The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

APIS: A REAL-TIME MESSAGE-ORIENTED MIDDLEWARE

by

Manie C. Steyn

A thesis submitted in partial fulfilment
of the requirements for the degree of

Master of Science (Electrical Engineering)

Electrical Engineering Department
University of Cape Town

February 2001

Approved by _____
Chairperson of Supervisory Committee

_____

_____

_____

Program Authorised
to Offer Degree _____

Date _____

University of Cape Town

Abstract

APIS: A REAL-TIME MESSAGE-ORIENTED MIDDLEWARE

by Manie C. Steyn

Chairperson of the Supervisory Committee: Mr. M. J. Ventura
Department of Electrical Engineering

This thesis presents an investigation and evaluation of a Real-Time Message-Oriented Middleware (MOM) implementation using commercial off-the-shelf (COTS) software components.

The Application Interface Services (APIS) is an implementation of a real-time MOM that provides network services to sub-systems of a large-scale distributed system.

It is shown that the characteristics of a MOM are well suited to a real-time message distribution application and that APIS, as an implementation of a real-time MOM, can provide a heterogeneous network interface to sub-systems of a distributed real-time nature. This simplifies the task of implementing information exchange and provides a definitive boundary for assigning responsibility during system design and development.

TABLE OF CONTENTS

1 INTODUCTION 1

1.1 SCOPE 2
1.2 SYSTEM ARCHITECTURE 3
1.3 RELATED WORK 7
1.4 DOCUMENT OVERVIEW 9

2 REAL-TIME NETWORK MIDDLEWARE 11

2.1 INFORMATION CHARACTERISTICS 11
2.1.1 Signals 11
2.1.2 Commands 13
2.1.3 State Variables 14
2.1.4 Requests 14
2.2 NETWORK SERVICE MODELS 15
2.2.1 Connection Oriented 15
2.2.2 Connectionless (Datagram) 15
2.2.3 Transaction 15
2.2.4 Broadcast 16
2.2.5 Multicast 16
2.3 REAL-TIME ATTRIBUTES 16
2.3.1 Determinism 16
2.3.2 Responsiveness 17
2.3.3 Priority and Precedence 17
2.3.4 Latency and Jitter 17
2.4 MESSAGE-ORIENTED MIDDLEWARE 19
2.5 APIS AS MIDDLEWARE 21

3 APPLICATION INTERFACE SERVICES (APIS) 22

3.1 IMS OVERVIEW 22
3.2 APIS REQUIREMENTS 23
3.3 APIS NOMENCLATURE 24
3.3.1 APIS Message 25
3.3.2 APIS Message ID 25
3.3.3 APIS Producer 26
3.3.4 APIS Consumer 26
3.4 THE APIS USER INTERFACE 26
3.4.1 Concept of Operation 28
3.4.2 Message Primitives 30

4 APIS ARCHITECTURE AND DESIGN 34

4.1 DESIGN OBJECTIVES 34
4.1.1 Deterministic Data (Signal) Distribution 34
4.1.2 Off-Host Communication Architecture 36
4.1.3 Use of COTS Technology 37
4.2 ARCHITECTURAL OVERVIEW 37
4.2.1 Real-Time Database Access 39
4.2.2 Real-Time Scheduled Tasks 40

5 APIS INFRASTRUCTURE 42

5.1 SELECTING COTS COMPONENTS 42
5.1.1 Media Access Control *43*
5.1.2 Transport Protocol *48*
5.1.3 Real-Time Operating Systems (RTOS) *49*
5.2 INTEGRATING COTS COMPONENTS 51
5.3 TARGET PLATFORM 51
5.4 MULTICAST GROUP MANAGEMENT 52
5.5 GUARANTEEING MESSAGE DEADLINES 54
5.5.1 Performance Parameters *55*
5.5.2 Synchronous Bandwidth Allocation *56*
5.5.3 Buffer Allocation *57*

6 LIMITATIONS OF MENTAT'S XTP 59

6.1 LACK OF UNACKNOWLEDGED MULTICAST SERVICE 59
6.2 FILE DESCRIPTOR LIMIT 60

7 PERFORMANCE MEASUREMENTS 62

7.1 END-TO-END LATENCY MEASUREMENTS 63
7.2 LATENCY PROFILE OF AN APIS MESSAGE 66

8 CONCLUSION 71

9 REFERENCES 73

APPENDIX A IMS SHORT-FORM SPECIFICATIONS 78

APPENDIX B FDDI ADAPTOR SPECIFICATIONS 79

APPENDIX C APIS INTERFACE CONTROL DOCUMENT 80

APPENDIX D APIS SOFTWARE DESIGN DOCUMENT 81

APPENDIX E APIS MESSAGE LATENCY PROFILE 82

LIST OF FIGURES

Number *Page*

Figure 1. APIS - ASU Interaction diagram 27
Figure 2. Off-Host Architecture 37
Figure 3. APIS Architecture 39
Figure 4. The Seven Layers of the ISO/OSI reference model and their mapping to the layers defined by the FDDI standard 43
Figure 5. APIS Implementation using XTP and FDDI 52
Figure 6. APIS Message IDs, Class D IP and IEEE MAC Addresses 53
Figure 7. APIS Implementation without using XTP. 61
Figure 8. APIS Latency vs. Message Size (Log Scale) 63
Figure 9. APIS Latency vs. Message Size (Linear Scale) 65
Figure 10. Logic Analyser Screencapture of a latency profile 66
Figure 11. Latency Timeline 67
Figure 12. Profile of a typical APIS Message transfer 69
Figure 13. Profile of a typical APIS Message transfer, percentage breakdown . 70

ACKNOWLEDGMENTS

The author wishes to acknowledge and thank C^2I^2 Systems (Pty) Ltd for the opportunity of presenting this project as a thesis. In particular the contributions of Dr. Richard Young, who is also the chief architect of APIS and Mr. Etienne de Villiers, the project manager of the IMS, are acknowledged.

Thanks are also due to Chelsea Henning for her proof reading of the final drafts of this text and to the author's wife Laura, for her continuous support and motivation during this project.

Finally the author wishes to thank his supervisor, Mr. Neco Ventura for his guidance.

Chapter 1

1 INTODUCTION

Interconnecting mission-critical, real-time distributed systems is a complex task in that many nodes, differing in hardware, software and functional requirements, interact with each other, requiring a mixture of reliable delivery and timing services.

A large-scale distributed system often integrates multiple heterogeneous sub-systems resulting in a configuration where applications, executing under many different real-time and non-real-time operating systems, on a variety of different hardware platforms, have to interact to complete a single mission or perform a single collaborative function.

Such a system is also often implemented by several contractors and concludes with a complex integration phase. Apart from the complexity of managing the networking issues of such a system, the contractual boundaries are difficult to define, making the assignment of responsibility for meeting network specifications e.g. latency and jitter, a complicated matter.

Being part of the system design team for one such a mission-critical, real-time distributed system, R. M. Young proposed an Information Management System (IMS), described in his PhD thesis [52]. The IMS realises the network backbone and network interface cards (NICs) as a separate sub-system responsible for timely delivery of messages and seamless integration of all other sub-systems.

Striving towards modularity and reusability, the IMS consists of various network services that combine to establish a seemingly simple interconnect medium for the sub-systems of a mission-critical, real-time distributed system.

1.1 Scope

This thesis is an investigation and evaluation of an implementation of one of the services, namely the Application Interface Services (APIS), of the IMS proposed by Young [52].

APIS[1] uses the principles of Message-Oriented Middleware (MOM) to distribute messages timely across the network without needing sub-systems to establish a connection between the sending and receiving application.

Using APIS, applications simply *produce* the data they know about and *demand* the data they need.

A MOM such as APIS provides high-level programming interfaces that abstract the underlying data communications and access components of each part of a distributed system. It isolates clients from back-end processes and decouples the application from the network process. This decoupling is the most important distinction between APIS and other connection-oriented middleware. For example, the *Common Object Request Broker Architecture* (CORBA) [2] provides an environment in which software objects can be shared across networks, but since it is essentially client-server architecture, the network address of the server, or *Object Request Broker* (ORB), needs to be known to the client. APIS on the other hand is message-oriented rather than connection-oriented. APIS utilises the uniqueness of the data identifier (APIS Message ID), rather than the network address, to deliver data messages between applications.

[1] Digressing from the technical nature of this thesis, we found it of interest to learn that the acronym also refers to a figure in ancient Egyptian mythology. The creator god PTAH of Memphis had as his herald the bull-god Apis to communicate with mankind on his behalf in the delivery of oracles. It is quite fitting that APIS existed as "communication middleware" even during the time of the Ancients.

This thesis will investigate the APIS system architecture as proposed by Young and then present a detailed implementation of APIS by the candidate followed by an evaluation of its performance.

Throughout the design and implementation of APIS the three main objectives were to provide data distribution in real-time, to provide an off-host communication architecture and to use commercial off-the-shelf (COTS) technology wherever possible.

In order to meet these objectives, this thesis sets out to identify suitable software and hardware components by means of a literature survey. It then details the specific issues pertaining to the implementation of APIS and presents subsequent performance measurements.

The design, implementation and qualification of APIS spans a four year period during which the author was responsible for the design and coding, in C++, of APIS, as well as the integration and testing of all the services of the IMS.

1.2 System Architecture

As previously stated the APIS real-time message-oriented middleware was conceptualised by Young in his PhD thesis [52]. According to Young, APIS is a network communications protocol that was designed to provide real-time data delivery capabilities to applications that can dynamically setup and manage the system dataflow. APIS do not detract from the real-time capabilities of the network layers beneath it nor impact on the real-time performance of the user system above it.

In order to achieve these goals the system architecture that is detailed in this thesis can be summarised as follows:

APIS implements Layers 5 to 7 of the ISO OSI Reference Model, interfacing to the Transport Layer (Layer 4) below and the APIS Service User (ASU)

above. The ASU is a *producer* and/or *consumer* of data of different types that are packaged in APIS Messages. APIS Messages are pre-defined by the system integrator as part of the system design and each is assigned a unique Message Identifier. The APIS protocol establishes the necessary communication channels between the producers and consumers of a particular APIS Message. LAN dataflow will therefore be determined by APIS Messages and not by predefined ASU addresses.

APIS achieves this *data driven approach* to dataflow management by making use of multicast group management. In contrast to a unicast-addressing scheme where a unique address is allocated to each node on the network, APIS allocates a unique multicast group address to each APIS Message that is defined in the system. A one-to-one mapping exist between the APIS Message ID assigned by the system integrator at design time and the multicast group address assigned by APIS at runtime, refer to paragraph 5.4 for more details. Making use of a multicast capable Transport Layer (OSI Layer 4) protocol such as the Xpress Transport Protocol (XTP), APIS simply joins a multicast group in order to produce or demand a specific APIS Message as specified by the ASU.

Initially XTP was chosen as the Transport Layer protocol for APIS since it was designed with real-time systems in mind, minimising the protocol overhead and connection setup time as well as supporting reliable multicast connections (refer to paragraph 5.1.2). Due to several problems that were encountered with the version of XTP available at the time (refer to chapter 6), XTP was removed from the network protocol stack for the second generation of APIS. It was then implemented directly on the Data Link Layer (OSI Layer 2) using the multicast services provided by the Logical Link Control (LLC) protocol.

To achieve real-time delivery of data and guaranteed Quality of Service (QOS) the underlying network topology was restricted to a Fibre Distributed Data

Interface (FDDI) LAN. The timed token protocol, synchronous transmission mode, bandwidth reservation and the coding schemes of FDDI to reduce packet errors, together with the very low Bit Error Rate (BER) of optical fibre, were some of the reasons for selecting this network technology as detailed in paragraph 5.1.1. Restricting the network to a *local area network* was required in order to make use of the guarantees provided by the FDDI timed token protocol. Once the FDDI LAN is expanded by means of network routers, multiple tokens will exist on the various segments making it impossible to guarantee the arrival of a token within a prescribed deadline.

Another important design decision taken to guarantee the real-time delivery of data and therefore QOS was to restrict the size of APIS Messages to 4 000 bytes. The APIS QOS guarantee relies on a data message being delivered within one rotation of the FDDI token, which has a guaranteed upper bound. The payload of an FDDI frame is just larger than 4 000 bytes. Restricting the size of an APIS Message to this value eliminates the possibility of fragmentation and hence late delivery.

QOS is thus guaranteed firstly by making use of the FDDI synchronous bandwidth allocation scheme to ensure that a producer of a high priority message never has to wait for network access. Secondly the maximum size of an APIS Message is chosen such that it is possible to deliver within one token rotation period, which has a guaranteed upper limit.

The APIS protocol was implemented in C++ for the VxWorks real-time operating system. It was implemented as a group of independent tasks, executing simultaneously to provide all the functionality to the ASU. A local database on each node keeps track of the ASUs and APIS Messages that concern this node only. The Administration Task is one of three types of tasks that execute as part of APIS on the node. It handles requests from the ASU to add or remove producers/consumers from the local database. A group of Send Tasks receives data from producers on this node and sends it

to the appropriate multicast group address. A group of Receive Tasks receive data from the LAN and delivers it to the appropriate consumers on this node. Paragraph 4.2 has a detailed explanation of each of these tasks and their interaction with the database.

The local database that exists on each node only has records associated with the ASUs of that node. Since a pre-defined one-to-one mapping exists between APIS Message IDs and multicast group addresses, there is no need to distribute the database across the network. The only purpose of the database is to associate the ASUs of a node as producers or consumers of a particular APIS Message, i.e. a particular multicast group address.

Priorities are assigned to the APIS Messages by ASUs when they register as producers creating the need for a priority based scheduler. The pre-emptive task scheduler of VxWorks is used to accomplish this. Each of the Send Tasks, there are eight in the current version of APIS, is assigned a different priority. High priority APIS Messages are then assigned to Send Tasks with correspondingly high priorities ensuring that they will be processed and delivered ahead of the lower priority messages.

APIS was implemented on a Pentium based Single Board Computer (SBC) referred to as the Network Interface Card (NIC). The NIC was completely self-contained and ran the VxWorks executable, consisting of the APIS tasks and network protocols, from FLASH memory. The NIC has a Multibus II parallel backplane interface that was used to interface to the ASU. The ASU is implemented on a separate SBC that is entirely under the control of the system integrator. This *off-host architecture* provides a heterogeneous distributed computing environment allowing the ASU's application to execute on its own processor under an operating system of its choice, preventing the host from impacting on the real-time processes of APIS and vice-versa. The off-host architecture is also the key element in defining the contractual boundary

between the implementers responsible for the applications and those responsible for the network interface.

1.3 Related Work

Although this thesis only details the implementation and evaluation of a network middleware conceptualised by Young [52], a literature survey was conducted to select components best suited to the task, and to set their parameters for optimum performance.

A very good introduction to the requirements and traffic characteristics of distributed real-time applications was presented in a paper by Aras et al. [4]. Other papers that examine hard real-time communication characteristics were published by Malcolm and Zhao [27], Clingiroglu et al. [10] and Simon [42]. These papers provide a broad background to communication in general.

The performance of real-time distributed systems as a whole (not only focussing on network performance) is discussed in two papers by Shin et al. [40] and [41], and also by Yen et al. [51] and Shankar [39].

Papers that deal with the issues of distributed real-time communication systems includes Chen et al. [7] who concludes that the transport layer plays a very important role in meeting message deadlines, Chlamtac et al. [9] who proves a worst case bound for FIFO-based networks, Feng et al. [13] who develops a method to test whether application-to-application message deadlines are met, and Zheng et al. [55] who addresses the issues of reliability when delivering messages within prespecified delay bounds.

The two papers that introduce the timed token access protocol of FDDI and provide guidelines for setting TTRT were published by Dykman and Bux [12] and Raj Jain [19]. Jain also published the *FDDI Handbook* [18]. In these works it was proved that the time elapsed between two consecutive visits of the token at a node is bounded by 2 × TTRT. The generalisation of this result

was reported by Agrawal et al. [1]; the time elapsed between any v consecutive visits is bound by $v \times$ TTRT, and the worst-case achievable utilisation was shown to be almost 33% using synchronous bandwidth allocation. Extensive results on the guarantee of hard real-time messages with arbitrary deadlines in FDDI networks, the buffer management, and the study of guarantee probability were reported by Nicholas Malcolm et al. in three papers [26], [28] and [29]. The properties of FDDI networks in general and its suitability towards real-time communication are also discussed in three more papers by Chen, Zhao et al. [6] [8] [54]. Hamdaoui and Ramanathan [16] address the selection of timed token protocol parameters to guarantee message deadlines.

