDINet system.

An experiment was set up as shown in figure 5.4. The aim of the experiment was to further investigate the non-deterministic nature of Ethernet. A MIDINet unit sends a byte via an Ethernet to another MIDINet unit. The receiving unit returned the same byte back via a MIDI cable to complete a cycle. The time difference between receipt and transmission was measured. There are other computers on the network continuously sending arbitrary packets on the network to create traffic. The experiment is repeated for different numbers of computers on the network, that is, a varied load.

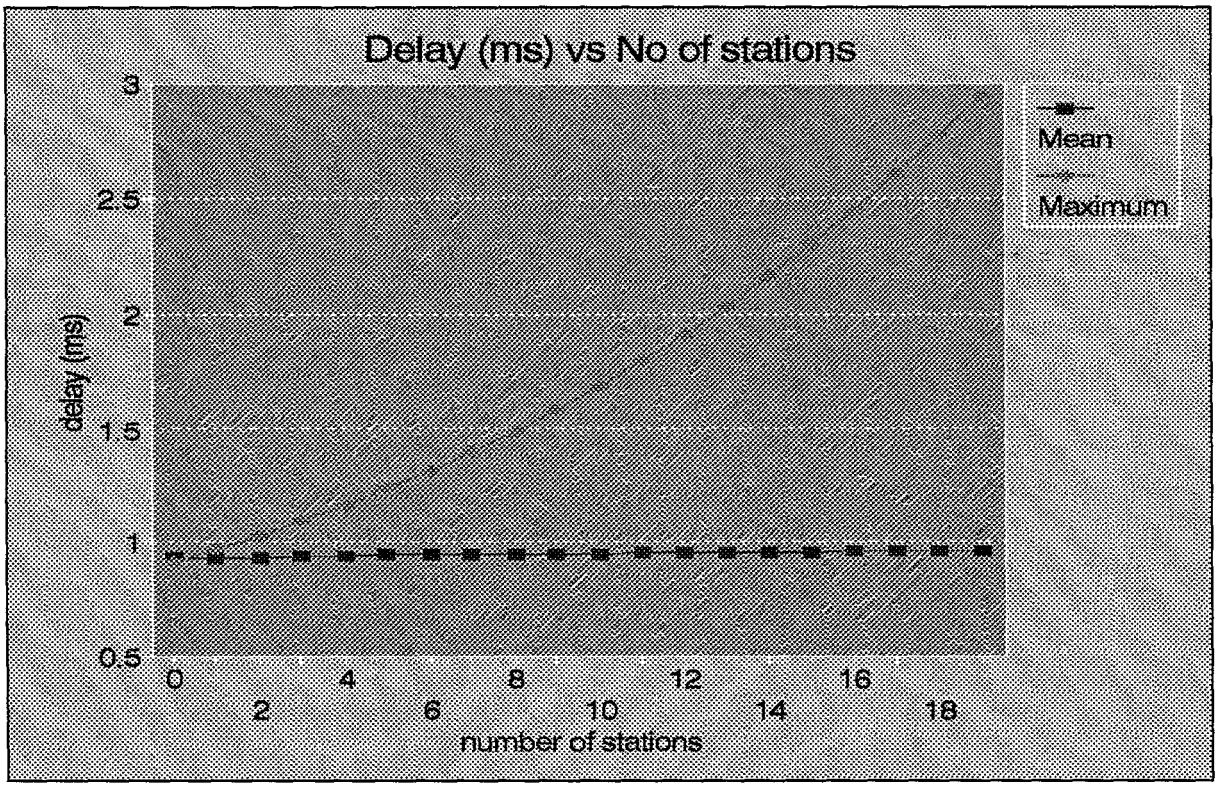


Figure 5.6 Response time as a function of number of station (Station sending data randomly)

The results of the experiment are shown in figure 5.5. The figure shows two graphs, one for maximum delay reached and the other shows the average of 10000 readings. The maximum delay increases exponentially with the number of computers on the network. From the graph it is clear that the response time goes beyond the 8ms limit.

However, having computers sending packets continuously does not model the real situation. In order to more realistically model the real world situation, the experiment was modified, so that computers send arbitrary numbers of packets (1-10) at random intervals (1-500ms). Figure 5.6 shows the results obtained in the experiment. From the graph, it is clear that the delays are fairly low, that is, they are within the limit of 8ms.

In conclusion, the mathematical analysis showed a stable system that could have reasonable buffer sizes. An additional MIDI card in a unit will cause a slight decline in the performance of the whole system. Simulation and operational analysis showed that the non-deterministic nature of Ethernet can introduce problems in the system. The results confirmed that Ethernet performs well under a light to medium load. However, as more devices are added to the

network, response time increases. The next chapter discusses measures taken to overcome Ethernet's transmission problems.

Protocols

6.1 Introduction

The previous chapter demonstrated the performance of an Ethernet network. Ethernet networks are well known for their non-deterministic nature. This usually makes them inappropriate for real-time communication applications. Ethernet networks operate efficiently under light to medium traffic. As more workstations attempt to access the network, network throughput declines [Kuo, 1981].

We have seen that response time increases as more workstations try to access the network. This is caused by packet collisions. When two packets collide, the CSMA/CD algorithm requires that both backoff and attempt the transmission later. More traffic creates more collisions. This is a drawback in a real-time application. This chapter explores other protocols that were considered for the MIDINet system.

6.2 Alternative protocols

Other protocols were taken into consideration in the quest to avoid Ethernet problems. Token Bus was the first one to be considered. It is deterministic and has the ability to prioritize the transmission of packets [Halsall, 1988]. Several queues are maintained for different priority levels. Data to be routed is checked for priority and gets queued in an appropriate queue. This priority scheme is suitable for the MidINet system. MIDI data can be assigned the highest priority. Any other data, such as device and connection data will have lower priority.

Token Ring, another alternative, offers high throughput and efficiency at high load. Like Token Bus, it permits prioritization. As has been pointed out, the priority scheme suits the MIDINet system. The major drawback of Token Ring is the presence of a centralized monitor function which introduces a critical component. A monitor detects and corrects errors in the system [Hutchison, 1988]. If the monitor does not perform properly, then the network becomes unstable.

High speed media like fiber optics were considered. FDDI (Fiber Distributed Data Interface) is a high performance fiber optic Token Ring. It can be used in the same way as any LANs mentioned (Token Ring, Token Bus and Ethernet), but with a greater bandwidth [Tanenbaum, 1988]. However, fiber optic and the associated FDDI interface cards are expensive.

Ethernet has the advantage of being widely used. This implies that a system built on top of it can interface easily with other systems. RHOCMN already uses Ethernet transmission. To find a solution to Ethernet transmission problems is preferable to changing the whole system.

The above considerations inspired the design of a protocol that would run on top of Ethernet, Netlink. The motivation behind Netlink is to have a protocol above Ethernet, that combines the good features of Ethernet and token passing protocols (their deterministic nature). It should therefore perform well in real-time applications.

6.3 Description of Netlink protocol

According to the ISO OSI standard: "the (N)-protocol is the set of semantic rules determining the information exchange among the (N)-entities in such a way that (N)-functions should be fulfilled" [Tarnay, 1991]. Where N is an arbitrary number. The (N)-protocol means a set on N rules of communication between two entities. The (N)-function is a set of function that can be carried out between two entities.

The syntactic rules determine the format of messages exchanged between communicating entities; the semantic rules specify the procedures, and the correct and proper sequence of instructions exchanged between communicating entities. The semantic rules also comprise a way of specifying the timing and the time specific characteristics.

The messages exchanged between entities either contain data fields, control information, or both data and control information. The procedures include the functions such as data transfer, addressing, error handling and flow control. The time specifications determine the time-out limits of waiting. Therefore, a protocol is defined by formats, procedures and time behaviour.

The protocol must recover from errors. Typical errors that could occur are: a packet is lost, an entity dies or the sequence of packets is mixed up. These errors may arise from erroneous packet formats or timing. The Netlink protocol described below takes these errors into

account. The Netlink protocol is closely related to the Token Bus protocol. It is derived from the concepts described by Tanenbaum [Tanenbaum, 1988].

Physically, the Netlink is a linear or tree-shaped cable onto which stations are attached as shown in Figure 6.1. Logically, the stations are organized into a ring, with each station

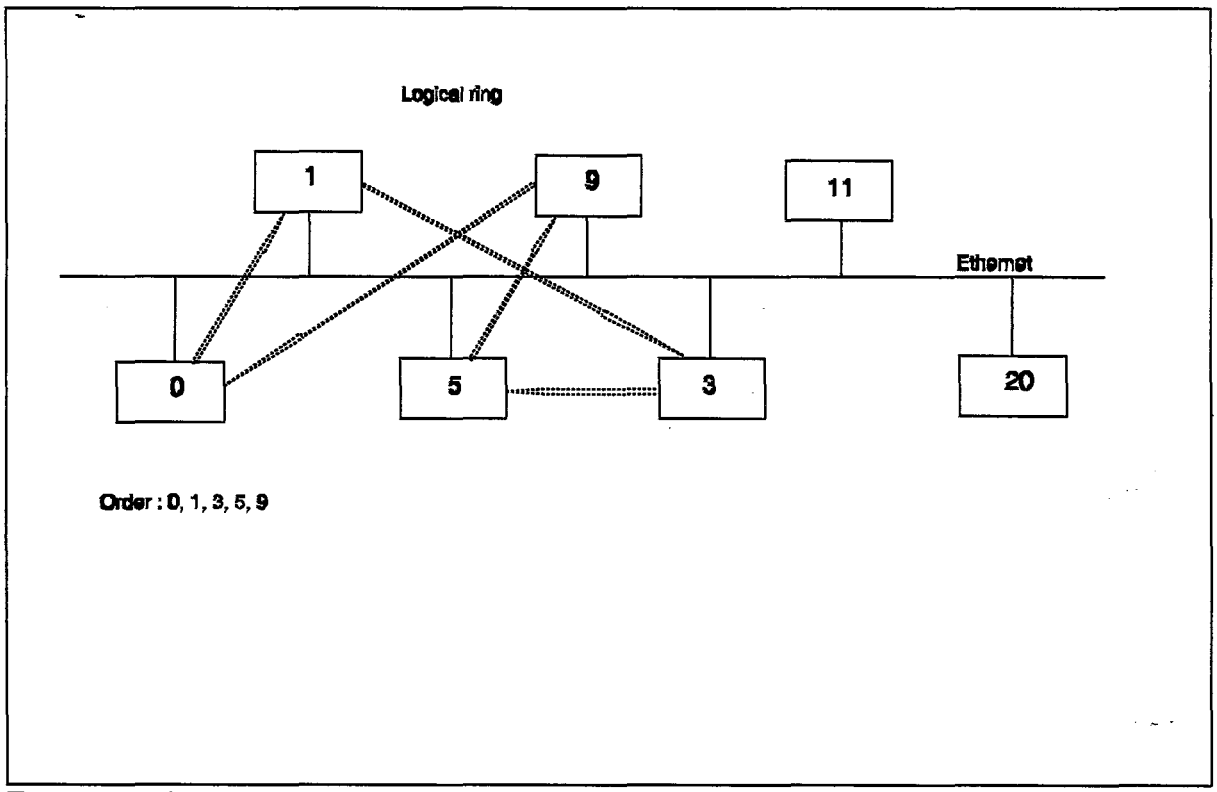


Figure 6.1 A netlink protocol physical connections and logical ring

knowing the address of the "previous" and "next" stations. Stations are assigned unique numbers. When the ring is initialized, the lowest numbered station (Controller) may send the first packet. Then, it passes permission to its immediate neighbour by sending the neighbour a special control packet called a **token**. The token propagates around the logical ring, with only the token holder being permitted to transmit packets. Since only one station at a time holds the token, collisions do not occur.

The physical order in which the stations are connected to the cable is not important. Since the cable is inherently a broadcast medium, each station receives each packet. A station should only transmit a single data packet when it receives a token. Since the packets are all

broadcast packets, the token can be passed encapsulated in a data packet. This reduces the overhead of having to transmit a separate packet (token) after transmitting data. If a station has no data to transmit, then it passes the token to its logical neighbour.

6.4 Communication in Netlink

The state transition diagram of the Netlink protocol is shown in figure 6.2. When the ring is initialized, stations are inserted in order of station number, from lowest to highest. Token passing is also done from lowest to highest. If a station has no data to transmit, then it passes the token immediately upon receiving it.

Netlink defines two priority classes for traffic, 0 and 1, with 0 being the lowest and 1 the highest. Each station maintains two packet queues for each priority level. When a token arrives at a station, the station drains the higher priority queue first. This priority scheme can be used to implement real-time traffic such as MIDI messages.

6.5 Ring Maintenance in Netlink

From time to time some stations will want to join the ring, others will want to leave the ring. Each station maintains the information about the current ring internally. Each station is aware of every other station in the ring.

A station can fall into one of three categories; **Non Active, Active Not Transmitting** and **Active & Transmitting**. Non active stations are those not participating at all. **Active Not Transmitting** comprise stations in the ring, but which have no data to transmit for a while. **Active & Transmitting** stations have data to transmit and are currently in the ring.

Everytime it gets the token, the controller solicits bids from stations currently in the **Active Not Transmitting** category that wish to enter the **Active & Transmitting** category, by sending the *SOLICIT_SUCCESSOR_ANT* packet shown in Figure 6.3. This packet is sent to a specific station. The stations are polled in numeric order. A single station is polled on each turn. A polled station responds by sending a *SET_SUC/PREDE_CESSOR* packet if it wants to join the ring.

Periodically, the Controller solicits bids from stations currently in the **Non Active** category

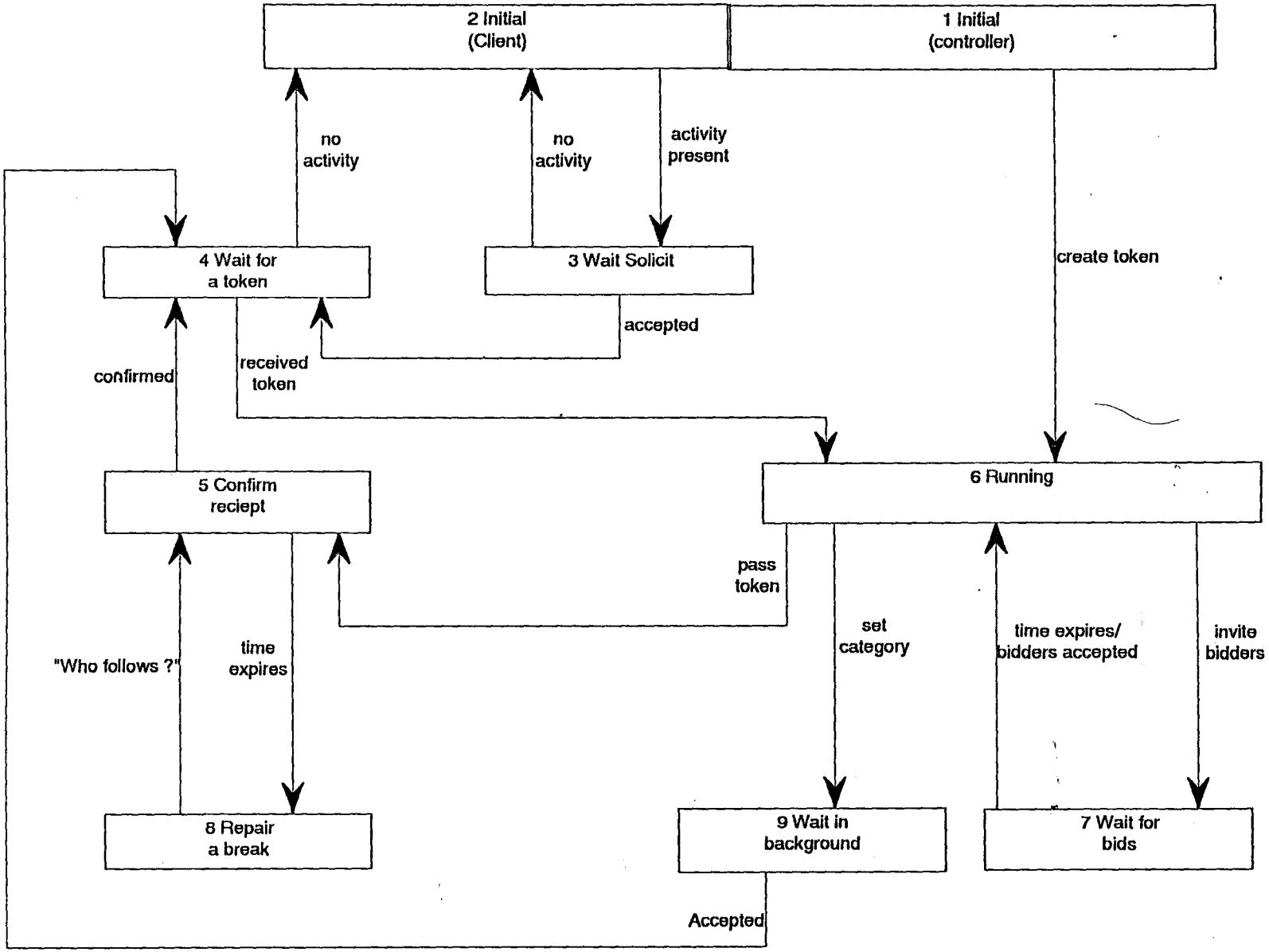


Figure 6.2 State transition diagram for Netlink protocol

that wish to enter **Active & Transmitting** category by sending the *SOLICIT_SUCCESSOR_NT* packet. This packet is sent to a specific station. The stations are polled in numeric order.

On Ethernet, the time it takes a bit to travel the length of the cable is called 'slot time', τ. It is a parameter used to characterize Ethernet collision handling. To accommodate the longest path allowed by Ethernet (2.5 km), this time is set to 51.2 microsec [Tanenbaum, 1988]. The controller waits for approximately **2τ+t** for a bidder to respond (**t**=100 microsec). The time **t** is added in order to cater for the processing that takes place on the bidder in order to respond. When the time expires, the controller polls another station or passes the token. If the polled station responds, then it is inserted into the ring, and assumes **Active & Transmitting** status. The bidding station responds with the *SET_SUC/PREDE_CESSOR* message.

Packet Code	Name	Description
00000001	Solicit_successor_ant	Allow stations to enter the ring from ANT to AT
00000010	Solicit_successor_nt	Allow stations to enter the ring from NA to AT
00000011	Who_follows	Recover from dead successor
00000100	Set_Category	Allows station to leave AT to ANT
00001000	Token	Pass the token
00001100	Set_suc/prede_cessor	Allow stations to join the ring
00001110	Set_successor	Allow stations to leave the ring
11111111	Data_with_token	Send data with the token

Figure 6.3 Netlink control packets.