Several papers have been published that address the allocation of synchronous capacity (bandwidth), also known as Synchronous Bandwidth Allocation (SBA). Chen et al. [5] developed and analysed an optimal scheme to ensure the transmission of synchronous messages before their deadlines. Zhang et al. [53] published an enhancement to this allocation scheme that also included an exact upper bound on the time between successive token arrivals. Others that analyse synchronous bandwidth allocation include Feng et al. [14] and [15], Kamat et al. [20] and [21], and Zheng and Shin [56].

Two papers that examine the features and functionality of XTP in relation to the requirements of distributed real-time systems were published by Strayer et al. [43] and [45]. These papers conclude that the multicast capability and the internal priority operation of XTP will prove to be of critical importance to modern real-time distributed systems. Michel et al. [32] evaluated an Off-Host Communication architecture consisting of XTP and a VME backplane-bus, addressing the advantages and performance concerns inherit to off-host protocol execution. Other works relating to XTP were published by Atwood et al. [3], Mechler et al. [30] and de Rezzende et al. [37].

More specifically related to APIS is the White Paper on Real-Time CORBA [2], the paper on Fast Socket, a high performance transport protocol

implementation based on the Berkeley Sockets API [38], and NDSS: A Real-Time Publish-Subscribe Network [36]. APIS is significantly different from all three these technologies in that it allows for an Off-Host Architecture implementation.

1.4 Document Overview

Chapter 2 introduces the concept of Message-Oriented Middleware. First there is an overview of real-time information characteristics and the network service models used to deal with them. We emphasise that APIS was designed to distribute real-time *signal* information. This type of information is periodic in nature and very latency critical; it is better to lose one message and wait for the next update than to receive stale data. Since multiple consumers could exist to receive the information it is best distributed by means of a multicast service. The APIS middleware uses multicasting technology to deliver signals (called APIS Messages) from multiple producers to multiple consumers without the need for the ASU to specify address information; only the APIS Message ID is significant.

Chapter 3 defines the APIS middleware and the services it provide. It starts of with an overview of the IMS of which APIS forms a part. The requirement specification for APIS is summarised, highlighting the distribution of 4 000 byte messages within 5 ms latencies to multiple consumers simultaneously as the most important requirement. The design of the user interface is also discussed in order to describe the services that APIS provides to the ASU.

Chapter 4 discusses the software architecture of APIS. The design objectives are identified and discussed under the following headings: Deterministic Data Distribution, Off-Host Communication Architecture and Use of COTS Technology. The design of the local database and the three types of real-time tasks i.e. Administration, Receive and Send Tasks that forms the core of APIS are detailed.

Chapter 5 details the selection of software and hardware components and their integration into APIS. The selection of FDDI as the physical medium because of its timed token protocol and synchronous bandwidth reservation properties are discussed. We also address the selection of XTP as the Transport Layer Protocol and VxWorks as the real-time operating system.

Chapter 6 lists some problems encountered and explains why a second implementation of APIS, without XTP, was considered.

The main performance requirement of APIS is the delivery of a 4 000 byte message within a latency of 5 ms. Hence in Chapter 7 we discuss how this requirement was tested for and present some measurement results both under no load and 80% network utilisation conditions. We also present a latency profile of an APIS Message transmission from which it is clear that the majority of the time (about 60%) is used to transfer the APIS message across the Multibus II interface to and from the ASU.

In Chapter 8 we conclude that message-oriented middleware is well suited to the requirements of distributing signals in a real-time distributed system. It is also concluded that APIS, as an implementation of a real-time MOM, meets the requirements for message distribution within deterministic latency bounds.

While conducting this project, the candidate also authored the Interface Design Document and the Software Design Document included in the appendices.

Chapter 2

2 REAL-TIME NETWORK MIDDLEWARE

In this chapter we introduce the concept of network middleware and specifically that of message-oriented middleware. We begin with an overview of real-time information characteristics, specifically that of *signals*, which, as will be detailed below, is the category of information best suited for distribution via MOM. We then consider the traditional network service models that are currently used to distribute information. Finally we discuss message-oriented middleware and APIS as an implementation of a mission critical real-time MOM.

2.1 Information Characteristics

Real-time distributed applications could have requirements for several types of information flows, each with vastly different characteristics and requirements. Pardo-Castellote, et al. [36] classifies four common types of information flow, which are discussed below:

2.1.1 Signals

Signals are unsolicited data messages containing measurements of quantities that change over time. They are usually updated repeatedly and may require delivery to more than one destination in the system. An example of signal data could be the wind speed and direction provided by an anemometer on a naval vessel; the latest signal data has to be distributed to both the Navigational Sub-System to plot a course and the Electronic Warfare Sub-Systems for ballistics calculations. Both sub-systems rely on receiving the latest information while it is still valid; receiving stale information could lead to a navigation error or worse even, the firing of a stray missile.

Signals are of particular interest to us since the bulk of the information exchange in the Combat Suite Architecture, for which Young proposed the APIS design, are of this type.

Some properties of signals (particularly as it applies to the Combat Suite Architecture) are:

- **Time-critical** - Signals have a well defined time-to-live and are useless if the data is old (also referred to as stale data).

- **Idempotent**[2] - Repeated updates are acceptable. *Producers* may provide information at a higher rate than *Consumer* need; furthermore, since a mission critical distributed architecture allows for multiple redundant sources, *Consumers* may receive duplicate (idempotent) information.

- **Last-is-best** - Latest information is more important than retransmitting missed samples.

- **High bandwidth** requirements - Due to the repetitive nature of signals, they require bandwidth resources directly proportional to the payload size and repetition rate of the signal.

Aras et al. [1] describes this source of data as "*a sensor which samples a physical quantity to produce a digital signal*". Moreover, three models are described that can approximate most of the sources of this nature:

- **Constant Bit Rate (CBR):** Fixed-size packets at deterministic intervals. Digitised voice would be and example of CBR signals.

[2] **Idempotent:** Relating to or being a mathematical quantity which when applied to itself under a given binary operation (as multiplication) equals itself. [Merriam-Webster's Collegiate Dictionary http://www.m-w.com]

- **Variable Bit Rate (VBR):** (on/off sources) The source alternates between a period in which fixed-size packets arrive with deterministic spacing (CBR) and an idle period. This model can be used to describe signals produced by a 'mouse' or 'trackball' device.

- **Periodic with Variable packet sizes:** Each period, the source submits a single packet of variable length to the network. This model best describe a stream of compressed video information. The size of each 'frame' will vary as the compression algorithm encodes and transmits the changes in the scene.

2.1.2 Commands

Commands are used to effect change in a system. It is paramount to a mission-critical system that sequences of commands are delivered once and only once; the system cannot miss any intermediate steps or execute a step twice. The *fire* command sequence from the Tracker Radar Sub-System to the Weapons Control Unit of a naval vessel is a good example of a sequence of (very critical) commands and confirmations.

Some properties of commands are:

- **Reliable** - The system cannot lose any commands.

- **Sequential** - The order of a sequence of commands must be preserved.

- **Singular** - Must be delivered reliably once and only once.

- Often **not time-critical** - Since reliability is the primary requirement for *Commands*, systems often tolerate retransmission of commands that could have impact on their latency.

2.1.3 State Variables

State Variables reflects the overall state or goals of the distributed application. On a naval vessel for example, it is important that every sub-system is aware of the current *combat state* of the vessel, sub-system responses to external information could be quite different depending on this system-wide State Variable. In such a distributed system it is very important that at any instant, all sub-systems have the exact same value for a specific State Variable; slow propagation of a state change could result in a *race* condition.

Pardo-Castellote, et al. refers to this type of information flow as *Status*. We however prefer the term *State Variable* to describe the application's system wide persistent data and use the term *Status* to refer to the current status of the network middleware.

Some properties of State Variables are:

- **Persistent** - State Variables usually persists for some time.
- **Idempotent** - Repeated updates are acceptable.
- Sometimes **time-critical.**
- Sometimes **reliable.**

2.1.4 Requests

Requests imply a two-way transaction; the client sends the request and the server returns a response.

Some properties of requests are:

- **Reliable** - The system cannot lose any requests.
- **Synchronous** - If the client waits until the request is fulfilled.
- **Asynchronous** - If the client does not wait until the request is fulfilled.

2.2 Network Service Models

Due to the unique requirements of each Information Type discussed in paragraph 2.1, different *Network Service Model* are used to implement each type in order to obtain optimal performance. The following traditional models have been summarised from [52]:

2.2.1 Connection Oriented

An association between two endpoints is established to transfer the user data. Usually an acknowledged service, this *virtual circuit* is useful for transferring reliable signals, commands or status type information between two previously established endpoints.

2.2.2 Connectionless (Datagram)

A datagram is a self-contained data entity. The absence of connection overhead makes it useful for transferring low latency signal type information. Usually this is an unacknowledged service model.

2.2.3 Transaction

In client-server architectures, nodes interact via transactions. The client initiates the transaction with a request, followed by a response from single or multiple servers.

2.2.4 Broadcast

This service model allows all nodes on the network to receive the same information simultaneously using a special broadcast address. Although it can be useful to distribute signal information, many nodes will receive and discard this information making it wasteful of processing power. Broadcast is an unreliable service.

2.2.5 Multicast

Multicast is a special case of broadcast where group-addresses are used to selectively deliver to a group of receivers. Multicast can be unreliable, partly reliable or completely reliable and is very useful in distributing signal type information.

2.3 Real-Time Attributes

Thus far in this chapter we have described several types of Information Flows as well as Network Service Models used during implementation. When implementing a network middleware for the distribution of data in a real-time distributed system, several real-time *attributes* have to be considered. Young [52] offers a detailed study of these *attributes*, a few of which are summarised below:

2.3.1 Determinism

The media access protocol is the key factor in establishing deterministic behaviour in distributed systems.

Distributed access schemes fall into two main categories, collision avoidance and token passing. Protocols such as Carrier Sense Multiple Access with Collision Detect (CSMA/CD) do not offer deterministic behaviour.

2.3.2 Responsiveness

Two approaches to real-time responsiveness exists:

A *time-triggered* approach is best suited where the external environment can be reasonably modelled. A time-triggered system is one that reacts to significant external events at pre-specified instants.

An *event-triggered* approach is best suited where the external environment is essentially random. An event-triggered system is one that reacts to significant external events directly and immediately.

Time-triggered systems are generally implemented using a pre-allocation of bandwidth, which is prone to saturation and collapse during an overload condition. Event-triggered systems on the other hand are able to deal with external events immediately and can handle further simultaneous unprioritised events with degraded performance. Such degradation can be made *graceful* by properly prioritising the events in a system.

2.3.3 Priority and Precedence

Young relates priority and precedence as follows: "In mission-critical systems, of equal significance to time-related *priority* is the characteristic of functional criticality, i.e. the importance of a message in relation to system functionality [or *precedence*]."

2.3.4 Latency and Jitter

Aras et al. [1] describe the distinguishing feature of real-time communication as "the fact that the value of the communication depends upon the times at which messages are successfully delivered to the recipient."

A specific maximum delay or *latency* bounds the desired delivery time for each message across the network, resulting in a deadline being associated with each message.

Just as the late arrival of a message can greatly reduce its value to the real-time application, the early arrival can also add to the complexity of the system, as it requires additional buffering at the receiver to achieve constant end-to-end latency.

Delay *jitter* is defined as the maximum variation in delay latency experienced by messages in a single connection.

Most real-time applications require a bound on jitter, in addition to a bound on the latency.

In control systems jitter is a much stricter requirement than latency that can be compensated for as a constant delay in the control algorithm.

Delays in computing algorithms are random due to data dependent conditional branches/loops and the unpredictable delays in sharing resources during execution.

It is therefore important that an upper bound for message delivery is known.

For a fixed sampling interval, Shin and Cui [41] classifies the negative effect of computing time delay on the stability of real-time control systems, into two problems. "The *delay problem* occurs when the computing time delay is nonzero but smaller than the sampling interval, while the *loss problem* occurs when the computing time delay is greater than, or equal to, the sampling interval, i.e., loss of the control output."

Their analysis modelled the computing time delay as a delay element in the control input after the DAC. Although their work assumed single processor architecture, a distributed system can be analysed, and the derived upper bounds can be applied, if one considers the computing delay to be the accumulated computing times of the various processors and the latency and jitter of the interconnecting network.

2.4 Message-Oriented Middleware

Having discussed Information Types, Network Service Models and Real-Time Attributes, we are now ready to introduce the concept of Message-Oriented Middleware (MOM).

Middleware, as the name suggests, aids to abstract the complexities of the service models described in paragraph 2.2 into a higher-level service. Most of the time these same service models are used in combinations to provide a new service through a new Application Programming Interface (API).

The traditional service models mentioned above, all require the sender to know the network address of the receiver and, in most cases, some kind of channel to be set up before data exchange can take place.

Message-Oriented Middleware (MOM), on the other hand, focuses on the message rather than the channel set-up to deliver the message. From the applications viewpoint, the communication is inherently connectionless; in other words, the sending and receiving applications do not establish a session; the sending application sends the message to the MOM, which is responsible for delivering it to the receiving application or multiple applications. This type of middleware has also been referred to as a *data driven* approach to dataflow management rather than the traditional *address driven* approach.

In their Frequently Asked Questions (FAQ) [17], the International Middleware Association (IMWA) answers the question, "What is MOM?" as follows:

> *"MOM is a specific class of middleware (software) that operates on the principles of message passing and/or message queuing. In general, MOM is characterised by a peer-to-peer distributed computing model supporting both synchronous and asynchronous interaction between distributed computing processes. MOM generally provides high level services, multi-protocol support and other system management services, thereby creating an infrastructure to support very reliable, scalable and performance-oriented distributed application networks in heterogeneous environments."*

MOM supports a wide range of communication models, including:

- One-way
- Request-Reply
- Store-and-Forward
- Publish-Subscribe

Publish-Subscribe MOM, also referred to as *dissemination architecture*, is of particular interest when distributing signals in distributed systems, and therefore to APIS. Nodes may produce data into 'the network' and consume data from 'the network' at will. Producers and Consumers are anonymous; neither knows where the data goes or originates. Publish-Subscribe MOM encourages many-to-many communications.

2.5 APIS as Middleware

APIS is intended as networking middleware for distributed navigational, command-and-control, or plant automation type applications. In such applications, various sensors or signal sources provide input to redundant control processors, which in turn command large amounts of actuators or other output devices to form a digital closed loop control system. The bulk of the information in such a system is of a latency-critical nature and falls into the *signal* category as discussed above. APIS is therefore concerned with the distribution of *signals* from multiple sources to many destinations within hard real-time deadlines.

The characteristics of publish-subscribe MOM as a specific class of middleware, set out in paragraph 2.4, is well suited to implement the services provided by APIS. In order to meet the requirement for message delivery within guaranteed deadlines, APIS must be implemented on a deterministic operating system as a real-time MOM.

It is imperative that the underlying network topology be chosen to support reliable and efficient message delivery for this many-to-many architecture.

3 APPLICATION INTERFACE SERVICES (APIS)

In this chapter we provide an introduction to the Application Interface Services (APIS). We start with an overview of the Information Management System (IMS) of which APIS forms a part. Next the requirement specification for APIS is summarised and the nomenclature of the APIS design is defined after which the design of the user interface is discussed.

3.1 IMS Overview

The Information Management Systems (IMS) introduced in Chapter 1 is an all-encompassing network solution for real-time distributed systems providing services in four categories:

- **Application Interface Services (APIS)**

  A publish-subscribe service is provided to sub-systems with the emphases on distributing latency critical data to other sub-systems without setting up network connections first.

- **Network Time Services (NTS)**

  NTS is built on the well-known Network Time Protocol (NTP), an Internet protocol optimised to keep the sub-systems synchronised to within sub-millisecond offsets.

- **File Transfer Services (FTS)**

  Large data transfers from sub-systems to print servers or backup servers are not latency critical and are handled by the FTS. The FTS operates at a

lower priority, allowing background file transfers to occur without interference to critical APIS data transfers.

- **Built-in Test Services (BITS)**

  A comprehensive user-interface allows the sub-system to query the state of the IMS or messages in the IMS at all times.