When a station receives a token, if it has no data to send, it increments a counter. When the counter reaches the threshold *N*, the station sends a *SET_CATEGORY* message. Its predecessor and successor will respond appropriately by taking note of their new successor and predecessor respectively. This station changes from the **Active & Transmitting** state to the **Active Not Transmitting** state.

Leaving the ring is simple. A station has to send a *SET_SUC/PREDE_CESSOR* packet to its predecessor and successor telling them that from now on the successor and the predecessor, respectively, have changed. Then, the station moves into the **Non Active** state. The successor

of the leaving station should treat this packet as a token. It would not be necessary to send the token again.

At initialization, the controller is the only station running. The controller creates a token and passes the token to itself. Periodically, it solicits bids from stations that want to join the ring. The controller will always have the lowest id. The controller is used principally for detecting and correcting errors in the system.

6.6 Error Handling in Netlink

It must be stated that, when the Netlink protocol was implemented, several assumptions were made in relation to errors that could occur. The assumptions were made to make the implementation simple. It was assumed that a station leaves the ring in a proper manner, that is, it sends an appropriate message. It was also assumed that the token will never get lost. However, the protocol can be improved by implementing error handling described below.

The protocol could take into account that stations may be halted due to power failure or system errors. If a station tries to pass the token to a station that is down, then what happens? After passing the token, a station listens to see if its successor either transmits a packet or a token. If it does neither, then the token is passed a second time.

If that also fails, then the station transmits a *WHO_FOLLOWS* packet, specifying the ID of its successor. This packet is sent to the dead machine's successor as its ID is determined from the internally maintained ring. This station responds by sending a *SET_SUCCESSOR* packet to the station whose successor failed, naming itself as a successor. This is a form of acknowledgement.

What if a station fails to pass a token and fails to locate its successor's successor? It sends another *WHO_FOLLOWS* packet to the next station in the ring. Eventually, the ring is established.

Another error occurs if the token holder goes down and takes the token with it. The controller has a timer that is reset each time a token is transmitted. When the timer reaches the threshold, the controller claims the token and processing continues. The dead station will be discarded from the ring by the *WHO_FOLLOWS* mechanism.

6.7 Packet Format

Netlink creates a further layer below the MIDINet layer but above Ethernet layer. Hence, Netlink attaches its control information to the packet received from the upper layers. Below is the format of an Ethernet packet and the size in bytes of its components.

Preamble	Source	Destination	Type	Data	CRC
8	6	6	2	46 - 1500	4

The Netlink data gets encapsulated in the data part of an Ethernet packet. The Netlink packet format has three parts: the type, predecessor, my_ID, and successor. The different types of Netlink packets are listed in Figure 6.2. **Predecessor, my_ID**, and **successor** are each one byte long, and contain stations' identification numbers. Below is a format of a Netlink packet

Type	Predecessor	My_ID	Successor	Data
1	1	1	1	0-1500

The MIDINet layer data gets encapsulated in the data part of a Netlink packet.

The Predecessor, My_ID and Successor are used to maintain the logical ring. As an example, a station knows that it is its turn to transmit if the packet type is *token* and the Successor matches its ID and the My_ID matches its own Predecessor.

6.8 Internal control information

When a station receives a packet or a token from any other station, it updates its internally maintained control information. Since a packet is broadcast, all the receiving stations do update their lists. This information includes the category the sender is currently in. The category is determined from the packet type. For instance, a data packet or a token means that the transmitting station is in the **Active** state. Each station maintains a list of all stations in the ring. As shown in section 6.7, a packet contains three IDs; the transmitting station, its predecessor and successor, and this information is used to update the list. Before it transmits, a station determines its successor and its predecessor from the list. Whenever a station

changes state or leaves the ring, the list is updated. Each station receives the appropriate information.

6.9 Advantages and disadvantages

The power of Netlink, as it has already been indicated, lies in combining the good features of Ethernet and Token passing protocols. Netlink is similar to the token bus protocol, but it has added features that make it a better protocol for our purposes.

As explained above, there is a controller in the Netlink protocol. The controller is the station that makes sure that the system recovers from errors, such as a token getting lost or a station going down. This is similar to a token ring's centralized monitor function. If the controller goes down, then it means the whole system cannot function. Unlike token ring, the controller cannot be replaced by any station on the system. To introduce that, the system must be restarted and reconfigured.

The functionality of a controller is assigned to the server in RHOCMN. This was decided for two major reasons : Firstly, RHOCMN has a server which is already a critical component in the network. Secondly, the server is not involved in the running of the MIDINet system.

Ethernet is inherently a broadcast medium; each station receives all the packets as long as the Ethernet destination address is a broadcast address. Netlink uses broadcast packets only. Every station gets every packet, whether it be control or data. Unlike many other protocols, Netlink control messages can be transmitted with data. Before any control messages are transmitted, the data buffer is checked to determine if it is empty. If data is present, it is packed with the control message. Any receiving station is responsible for extracting this data from the control message.

Passing a token to stations with no data to transmit wastes network bandwidth and imposes unnecessary delays on stations with data. Netlink overcomes the problem by introducing categories. There are three categories as has been mentioned: **Non Active, Active Not Transmitting** and **Active & Transmitting.** Through this category scheme, the system ensures that only stations with data to transmit are in the ring.

The major drawback behind the protocol is that the protocol implementation impinges on host

processor time. Under PC-Xinu, the process that runs the protocol must be of the highest priority. No packets should be missed as this could destroy the protocol. Consequently, other processes get less access to the processor. As a result, the performance of the MIDINet system declines.

6.10 Performance Evaluation

When implementing the Netlink protocol, the aim was to overcome the non-deterministic nature of Ethernet. In the previous chapter, we have seen the analysis carried out to evaluate the performance of the MidINet system. Similar experiments were carried out to evaluate the performance of the system with the Netlink protocol.

The hardware setup was similar to the one used in the previous experiments. Two MIDINet units were used. At irregular intervals, one MIDINet unit transferred a MIDI message over Ethernet to the second MIDINet unit. The second unit returned this message via a MIDI link. The time difference between receipt and transmission was measured. More PCs were added to the network to create traffic. In this experiment, MIDINet units were PCs employing the Netlink protocol. Figure 6.3 shows the results of the experiment (see graph labelled **propagation delay**).

The graph is roughly a horizontal line. The protocol controls access to the medium. At any point in time, one station has access to the medium. There is no contention and hence, no collisions. A message is successfully transmitted over the network without any delays. However, the experiment does not show the delay that is imposed by adding more stations on the network. It shows that the non-deterministic nature of the medium is overcome.

Each station on the network gets a chance to transmit data; this means that the token is passed from one station to the next. Every additional station on the network imposes a delay on the whole system. If a station gets a chance to transmit data, then how long should it wait to get another chance to transmit again ? Further experiments were conducted to determine the time a token takes to complete a round trip through the stations. Figure 6.3 shows the results of the experiment (see graph labelled **token passing protocols**). From the graph, it is clear that adding a station to the network imposes a delay. The delay increases constantly; hence, the number of stations that impose a tolerable delay can be determined.

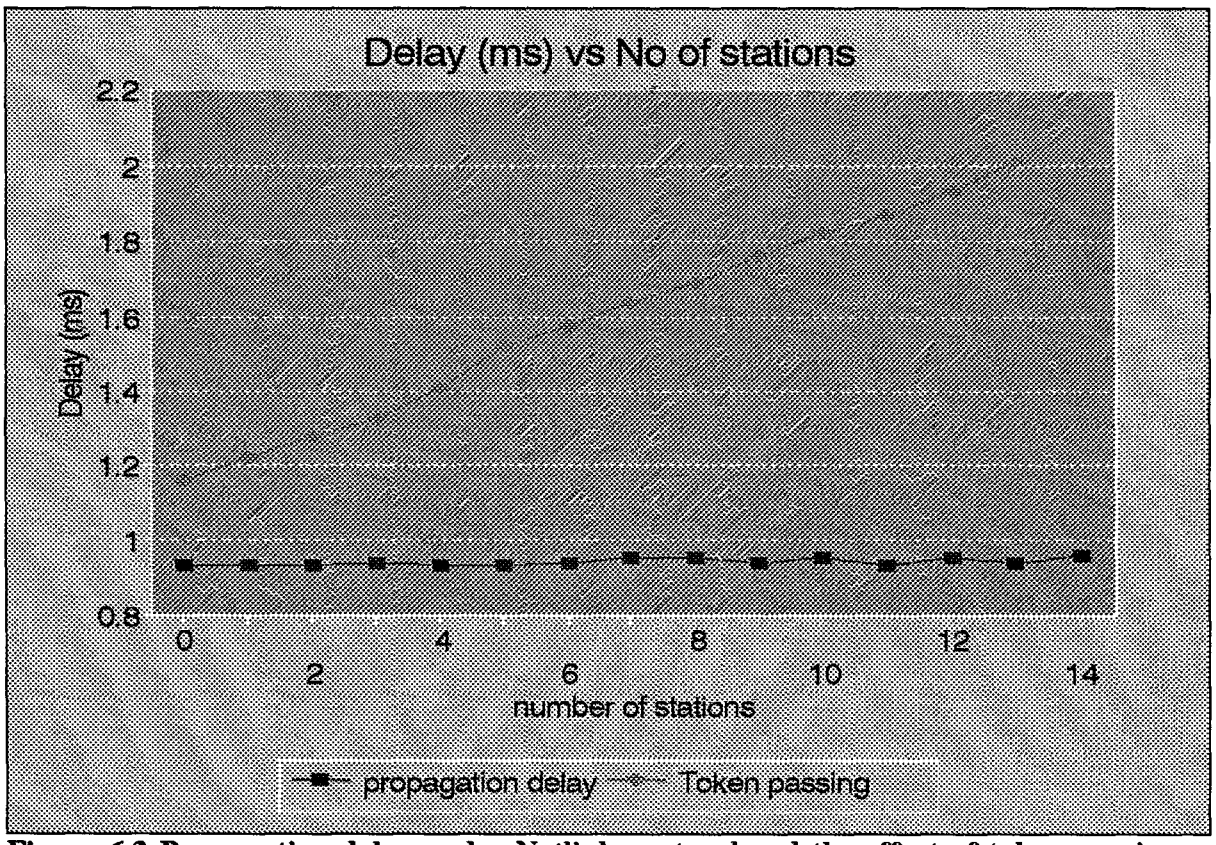


Figure 6.3 Propagation delay under Netlink protocol and the effect of token passing

In conclusion, Netlink combines the good features of Ethernet and token passing protocols. It overcomes the non-deterministic nature of Ethernet. It has a priority scheme which ensures that any data that needs to be communicated in real-time is assigned higher priority and the data endures minimal delays. Having various categories ensures that only stations with data to transmit are in the ring. The performance analysis showed that the time a token takes to complete a circle around the stations increase constantly as stations are added to the network. The category scheme minimise the number of stations that could be in a ring at any point in time. Through this scheme, the system should perform well under any load. When load decreases it means stations are running out of data to transmit, hence they are excluded from the ring. Under heavy load, all stations get a fair chance to transmit data.

Conclusion

7.1 MIDI problems addressed.

This thesis has introduced a means of addressing the problems of MIDI transmission in RHOCMN. A system was described, the MIDINet system, which is a networked system used to interconnect workstations and several MIDINet units. A MIDINet unit forms an interface between MIDI devices and a network. Any device connected to any MIDINet unit can communicate with any other device. A MIDINet unit permits MIDI data to be locally processed.

The MIDINet system has adequately solved problems related to MIDI transmission in RHOCMN. The first problem was wiring. As has been indicated, in the main studio, a star configuration is currently used for transmitting MIDI from the MIDI patch bay to the workstations and MIDI devices. This leads to a complex hardware configuration.

The MIDINet unit provides the functionality of the MIDI patch bay. Each device is independently connected to a unit. The star configuration is dissolved, which leads to reduced wiring complexity. Routing is performed via the network. The MIDINet system allows the user to send an appropriate message to 'patch' devices together. With the MIDINet unit, a MIDI message is routed to a specific device. Hence, devices can be easily shared amongst the users. Sharing devices is the main goal behind RHOCMN.

Furthermore, MIDI cables are restricted to fifteen metres in length. This imposes constraints on remote access to the studio. Workstations must be within 15 metres of the main studio. Some manufacturers have provided RS422 or fibre optic links which extend the carrying length of MIDI [Lone Wolf, 1989]. They are typically expensive solutions and do not solve the configuration problem. The MIDINet system uses Ethernet for networking. Ethernet is cheap and readily available. Ethernet LANs can extend to 2.5 km. The MIDINet system permits users outside the studio to have access to MIDI devices in the main studio.

The other problem related to MIDI in RHOCMN is channelisation. As has been discussed,

in RHOCMN, the server sends system exclusive messages to MIDI synthesizers to select the channel numbers on which they should send and receive MIDI data. This means that manufacturer specific information must be kept in the system. When a new MIDI device is installed in the studio, the server has to be loaded with the device's system exclusive message type for changing channels. The MIDINet system solves the channelisation problem. Since devices are identified uniquely within the MIDINet system, it is immaterial whether there is more than one device using the same channel number. Any MIDI message transmitted bears the source device's identification. Hence, the message is routed to the destination only, not broadcast as it is in the original system. Channel mapping occurs at the destination. The server is not required to send system exclusive messages to select channel numbers.

The MIDINet LAN offers other advantages. It is a high-speed network capable of distributing MIDI information in a complex system. It offers the advantages of bidirectionality and greater bandwidth. Although MIDI data is flowing with other information on the network, the 10 Mbps capacity of Ethernet is not saturated. A potential problem was the non-deterministic nature of Ethernet. Empirical analysis showed that the non-deterministic nature of Ethernet can introduce problems in the MIDINet system under extreme loads. The results confirmed that Ethernet performs well under a light to medium load. However, as more devices are added to the network, response time increases. This led to an implementation of the Netlink protocol.

Netlink combines the good features of Ethernet and token passing protocols. It overcomes the non-deterministic nature of Ethernet. It has a priority scheme which ensures that any data that needs to be communicated in real-time is assigned higher priority, and this data endures minimum delays.

In a typical single user studio, the MIDINet system offers features such as MIDI merging and mapping. Merging is achieved by creating a connection between more than one source device and a single destination. An example of mapping is the mapping of the channel number on the incoming MIDI message to the channel number set on the destination device.

The main advantage of the MIDINet system approach is that the problems of MIDI are addressed using cheap and readily available technology and without redefining the MIDI protocol.

7.2 Possible extensions

The MIDINet system can be developed further to make it more powerful. Developments can be made in the area of MIDI data processing, such that more MIDI mapping and filtering functions are implemented. On the other hand, the hardware of the MIDINet system can be improved or new technologies can be adopted in order to add new features.

7.2.1 Hardware

In the previous chapters, it was indicated that the MIDINet system will still perform well with an additional MUART card. With an additional MUART card, each unit will provide 6 Ins and 6 Outs. More devices can be connected directly to the MIDINet units, rather than daisy chained.

7.2.2 Software

There are several extensions that could be effected on the software that implements the MIDINet system. The system can be improved with regard to the processing of MIDI data. These functions include filtering and mapping.

7.2.2.1 MIDI Filtering

MIDI filters allow a parsing program to selectively ignore bytes in a given range of values. Filters are typically used to enable/disable recognition of different types of messages or to enable/disable recognition of messages on any given channel. In the MIDINet system, this functionality can be effected within the parsing routines by use of flags. If a user requires the system to filter a certain type of message, then a corresponding flag is set. The parser filters this type of message according to the value of this flag. This capability is a possible extension to the MIDINet system.

7.2.2.2 MIDI Mapping

MIDI Mapping refers to the reassignment of the value of a byte in a MIDI message. The following are some ways of using mapping algorithms :-

- Reassign program numbers
- Reassign channel numbers
- Reassign note number to create transposition
- Scale velocity data to any curve
- Scale controller data to any curve

Channel reassign mapping has been used in the MIDINet system. Other forms of mapping can be implemented easily by using procedures and flags that control the user's requirements.

7.3 Netlink protocol

The Netlink protocol as it has been implemented, is incomplete. Error handling has not been effected, although it has been stated in the rules (appendix 4) and shown in the state transition diagram.

When the Netlink protocol was implemented, several assumptions were made in relation to errors that could occur. The assumptions were made to make the implementation simple. It was assumed that a station leaves the ring in a proper manner, that is, it sends an appropriate message. It was also assumed that the token would never get lost. However, the protocol can be improved by taking into account the following errors. These improvements have been indicated in chapter 6.

The MIDINet system utilises network technology and protocols to overcome the shortcomings and MIDI transmission problems in RHOCMN. Ethernet technology is employed since RHOCMN is built on top of Ethernet. The MIDINet system forms part of the few attempts that have been made so far to solve MIDI problems. These attempts include MediaLink [Westfall, 1989], MIDIShare [Fober, 1994] and ZIPI [McMillen, 1994]. When it was implemented, several proposals made for similar networks were taken into consideration. These proposals include ArcoNet [Allik, 1990] and MIDI server [Buxton, 1987]

References

Aikin, J., "PLUG IN HERE, The ABC's of Techno-Music Literacy", *Keyboard*, June 1988, p34-56.

Aikin, J, Milano D., "MIDI-Controlled Digital Mixer", *Keyboard*, August 1987, p110-146

Alki K., "ArcoNet: A Proposal for a Standard Network for Communication and Control in Real-Time Performance", Proceedings of the International Computer Music Conference 1986.

Allik, K., Dunna, S., Mulder, R., "ArcoNet: A proposal for a Standard Network for Communication and Control in Real-time Performance", *LEONARD*, Vol. 23, No. 1, 9p91-97), 1990

Brighton, N., "Putting a sparkle on Analog", *Electronic Musician*, September 1993

Brighton, N., "The Patch Bay", *Electronic Musician*, May 1992, p88

Buxton W., "Masters and Slaves Versus Democracy: MIDI and Local Area Networks", Proceedings of the AES 5th International Conference.

CACI Products Company, "NETWORK II.5 Users' manual", Pennysilvania 1990.

Claiborne, D., *Mathematical Preliminaries for Computer Networking*, John Wiley & Sons, Inc. USA 1990.

[Comer, 1988a] Comer, D., *Internetworking with TCP/IP*, Printice-Hall, Inc. Englewood Cliffs, NJ 1988.

[Comer, 1988b] Comer, D., Fossum, T., *Operating System Design Volume I The Xinu approach*, Printice-Hall, Inc. Englewood Cliffs, NJ 1988.

[Comer, 1989a] Comer, D., *Internetworking With TCP/IP Volume II*, Printice-Hall, Inc. Englewood Cliffs, NJ 1988.

Comer D., *Operating System Design - Volume II: Internetworking with Xinu*, Printice-Hall, 1987.

De Furia., Scacciaferro J., *The MIDI Programmer's Handbook*, M&T Publishing, Inc. Redwood City, California.

DeMarco T., *Structured Analysis and System Specification*

Dimuzio, T., "The Magic of the real-time MIDI control", *Electronic Musician*, August 1990, p70

Fortier,J., Desrochers, G., *Modeling and Analysis of Local Area Networks*, CRC Press, Inc. Florida 1990.

Foss R., Wilks, A., "A Network Approach to the Problem of Sharing Music Studio Resources", ICMC Glasgow 1990 Proceedings.

FTP Software Inc., "PC/TCP Version 1.09 Packet Driver Specification", *FTP Software, Inc.* Wakefield, MA 1989.

Halsall, F., *Data Communications, Computer Networks and OSI*, Addison-Wesley Publishers Limited, 1988.

Holt, C., *Microcomputers Organization - Hardware and Software*, Macmillan Publishing Company, Inc. New York 10022, USA 1985.

Huber, D., "Riding the bus", *Electronic Musician*, March 1991, p46

Hurtig, B., "The Future of Analog", *Electronic Musician*, May 1992, p38

Hutchison, D., *Local Area Networks Architectures*, Addison-Wesley, Inc. 1988.

the IMA Bulletin, "MIDI Time Code, The Linking of MIDI and SMPTE", *Electronic Musician*, the IMA Bulletin, July 1986

[IMA, 1989a] the IMA Bulletin, "The MidiTap from Lone Wolf", the IMA Bulletin, June 1989.

[IMA, 1989b] the IMA Bulletin, "MIDI 1.0 Detailed Specification,

Document Ver 4.1" the IMA Bulletin, 1989

Kane, J., Osborne,A., *An Introduction to microcomputers Vol 3.* Osborne & Associates, Inc. Berkeley, California 1975.

Keiser, G.E., *Local Area Networks*, Mcgraw-Hill, Inc. 1989.

Kuo, F., *Protocols & Techniques for data communication Networks*, Printice-Hall, Inc., Englewood Cliffs NJ, 1981.

Jain, Raj., *The art of computer performance analysis*, John Wiley & Sons, Inc. USA 1991

Lone Wolf, MidiTap User Manual, *Lone Wolf*, Inc 1990.

Loy, G., "Musicians Make a Standard: The MIDI Phenomenon", *Computer Music Journal*, Vol. 9, No. 4, Winter 1985.

"MIDI Machine Control Ratified", *the IMA Bulletin* 9(1), Summer 1992.

McMillen, K., Simon, D., Wessel, D., Wright, M., "A New Network and Communications Protocol for Electronic Musical Devices", *Audio Hardware, Networking,* ICMC Proceedings 1994.

Meyer, C., Hall, G., "An introduction to Hard Disk recording and editing", *Electronic Musician*, October 1990.

Miller, M.A., *INTERNETWORKING A guide to Network Communications LAN to LAN; LAN to WAN*, M&T Books, Inc., Redwood City, CA 94063, 1991.

Moore F., *Elements of computer Music*, Prentice Hall, Englewood Cliffs, New Jersey.

Mosala, T., "Simulation of an Ethernet network using NETWORK II.5" Rhodes University 1992 (unpublished)

Oppenheimer, L., "Intone MIDI Maestro Audio and MIDI patch bay", *Electrinic Musician*, p94-97, March 1991.

Rehmet G., "PC-Xinu Event Driven Windowing System", Rhodes University, 1991 (unpublished).

Rona J., "MIDI The Ins, Outs & Thrus", Hal Leonard Books, Wisconsin, 1987.

Schoemaker, S., *Computer Networks and Simulation*, Elsévier North-Holland Inc. New York 1978.

Shlear, S., Mellor, S., *Object Oriented Systems Analysis : modeling the world in data*, Yourdon Press, Englewood Cliffs, New Jersey, 1988.

Tanenbaum A., *Computer Networks*, Prentice-Hall, Inc. Englewood Cliffs, New Jersey 1988.

Tarnay, K, *Protocol Specification and Testing*, Plenum Press, NY. 1991.

[Ward, 1985a] Ward, P., Mellor, S., *Structured Development for Real-Time Systems Vol 1*, Yourdon Press, NY. 1985.

[Ward, 1985b] Ward, P., Mellor, S., *Structured Development for Real-Time Systems Vol 2*, Yourdon Press, NY. 1985.

[Ward, 1985c] Ward, P., Mellor, S., *Structured Development for Real-Time Systems Vol 3*, Yourdon Press, NY. 1985.

Watkins, R., *A memory resident implementation of PC-XINU*, Rhodes University, 1990 (unpublished).

Westfall, L., "The Local Area Network: MIDI's next step ?" *Electronic Musician*, November 1989, p64-119

Wilkinson, S., "Sound all Around", *Electronic Musician*, October 1993, p127.

Wilkinson, S., "From the Top : MIDI Basics, Part 1", *Electronic Musician*, August 1993, p73.

Wilkinson, S., "From the Top : MIDI Basics, Part 2", *Electronic*

Musician, September 1993, p70.

Wilkinson, S., "From the Top : MIDI Implementation Charts", *Electronic Musician*, March 1993, p69.

Wilks, A., "The analysis of Computer Music Network and the implementation of essential sub-systems", M.Sc. Thesis, Rhodes University, 1994.

Yourdon, E., Constantine, L., *Structured Design*, Printice-Hall, Inc., Englewood Cliffs, N.J. 1979.

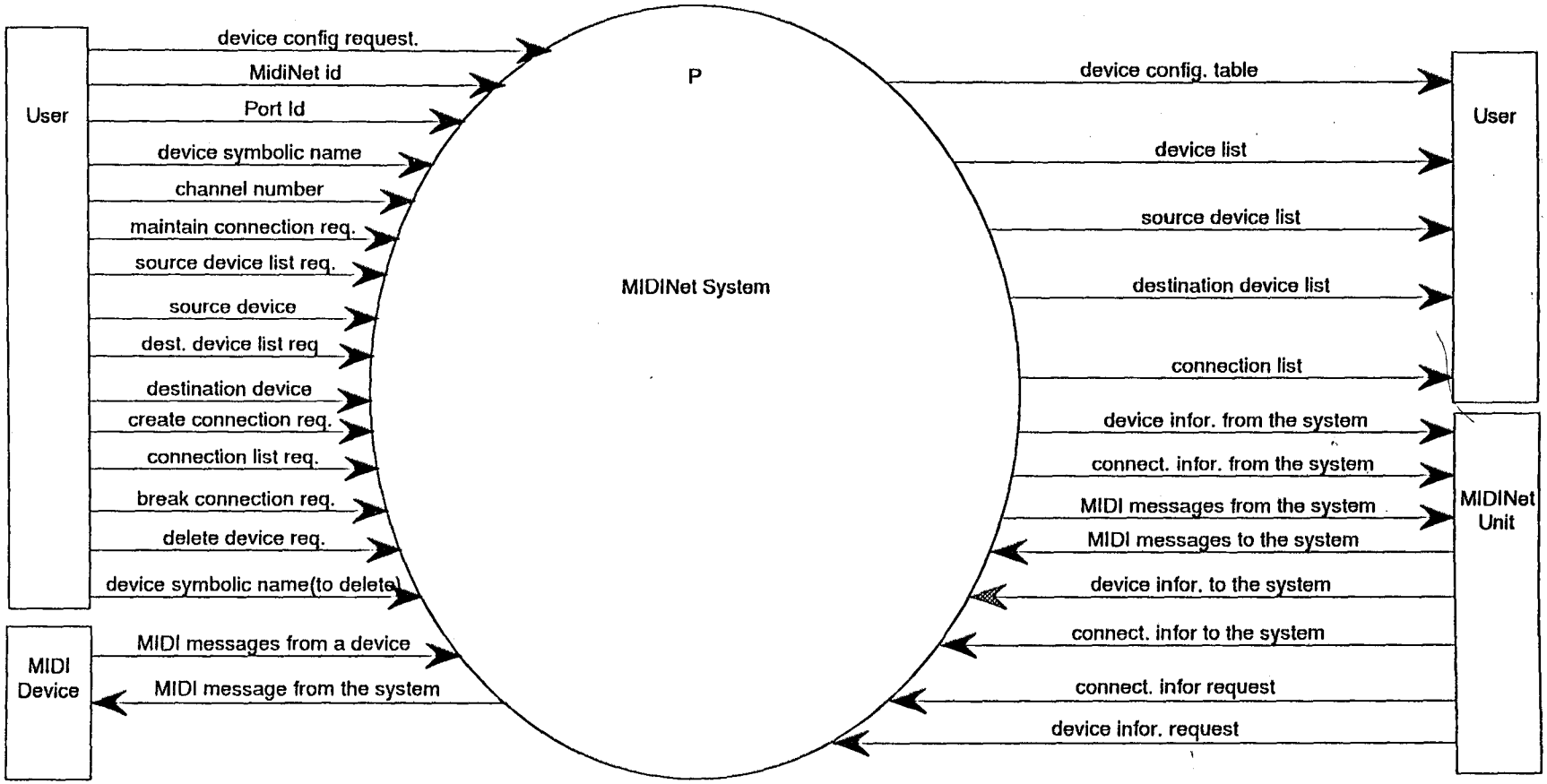
Ziffer, A., "Multitrack Recording", *Eletronic Musician*, January 1991, p44

Appendix 1

Essential Modelling

1.1 Enviromental Model

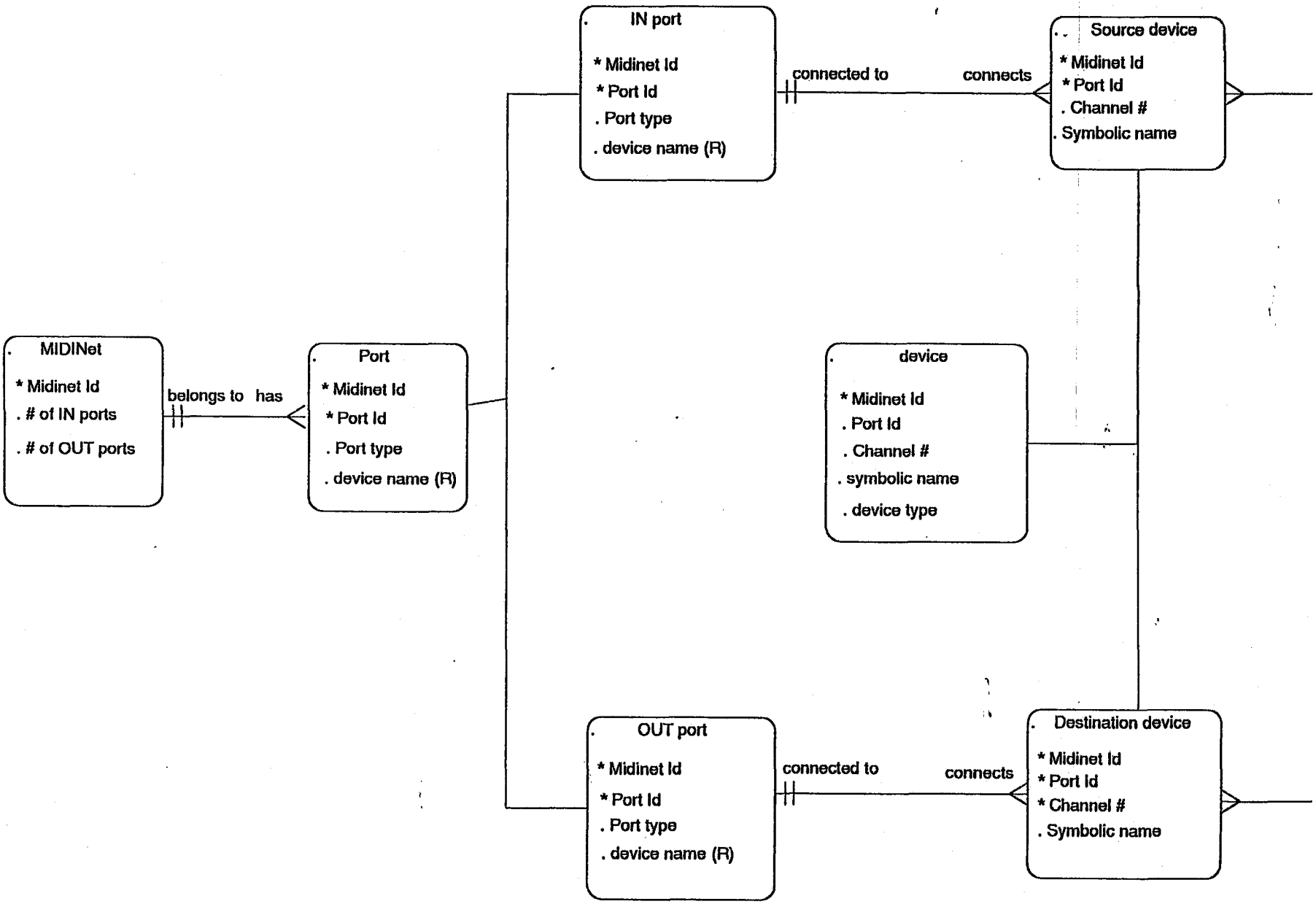
1.1.1 Context Schema

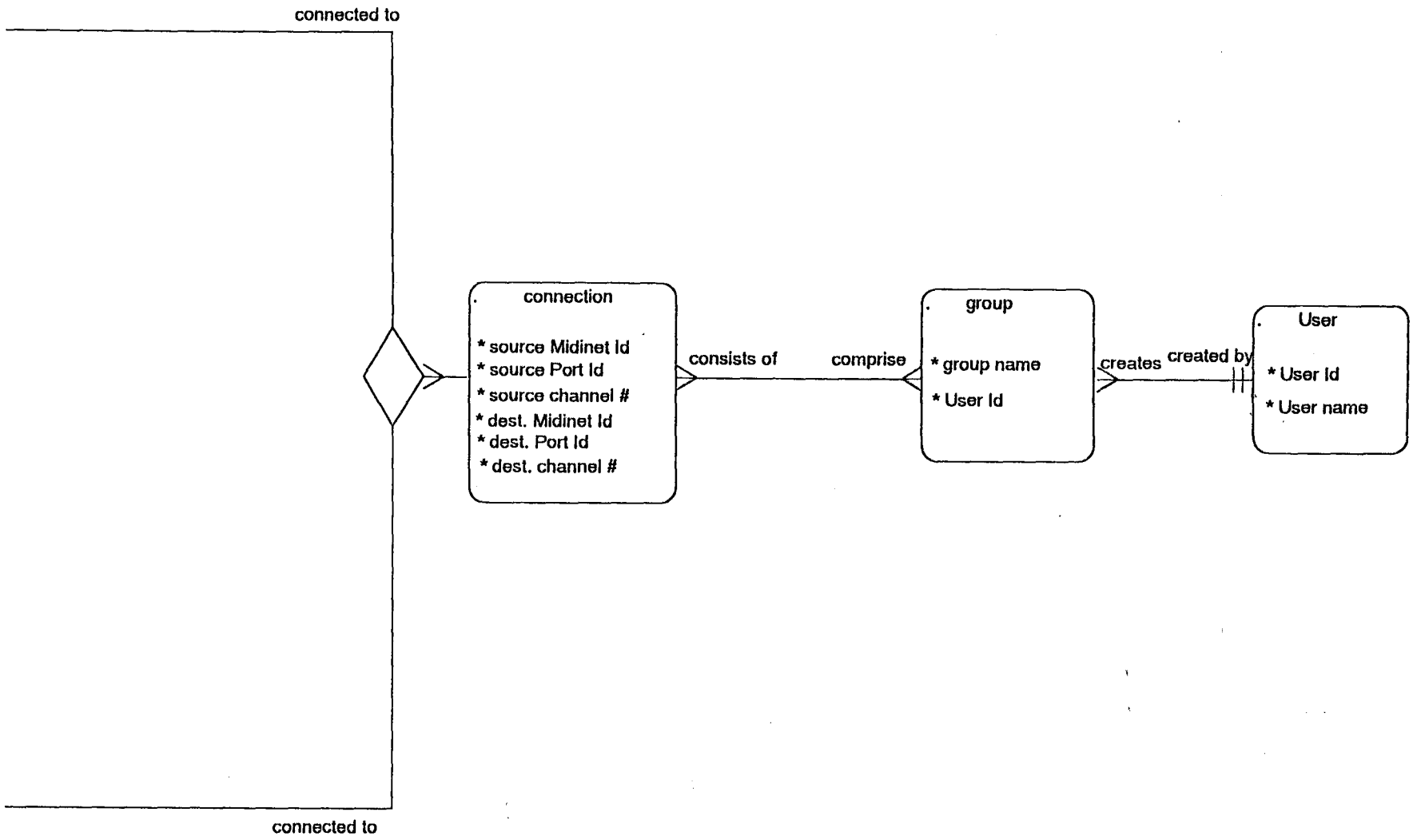

User
device config request.
MidiNet id
Port Id
device symbolic name
channel number
maintain connection req.
source device list req.
source device
dest. device list req
destination device
create connection req.
connection list req.
break connection req.
delete device req.
device symbolic name(to delete)
MIDI Device
MIDI messages from a device
MIDI message from the system
P
MIDINet System
device config. table
device list
source device list
destination device list
connection list
device infor. from the system
connect. infor. from the system
MIDI messages from the system
MIDI messages to the system
device infor. to the system
connect. infor to the system
connect. infor request
device infor. request
User
MIDINet Unit

1.1.2 Event List

User make a device configuration request.
User enters the MIDINet ID.
User enters the Port ID.
User enters device symbolic name.
User enters the channel number.
User make delete devices request.
User enters device to be deleted.
User requests a device list.
User make establish connection request.
User requests source devices' list.
User requests destination devices' list
User enters source device.
User enters destination device.
User requests connection table.
User make a 'break a connection' request.
User enters a connection to break
MIDI Messages enter the MIDINet from a port.
MIDI messages from an Ethernet.
Devices config. information from an Ethernet.
Delete device data from Ethernet.
Establish connection data from an Ethernet.
Break connection data from an Ethernet.
A unit make load device information request.
A unit make load connection information request.