This thesis only addresses the implementation and evaluation of the APIS component of the IMS. The reader is referred to [52] for detailed descriptions of the other three components of the IMS. The short-form specifications for the IMS and its services, as published by C^2I^2 Systems, are included in Appendix A.

3.2 APIS Requirements

As already mentioned in Chapter 1, Young sets out the requirements for the IMS and APIS in his PhD Dissertation [52]. The short-form versions of these, as published by C^2I^2 Systems, are included in Appendix A.

Some of the requirements that have direct bearing on the implementation of APIS are discussed below:

- **Real-Time message delivery**

  The single most important requirement is the maximum latency of 5 ms for delivery of data messages (signals) of up to 4 000 bytes in size. This requirement is the primary consideration when selecting all the components for the APIS implementation. In order to meet this requirement the chosen network architecture, both the physical layers and the logical protocol stack, needs to be of a high-speed deterministic nature. The chosen operating system must also be of a real-time nature, able to provide deterministic scheduling.

- **Multiple Producers and Consumers**

  Another important requirement of APIS stipulates that multiple sub-systems shall be able to receive messages produced by any other sub-system, all within the specified latency. This requirement clearly eliminates the unicast connection oriented transport protocols in favour of a multicast protocol that delivers the message to all receivers at once; it would be impossible to meet the latency deadline by delivering the message sequentially via multiple virtual circuits.

- **Heterogeneous sub-system support**

  Supporting various host platforms can be achieved in one of two ways. Either implement the APIS middleware for each of the required combinations of hardware platform and operating system, or employ off-host architecture in which a backplane bus interconnects the host processor to the intelligent network card hosting the APIS middleware. Such architecture allows sub-systems contractors to implement their applications on many different hardware platforms while still using the same hardware for the network middleware. In a distributed system where different contractors often implement sub-systems, the extra cost of off-host network architecture is more often than not justified by the advantages of separating the two processes.

3.3 APIS Nomenclature

A number of concepts are fundamental to the way in which the APIS architecture operates. The following terms are used to describe the APIS architecture and concept of execution:

3.3.1 APIS Message

Applications communicate by sending user-defined APIS Messages, identified only by a user-provided *APIS Message ID*. There is no need for users to specify computer addresses, routes, port numbers, etc. The data contained in one APIS Message can be as large as 4 000 bytes[3].

3.3.2 APIS Message ID

A unique Message ID identifies an APIS Message on the LAN.

Four fields, each 16 bits in size, are provided for the user to name and categorise messages. A wildcard value of zero can be used in any of the fields to refer to a sub-set of messages. This mechanism allows the user to classify messages by *type*, *sub-type* and *identifier* (all user defined) as required by the specific distributed system implementation.

For instance, a certain distributed system defines the fourth field of the Message ID to reflect a certain class of event, and also defines an event type *ALARM* to be equal to 4. An APIS Message ID of 0:0:0:4 then refers to all messages in the system that report ALARM conditions.

The Message ID is a convenient identifier assigned by the system developer and is unrelated to either the sending or the receiving sub-system of the data Message.

[3] The limit of 4 000 bytes stems from the size of the data segment of an FDDI frame e.g. 4 478 bytes. FDDI guarantees the arrival of a frame in synchronous mode to be $\leq 2 \times$ TTRT. A value for the TTRT can thus be selected to ensure that an FDDI frame, and therefore an APIS message of max size 4 000 bytes, will meet a specified latency.

3.3.3 APIS Producer

An APIS Services User (ASU) registers with APIS as a Producer to *publish* information that is useful to other ASUs on the system. The Producer has no knowledge of the end-users of the data, only the system-wide unique APIS Message ID is specified.

The Producer supplies information regarding the size and repetition interval of the data, this information is used to allocate resources and guard against over-utilisation of the network.

Upon registration the Producer also assigns a priority to each APIS Message ID that will be produced. This is a local priority only that will determine the scheduling of messages onto the LAN when multiple message streams are being produces. APIS supports a total of eight priority levels at present.

3.3.4 APIS Consumer

An ASU registers with APIS as a Consumer to *subscribe* to information produced by other ASUs on the system. The Consumer has no knowledge of the originator of the data, only the system-wide unique APIS Message ID is specified.

The Consumer specifies a window during which no messages should be delivered; this *dead-time* guards against a fast producer overrunning a slow consumer.

3.4 The APIS User Interface

The design of the user interface or the Application Programming Interface (API) is a very important one since it offers the only interaction between the user's application and the network. The design of the API has to be of such a nature that it is clear and easy to implement by the user, leaving no room for

ambiguity. Just as important is the ability of the API to make available the features of the network middleware without degrading any functionality or performance offered by underlying protocols.

The APIS sub-system Interface Control Document (APIS ICD), included in Appendix C, defines a set of message primitives for communication between APIS and the sub-system application, also referred to as the Application Service User (ASU). This set of primitives can be implemented as function calls in a software library or, in the case of off-host architecture, as a set of messages to be sent across the backplane bus.

A concise description of the ASU's interaction with APIS using these primitives is presented (refer to Figure 1 below), followed by a short description of each message.

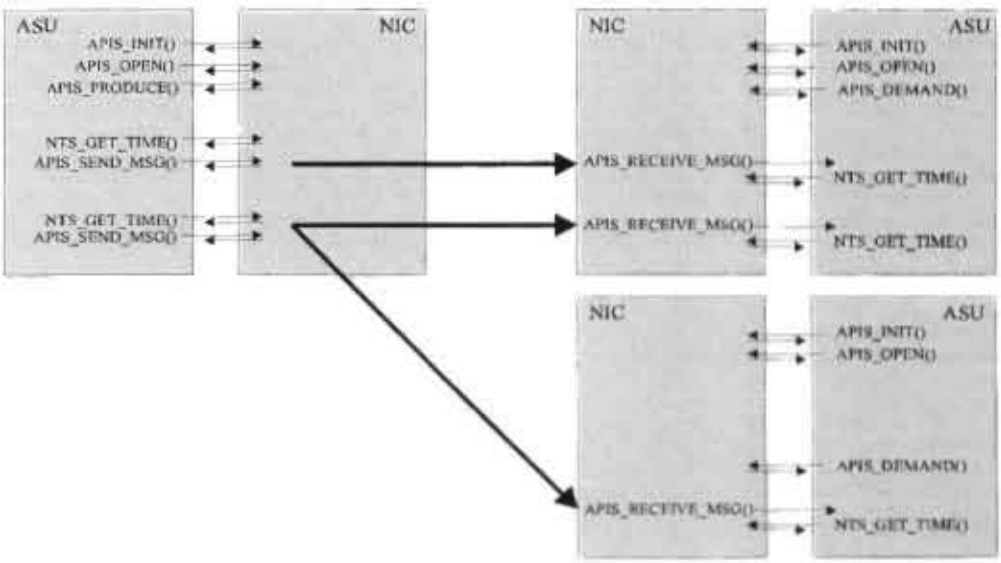


Figure 1. APIS - ASU Interaction diagram

Figure 1 also illustrates how the ASUs would use the NTS service that the IMS provide to time-stamp messages when they are transmitted and received.

3.4.1 Concept of Operation

As mentioned above, the set of primitives that defines the APIS API can either be implemented as a software library of function, or as a set of Protocol Data Units (PDUs) when off-host architecture is deployed. The PDUs define communication only between the ASU's application and the APIS middleware; they should not be mistaken for APIS Messages, which are the data carrying entities between a producing and consuming ASU. In fact, the producing and consuming ASUs use these API primitives to configure the APIS middleware, enabling the flow of APIS Messages between them.

To keep the discussion of the API architecture-independent, we shall refer only to *primitives* and their associated *parameters*. Details of implementing the API primitives as a set of PDUs for the Multibus II architecture, as well as a VxWorks message queue implementation are provided in Appendix C.

The use of each API primitive in a typical message exchange session is discussed below. A more detailed description of each primitive follows in paragraph 3.4.2.

Following a cold or warm start of a sub-system processor host, a once only APIS_INIT primitive is issued by the ASU to APIS. This allows APIS to initialise its database in respect of that host.

Before an ASU can make use of the services that APIS provides, it must register with APIS. This registration is required for authentication and resource allocation. An ASU registers with APIS utilising the APIS_OPEN primitive.

In order to transmit APIS Messages into the network, the producing ASU must inform APIS by using the APIS_PRODUCE primitive

and also provide, in the form of parameters, the APIS Message ID for the message it wants to produce as well as the message size and indented repetition rate. APIS will use this information to allocate bandwidth for this message stream.

Similarly, to receive APIS Messages from the network, the consuming ASU must demand it from APIS by using the APIS_DEMAND primitive. Once again, only the APIS Message ID is supplied as a parameter; the ASU does not specify the source or destination of any of these messages. APIS will determine the destination and source addresses for each message that is registered.

To distribute a message into the network, the producing ASU uses the APIS_SEND_MSG primitive. To deliver the messages from the Network to the consuming ASU, APIS uses the APIS_RECEIVE_MSG primitive.

An ASU can dynamically register and deregister APIS Messages by using the APIS_PRODUCE, APIS_DEMAND, APIS_REMOVE_ PRODUCE and APIS_REMOVE_DEMAND primitives.

When an ASU issues the APIS_CLOSE primitive, all the messages transferred to and from it will be removed from the network.

A unique APIS Message ID identifies a message on the LAN. APIS interprets the Message ID as a number consisting of four 16-bit fields. The System Contractor can allocate meaning to any field in the Message ID without effecting the operation of the APIS.

The only APIS requirement is that the Message ID uniquely identifies a message on the LAN.

3.4.2 Message Primitives

The following APIS API primitives are defined in the APIS ICD:

- **APIS_INIT**

  This API primitive assists APIS with database administration. It allows for the removal of all ASU information linked to the ASU host that issued the primitive as well as freeing associated unused memory buffers. This primitive should only be issued once per ASU host after start-up.

- **APIS_OPEN**

  This API primitive provides for the registration of the application with APIS as a potential ASU. It allows for the bi-directional identification of ASUs with APIS, through the exchanging of Application IDs and Application Service Access Point (ASAP) descriptors. This primitive has no bearing on the network portion of the middleware; it only authenticates the ASU and creates a database entry.

- **APIS_CLOSE**

  This API primitive provides for the closure of the ASAP, removing the ASU from the APIS database.

- **APIS_PRODUCE**

  This API primitive registers an APIS Message with APIS for transmission. APIS captures the parameters supplied by the ASU when this primitive is issued in the local database. This information includes the size and repetition rate of the message, which APIS will use to reserve bandwidth, as well as the APIS

Message ID which is used to derive a multicast group address for the transmission of this particular message stream. Once the multicast group address has been determined, APIS issues a call to the transport layer protocol to open a socket and create a new multicast group using this address. It also announces, by means of a network broadcast, the creation of this new group to all NICs on the network. If a demand for this APIS Message ID had been registered with any NIC on the network prior to the creation of this multicast group, that NIC will issue a call to its transport layer to *join* this group after receiving the broadcast announcement.

- APIS_DEMAND

  This API primitive registers an APIS Message with APIS for reception. As with the APIS_PRODUCE message, the parameters supplied by the ASU when this primitive is issued is also captured in the local database. APIS will use the supplied Message ID to derive the address of the multicast group associated with this Message ID. Once the multicast group address has been determined, it issues a call to the transport layer protocol to open a socket and *join* this group. Messages from all ASUs that registered to produce this particular message will now be received on this connection, and can be delivered to the ASU using the APIS_RECEIVE_MSG primitive.

- APIS_REMOVE_PRODUCE

  This API primitive provides for the removal of an APIS Message ID that has previously been registered using the APIS_PRODUCE primitive, from APIS.

- APIS_REMOVE_DEMAND

  This API primitive provides for the removal of APIS Message ID that has previously been registered using the APIS_DEMAND primitive, from APIS.

- APIS_SEND_MSG

  This API primitive allows for the transmission of an APIS Message from the *producing* ASU to other ASUs on the network that are registered to demand that particular message.

  Upon this primitive being issued, APIS will query the local database with the ASAP (provided as a parameter for authentication) of the sending ASU and the Message ID of the message being sent. It will then verify that this particular ASU has requested bandwidth for transmission of this APIS Message with the APIS_PRODUCE primitive, and that the size of the data is not greater than that specified at registration. This verification process is necessary to prevent over utilisation of the network. It will then issue a call to the transport layer protocol to transmit this message via the socket, created during the processing of the APIS_PRODUCE primitive, to the multicast group associated with his particular APIS Message ID.

- APIS_RECEIVE_MSG

  This API primitive allows for the received APIS Message to be delivered to the *consuming* ASU. This is not a request made by the ASU, but an event generated by the APIS.

  Once APIS receives data from one of the multicast groups that were joined as a result of processing previous APIS_DEMAND

primitives, it queries the local database and delivers the APIS Message to the appropriate ASUs that *demanded* this information.

Chapter 4

4 APIS ARCHITECTURE AND DESIGN

In Chapter 2 we discussed the characteristics of real-time information and in Chapter 3 we addressed the requirements set out for the APIS Publish-Subscribe MOM. In this chapter we use that information to derive some design objectives. We also discuss the choice of architecture to meet these objectives.

4.1 Design Objectives

Drawing from the requirements listed for APIS in paragraph 3.2, three cardinal objectives for the design and implementation of APIS are derived:

4.1.1 Deterministic Data (Signal) Distribution

The first architectural consideration for all components chosen to design and implement APIS is the requirement to provide deterministic data distribution. In the context of APIS, deterministic data distribution refers to application-to-application delivery of APIS Messages, up to a maximum of 4 000 bytes in size, within 5 ms.

All components in the data path should be responsive enough to guarantee timely delivery of the APIS Message. Critical component in the data path include:

- **Physical Medium (Network Hardware)**

  The line speed of the physical network hardware ultimately bounds the maximum performance that any application can achieve. For example, traditional 10Mbps Ethernet can transmit 1 bit every 0.1μs, in the case of our 4 000 byte message, the

transmit time alone accumulates to nearly 4ms. This latency is insufficient in meeting the deadline of 5ms if one still has to consider the overhead of the protocol stack and copying of the message to and from the network hardware. Clearly a technology employing at least 100 Mbps line speed, such as Fast Ethernet or FDDI is required.

- **Protocol Stack (Network Software)**

  A 'light weight' protocol is required to encode and transport the data. Such a protocol should add the minimum of overhead to the data, both in size, i.e. small protocol headers, and processing complexity, i.e. the use of non-complex integrity checks. Moreover, the implementation of the protocol should be streamlined to process the data with the minimum of processing cycles.

- **APIS Middleware**

  The APIS Middleware associates the APIS Message IDs, which the ASUs use to describe the data, with the particular multicast associations used to deliver the data across the network. In addition to this, it also authenticates users and data every time a data message is sent.

- **User API**

  In the case of off-host architecture, data messages have to be passed between the ASU and APIS across a backplane bus. The design of the user interface or API becomes important in minimising the latency of getting the data in and out of APIS.

Apart from the determinism required, the distribution of data in a many-to-many architecture also implies that multicast services be provided to the middleware by the Transport Layer of the chosen network stack.

4.1.2 Off-Host Communication Architecture

Network protocols that are implemented by means of software all require some degree of protocol processing. In *off-host* architecture the network interface is provided with its own processing capability, relieving the sub-system host of this task and enhancing its performance significantly. In the same way the *off-host* architecture also guarantees that the non-protocol processing of the sub-system host does not interfere with the deadline critical scheduling required for real-time message flow.

The separation of the distributed application and the network middleware onto different processors, does not only create a well defined contractual boundary between implementers, but also enables a heterogeneous distributed system to access a totally homogeneous LAN. A *heterogeneous* architecture is preferred by the distributed system, allowing sub-systems implemented under various operating systems on a large variety of hardware, to interface via the LAN. A *homogeneous* LAN architecture is preferred to ensure that all equipment interfacing directly onto the LAN media do not jeopardise critical message latencies.

Moreover, in a latency-critical network application, the priority design of user-tasks and network-tasks executing on the same processor, will be extremely complex, especially if different sub-contractors implement the two software elements.

Figure 2 shows several APIS Service User (ASU) applications communicating via Network Interface Cards (NICs) using Off-Host architecture.

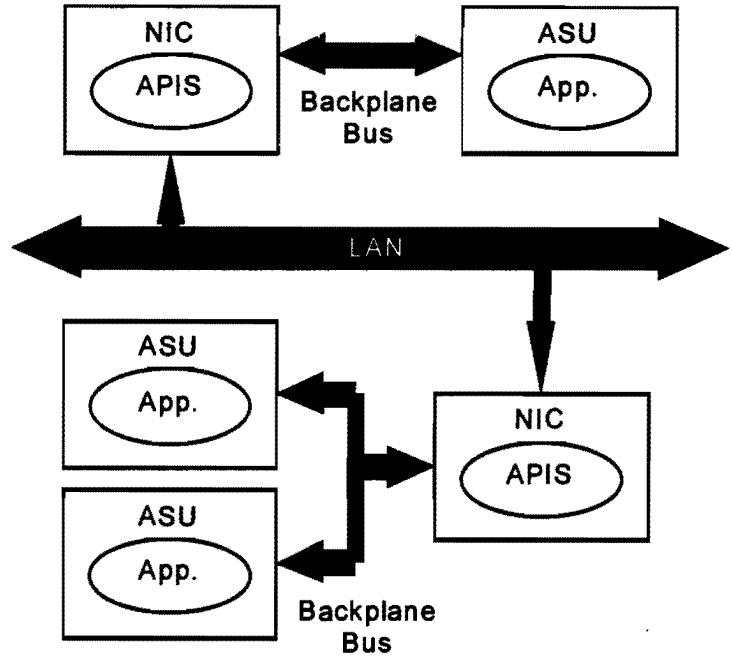


Figure 2. Off-Host Architecture

4.1.3 Use of COTS Technology

In order for the system to be affordable, Commercial Off-The-Shelf (COTS) components should be used. This also implies that an *open systems* architecture is adopted to ensure that these components conform to an acceptable standard, i.e. they are non-proprietary.

4.2 Architectural Overview

The primary functions of the APIS middleware can be listed as follows:

a. To distribute data from local producers to remote consumers across the network.

b. To receive data from remote producers via the network and deliver it to local consumers.

c. To associate producers and consumers of data with each other and maintain multicast associations for the transport of data in a manner that is transparent to the producers and consumers.

In order to achieve the above functionality, APIS is implemented as a collection of concurrent processes. Assigning each function to an individual processes rather than one process with many functions, allows us to prioritise functions relative to each other. It also provides modularity, which allows us to design and test the middleware in smaller, more manageable software units.

A local database is maintained on each of the APIS nodes to keep track of producers, consumers and the messages they have registered. This database is referenced each time an APIS Message is transmitted and received to translate its APIS Message ID to a multicast network address. Since the database is a shared resource amongst all the processes of APIS, its access routines should be *re-entrant* and *tread-safe* to avoid corruption of data records during simultaneous access attempts.

Drawing from the previous two paragraphs it is evident that an operating system that employs a pre-emptive, real-time scheduler as well as mechanisms to facilitate real-time access to a local database, is essential in meeting the requirements of APIS (refer to Figure 3 for an architectural overview).

In the next two paragraphs we shall discuss the architecture of the concurrent processes and the local database in more detail.

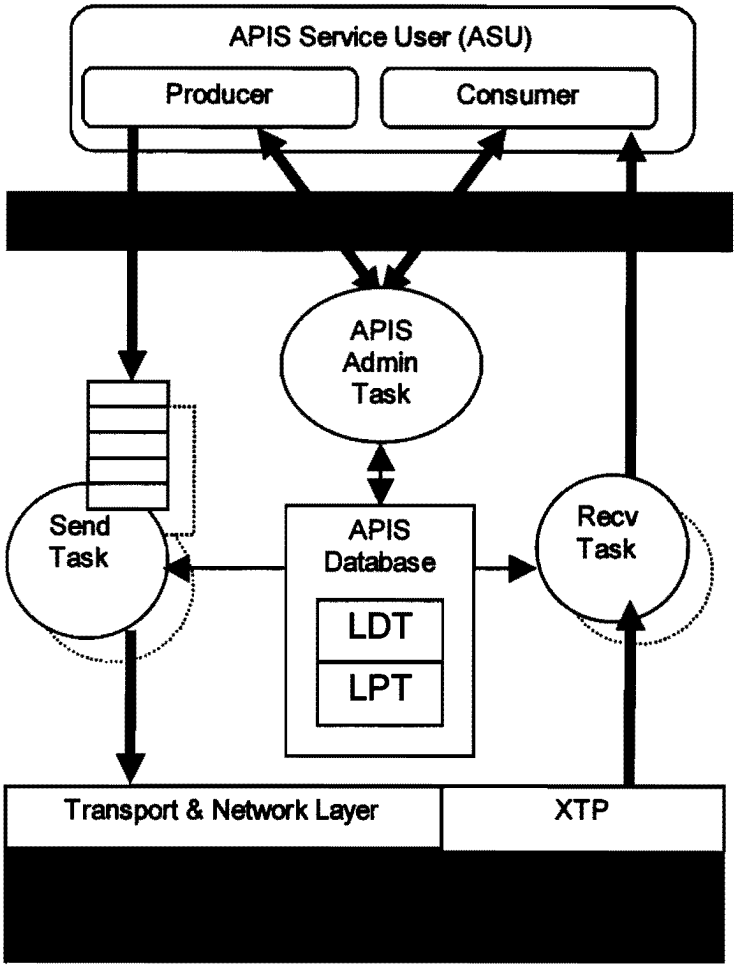


Figure 3. APIS Architecture

4.2.1 Real-Time Database Access

Two tables comprise the local database for all messages registered on the NIC. The first is named the *Local Produced Table* (LPT), containing information about producers registered on this NIC and the messages they produce. The second is the *Local Demand Table* (LDT), containing the information of the consumers registered on this NIC and the messages they demand.

The database tables are implemented by means of linked lists from the Standard Template Library (STL) provided for C++.

The APIS Database is shared amongst all the tasks that comprise APIS; it is therefore necessary to protect it with a record locking mechanism such as a semaphore. Once a semaphore is used to share a resource between tasks of different priorities, there is a chance that *priority inversion* might occur. Priority inversion is the condition where a task of lower priority can, under a certain set of circumstances, indefinitely deny a task of higher priority access to a shared resource. Some real-time operating systems provide a special set of semaphores (VxWorks calls them "mutual exclusion semaphores") with priority inheritance protocol to prevent such an occurrence.

4.2.2 Real-Time Scheduled Tasks

As previously indicated, APIS is architectured as a group of concurrent processes, or tasks in the terminology of VxWorks.

Apart from separating the sending, receiving and administrating functionalities into separate tasks, APIS also employ tasks of different priorities to allow the user to assign a relative priority to messages when sending. Eight *Send Tasks*, each with an associated message queue to buffer incoming data while the others are sending, are used to realise this prioritisation of messages. Since the scheduler schedules the tasks based on their priorities, this strategy effectively removes the need to implement an additional message scheduler; the RTOS interrupts the processing of a lower priority message as soon as a higher priority message arrives.

In addition to the eight *Send Tasks*, there are also several *Receive Tasks* and an *Administrative Task*.

Once again it is emphasised that this architecture relies heavily on the performance of the operating system and that a pre-emptive, hard

real-time task scheduler is paramount to meet the requirements specified for APIS.

- **Send Tasks**

  Tasks that receive a data message from the producer via the user interface, validates the producer as registered and passes it to the physical network media via the Transport Protocol. The List of Produced Messages will have an entry for this message containing the valid producers as well as the Transport Layer context information. Several Send Tasks may operate simultaneously. In the present design there are eight Send Tasks, one for each message priority that APIS supports.

- **Receive Tasks**

  Tasks that receive a data message via the Transport Protocol and dispatch it to all the consumers that are registered on this NIC to receive this message, via the user interface. The List of Demanded Messages on this NIC will have an entry for this message, with subsequent entries for all the consumers to which APIS should deliver this message. Several of these Receive Tasks may operate simultaneously.

- **Admin Task**

  A task that receives control messages from the producers and consumers via the user interface. Control messages are used to register/de-register producers for sending a particular message or consumers to receive a particular message. Only one Admin Tasks exists on each NIC.

Chapter 5

5 APIS INFRASTRUCTURE

In the previous chapters the architecture of the APIS middleware was outlined as a collection of software modules or tasks that collaborate to distribute data between producers and consumers. To meet this goal, the software modules must be implemented on a platform that supports the requirements of APIS both in terms of the operating system and the processor hardware. The infrastructure required by APIS includes a pre-emptive scheduler in the operating system to allow tasks to respond in real-time to service requests from the users, as well as a very responsive communication protocol stack along with associated network hardware and device drivers.

In this chapter we discuss the software components selected to support the APIS middleware. We also address certain details regarding implementation and integration of these components.

5.1 Selecting COTS Components

As outlined in paragraph 4.1.3, one of the design objectives was to use commercial off-the-shelf components wherever possible. This prompted us to consider existing protocols and technologies rather than design our own proprietary ones.

The most important software *building blocks* to consider for the implementation of APIS are the operating system and the communication stack. Our definition for the communication stack includes both the higher layer protocols that interface to APIS, i.e. the Transport Layer, as well as the lower layers such as the media access control (MAC) layer that is usually implemented on the network interface hardware itself.

The characteristics and performance of the communication stack and the operating system will determine to what extent APIS will be able to meet the required deadlines in message delivery.

The selection of each of these components and their effect on the performance of APIS are discussed below:

5.1.1 Media Access Control

According to Feng [13], the timeliness of message delivery by a network middleware such as APIS, depends on the network topology, media access control (MAC) protocol, number of nodes in the network, and the setting of various network parameters. All of these issues have been addressed with the focus on guaranteeing message deadlines by a set of ANSI and ISO standards for fibre optic networks, the Fibre Distributed Data Interface (FDDI). The FDDI standards are defined for the Physical Layer (Layer 1) and the MAC sub-layer, the lower portion of the Data Link Layer (Layer 2), of the ISO/OSI reference model for network protocols, Figure 4.

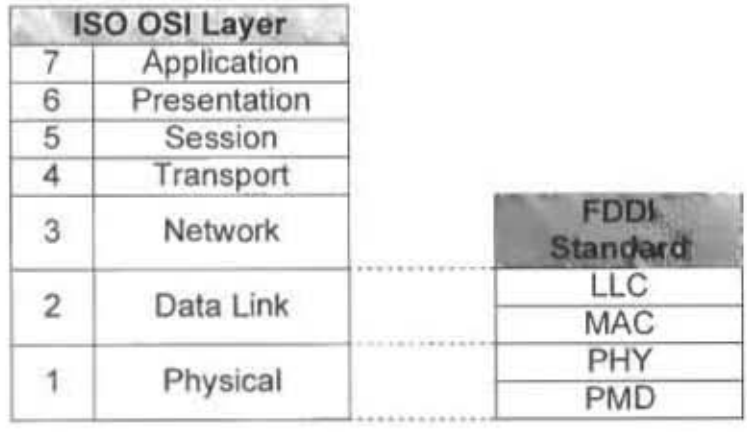


Figure 4. The Seven Layers of the ISO/OSI reference model and their mapping to the layers defined by the FDDI standard

The ten most important properties of FDDI, as detailed by Jain [18], and their relevance to APIS are summarised below:

- **High Bandwidth**

  FDDI provides a bandwidth of 100 Mbps. Even though APIS is designed primarily as a low latency LAN and not a high throughput LAN, the high available bandwidth has an impact on the access time and responsiveness of the LAN and hence also on the message delivery latency.

- **Long Interstation Distance**

  Multimode[4] fibres connecting two nodes are limited to a length of 2 km. With single-mode[5] fibres, distances as long as 60 km can be reached depending on the quality of the fibre.

  Multimode fibres were selected for our implementation since nodes in a process control plant or computer consoles on a ship are usually in close proximity. There is also a substantial cost saving involved when using multimode fibre.

- **Large Extent**

  While most other network's cable lengths are limited to less than 10 km, FDDI allows up to 200 km of fibre or, equivalently, 100km of two-fibre cable.

---

[4] Multimode fibre is so termed due to the multiple modes of light propagation within the optical media. The default fibre size is an internal diameter of 62,5 μm and external diameter of 125 μm. An LED source with a wavelength of 1300 nm is used as the transmitter.

[5] Single-mode fibre is so termed due to the single mode of light propagation within the optical media. The default fibre size is an internal diameter of 8,7 μm and external diameter of 125 μm. A laser source with a wavelength of 1300 nm is used as the transmitter.

Although this was not an important consideration for the current implementation of APIS onboard a sea-going vessel, it would be valuable when implementing a distributed system on land that might include several off-site locations.

- **Large Number of Nodes**

  FDDI allows up to 500 stations on a single network. Unlike some other networks where efficiency decreases significantly as the number of nodes that share the medium is increased, the timed token ring access method of FDDI does not show a significant decrease in efficiency. Since it is not a 'carrier sense multiple access' technology, every additional node in the ring only effects the time it takes for the token to rotate through the ring by a small 'token processing' time, in the order of 1μs per node.

- **Real-Time Traffic**

  FDDI guarantees a bound on the waiting time to obtain a usable token. This service class, called *synchronous service*, is suitable for real-time applications, in which it is important that certain messages get through even during times of heavy load.

  Since the maximum time to obtain a usable token is bound and since this time is a function of the FDDI ring parameters (refer to paragraph 5.5.1), the FDDI ring parameters can be chosen to make the network as responsive as needed. APIS uses this feature to guarantee delivery of a message within its deadline.

- **Reserved Bandwidth**

  In addition to bounded access time, the synchronous service of FDDI also has a reserved bandwidth capability. This feature

allows APIS to reserve a timeslot in every token rotation in order to deliver messages in real-time even when high network traffic is present.

- **High Availability**

  Availability refers to the percentage of time the network is usable. FDDI uses a number of automatic fault recovery mechanisms, which allow the network to come back up quickly after a failure.

  An important feature of FDDI is its distributed nature. All algorithms are distributed in the sense that the control of the ring is not centralised. When any component fails, other components can reorganise and continue to function. This includes fault recovery, clock synchronisation, token initialisation, and topology control.

  These fault tolerant features make FDDI a good candidate for deploying APIS in a mission critical situation.

- **High Reliability**

  Reliability here refers to the error rates in the network while it is operating. In general, the error rates of fibre are lower than those of copper cables of equivalent length. In addition, the coding scheme[6] and the cyclic redundancy mechanisms that FDDI use result in the detection of most errors. The undetected error rate is in the order of 10^{-20} or less.

[6] A 4b/5b code is used along with *nonreturn to zero inverted* (NRZI) encoding. This combination provides for error protection, control symbols to relay line state information, and easy recovery of the clock signal.

- **High Security**

  Fibre provides a secure medium because it is not possible to read or change optical signals without physical disruption.

- **Noise Immunity**

  Fibre is not susceptible nor does it emit electromagnetic interference, providing a noise-free medium for communication.

  This feature is particularly important for the naval implementation of APIS where cable runs will pass through noisy engine and generator rooms etc. onboard the ship.

The feature set listed above indicates that FDDI is a good candidate for implementing APIS in a real-time mission critical environment.

FDDI was selected for the APIS implementation not only for its high bandwidth and its property of a bounded access time, but also for its dual-redundant ring architecture. The bounded access time provides a necessary condition to guarantee real-time deadlines. The dual-redundant ring architecture allows continuous real-time service even under some failure conditions.

Over the years FDDI has been selected for various other embedded distributed real-time applications. For example, as the backbone network for NASA's Space Station Freedom. FDDI has also been adopted by the U.S. Navy's Next Generation Computer Resource Program, as part of its Survivable Adaptable Fibre Optic Embedded Network (SAFENET) [47].

The network interface card selected is a PMC FDDI adapter from C^2I^2 Systems, refer to Appendix B for a short-form specification. It is designed around the Supernet 3 FDDI chipset technology from

AMD. The MAC and physical layer protocols are implemented in hardware on this chipset. A PCI bus interface is used to connect the MAC layer to the higher-level communication stack protocols by means of DMA memory.

5.1.2 Transport Protocol

The task of the Transport Protocol is to provide reliable host-to-host communication for use by the higher-level protocols [46]. In general the Transport Protocol provides features such as: flow control, error control, delivery priority, unicasting/multicasting and parametric addressing.