1.2 Information Model


IN port
* Midinet Id
* Port Id
. Port type
. device name (R)
connected to
connects
Source device
* Midinet Id
* Port Id
. Channel #
. Symbolic name
MIDINet
* Midinet Id
. # of IN ports
. # of OUT ports
belongs to
has
Port
* Midinet Id
* Port Id
. Port type
. device name (R)
device
* Midinet Id
. Port Id
. Channel #
. symbolic name
. device type
OUT port
* Midinet Id
* Port Id
. Port type
. device name (R)
connected to
connects
Destination device
* Midinet Id
* Port Id
* Channel #
. Symbolic name


connected to
connection
* source Midinet Id
* source Port Id
* source channel #
* dest. Midinet Id
* dest. Port Id
* dest. channel #
consists of
comprise
group
* group name
* User Id
creates
created by
User
* User Id
* User name
connected to

1.3 Behavioural Model

1.3.1 Response List

Event : User make a device configuration reguest.
Resp : The system prompts the user to enter the particulars of a device.

Event : User enters the MIDINet ID.
Resp : System checks validity of the ID and respond with an appropriaate message.

Event : User enters the Port ID.
Resp : System checks validity of the ID and respond with appropriate message.

Event : User enters device symbolic name.
Resp : The system checks validity of the name i.e should be unique and respond with an appropriate message.

Event : User enters the channel number.
Resp : The channel number is verified and appropriate message is issued. The system checks the combination of IDs for validity. The device is included to the device's list and is passed to other units.

Event : User make a connection table request.
Resp : The system displays the device list and the connection table.

Event : User requests source devices' list.
Resp : The source devices' list is displayed. If the list is empty, then a message is issued,

Event : User requests destination devices' list.
Resp : The destination devices' list is displayed. If the list is empty, then a message is issued.

Event : User enters source device.
Resp : Source device is highligted or the message is issued if already involved in a connection and the user is

warned that the connection will be broken if he insisted.

Event : User enters destination device.
Resp : Destination device is highlighted but the user is warned if the device is involved in another connection and the connection is established i.e an entry is created on the connection table old connections if any are broken.

Event : User enters a connection to break.
Resp : The connection is highlighted and the user is prompted to break or abandon.

Event : User make a 'break a connection' request.
Resp : The connection is broken.

Event : User requests device list

Resp : The device list is displayed.

Event : User enters device to delete.
Resp : The device is deleted from the tables and the particulars are deleted also and the connections are broken.

Event : MIDI Messages enter the MIDINet from a port.
Resp : The system identifies the message by the MIDINet ID, port ID and the channel number it bears. The system checks from the connection table if one of the destinations is a local device. If its the case, it does channel mapping and sent the message to the device, else attaches the particulars of device and send the message as a packet over the network.

Event : Receives MIDI messages from an Ethernet.
Rest : Checks from the connection table for the destination(s). Get the actual number of the destination device and send the message through the appropriate port if the destination is not one of the

local device then it ignores it.

Event : Devices configuration data from an Ethernet.
Resp : Updates the device list.

Event : Delete device data from an Ethernet.
Resp : Device is deleted from the device list.

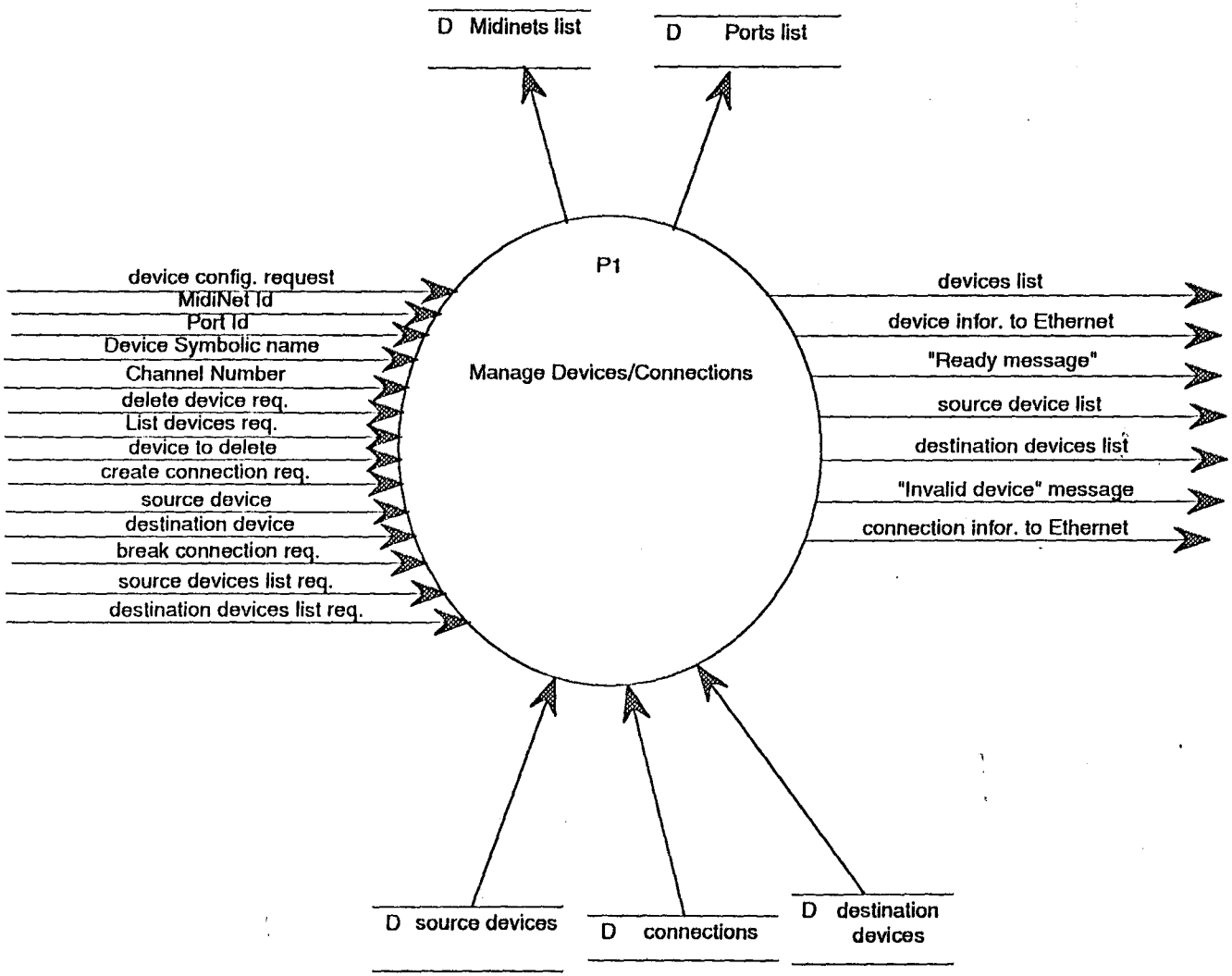
Event : Establish connection data from an Ethernet.
Resp : Connection table is updated :- connection is established.

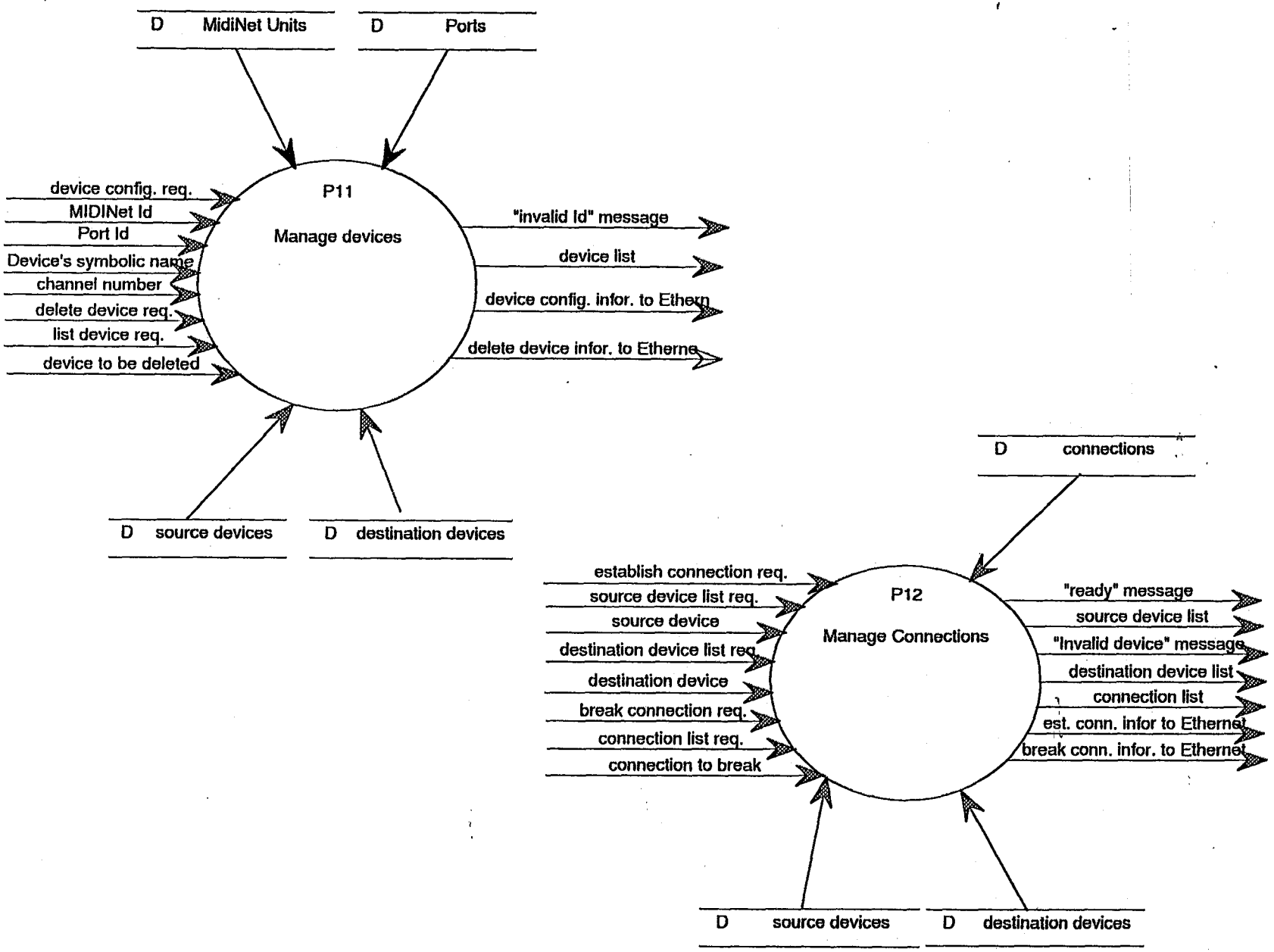
Event : Break connection data from an Ethernet.
Resp : Connection table is updated :- connection broken.

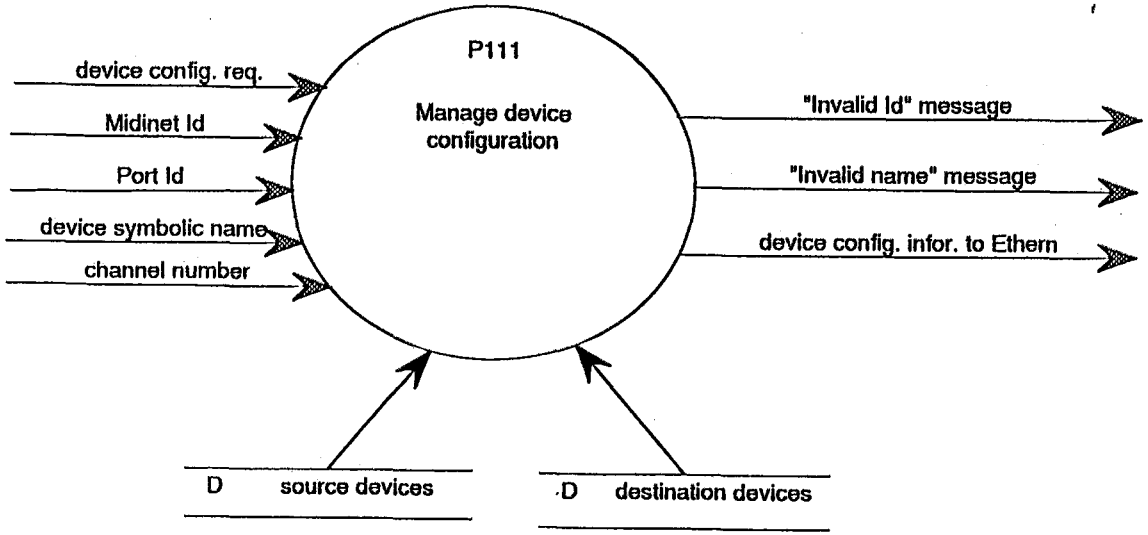
Event : A unit make load device information request.
Resp : The unit send all the information related configured devices.

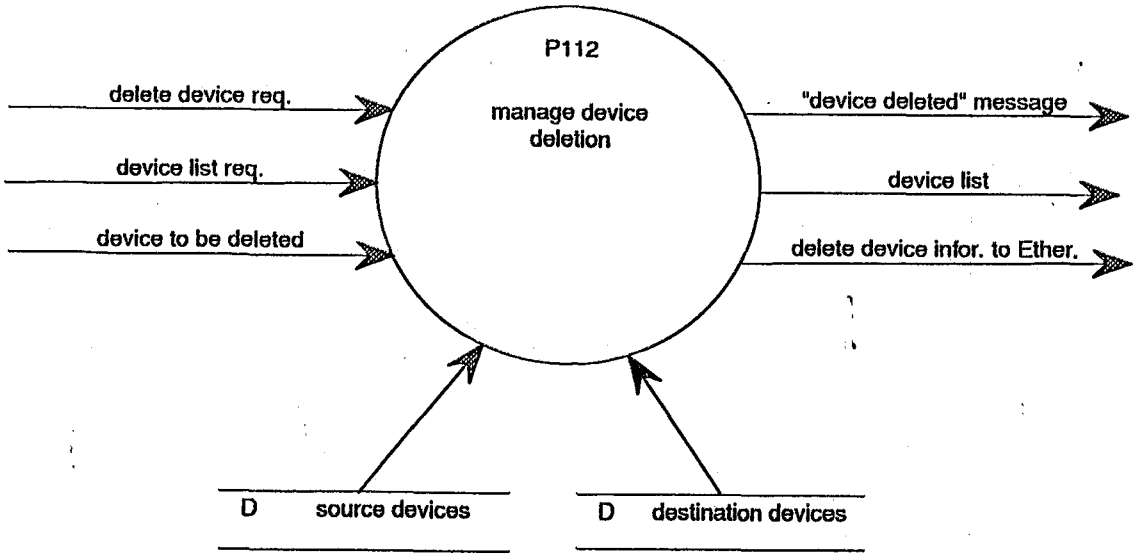
Event : A unit make load connection information request.
Resp : The unit sends all the information related to established connection i.e the whole of connection table.