The APIS software modules use the Transport Protocol to distribute the data between nodes on the FDDI ring. In the previous chapters it was established that APIS would need a transport protocol that could not only provide multicasting capabilities, but is also very responsive in handling the data with a minimal protocol overhead.

The Xpress Transport Protocol (XTP) [44][50] is a high performance ISO OSI Transport Layer protocol. Developed under co-ordination of the XTP Forum, it draws extensively from a number of earlier experimental transport protocols developed to meet real-time requirements [50]. It has received considerable support from the US Navy and is specified as the real-time transport protocol for the Survivable Adaptable Fibre Optic Embedded Network (SAFENET) standards suite.

XTP was chosen mainly because of its multicast capability and user-defined flexibility but also because it was "*explicitly designed for high throughput, low latency implementations.*" [44]

Mentat Inc. [31] implemented the version of XTP used for APIS. This version is based on the UNIX STREAMS network interface.

5.1.3 Real-Time Operating Systems (RTOS)

Network protocols implemented in software can only perform as well as the underlying operating system allows. In such an environment where network protocols are implemented as a collection of interrupt service routines and independent processor tasks, scheduling policy of the tasks is the key factor that determines performance. Adherence to industry standard operating system interfaces is also vital when combining COTS software components from different sources.

The VxWorks Real-Time Operating Systems [49] from Wind River Systems was chosen mainly because it meets the following requirement:

- **High-Performance Real-Time Multitasking Kernel**

  The highly efficient micro kernel architecture is scalable from deeply embedded applications to complex distributed systems.

- **Pre-emptive Task Scheduling**

  VxWorks provides round robin as well as pre-emptive task scheduling. Building APIS on top of a pre-emptive task scheduler allows implementation of a middleware that is responsive to message priorities assigned by the user. Pre-emptive scheduling allows execution of higher priority task to interrupt and defer the execution of lower priority tasks. APIS uses this feature to give preference to higher priority APIS Messages by assigning them to higher priority tasks than the low priority APIS Messages.

By implementing a flat memory model and executing tasks as threads in a globally shared memory environment, VxWorks can guarantee very low and deterministic interrupt latencies and context switching latencies between tasks.

- **Real-Time Extensions**

  VxWorks adheres to the POSIX 1003.1b Real-Time Extensions standard. Thread-safe version of message queues, signals, memory management and semaphores with priority inheritance are provided for controlling critical system resources.

  These features are vital to the implementation of the local database on each APIS node that requires real-time access from multiple tasks.

- **STREAMS Networking Support**

  XTP from Mentat Inc., which was selected as the transport protocol for APIS, uses the STREAMS framework for interconnecting the various layers of network protocol software. Thus, an important feature of VxWorks was its support for STREAMS as an optional module.

- **Advanced Development Environment**

  The VxWorks development environment entitled Tornado provided a set of debugging, diagnostic and analysis tools specifically designed for real-time embedded applications [48].

5.2 Integrating COTS Components

Although this thesis focuses on the design and implementation of the APIS MOM, it is worthwhile recording the effort that was required to establish the infrastructure needed by APIS:

1. An FDDI network driver was not available at the time for the VxWorks RTOS. The source code for a STREAMS version of the driver for the SCO operating system had to be purchased and ported to VxWorks.

2. An FDDI network card was not available in the PMC format at the time. The hardware design for a PCI version of the card had to be purchased and re-engineered to the PMC standard. This work included the layout and manufacture of a multi-layer PC board to the IEEE PMC specification.

3. Mentat XTP was not available for VxWorks at the time. The XTP source code for the Sun Solaris operating system had to be purchased and ported to VxWorks.

5.3 Target Platform

The chosen target platform is an Intel Pentium-based SBC with a Multibus II Parallel Backplane Bus (PBB) and a PMC slot to host an FDDI card.

The Multibus II PBB is used to interface the ASU's application, running on a similar SBC in another slot in the card cage, to APIS. All communication between APIS and the ASU is via the Multibus II PBB.

An advantage of using MBII technology as the ASU interface is that it uses message-passing architecture instead of shared-memory architecture as implemented by VME or CompactPCI. It is thus much simpler to implement a message-passing middleware onto the already existing MBII Transport Protocol.

Message passing effectively passes responsibility of timely delivery of the message to APIS as soon as the ASU *posts* the message onto the PBB.

Figure 5 illustrates the selection of the components chosen for the implementation. From the figure it is clear that the ASU, on one SBC, will communicate with APIS, on another SBC, via the Multibus II PBB and the Multibus II Transport Protocol. APIS in turn will communicate with other APIS nodes on the FDDI LAN via the XTP and FDDI protocols.

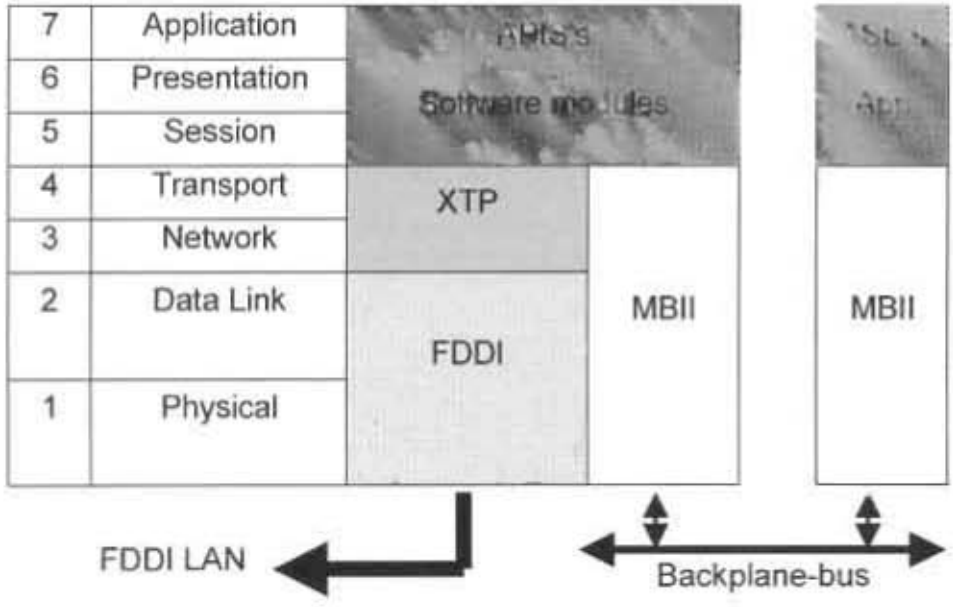


Figure 5. APIS Implementation using XTP and FDDI

5.4 Multicast Group Management

To maintain connectivity and timely delivery to multiple consumers, APIS employs multicasting technology. APIS assigns every Message ID that is produced or consumed by an ASU, to a unique multicast group address. This design paradigm enables APIS to simply *join* the multicast group associated with a particular Message ID when a consumer registers the demand and *leave* the group when the consumer de-registers.

Assigning a unique multicast group address to each Message ID also limits the amount of load on the network stack by filtering unwanted addresses at an early stage i.e. at the Data Link Layer (DLL) or more specifically the Media Access Control (MAC) sub-layer. This would not be the case if a broadcast type address were used; filtering would then only take place at the Transport Layer.

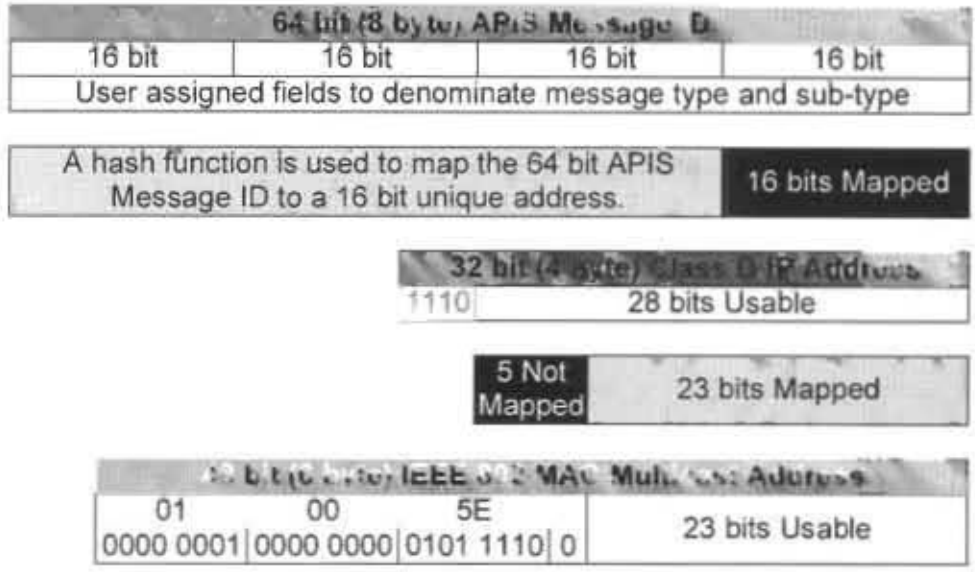


Figure 6. APIS Message IDs, Class D IP and IEEE MAC Addresses

The Class D IP address space, used for multicast transmissions, occupies the range from 224.0.0.0 to 239.255.255.255. Class D addresses are bound to MAC addresses on a LAN differently than Class A, B, or C unicast addresses. A unicast IP address is explicitly bound to a MAC address; in contrast a Class D address is automatically mapped to a MAC multicast address by a simple procedure. A MAC layer multicast address is 48 bits (6 bytes) long, of which the 25 high-order bits contain a fixed identifier (01-00-5E-00), leaving only 23 significant bits. A Class D IP address, on the other hand, is 32 bits (4 bytes) long of which 28 bits are significant. The mapping procedure simply maps the low-order 23 bits of the Class D address onto the 23 low-order bits

of the MAC address, leaving 5 bits unmapped. Thus 2^5 or 32 Class D IP addresses exist for every MAC address [33].

APIS relies on each Message ID being mapped to a unique MAC address in order to reduce traffic load on the network stack; for this reason only the low-order 16 bits of a Class D IP address are used to map 2^{16} unique APIS Message IDs. A hashing function is used to map the user-assigned 64 bit APIS Message ID to a 16 bit value that is used to construct the address.

Figure 6 illustrates the relationship between a MAC multicast address, a Class D IP address and an APIS Message ID.

5.5 Guaranteeing Message Deadlines

Fibre Distributed Data Interface (FDDI) is a set of standards developed by the American National Standards Institute's (ANSI) Accredited Standards Committee (ASC) Task Group X3T9.5 [18].

In order to guarantee that the deadlines of synchronous messages are met when using FDDI, network parameters such as the synchronous bandwidth, the target token rotation time, and the buffer size must be chosen carefully [28]. Each of these will be discussed below.

FDDI uses a *timed token* access method to share the medium amongst stations. This access method is different from the traditional token ring access method in that the time taken to travel around the ring, the Target Rotation Time (TRT), is accurately measured by each station and is used to determine the media access opportunity upon token arrival. In other words, each time a node receives the token it calculates how much data it can transmit before it has to release the token to prevent it from becoming late.

FDDI allows two types of traffic: synchronous and asynchronous. Synchronous traffic consists of delay-sensitive traffic, which needs to be transmitted within a certain time interval. The asynchronous traffic consists of data packets that can sustain some reasonable delay and is generally throughput sensitive in the sense that higher throughput (bytes per second) is more important than the time taken by the bits to travel over the network.

5.5.1 Performance Parameters

All stations on the network must negotiate and agree on the value of the Target Token Rotation Time (TTRT), it is the key network parameter that affects performance [19]. Each station measures the TRT, the time since it last received the token. On a token arrival it computes a Target Holding Time (THT), the difference between the TTRT and TRT. A station can transmit synchronous traffic whenever it receives a token, asynchronous traffic can be transmitted only if THT is positive.

The guidelines for setting TTRT are set out in [19] as follows:

1. The TRT can be as long as 2 × TTRT. Thus synchronous stations should set TTRT to half of the required repetition interval.

2. TTRT should allow for at least one maximum size frame along with the synchronous time allocation, if any. That is:

$$TTTR \geq D_{max} + \text{Token Time} + F_{max} + \Sigma SA$$

Where: D_{max}, the Ring Latency[7], equals 0,06017 ms

F_{max}, the Maximum Frame Time[8] is 0,360 ms

[7] For 2 km of optical fibre and 50 stations, D_{max} = (2 km) × (5,085 μs) + (50 stations) × (1 μs/stations) = 60,17 μs

Token Time, for 11 bytes, is 0,00088 ms

∑SA is total Synchronous Allocations

Thus: $TTRT \geq \sum SA + 0{,}42$ ms

To achieve a latency of less than 5 ms for our current APIS application, the TTRT was chosen as 2 ms. This value allows ∑SA to be at least 1,5 ms or 18 750 bytes during each TRT.

At the chosen TTRT the efficiency and maximum access delay can be calculated from the formulas derived in [19] as follows:

$$\begin{aligned}\text{Efficiency} &= \frac{n(TTRT - D_{max})}{nTTRT + D_{max}} \\ &= \frac{50(2 - 0{,}06017)}{50 \times 2 + 0{,}06017} \\ &= 96{,}93\,\%\end{aligned}$$

$$\begin{aligned}\text{Max access delay} &= (n-1)TTRT + 2D_{max} \\ &= (50-1) \times 2 + 2 \times 0{,}06017 \\ &= 98 \text{ ms}\end{aligned}$$

The calculated maximum access delay is the time that an asynchronous message may have to wait for LAN access under heavy network load conditions. The access time of synchronous messages is, of course, twice the TTRT, or 4 ms in our case.

5.5.2 Synchronous Bandwidth Allocation

Preferential traffic in FDDI is known as synchronous traffic. A node may transmit synchronous data whenever the token is received; asynchronous data may only be sent once synchronous data

---

[4] Maximum Frame Time = (4 500 bytes) / (125 MHz) = 0,360 ms

transmission is complete on condition that the THT has not expired as explained above.

Since there is only a limited amount of synchronous bandwidth available (calculated above for our case as 1,5 ms or 18 750 bytes per TRT), nodes have to request bandwidth from a designated node known as the Synchronous Bandwidth Allocator (SBA). The SBA functionality is implemented as part of the FDDI driver software.

A problem exists with the SBA in that it does not really reserve a portion of the available bandwidth as the name suggests, but rather guarantees a time slot in each TRT during which the node will be able to access the LAN. The value of TRT varies with the load on the LAN making it difficult to calculate, in advance, the portion of TRT needed to reserve a certain percentage of the total bandwidth. Various allocation schemes have been suggested to overcome this problem.

In order to guarantee the real-time delivery of messages, APIS allocates time for all high priority messages to be transmitted during each TRT. Although this allocation scheme is wasteful, it ensures that bandwidth is available for the worst case when all high priority messages are transmitted simultaneously.

5.5.3 Buffer Allocation

Each APIS node has buffers for incoming and outgoing synchronous messages. Message loss can occur if a buffer that is to small overflows. On the other hand, buffers that are too large are wasteful of memory.

There is an added complexity on the receiving side in that the processor must be able to keep pace with the incoming messages to avoid message loss.

Malcolm and Zhao [28] prove that the maximum queue length is independent from message deadlines. Intuitively one would expect that large message deadlines might lead to buffers overflowing. However it is proved that this is not true when synchronous bandwidth is allocated. They show that the size of the buffer need be no more than the size of the synchronous message multiplied by the upper bound on the number of messages waiting to be transmitted.

Since the largest message size allowed by APIS is restricted to 4 000 bytes, and due to the large amount of memory available on the chosen hardware platform, it was decided to allow for buffers of ten times the size of the largest message, i.e. 40 000 bytes.

6 LIMITATIONS OF MENTAT'S XTP

During the implementation of APIS, it became apparent that the XTP implementation obtained from Mentat Inc. was not adequate in allowing APIS to meet its requirements. It was decided to remove XTP from the network protocol stack until a more advanced implementation became available.

Without a transport layer, APIS now manages the associations between producers and consumers and only uses the 'raw' multicast services provided by Logical Link Control (LLC) 'sub' Layer of the OSI Data Link Layer.

The limitations that were observed and their impact on the performance of APIS are discussed below:

6.1 Lack of Unacknowledged Multicast Service

To implement the latency-critical distribution of signals, it was intended to use the *Unacknowledged Multicast Stream Service*, as defined in the XTP specification [50]. Unfortunately Mentat Inc.'s implementation of XTP at the time only provided the *Reliable Multicast Stream Service*.