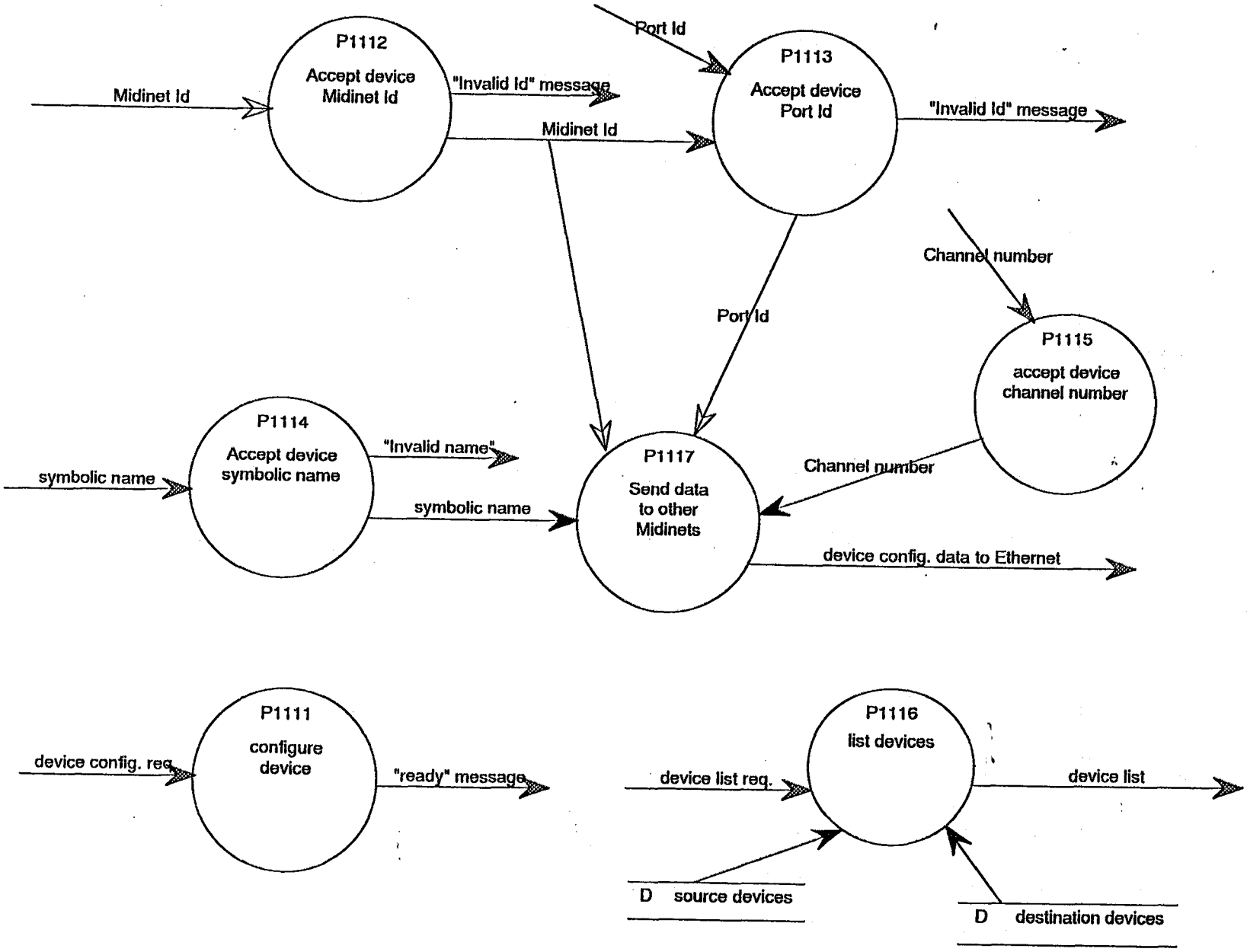
1.3.2 Transformaton shema

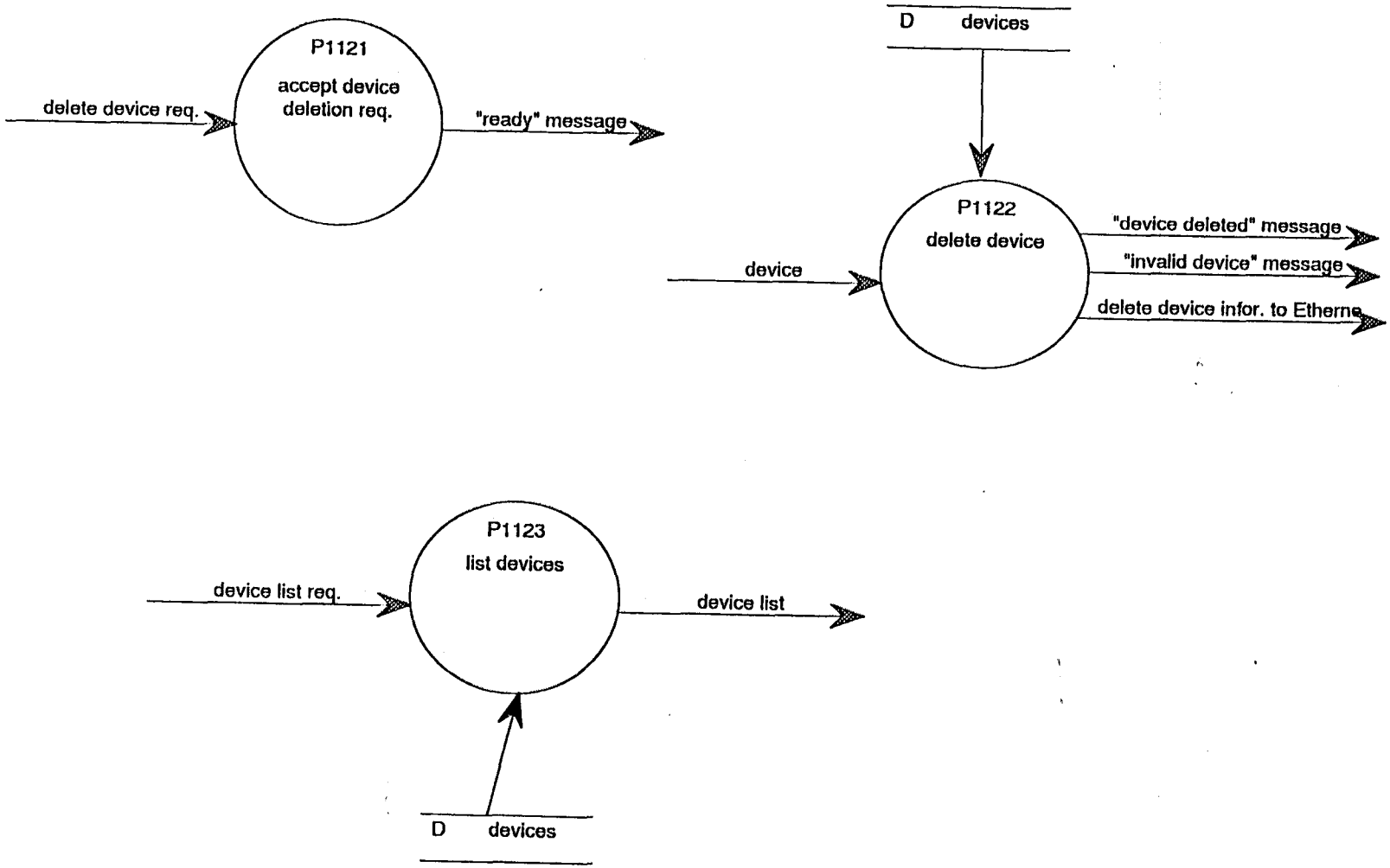

D Midinets list
D Ports list
P1
Manage Devices/Connections
device config. request
MidiNet Id
Port Id
Device Symbolic name
Channel Number
delete device req.
List devices req.
device to delete
create connection req.
source device
destination device
break connection req.
source devices list req.
destination devices list req.
devices list
device infor. to Ethernet
"Ready message"
source device list
destination devices list
"Invalid device" message
connection infor. to Ethernet
D source devices
D connections
D destination devices

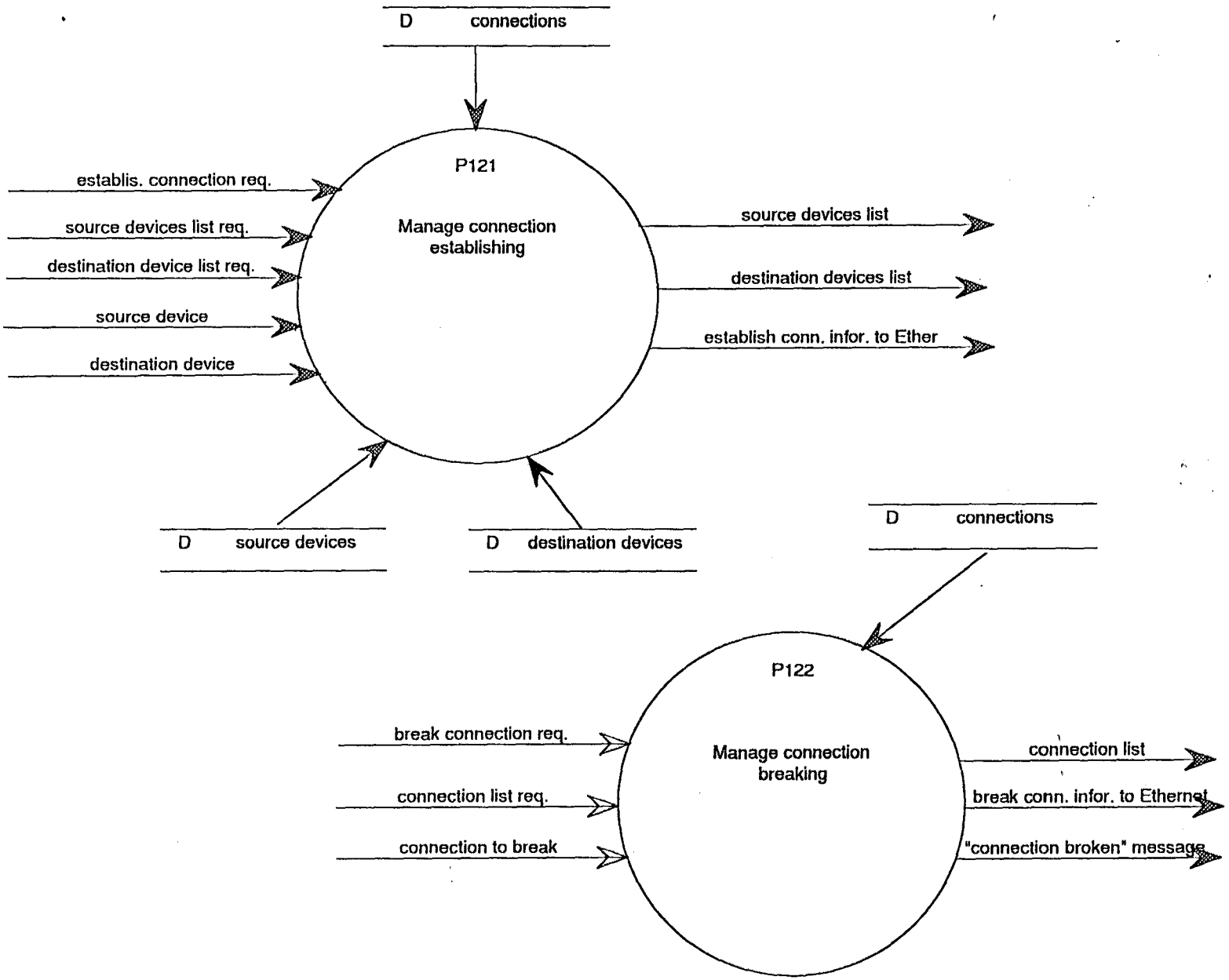

D MidiNet Units
D Ports
P11
Manage devices
device config. req.
MIDINet Id
Port Id
Device's symbolic name
channel number
delete device req.
list device req.
device to be deleted
"invalid Id" message
device list
device config. infor. to Ethern
delete device infor. to Etherne
D source devices
D destination devices
D connections
P12
Manage Connections
establish connection req.
source device list req.
source device
destination device list req
destination device
break connection req.
connection list req.
connection to break
"ready" message
source device list
"Invalid device" message
destination device list
connection list
est. conn. infor to Ethernet
break conn. infor. to Ethernet
D source devices
D destination devices

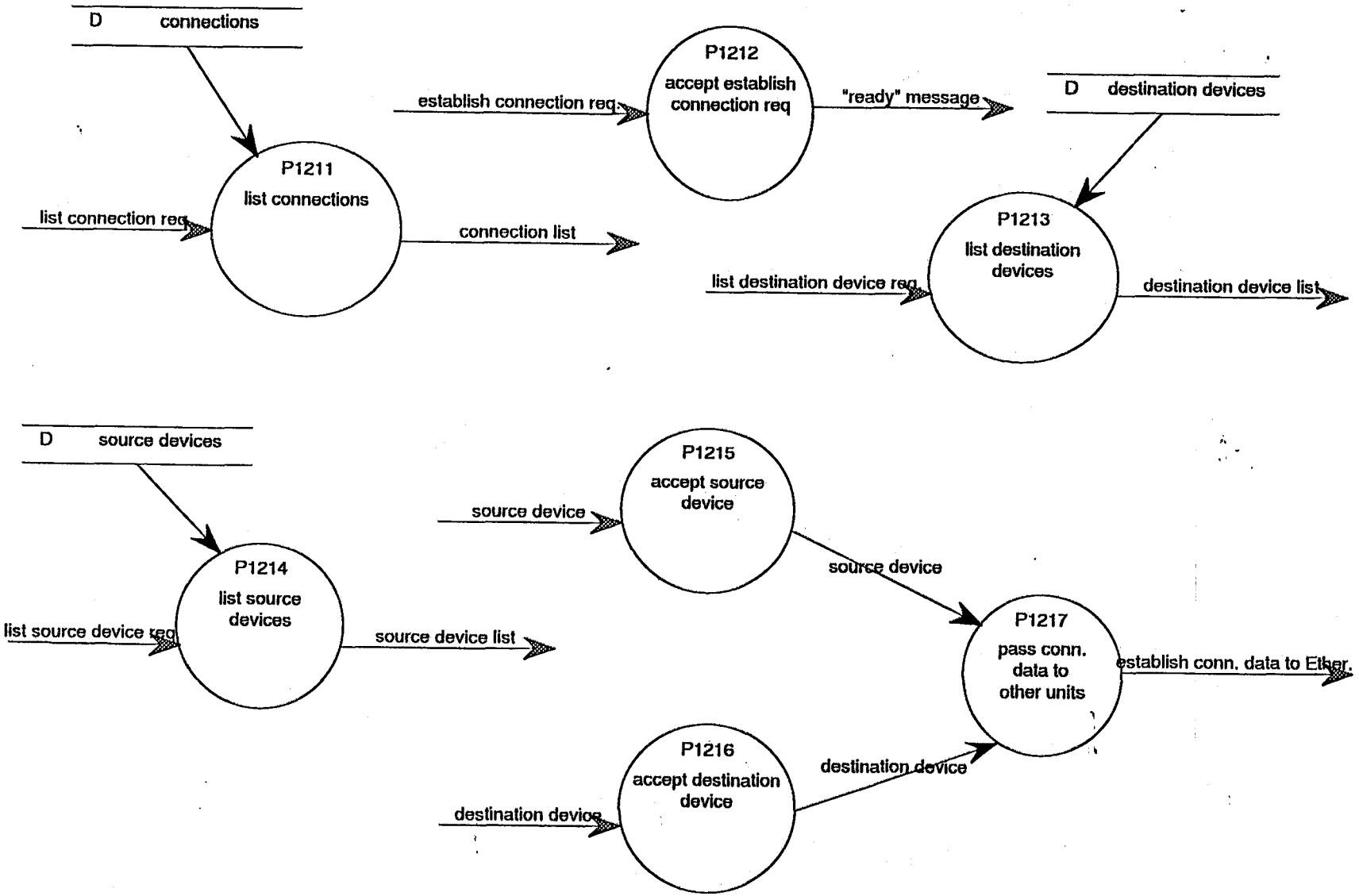

P111
Manage device configuration
device config. req.
Midinet Id
Port Id
device symbolic name
channel number
"Invalid Id" message
"Invalid name" message
device config. infor. to Ethern
D source devices
D destination devices

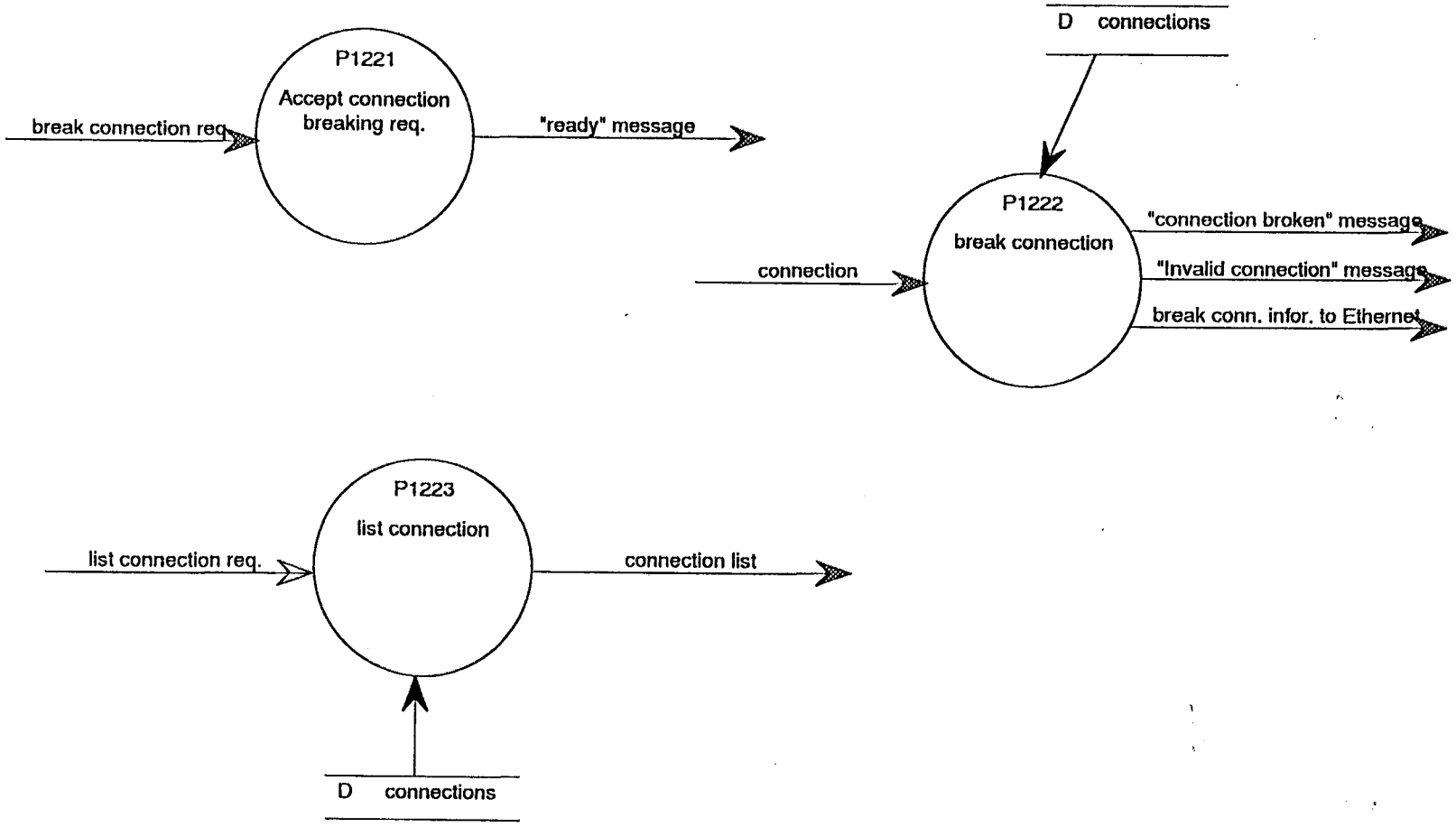

P112
manage device deletion
delete device req.
device list req.
device to be deleted
"device deleted" message
device list
delete device infor. to Ether.
D source devices
D destination devices