The *Unacknowledged Multicast Stream Service* is analogous to the UDP/IP service where no connection is set up between the sending and receiving party. The *Reliable Multicast Stream Service*, on the other hand, is analogous to the connection oriented TCP/IP service. It ensures that a message is delivered, in the case of a lost message a retransmission would occur, potentially jeopardising the latency of all subsequent messages in that stream. This is not always desirable, especially in the case of distributing real-time signals where

missing one data value in favour of receiving the next one on time is preferred. Refer to the properties of real-time signals in paragraph 2.1.1.

Although we were implementing APIS on a very low BER ($2{,}5 \times 10^{-10}$)[18] fibre LAN, the chance did exist that a lost message could be retransmitted, thereby missing its deadline.

6.2 File Descriptor Limit

A more serious trait of the XTP implementation pertaining to file descriptors was also observed. Since the Mentat Inc. implementation of XTP was geared towards reliable multicasting, a separate UNIX network STREAM was allocated for every multicast group association, requiring a unique file descriptor from the operating system for each. APIS allows up to 2^{16} different Message IDs to be registered; VxWorks, on the other hand, as a real-time embeddable kernel, only allows a maximum of 250 open file descriptors at a time to conserve resources and response time.

The implementation of the *Unacknowledged Multicast Stream Service*, had it been available, would have required one UNIX network STREAM and file descriptor to send messages to different multicast groups since there are no connections between producers and consumers at the transport layer. A seconds UNIX network STREAM and file descriptor would be required to receive messages from different multicast groups when they are 'joined' (XTP terminology) by doing a 'bind' (UNIX sockets terminology) for every address to the same receiving socket. In this way, two file descriptors could have been used to implement the basic APIS services.

The large number of file descriptors required by the XTP implementation and its lack of support for the unacknowledged multicast service, lead us to remove XTP from the network stack as an interim measure. APIS was then implemented directly on the Logical Link Control (LLC) Layer of the OSI

Data Link Layer, which provides basic services to transmit and receive from multicast groups.

It was also argued that the 4B/5B-encoding scheme and the *Frame Check Sequence* (FCS) used by FDDI eliminated the need for additional error checking by an advanced transport layer protocol such as XTP.

Figure 7 illustrates the implementation of APIS without the XTP layer.

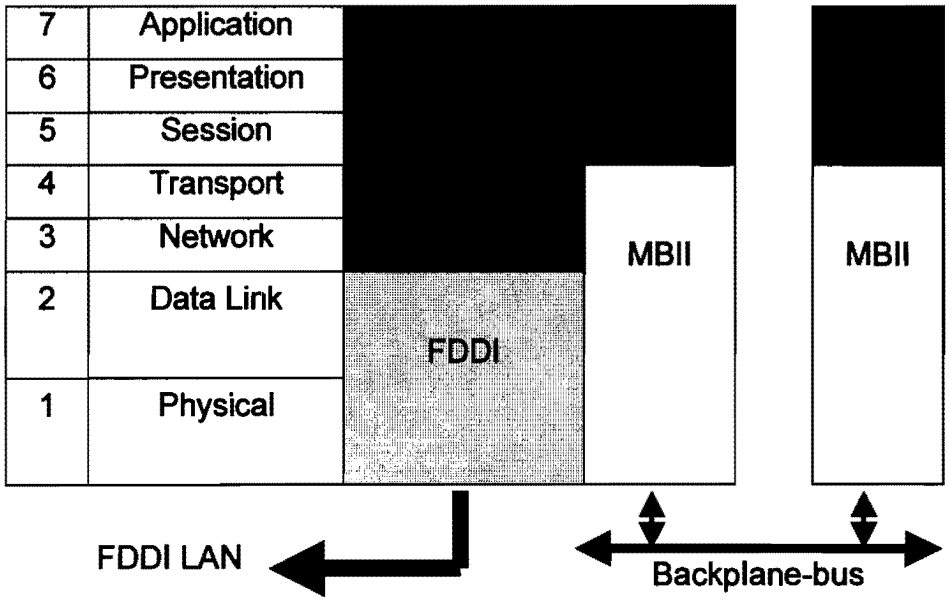


Figure 7. APIS Implementation without using XTP.

Chapter 7

7 PERFORMANCE MEASUREMENTS

The APIS middleware was implemented on a 200 MHz Pentium-based single board computer (SBC), referred to as the NIC, using message passing on a Multibus II backplane as the interface to the user. The test application was implemented on a second, similar, SBC referred to as the ASU. In total four SBCs were used to obtain the measurements for the preliminary results reported in this thesis. One pair of SBCs constituted a Producer ASU and a NIC, while a second pair constituted a Consumer ASU and a NIC.

After these initial tests were completed the APIS project entered a qualification and test phase, which is outside the scope of this thesis. During that phase, the APIS hardware and software successfully complied with the requirements set by our client, the SA Navy, as the real-time data bus proposed for their Combat Suite program.

Two techniques were used to obtain latency measurements reported here:

- The relative time between events occurring on the same SBC were measured using a timer provided by the Pentium processor architecture. The timer increments at the rate of the processor's clock (200 MHz) and can be read with a few simple assembler language instructions with little impact on the performance of the existing source code. Using this timer, timestamps were taken at various points in the software 'data path' to obtain a latency profile of an APIS Message.

- Events that occurred on separate SBCs, i.e. the start of a message transmission on the Producer ASU and the reception of it on the Consumer ASU, were measured by asserting control pins on the serial

ports of the SBCs at various points in the source code. VxWorks allows direct access to the registers of the serial port devices, enabling us to effect change of the RTS and DTR pins with simple read and write instructions. The signals were monitored on a digital oscilloscope and logic analyser to calculate offsets.

7.1 End-to-End Latency Measurements

The latencies depicted in Figure 8 represent the end-to-end message transfer latencies between two test applications for messages up to 4 000 bytes long. Each measurement includes the latency of a Multibus II transfer, validation and processing in APIS and the processing latency of the network stack on each of the two NICs, as well as the FDDI transfer between them.

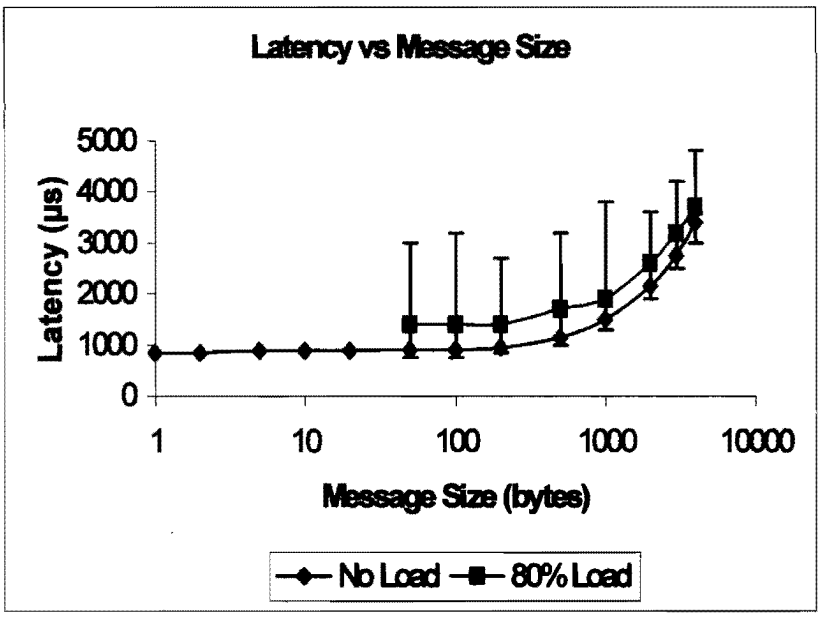


Figure 8. APIS Latency vs. Message Size (Log Scale)

Measurements were taken both on an unloaded FDDI ring as well as during the presence of an 80% load on the network.

When testing the system under load conditions, we need to consider both *network load* and *stack load*. Network Load refers to the absolute traffic load measured on the FDDI ring itself. Because of the nature of the FDDI timed token protocol, nodes using the *synchronous service* have guaranteed access to the LAN, minimising the affect that the overall network load has on the data being sent from such a node. Stack Load refers to the traffic load measured on each individual node's protocol stack and is limited by the throughput of the network card as well as the amount of processor cycles available to send and receive the data.

The 80% network load consisted of a 12 Mbit/s (12%) data transfer between the two ASU's in order to load the network stack of each NIC, as well as an additional 68 Mbit/s (68%) load created by an separate traffic generating tool.

For the unloaded case, latencies were also measures for 1, 2, 5, 10 and 20 bytes of data per message. The graph shows that the latencies measured for all messages of 100 bytes or smaller are nearly equal. This constant (minimum) latency for small messages can be contributed to the packet header of 33 bytes added to all APIS messages on the network. For larger messages, the overhead of the packet header becomes insignificant and the relationship between message size and transfer latency becomes more-or-less linear as can be seen from Figure 9.

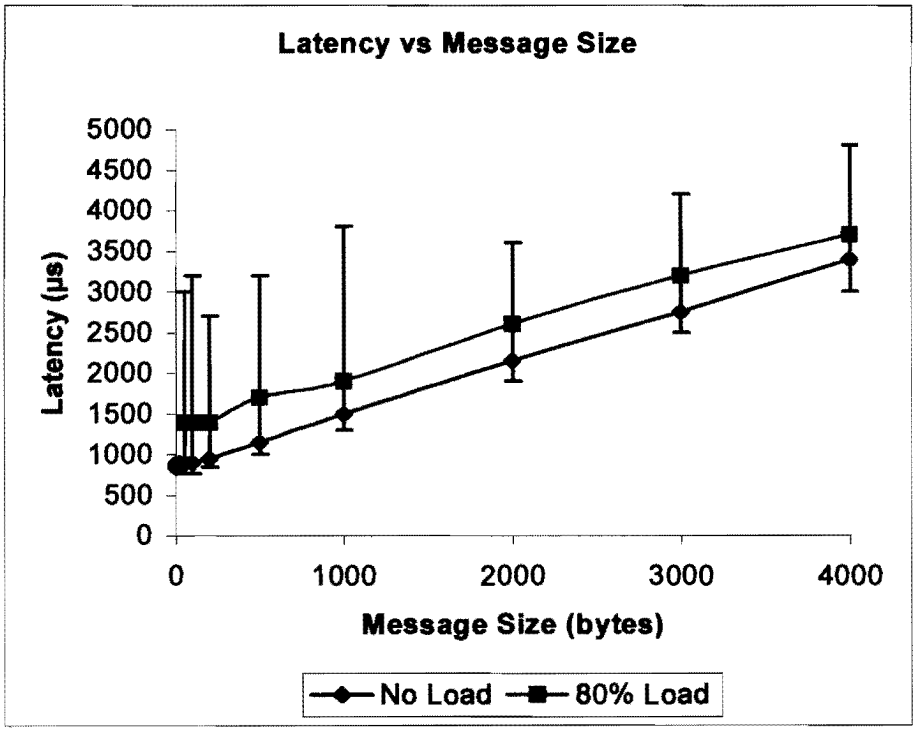


Figure 9. APIS Latency vs. Message Size (Linear Scale)

Under heavy load it is observed that although the average transfer latency remains directly proportional to the message size, the maximum latency (or jitter) measured for each message size is not a function of the message size but rather tends towards a constant maximum value. For example, the 1000 byte message has an average latency of 1,7ms but a maximum jitter value of 3,8ms. It shows that when the network is under load, the time to gain access to the network becomes more significant than the time it takes to output the bytes onto the media. This is the expected behaviour for FDDI and bounds on the access time were calculated in paragraph 5.5.1. For our implementation, the TTRT was chosen as 2 ms to limit the maximum FDDI access time to 4 ms. This allows for 1 ms of processing time to meet a delivery deadline to the application of no more than 5 ms. The worst case application to application latency recorded over the test period was 4,8 ms, which was less than the 5 ms expected.

7.2 Latency Profile of an APIS Message

The source code was modified to toggle serial port pins when sending and receiving APIS Messages and monitored with a logic analyser to capture the latency profile of an APIS message transfer. Figure 10 is an image that was captured from the logic analyser for a 4 000 byte message transfer. Appendix E contains additional screen shots that were captured during these tests.

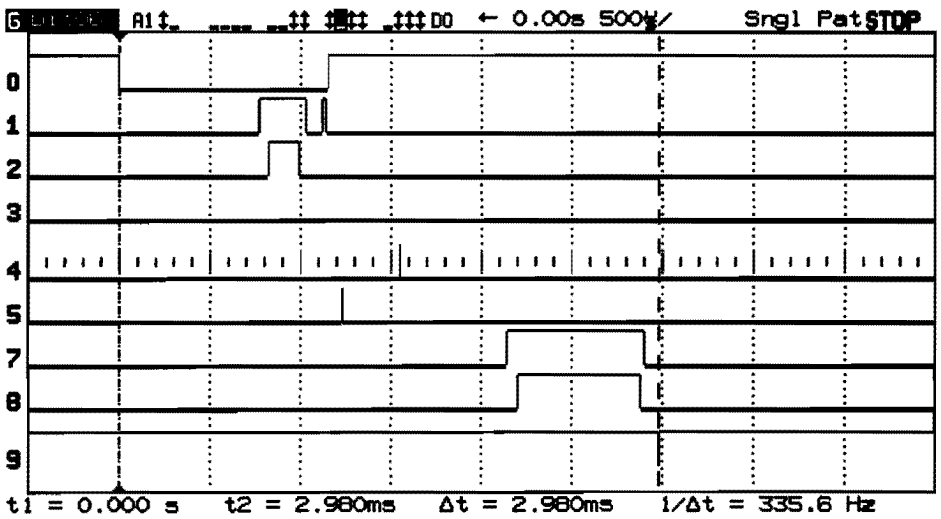


Figure 10. Logic Analyser Screencapture of a latency profile

Figure 11 depicts the timeline of a message delivery in APIS. The figure also indicates which events occurred on the Producer ASU, the Producer NIC, the Consumer NIC and the Consumer ASU.

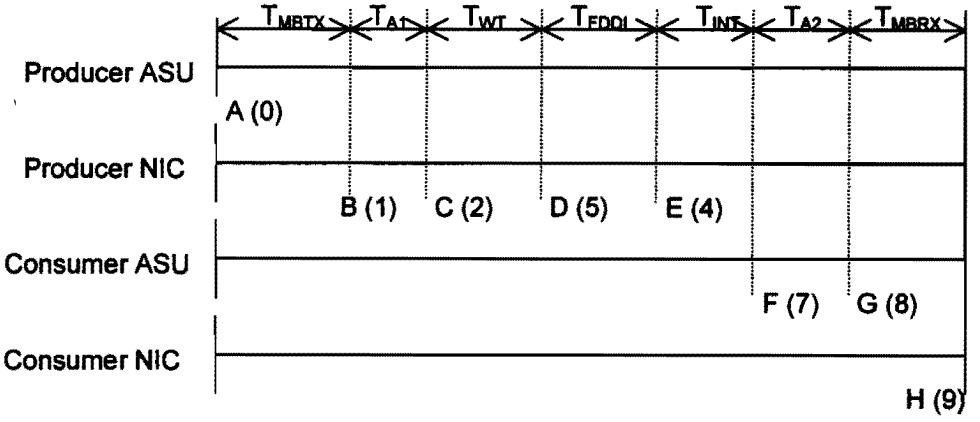


Figure 11. Latency Timeline

The events on the timeline correspond to the first transitions on each of the traces (trace numbers follows event numbers in parentheses) of Figure 10. The events are described below:

A: (Trace 0) The Producer ASU starts to transmit the message via Multibus.

B: (Trace 1) APIS finishes receiving the message via Multibus and starts processing the message.

C: (Trace 2) APIS starts copying the message to the FDDI driver.

D: (Trace 5) Traces 3, 4 and 5 are bits of a hardware register on the SuperNet 3 FDDI chipset that indicate the status of the FDDI protocol processor. When the bit that is monitored on trace 5 is set, it indicates that a valid token has been received.

E: (Trace 4) When the register bit monitored by trace 4 is set, it indicates that the token is being released, implying that all synchronous data has been sent onto the LAN.

F: (Trace 7) The FDDI driver indicates to APIS that a message has been received.

G: (Trace 8) APIS starts the Multibus transfer of the message to the Consumer ASU.

H: (Trace 9) The Consumer ASU received the message, signifying the end of the Multibus transfer as well as the end of the APIS message transfer.

The relative times between events are also indicated on the timeline and are labelled as follows:

T_{MBTX}: The time required to transmit the message via the Multibus from the Producer ASU to the Producer NIC.

T_{A1}: The time required by APIS to query the database in order to validate the Producer ASU as well as retrieve routing information for this message.

T_{WT}: The time that APIS had to wait for a valid token to become available in order to send the message onto the LAN.

T_{FDDI}: The time required to transmit the message onto the physical media.

T_{INT}: The time since the transmission ends and the FDDI driver indicate to APIS that a message was received.

T_{A2}: The time required by APIS to query the database to locate the local Consumer ASUs for this message.

T_{MBRX}: The time required to transmit the message via the Multibus from the Consumer NIC to the Consumer ASU.

Figure 12 shows a breakdown of the latencies that comprise a single APIS message transfer under no load conditions for various message lengths.

The graph shows the latencies of Multibus II, APIS, FDDI media and the FDDI driver. These latencies were computed from the timeline as follows:

$$\text{Total Time} = T_{MBTX} + T_{A1} + T_{WT} + T_{FDDI} + T_{INT} + T_{A2} + T_{MBRX}$$

$$\text{When : MBII} = T_{MBTX} + T_{MBRX}$$

$$\text{APIS} = T_{A1} + T_{A2}$$

$$\text{FDDI} = T_{FDDI}$$

$$\text{Driver} = T_{WT} + T_{INT}$$

We have: Total Time = MBII + APIS + FDDI + Driver

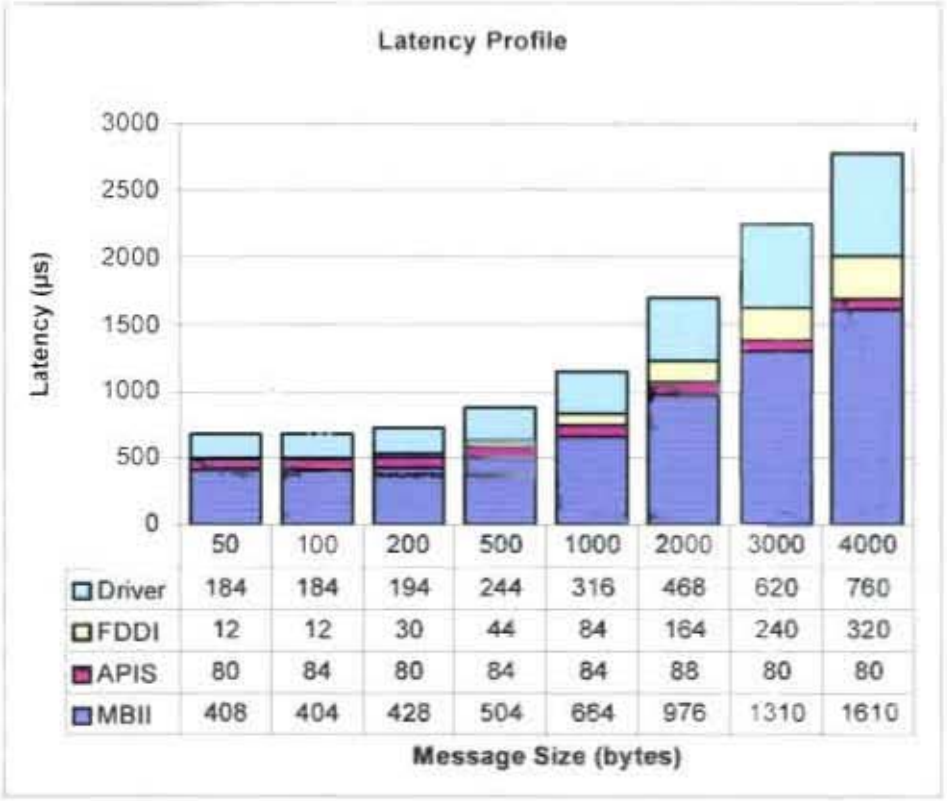


Figure 12. Profile of a typical APIS Message transfer

Figure 13 presents the same data as a percentage as the total transfer latency. It can be seen that Multibus II transfers make up a substantial part of this latency (always about 60% of the total latency). Technology such as VME or Compact PCI could reduce this amount and thus also the overall latency of the transfer. The contribution of the driver to the latency is also proportional to the size of the message, about 25%, while the latency of APIS processing is constant. This is expected, as the processing in APIS is not dependent on the size of the message, it only involves Message and Producer verification to the database.

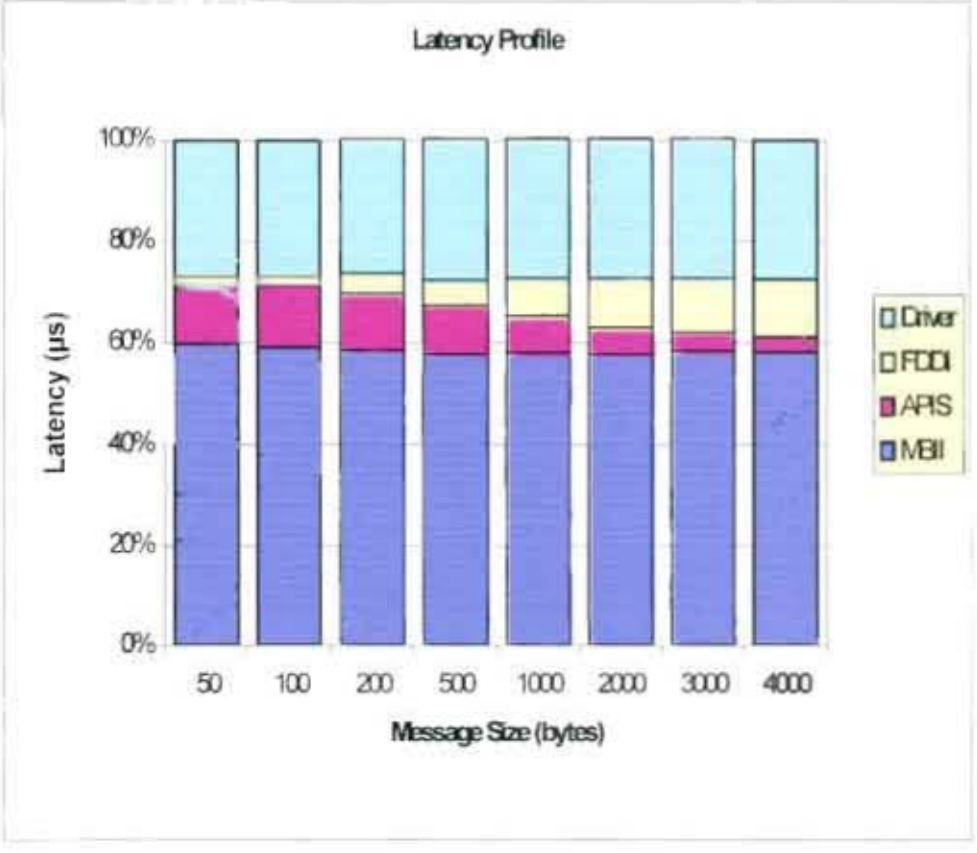


Figure 13. Profile of a typical APIS Message transfer, percentage breakdown

Chapter 8

8 CONCLUSION

In this thesis we investigated the suitability of sharing information in a distributed system by making use of APIS, the real-time message-oriented middleware proposed by Young in his PhD thesis, and the feasibility of implementing APIS by means of COTS software components only.

To a large extent the information in a real-time distributed control system can be categorised as being of the repetitive *signal* variety. These signals are latency-critical and therefore not necessarily always reliable; it is better to lose one message and wait for its next update than to receive stale data. It is concluded that the properties of Publish-Subscribe MOM, as utilised in the APIS architecture, are well suited to distributing these signals within a real-time distributed system.

The results have shown that such a system can be implemented by means of the *Unacknowledged Multicast Stream Service* provided by XTP, or the *raw* multicast service provided by LLC.

The results of latency tests suggest that the synchronous data transfer provided by the timed token protocol of FDDI is well suited to this type of service. An upper bound can be placed on the message transfer latency by selecting an appropriate TTRT.

The off-host architecture of the APIS design was found favourable in providing a homogeneous LAN in a heterogeneous distributed system as well as protecting latency critical network processes from user implemented host applications. Unfortunately this off-host architecture is also the biggest contributor (60% for Multibus II transfers) to message delivery latency. It is

therefore suggested that alternative backplane technologies such as VME or CompactPCI are investigated to replace the Multibus II backplane bus.

9 REFERENCES

[1] Gopal Agrawal, Biao Chen, Wei Zhao, Sadegh Davari, "Guaranteeing Synchronous Message Deadlines with the Timed Token Medium Access Control Protocol", *IEEE Transactions on Computers*, Vol. 43, No. 3, March 1994.

[2] Dock Allen, Peter Krupp, *White Paper on Realtime CORBA*, OMG Realtime Platform SIG, Issue 1.0, December 1996.

[3] J. William Atwood, Octavian Catrina, John Fenton, W. Timothey Strayer, "Reliable Multicasting in the Xpress Transport Protocol", *Proceedings of the 21st Local Computer Networks Conference*, Minneapolis, October 13-16, 1996.

[4] Çağlan M. Aras, James F. Kurose, Douglas S. Reeves, Henning Schulzrinne, "Real-Time Communication in Packet-Switched Networks", *IEEE Proceedings*, Vol. 82, No. 1, pp. 122-139, Jan. 1994.

[5] Biao Chen, Gopal Agrawal, Wei Zhao, "Optimal Synchronous Capacity Allocation for Hard Real-Time Communications with the Timed Token Protocol", *Real-Time Systems Symposium*, Dec. 1992.

[6] Biao Chen, Sanjay Kamat, Wei Zhao, "Fault-Tolerant Real-Time Communication in FDDI-Based Networks", *Proceedings of Real-Time Systems Symposium*, Dec. 1995.

[7] Biao Chen, Hong Li, Wei Zhao, "Meeting Delay Requirements in Computer Networks with Wormhole Routing", *Proceedings of the 16th International Conference on Distributed Computing Systems*, 1996.

[8] Biao Chen, Wei Zhao, "Properties of the Timed Token Protocol", Technical Report 92-038, Texas A&M University, 1992.

[9] Imrich Chlamtac, András Faragó, Hongbiao Zhang, Andrea Fumagalli, "A Deterministic Approach to the End-to-End Analysis of Packet Flows in Connection-Oriented Networks", *IEEE/ACM Transactions on Networking*, Vol. 6, No. 4, August 1998.

[10] Ardas Cilingiroglu, Sung Lee, Ashok Agrawala, "Real-time Communication", Technical Report CS-TR-3740, University of Maryland, January 1997.

[11] David D. Clark, Wenjia Fang, "Explicit Allocation of Best-Effort Packet Delivery Service", *IEEE/ACM Transactions on Networking*, Vol. 6, No. 4, August 1998.

[12] Doug Dykeman, Werner Bux, "Analysis and Tuning of the FDDI Media Access Control Protocol", *IEEE Journal on Selected Areas in Communication*, Vol. 6, No. 6, July 1988.

[13] Fang Feng, Sanjay Kamat, Wei Zhao, "Guaranteeing Application-to-Application Deadlines in Distributed Real-Time Systems", *Proceedings of the 20th Conference on Local Computer Networks*, Minneapolis, 1995.

[14] Fang Feng, Amit Kumar, Wei Zhao, "Investigating the Synchronous Bandwidth Allocation in an FDDI Network for Real-Time Communications", Technical Report 95-022, Texas A&M University, 1995.

[15] Fang Feng, Wei Zhao, Amit Kumar, "Bounding Application-to-Application Delays for Multimedia Traffic in FDDI-Based Communication Systems", *Proceedings of Multimedia Computing and Networking*, San Jose, Jan. 1996.

[16] Moncef Hamdaoui, Parameswaran Ramanathan, "Selection of Timed Token Protocol Parameters to Guarantee Message Deadlines", IEEE/ACM Transactions on Networking, Vol. 3, No. 3, June 1995.

[17] International Middleware Association, "Frequently Asked Questions", http://www.imwa.org/utilities/faq.html.

[18] Raj Jain, *FDDI Handbook: High-Speed Networking Using Fiber and Other Media*, Addison-Wesley Publishing Company, 1994.

[19] Raj Jain, "Performance Analysis of FDDI Token Ring Networks: Effect of Parameters and Guidelines for Setting TTRT", *IEEE LTS*, pp. 16-22, May 1991.

[20] Raj Jain, "FDDI: Current Issues and Future Plans", Technical Report 9809086, Computing Research Repository: Networking and Internet Archive, 1993.

[21] Sanjay Kamat, Gopal Agrawal, Wei Zhao, "On Available Bandwidth in FDDI-Based Reconfigurable Networks", *IEEE INFOCOM '94*, 1994

[22] Sanjay Kamat, Nicholas Malcolm, Wei Zhao, "Performance Evaluation of a Bandwidth Allocation Scheme for Guaranteeing

Synchronous Messages with Arbitrary Deadlines in an FDDI Network", Technical Report 94-021, Texas A&M University, 1994.

[23] Sanjay Kamat, Wei Zhao, "Real-Time Schedulability of Two Token Ring Protocols", *Proceedings of the 13th IEEE International Conference on Distributed Computing Systems*, June 1993.

[24] Sanjay Kamat, Wei Zhao, "Real-Time Performance of Two Token Ring Protocols", *Advances in Real-Time Systems*, chapter 6, pages 117-147, Prentice-Hall, 1995.

[25] Darrell D. E. Long, Carol Osterbrock, Luis-Felipe Cabrera, "Providing Performance Guarantees in an FDDI Network", Technical Report UCSC-CRL-93-23, University of California, 1992.

[26] Nicholas Malcolm, Sanjay Kamat, Wei Zhao, "Real-Time Communication in FDDI Networks", *Journal of Real-Time Systems*, Vol. 10, No. 1, Jan. 1996.

[27] Nicholas Malcolm, Wei Zhao, "Hard Real-Time Communication in Multiple-Access Networks", *Journal of Real-Time Systems*, Vol. 9, No. 1, pp. 75-107, Jan. 1995.

[28] Nicholas Malcolm, Wei Zhao, "Guaranteeing Synchronous Messages with Arbitrary Deadline Constraints in an FDDI Network", *Proceedings of the 13th Conference on Local Computer Networks*, Minneapolis, 1993.

[29] Nicholas Malcolm, Wei Zhao, "Use of the Timed Token Protocol for Real-Time Communications", *IEEE Computer*, Vol. 27, No. 3, March 1994.

[30] Roland Mechler, Gerald W. Neufeld, "XTP Application Programming Interface", Technical Report TR-95-17, University of British Columbia, July, 1995.

[31] Mentat Inc., *Mentat XTP Programmer's Manual*, Ver. 4.0-01, 2nd Edition Corrected, Oct. 1998.

[32] Jeffrey R. Michel, Alexander S. Waterman, Alfred C. Weaver, "Performance Evaluation of an Off-Host Communication Architecture", *Proceedings of Local Computer Networks Conference*, Minneapolis, September 1993.

[33] C. Kennith Miller, *Multicast Networking and Applications*, Addison-Wesley Publishing Company, 1998.

[34] Sonu Mirchandani, Raman Khanna, *FDDI, Technology and Applications*, John Wiley & Sons Inc., 1993.

[35] Saurab Nog, David Kotz, "A Performance Comparison of TCP/IP and MPI on FDDI, Fast Ethernet, and Ethernet", Technical Report PCS-TR95-273, Dartmouth College, Hanover, 1996.

[36] Gerardo Pardo-Castellote, Stan Schneider, Mark Hamilton, "NDDS: The Real-Time Publish-Subscribe Network", Real-Time Innovations Inc. White Paper http://www.rti.com, 1997.

[37] José F. de Rezende, Andreas Mauthe, David Hutchinson, Serge Fdida, "M-Connection Service: A Multicast Service for Distributed Multimedia Applications", *Proceedings of Multimedia Transport and Teleservices, International COST237 Workshop*, Copenhagen, Denmark, November 1995.

[38] Steven H. Rodrigues, Thomas E. Anderson, David E. Culler, "High-Performance Local Area Communication With Fast Sockets", USENIX '97, 1997.