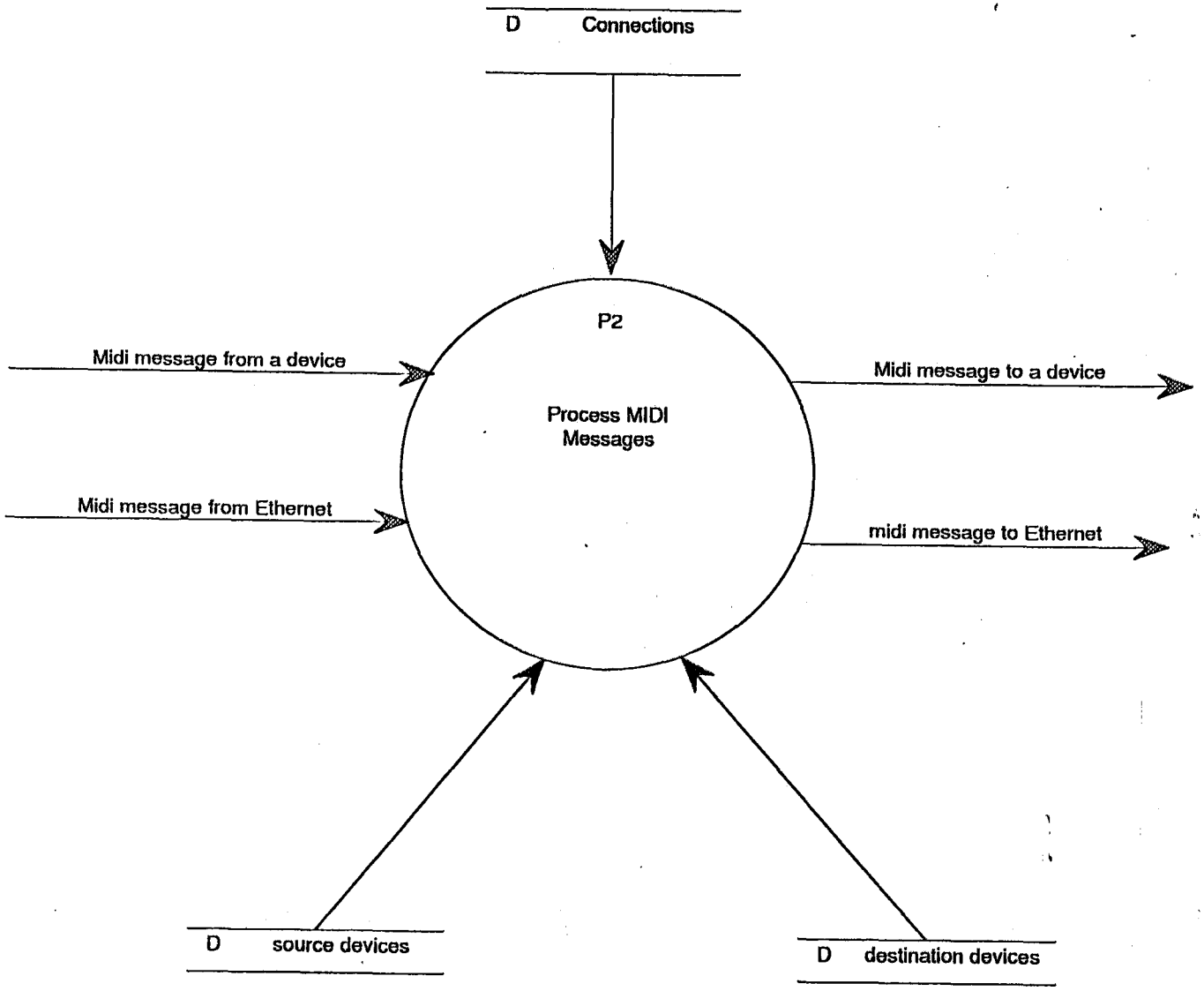

P1112
Accept device Midinet Id
Midinet Id
"Invalid Id" message
Midinet Id
Port Id
P1113
Accept device Port Id
"Invalid Id" message
Channel number
Port Id
P1115
accept device channel number
P1114
Accept device symbolic name
symbolic name
"Invalid name"
symbolic name
P1117
Send data to other Midinets
Channel number
device config. data to Ethernet
P1111
configure device
device config. req.
"ready" message
P1116
list devices
device list req.
device list
D source devices
D destination devices

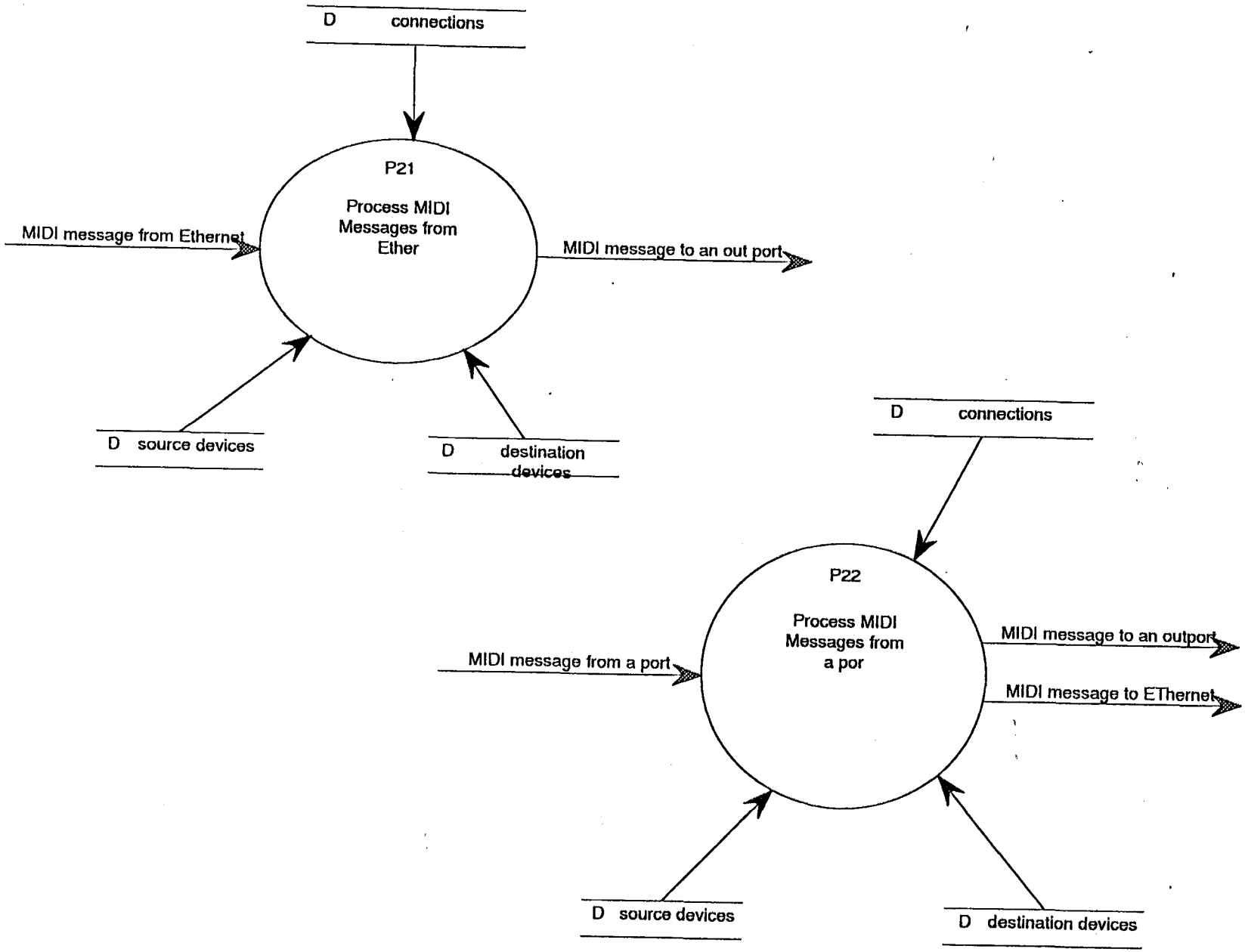

P1121
accept device deletion req.
delete device req.
"ready" message
D devices
P1122
delete device
device
"device deleted" message
"invalid device" message
delete device infor. to Etherne
P1123
list devices
device list req.
device list
D devices

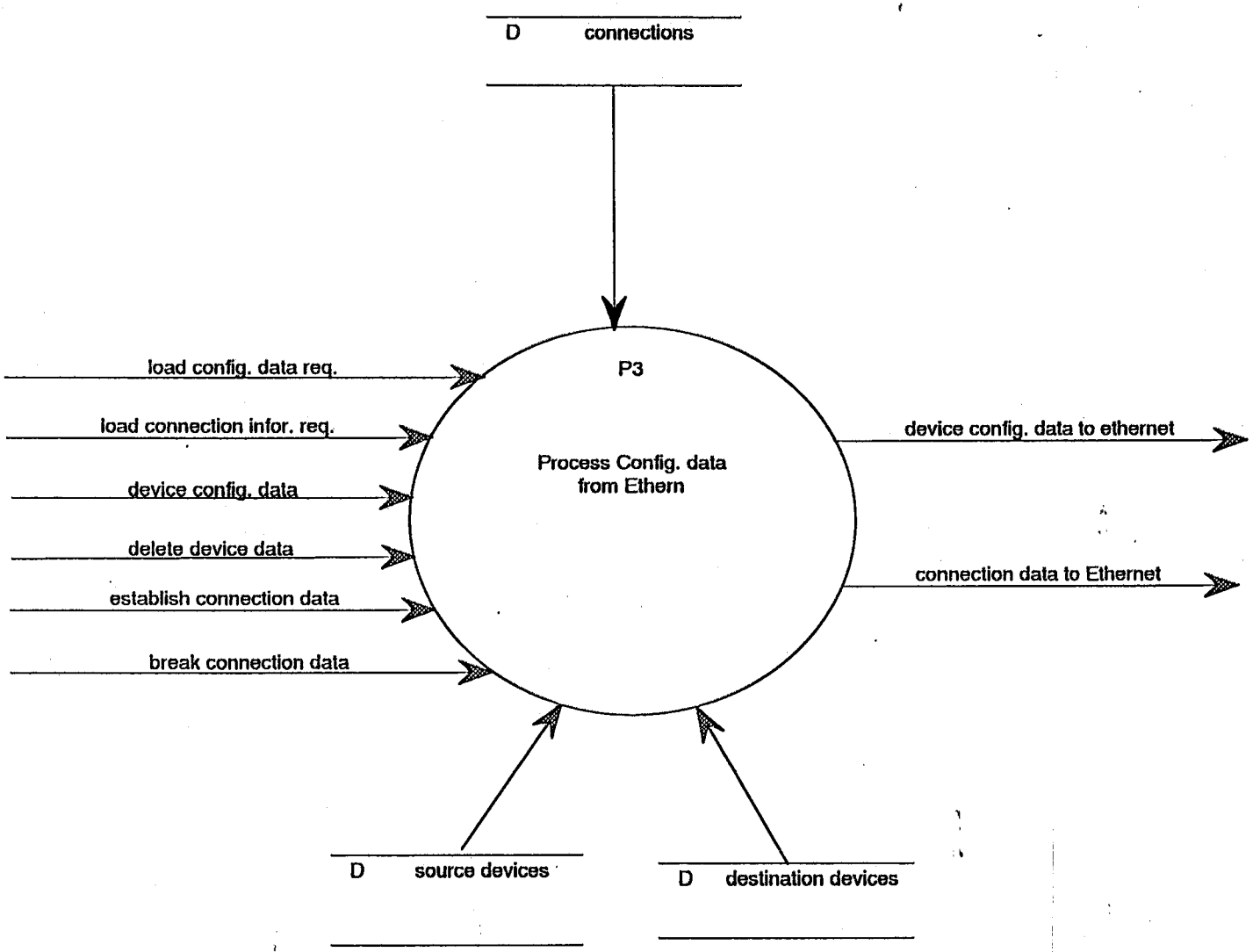

D connections
P121
Manage connection establishing
establis. connection req.
source devices list req.
destination device list req.
source device
destination device
source devices list
destination devices list
establish conn. infor. to Ether
D source devices
D destination devices
D connections
P122
Manage connection breaking
break connection req.
connection list req.
connection to break
connection list
break conn. infor. to Ethernet
"connection broken" message

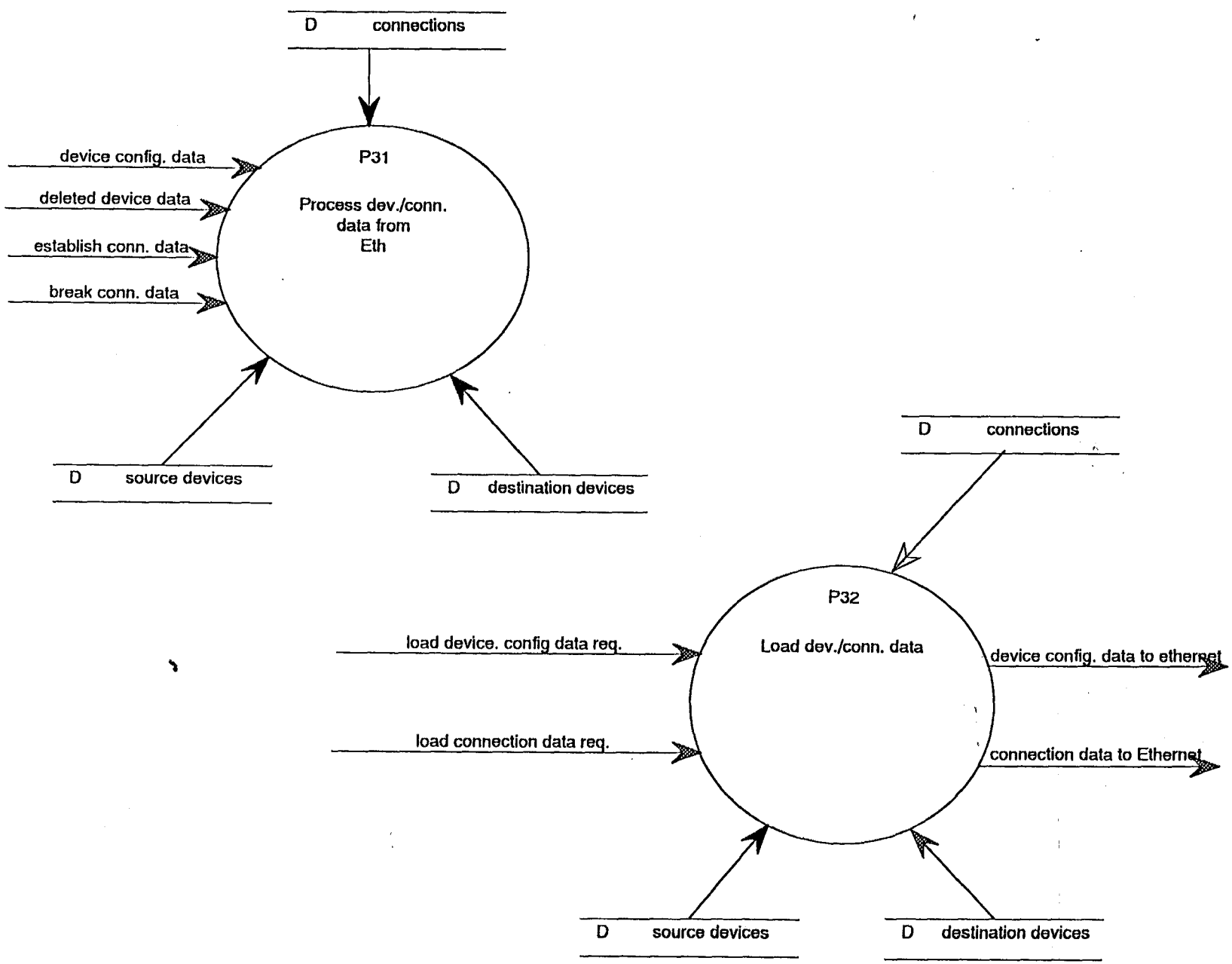

D connections
P1211
list connections
list connection req
connection list
establish connection req
P1212
accept establish connection req
"ready" message
D destination devices
P1213
list destination devices
list destination device req
destination device list
D source devices
P1214
list source devices
list source device req
source device list
source device
P1215
accept source device
source device
P1217
pass conn. data to other units
establish conn. data to Ether.
destination device
P1216
accept destination device
destination device


P1221
Accept connection breaking req.
break connection req.
"ready" message
D connections
P1222
break connection
connection
"connection broken" message
"Invalid connection" message
break conn. infor. to Ethernet
P1223
list connection
list connection req.
connection list
D connections


D Connections
P2
Process MIDI Messages
Midi message from a device
Midi message from Ethernet
Midi message to a device
midi message to Ethernet
D source devices
D destination devices


D connections
P21
Process MIDI Messages from Ether
MIDI message from Ethernet
MIDI message to an out port
D source devices
D destination devices
D connections
P22
Process MIDI Messages from a por
MIDI message from a port
MIDI message to an outport
MIDI message to EThernet
D source devices
D destination devices


D connections
P3
Process Config. data from Ethern
load config. data req.
load connection infor. req.
device config. data
delete device data
establish connection data
break connection data
device config. data to ethernet
connection data to Ethernet
D source devices
D destination devices


D connections
device config. data
deleted device data
establish conn. data
break conn. data
P31
Process dev./conn. data from Eth
D source devices
D destination devices
D connections
load device. config data req.
load connection data req.
P32
Load dev./conn. data
device config. data to ethernet
connection data to Ethernet
D source devices
D destination devices

1.3.3 Data Dictionary

Device Config. request = * selecting from the menu the option to configure a device *

MIDINet ID = 1 {digit} 10
* A byte long number which uniquely identifies a MIDINet unit *

Port ID = 1 {digit} 4
* A byte long number which uniquely identifies a port on a MIDINet unit *

Port Type = {char - I/O}
* Indicate whether a port is an In or Out port *

Devices' Symbolic name = 4 {character} 10
* ten character long string that uniquely identifies a device within a MIDINet unit i.e identical symbolic names can exist but on different MIDINet units *

Channel Number = 0 {channel level} 16
* channel number that is set for a device on which MIDI messages are to be transmitted or received *

Device = MIDINet ID + Port ID + Symbolic name + Channel #

Source Device = Device

Destination Device = Device

Establish Connection request = * selecting on the menu the option to maintain devices *

Source device list request = * selecting on the sub menu the option to display the source devices' list *

Source Device list = * a list of all devices (source devices) connected to In ports on every MIDINet unit within the system *

Destination Devices List request = * selecting on the sub-menu

the option to display the destination devices' list *

Destination Device List = * a list of all devices connected to Out ports on every MIDINet unit within the system *

User enters a source device = * A click on one of the source devices on the list *

User enters a destination device = * A click on one of destination devices on the list *

Establish a connection request = * confirmation command to create a connection *

Connection = Source Device + Destination Device.
* an established route for message between two devices *

Connection table = * A list of a created connections, showing the source and the destination device together with MIDINets and Ports IDs to which they are connected *

Break connection request = * selecting from the sub-menu the option to break a connection *

MIDI message = * a standard three byte long MIDI Message *

MIDI message to an out port = MIDI message
* a standard MIDI message sent out to a port *

MIDI message to Ethernet = Source device + MIDI message
* altered standard MIDI message :- packed with the source device into a packet *

Device config. data = MIDINet ID + Port ID + Symbolic name + device config code
* a configuration information for a particular device. Device config code (**DA - Device to Add**) *

Delete device data = MIDINet ID + Port ID + Symbolic name + Delete code
* the particulars of a devices to be deleted from the system :-

Delete code (**DR - Device to Remove**) *

Establish connection data = Source device + Destination device + establish connection code
* connection information and the code (**LE - Link** (connection to be **Established**) *

Break connection data = Source device + Destination device + break connection code.
* connection information and the code (**LB - Link** (connection) to **Break**). *

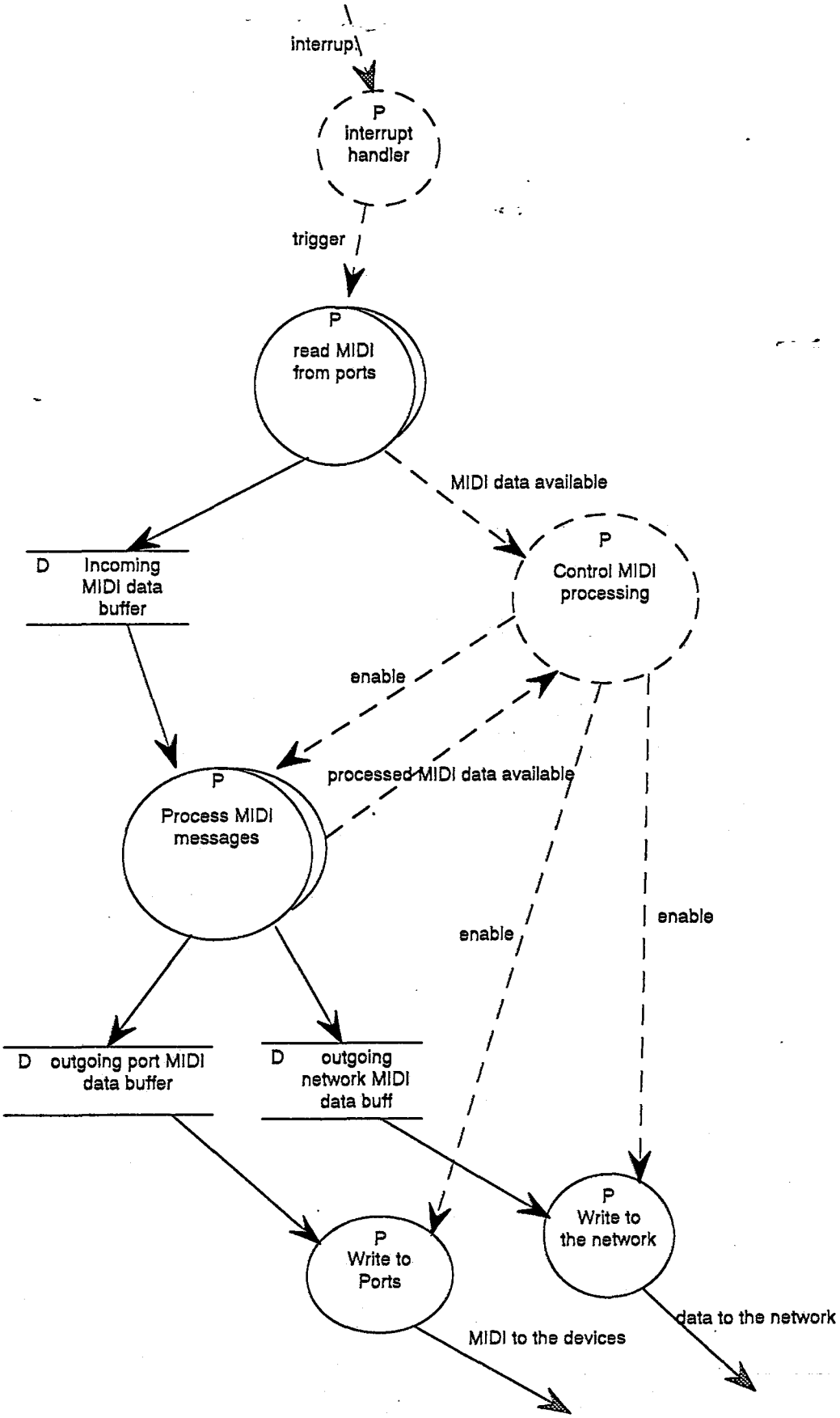
Load device information request = * an instruction for a unit with the lowest identification number to send device configuration data *

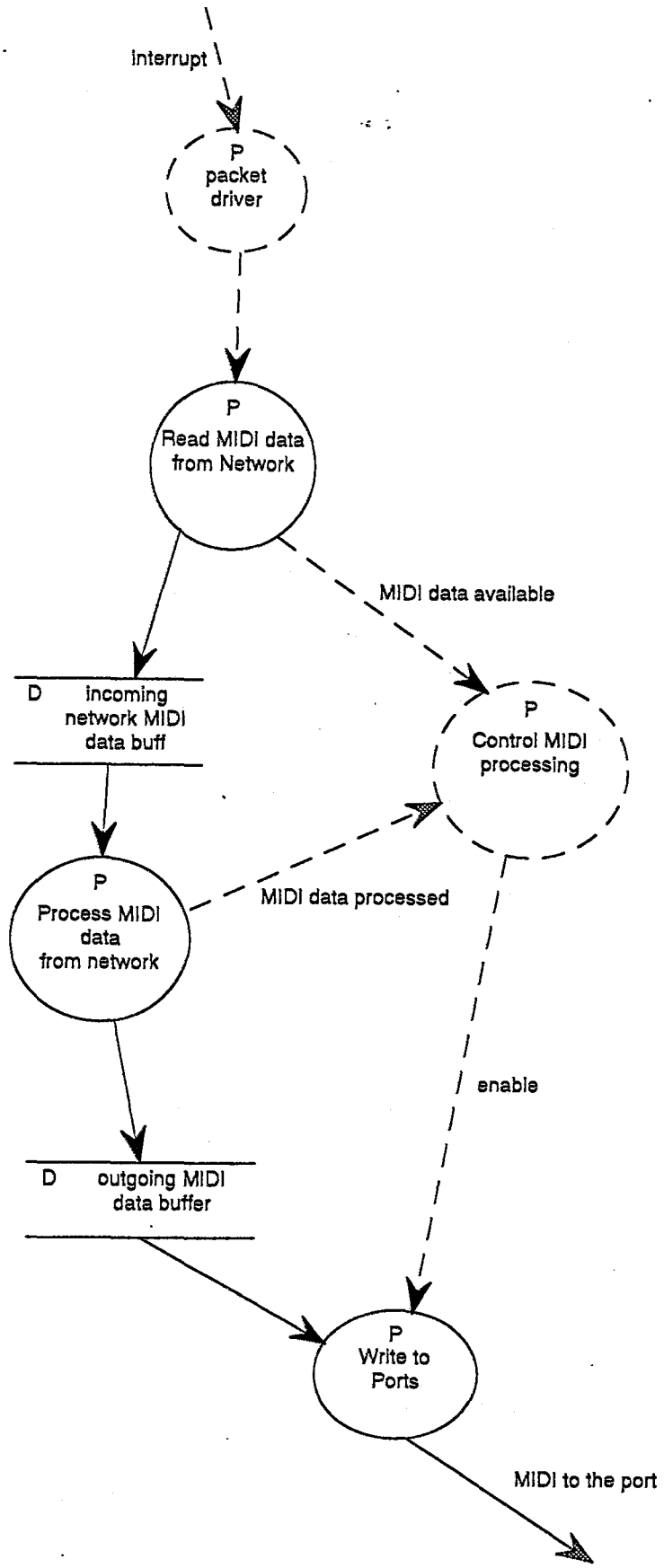
Load connection information request = * an instruction for a unit with the lowest identification number to send connection data i.e the connection table information *

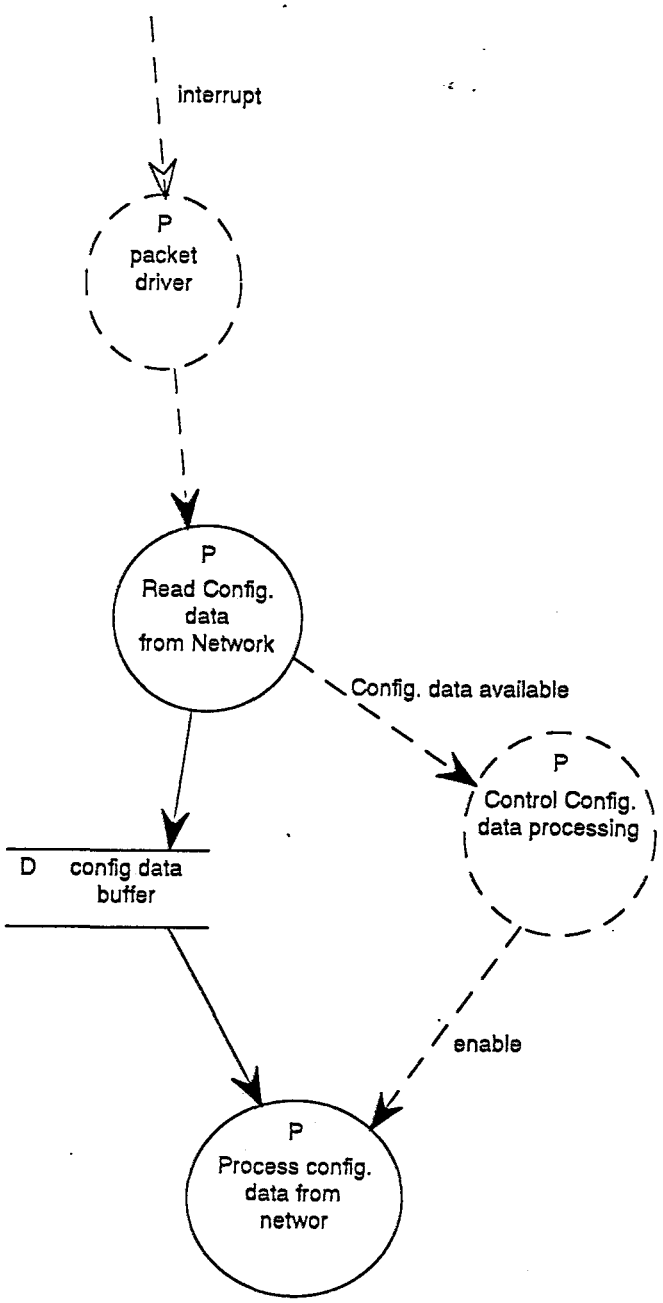
Appendix 2

Implementation Modelling

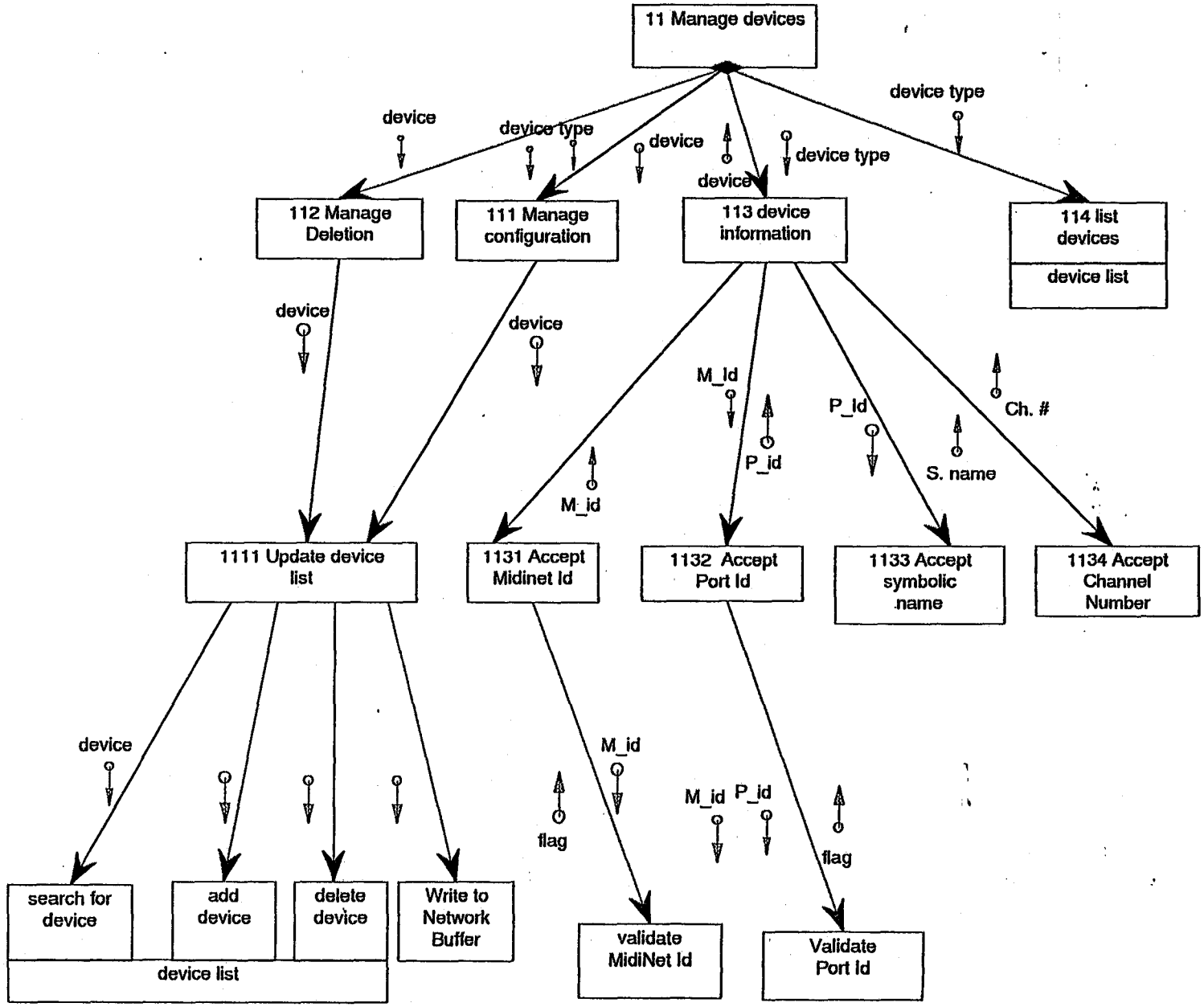
2.1 Modelling the interfaces

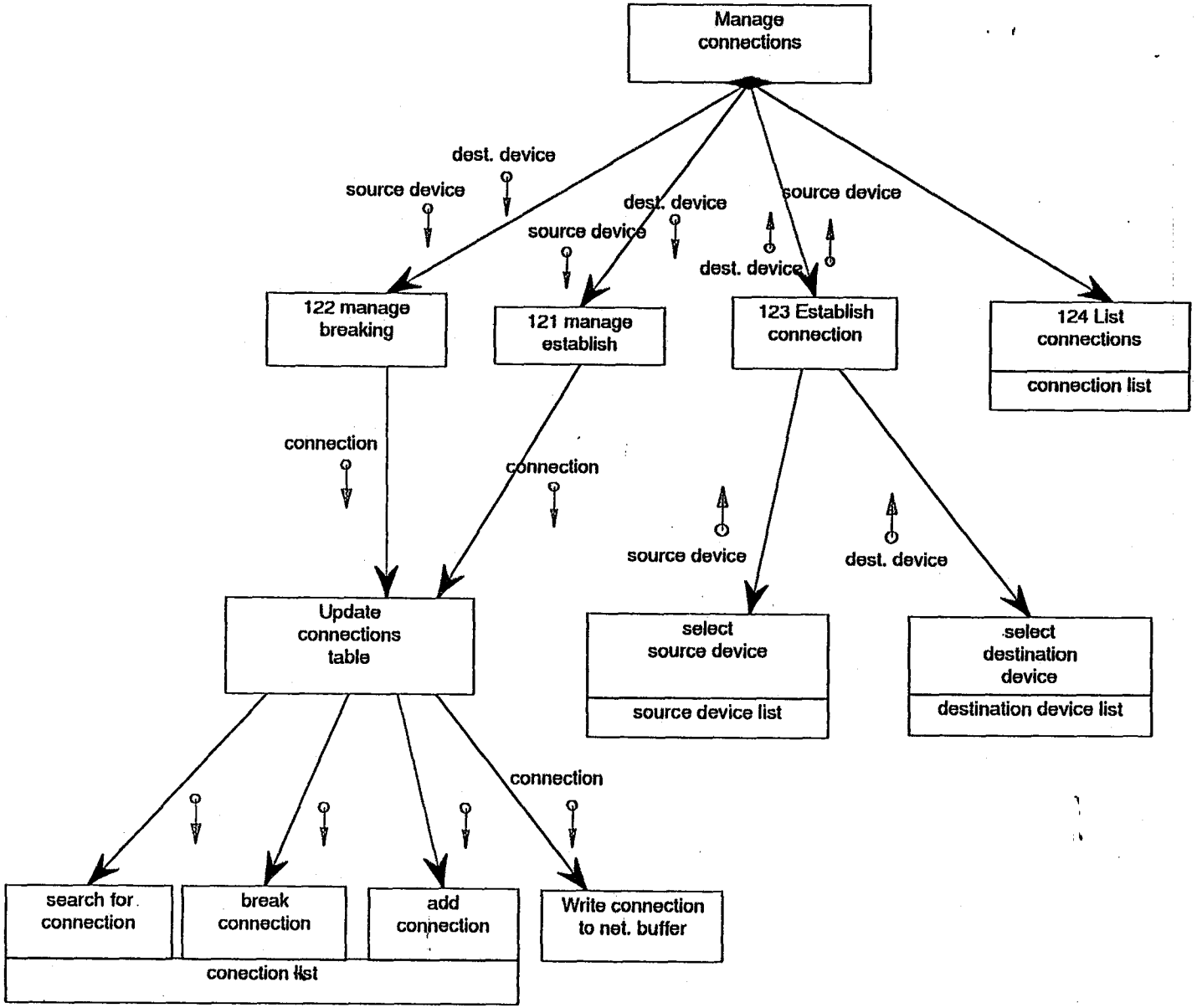

interrupt
P
interrupt
handler
trigger
P
read MIDI
from ports
MIDI data available
D
Incoming
MIDI data
buffer
P
Control MIDI
processing
enable
processed MIDI data available
P
Process MIDI
messages
enable
enable
D
outgoing port MIDI
data buffer
D
outgoing
network MIDI
data buff
P
Write to
the network
P
Write to
Ports
data to the network
MIDI to the devices

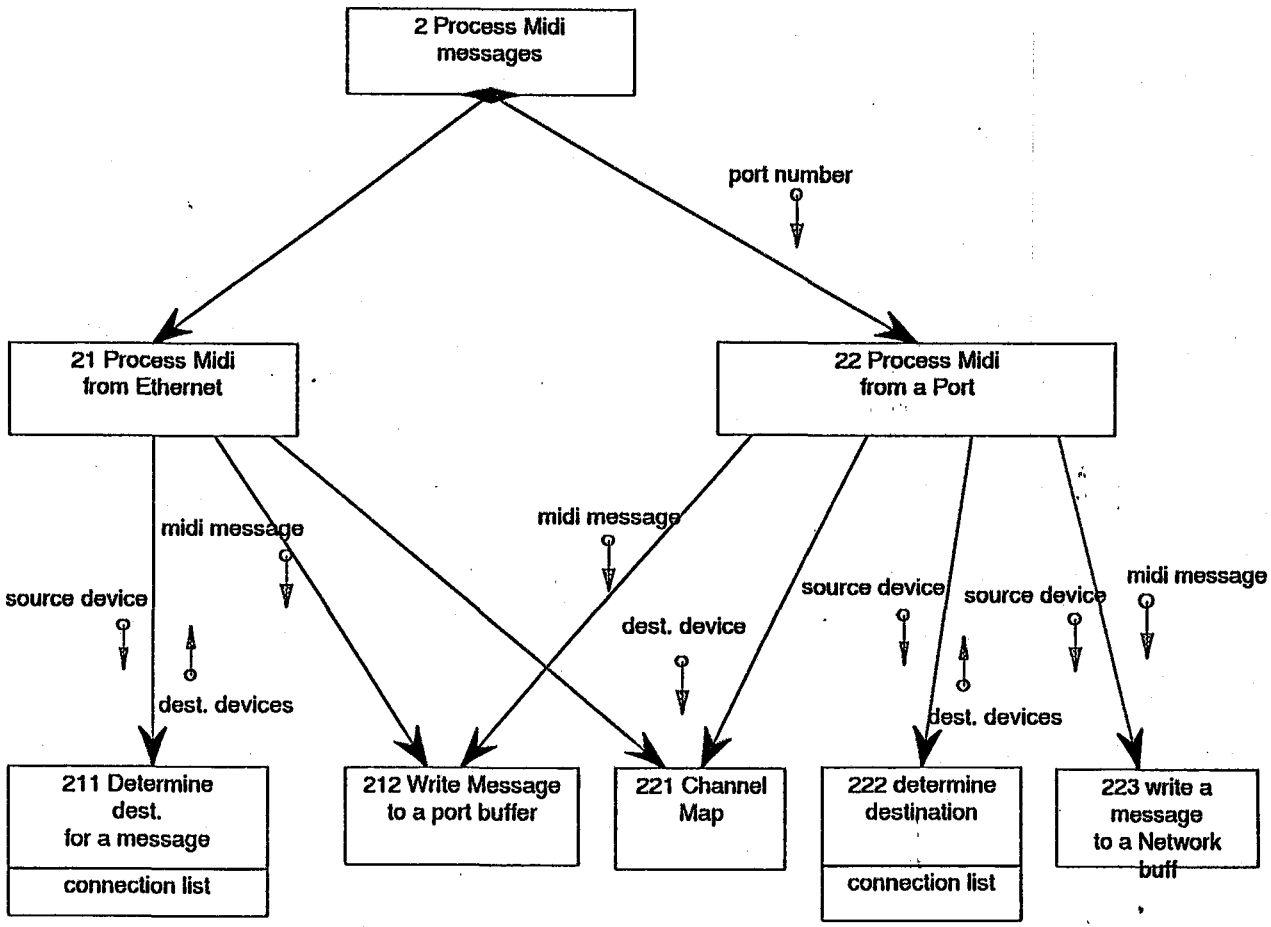

interrupt
P
packet
driver
P
Read MIDI data
from Network
MIDI data available
D
incoming
network MIDI
data buff
P
Control MIDI
processing
P
Process MIDI
data
from network
MIDI data processed
enable
D
outgoing MIDI
data buffer
P
Write to
Ports
MIDI to the port

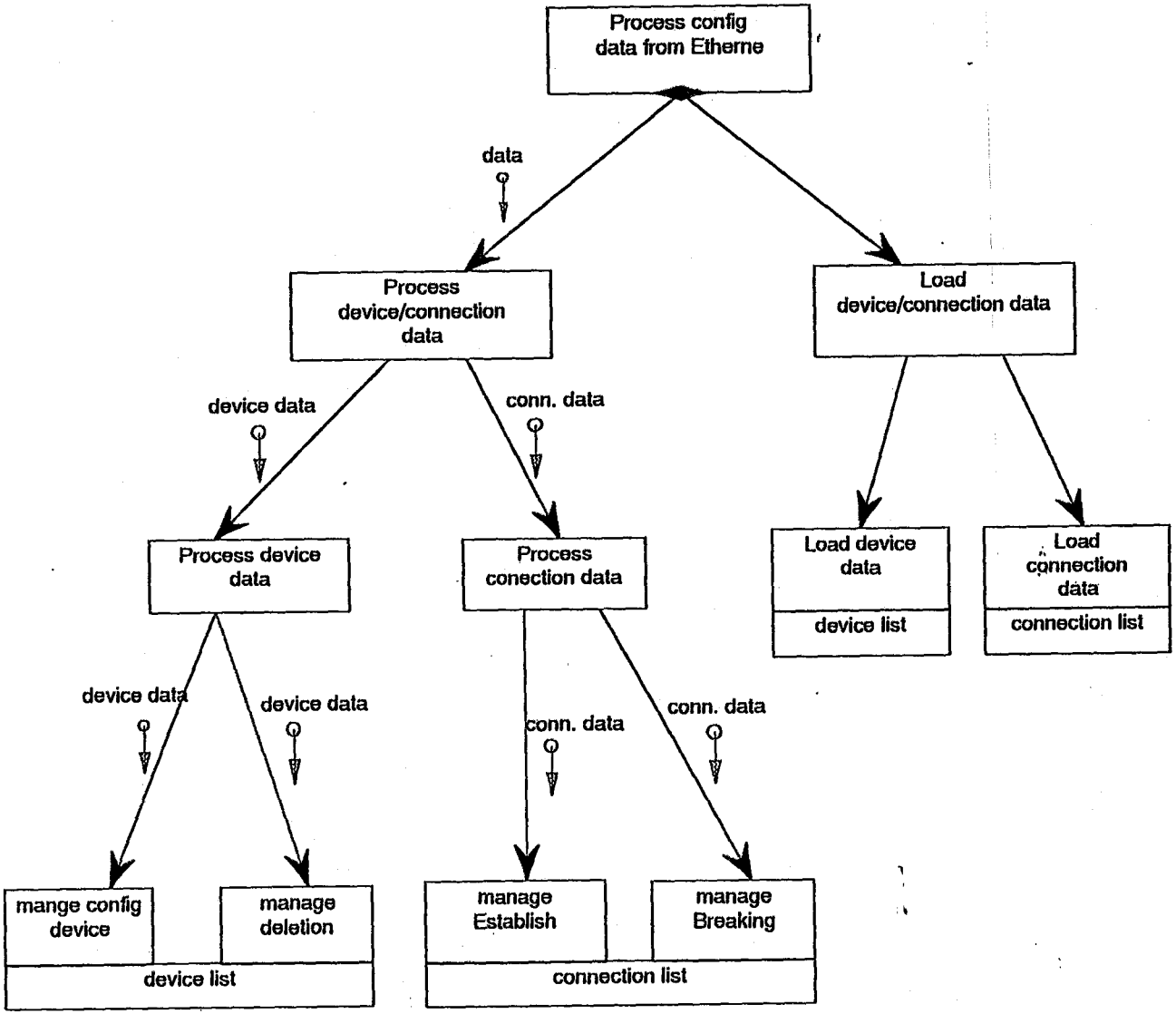

interrupt
P
packet
driver
P
Read Config.
data
from Network
Config. data available
P
Control Config.
data processing
D
config data
buffer
enable
P
Process config.
data from
networ

2.2 Stucture Charts


11 Manage devices
device
device type
device
device
device type
device type
112 Manage Deletion
111 Manage configuration
113 device information
114 list devices
device list
device
device
M_Id
P_Id
P_Id
Ch. #
S. name
M_id
1111 Update device list
1131 Accept Midinet Id
1132 Accept Port Id
1133 Accept symbolic name
1134 Accept Channel Number
device
M_id
flag
M_id
P_id
flag
search for device
add device
delete device
Write to Network Buffer
device list
validate MidiNet Id
Validate Port Id


Manage connections
dest. device
source device
source device
dest. device
source device
dest. device
source device
122 manage breaking
121 manage establish
123 Establish connection
124 List connections
connection list
connection
connection
source device
dest. device
Update connections table
select source device
source device list
select destination device
destination device list
connection
search for connection
break connection
add connection
Write connection to net. buffer
conection list


2 Process Midi messages
port number
21 Process Midi from Ethernet
22 Process Midi from a Port
midi message
source device
dest. devices
midi message
dest. device
source device
source device
midi message
dest. devices
211 Determine dest. for a message
connection list
212 Write Message to a port buffer
221 Channel Map
222 determine destination
connection list
223 write a message to a Network buff


Process config data from Etherne
data
Process device/connection data
Load device/connection data
device data
conn. data
Process device data
Process conection data
Load device data
device list
Load connection data
connection list
device data
device data
conn. data
conn. data
mange config device
manage deletion
device list
manage Establish
manage Breaking
connection list

Appendix 3

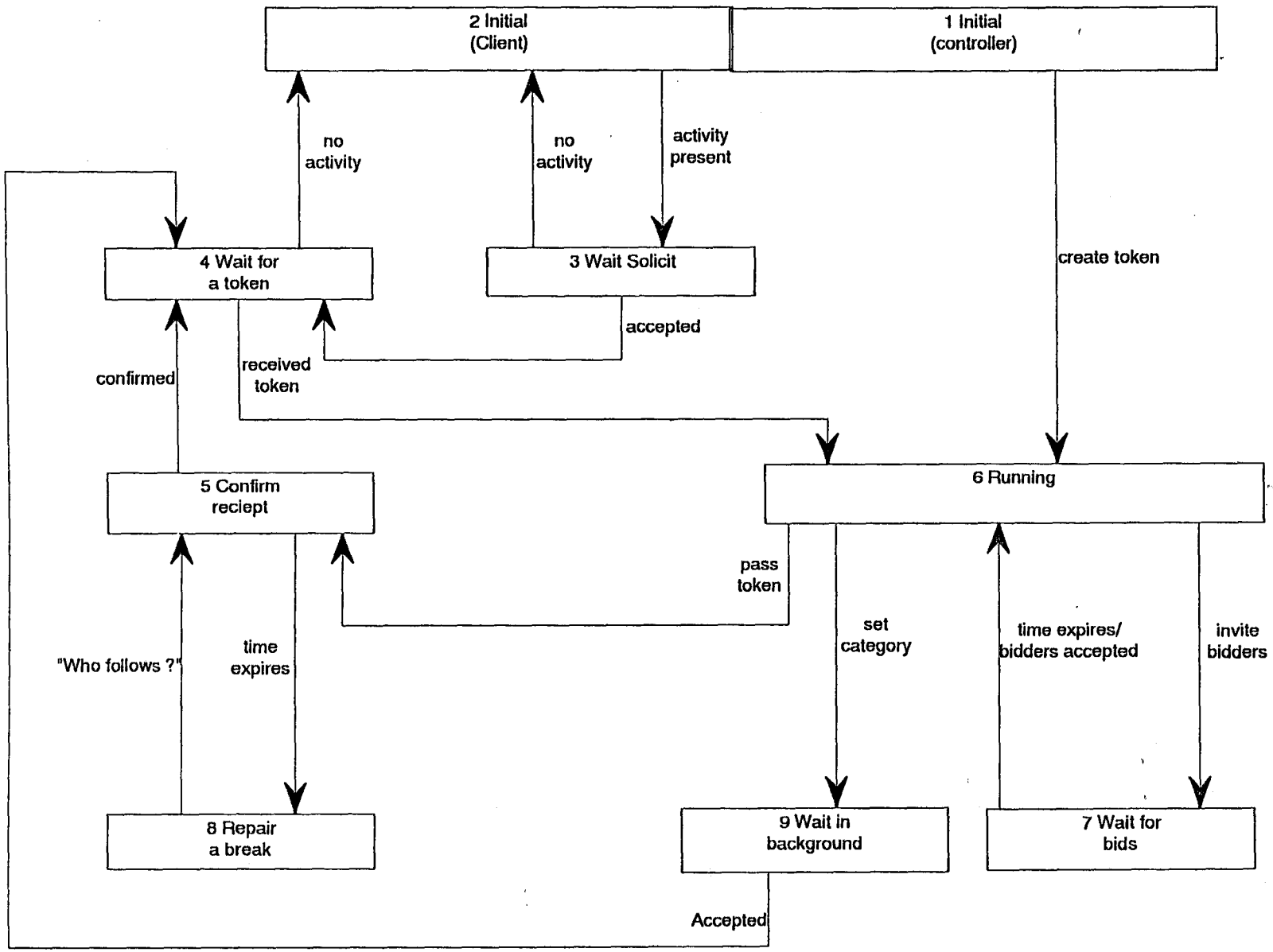
MIDI Messages

Data	00 - 7F	0 - 127	MIDI Data
Channel Status Bytes	80 - 8F	128 - 143	Note Off
	90 - 9F	144 - 159	Note On
	A0 - AF	160 - 175	Poly Key Pressure
	B0 - BF	176 - 191	Control Change
	C0 - CF	192 - 207	Program Change
	D0 - DF	208 - 223	Channel Pressure
	E0 - EF	224 - 239	Pitch Wheel Change
System Common Status Bytes	F0	240	System Exclusive
	F1	241	MTC Quater Frame
	F2	242	Song Position Pointer
	F3	243	Song Select
	F4	244	Undefined
	F5	245	Undefined
	F6	246	Tune Request
	F7	247	End of Exclusive
System Real Time Status Bytes	F8	248	MIDI Clock
	F9	249	Undefined
	FA	250	Start
	FB	251	Continue
	FC	252	Stop
	FD	253	Undefined
	FE	254	Active Sensing
	FF	255	System Reset

Appendix 4

Netlink Protocol

4.1 Data flow diagram


2 Initial
(Client)
1 Initial
(controller)
no
activity
no
activity
activity
present
create token
4 Wait for
a token
3 Wait Solicit
accepted
confirmed
received
token
5 Confirm
reciept
6 Running
pass
token
"Who follows ?"
time
expires
set
category
time expires/
bidders accepted
invite
bidders
8 Repair
a break
9 Wait in
background
7 Wait for
bids
Accepted

4.2 Algorithm/Rules

SOLICIT

```
Controller          (* running state *)

    if  ANT category not empty then

              invite a bIDder in ANT category

              time = Slot time
              wait for 'time'

              If 'time' expires then
                   try next station
              else
                   set successor     (* there is a bidder *)
    else
         pass token        (* Waiting for token state *)


    If NA category not empty
         invite a bidder in NA category (as above)


Station waiting to join   (* solicit wait state *)

    wait for invitation

    if solicit invitation then
         send request to join
         set predecessor        (* from the invitation *)
         set successor          (* wait for token state *)
    else
         wait                   (* wait solicits state *)


station successor   (* wait for token state *)
```

```
receive join ring packet
     set predecessor
```

Station predecessor

```
receive join ring packet
     set predecessor
```

Leave ring

Token holder (* running state *)

```
send set suc/prede-cessor message
pass token
exit
```

Token holder's successor (* wait for token state *)

```
receive suc/prede-cessor message
set predecessor
```

Token holder's predecessor (* wait for token state *)

```
receive suc/prede-cessor message
set successor
```

Change Category

Token Holder

```
if No data and count reached N and ring > 2 then
     send set category
else
     pass token
```

Token holder's successor

```
receive set category
set predecessor
update internal current ring
```

Token Holder's predecessor

```
receive set category
set successor
update intenal current ring
```

Other stations

```
update internal current ring
```

Ring Initialization

Controller (* initilise state *)

```
create token
successor = predecessor = myid (* single *)
```

Dead Successor

Token holder

```
pass token
wait for slot time

if nothing from successor then
     pass token again
     wait for slot time

     If nothing again and sid-mid # 2 then
```

```
repeat
     send 'who follows'  (* Repair break state *)
     wait for slot time  (*Confirm receipt state*)

     if nothing then
          try next station
     else
          set successor
          pass token
until reached successor or successful bidding
```

Token holder's successor (* wait for token state *)

```
receives 'who follows' message
send 'set successor' message
set predecessor
```

Any station

```
update internal link
```

No token

Controller

```
(* wait for token state or wait for solicit state *)

start 'last time heard of token' clock
if expires and no token and lowest ID
     start ring initializing  (* initialise state *)
```

GLOSSARY

Bandwidth: The information capacity of a communications line, this is measured in Hertz (Hz) and is the difference between the lowest and the highest frequency which can be carried on the line.

Channel: An information pathway over which MIDI data is transmitted or received.

Channel number: The lower four bits of each MIDI status byte which indicate the MIDI channel number for the data bytes that follow.

CSMA/CD: Carrier Sense multiple access with collision detection, the well known local network medium access protocol which originated with Ethernet.

Daisy chain: MIDI devices connected in series.

DIN - Deutche Industrie-Norm: A standard type of electrical connector, developed in Germany.

Effects: Devices that change the characteristics of an audio signal passed through them.

Fiber Optic: An increasingly popular type of communications medium in which light travelling within very fine fibres carries information: offer very high bandwidth, and is immune to electromagnetic noise.

FTP: File transfer protocol, a generic name for file transfer.

IEEE: The International Electrotechnical Commission, responsible for standardization in the area of electrical engineering.

IP: The Internet protocol developed for the US Arpenet.

LLC: Logical link control, a sublayer of the IEEE/ISO LAN

reference model.

MAC: Medium access control, also a sublayer of the LAN reference model. Together with the higher LLC, corresponds to the OSI data link layer.

Mixer: Sound mixer.

MTC - MIDI Time Code: A means of synchronizing events between devices.

Patch bay: Unit for connecting sound devices.

Packet: A block of information.

Pitch shift: Split signal producing thickening of a sound or a harmony.

Preamble: Precedes the operational part of a packet and is used for receiver synchronization.

Quantize: The rounding off of rhythmic values to a particular value, usually used to "correct" rhythmic errors.

Reverb: An effects device used to simulate the ambience of a room.

Sample: The digital recording of a sound. Sampling instruments have the ability to record, store, manipulate, and then play back sounds.

SMPTE - Society of Motion Picture and Television Engineers: This technical union created the standard synchronization code for film and video.

System Common Messages: The group of MIDI messages used primarily to enhance the functions of other commands.

System Exclusive Messages: MIDI codes for sending data for a specific instrument.

System Real Time Messages: MIDI messages that synchronize devices.

TCP: Transmission control protocol, the host-to-host protocol which all Internet hosts use to communicate.

Track: Band running along a tape (or area of computer memory) which can be recorded independently of other tracks.

Velocity: In MIDI, the speed with which a key, drum pad, or string is hit or released.

Volume: The MIDI code used to adjust the overall output level of an instrument.

UNIX: A well known computer operating system which was developed at AT&T Bell Laboratories in the USA.