[39] A. Udaya Shankar, "Reasoning Assertionally about Real-Time Systems", *Proceedings of the IEEE*, Vol. 82, No. 1, January 1994.

[40] Kang G. Shin, Parameswaran Ramanathan, "Real-Time Computing: A New Discipline of Computer Science and Engineering", *Proceedings of the IEEE*, Vol. 82, No. 1, January 1994.

[41] Kang G. Shin, Xianzhong Cui, "Computing Time Delay and Its Effects on Real-Time Control Systems", *IEEE Transactions on Control Systems Technology*, Vol. 3, No. 2, pp. 218-224, June 1995.

[42] Robert Simon, "Reliable and Predictable Communication for Distributed Real-Time Systems", Technical Report TR98-07, George Mason University, 1998.

[43] W. Timothy Strayer, Michael J. Lewis, Raymond E. Cline Jr., "XTP as a Transport Protocol for Distributed Parallel Processing", *Proceedings of the USENIX Symposium on High-Speed Networking*, Oakland, August 1-3, 1994.

[44] W. Timothy Strayer, Bert J. Dempsey, Alfred C. Weaver, *XTP: The Xpress Transfer Protocol*, Addison-Wesley Publishing Company, 1992.

[45] W. Timothy Strayer, Alfred C. Weaver, "Is XTP Suitable for Distributed Real-Time Systems?", *Proceedings of the International*

Workshop on Advanced Communications and Applications for High-Speed Networks (IWACA), Munich, Germany, March 16-19, 1992.

[46] Andrew S. Tanenbaum, "Network Protocols", *Computing Surveys*, Vol. 13, No. 4, pp. 453-489, Dec. 1981.

[47] U.S. Department of Defence, MIL-STD-2204, *Survivable Adaptable Fibre Optic Embedded Network*, Sept. 1992.

[48] Wind River Systems, "The Next Generation of Embedded Development Tools", WRS White Paper MCL-WHP-TO-9612, 1996.

[49] Wind River Systems, "VxWorks 5.4", WRS White Paper MCL-DS-VXW-996, 1999.

[50] XTP Forum, "Xpress Transport Protocol Specification", XTP 95-20, Revision 4.0, 1 March 1995.

[51] Ti-Yen Yen, Wayne Wolf, "Performance Estimation for Real-Time Distributed Embedded Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 11, November 1998.

[52] Richard M. Young, "Real-Time Protocol Strategies for Mission-Critical, Distributed Systems", PhD Dissertation, University of Witwatersrand, July 1996.

[53] Sijing Zhang, Alan Burns, "An Optimal Synchronous Bandswidth Allocation Scheme for Guaranteeing Synchronous Message Deadlines with the Timed-Token MAC Protocol", *IEEE/ACM Transactions on Networking*, Vol. 3, No. 6, December 1995.

[54] Wei Zhao, Amit Kumar, Gopal Agrawal, Sanjay Kamat, Nicholas Malcolm, Biao Chen, "Real-Time Communcation in FDDI-Based Reconfigurable Networks", *RTOSS '94*, 1994.

[55] Qin Zheng, Kang G. Shin, "Fault-Tolerant Real-Time Communication in Distributed Computing Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 5, pp. 470-480, May 1998.

[56] Qin Zheng, Kang G. Shin, "Synchronous Bandwidth Allocation in FDDI Networks", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 12, pp. 1332-1338, December 1995.

APPENDIX A IMS SHORT-FORM SPECIFICATIONS

Information Management System

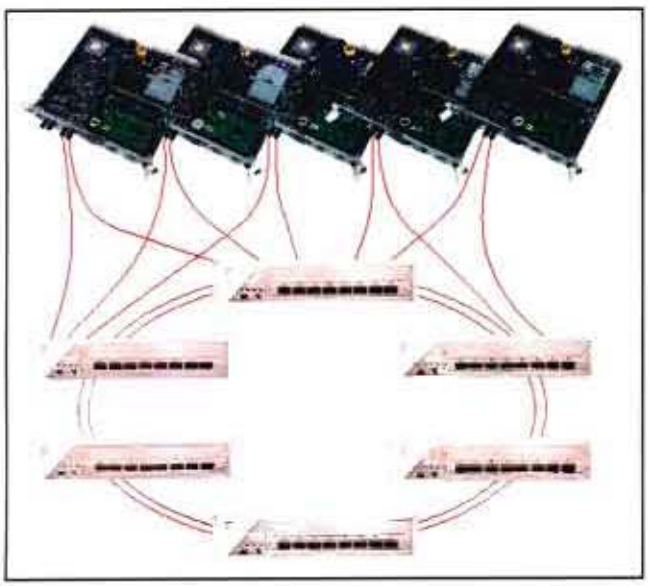
The Information Management System (IMS) is a ship–borne network, based on SAFENET, that manages the transfer of time–critical command and control messages, multimedia streams and background file transfer from many sources to many destinations. The IMS architecture supports unicast, broadcast and reliable multicast transfer types. It also provides for network synchronisation and message timestamping as well as sophisticated built-in test and network management.

The IMS offers bounded packet latencies, message-level priorities, synchronous bandwidth allocation, high overall performance, determinism and reliability.

Apart from ship-borne applications, IMS is also finding application in real–time vetronics systems as well as tactical command and control systems.

IMS Architecture

The C²I² Systems Information Management System (IMS) communications architecture follows the dissemination architecture designed for real–time communications, refer to Figure 1 : *Dissemination Architecture* below. The IMS communications architecture supports many–to–many connections which is best suited to distributed, time–critical information flow. The IMS allows nodes on the network to produce and consume data on the physical network and provides distributed control of the network. The IMS manages the actual data transfers between nodes on the network. Each node dynamically registers with the IMS and then becomes a producer and/or consumer until deregistration.



The IMS handles the multicasting of all data on the network, thereby allowing virtual links to be setup between the nodes on the network. This architecture is symmetric, robust to changes and failures and is very efficient.

The IMS communications architecture differs significantly from the older *point–to–point* (e.g. TCP) and *client–server* (e.g. RPC) architectures. These architectures suffer from complex connection and error recovery problems as well as single points of failure.

Functions

Transfer Control Data

- Application Interface Services (APIS)
- Xpress Transport Protocol (XTP)
- Internet Protocol (IP)
- IEEE 802.2 Data Link Layer (DLL)
- ISO FDDI MAC Layer

Information Management System

Transfer Bulk Data

- File Transfer Services (FTS)
- User Datagram Protocol (UDP)

Network Time Services

- Network Time Protocol (NTP)
- Network Synchronisation Services
- Packet Timestamping

Network Management Services

- Built-in Test Services (BITS)
  - LAN Adapters
  - Cable Plant
- Simple Network Management Protcol (SNMP V2.0)
- Management Information Bases
- Managed Agent
- Managing Application
- Graphical Man-Machine Interface
- Operator-Assisted Trouble-Shooting, Maintenance and Reconfiguration

Cable Plant

- 62,5 µm / 125 µm Multimode Fibre Cable Plant
- Dual-Redundant Standard
- Quad-Redundant Optional
- Optical Bypass Switches
- Trunk Coupling Units
- Fibre Amplifiers

Features

- Critical Virtual Circuit Capability
- Multi-Level Packet Priority
- FDDI Synchronous Mode
  - Synchronous Bandwidth Allocator
  - End Station Support
- Multi-Protocol Support
- SAFENET-Compatible
- Ruggedised FDDI Connectors (ST)
- Rugged miniature circular OBS connector
- Copper Media (CDDI) Optional

Performance

- < 950 µs end-to-end latency (< 200 byte messages)
- > 15 $Mbits^{-1}$ end-to-end throughput (> 4 000 byte messages)
- < 250 µs node-to-node synchronisation accuracy (2 σ)

C²I² Systems (Pty) Ltd • P.O. Box 171 • Rondebosch 7701 • South Africa • Telephone : (+27 21) 683 5490 • Facsimile : (+27 21) 683 5435 • URL : http://www.cci.co.za/

Application Interface Services

Description

Application Interface Services (APIS) is an extended profile network communications protocol designed for the exchange of information between functionally-independent applications incorporated in a distributed real-time system.

APIS conceptually encompasses Layers 5 to 7 of the ISO OSI Reference Model and so interfaces below to Layer 4, the Transport Layer and above to the Application Service Service (ASU) which will normally be a collaborative, networked, software application.

The ASU is a producer and/or consumer of data of different types. Data types are pre-defined by an ASU administration authority, as part of the network system design and each data type is ascribed a unique identification code or Message Identifier. The APIS protocol establishes the necessary communication channels between ASUs by registering and matching their producers and consumers. LAN dataflow will therefore be determined by the data type of ASU messages and not by predefined ASU addresses.

This data-driven approach to dataflow management provides a higher level of flexibility than the traditional addressed-point-to-addressed-point facilities provided by general purpose LAN protocols. The objective of this approach is to simplify ASU communication and configuration logic, thereby decoupling system design from network design.

Principles of Operation

The intrinsic operation of APIS requires a number of specific services from the lower layer protocols. These are broadcast, reliable multicast and multicast group management.

Message classification characterises each message by type, sub-type and identifier (ID). Messages are grouped by type and sub-type and uniquely identified by Message ID. Such classification is made on the basis of origin (i.e. producer) and application category (e.g. contact, target, navigation data, etc.).

Producers and consumers register with their own (local) APIS which identifies them to the system by means of an APIS broadcast. The status of all producers and consumers is then maintained in a local status table which is updated periodically as well as at each significant status event. The aggregate of this process provides a distributed, real-time, dataflow management agent.

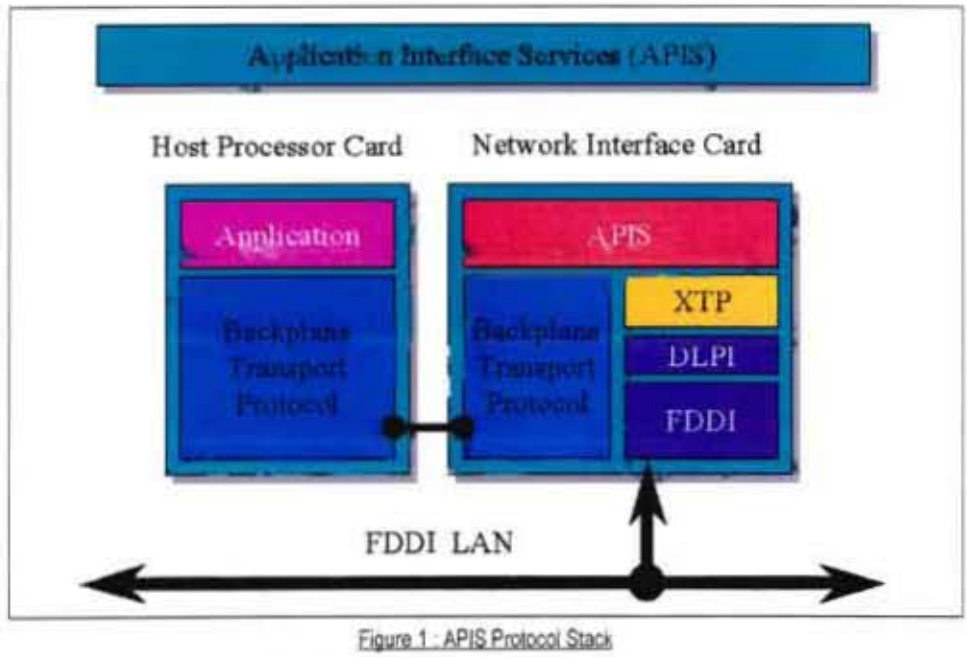


Figure 1 : APIS Protocol Stack

Application Interface Services

When a consumer requires a message it registers its requirement (i.e. demands the message) and APIS sets up all necessary internal mechanisms and control messages to ensure that the producer provides this message.

The Message Identification scheme is designed in such a way as to support wildcarding. Wildcarding is defined as group addressing by means of address subsets. Thus groups of producers and consumers can be accessed using a generic addressing scheme. APIS employs wildcarding to manage production and consumption of messages, both individually as well as by group (type) and sub-group (sub-type). Wildcard demand and limited wildcard produce are both supported as APIS service options.

APIS employes multicast and multicast group management to manage production and consumption of messages. A Producer uses reliable multicast to transmit to all Consumers requiring a particular message. Consumer groups, including the joining and leaving of groups after strartup, are managed by multicast group management facilities.

Performance

- < 60 us APIS layer transition (< 200 byte messages)
- < 950 us end-to-end latency (< 200 byte messages)
- > 15 $Mbits^{-1}$ end-to-end throughput (4 000 byte messages)

APIS/Application Interface

The interface to APIS is defined by the following service requests :

- APIS Initialise
- APIS Open
- APIS Close
- APIS Produce
- APIS Demand
- APIS Remove Produce
- APIS Remove Demand
- APIS Send Message
- APIS Receive Message

Network Time Services

Description

Network Time Services (NTS) is an extended profile protocol with the following capabilities :

- Effects system syncronisation by means of a Network Time Protocol (NTP).
- Provides timestamping of messages.

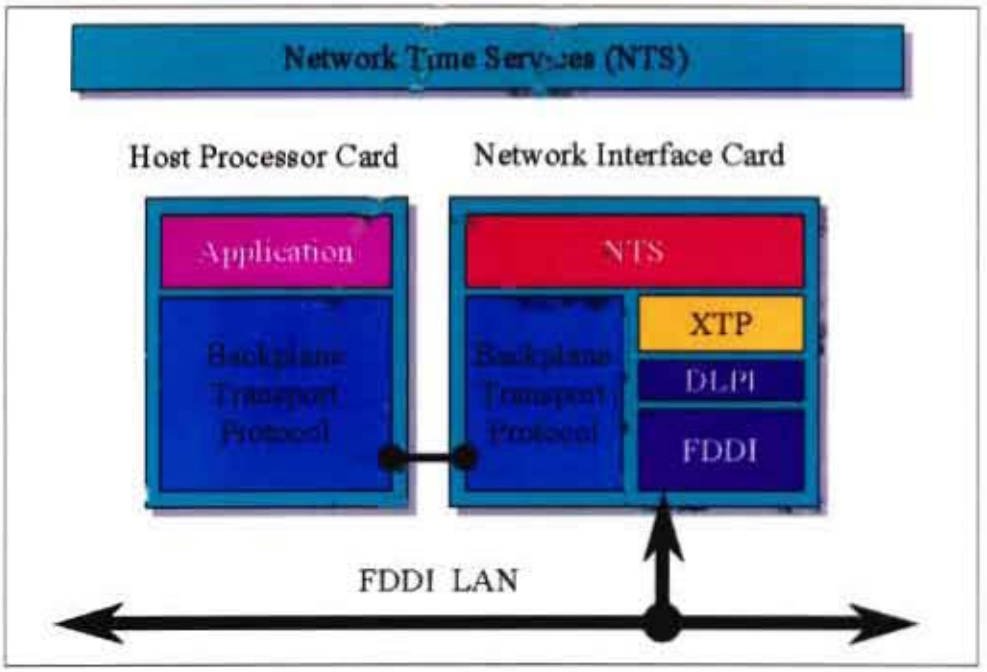


Figure 1 : NTS Protocol Stack

Application

Despite the raw performance available at the lower layers of the LAN profile, networked systems can still suffer from transfer latency and jitter. Latency and jitter are problems as they may lead to instability in distributed control algorithms. However, the data repetition rates required by the collaborative distributed algorithms can be sufficient if accurate timing of the data samples can be recovered. Real-time systems therefore require services from the network which circumvent the problems of latency and jitter. Timestamping of data by the producer, using accurate network time derived from NTP, allows a consumer to reconstruct critical timing information by determining the age of the data.

Network Time Services

Principles of Operation

NTP implements timing mechanisms between all participating sub-systems over the network. The protocol does not rely on the accuracy of the clock of a single peer, but rather attempts to find the most accurate time source available to it. Each node maintains a local clock to provide time values and to synchronise with all other clocks on the network. NTP provides basic functionality such as syncronisation and timestamping to NTS. NTS in turn provides user-level time services to the application.

Network Time Services allow a user to obtain the current clock and an indication of the accuracy and quality of the value. NTS can provide synchronisation services, both with respect to calendar time (i.e. absolute time) as well as relative time between distributed clocks. For many distributed applications, only relative time synchronisation is required. For certain applications, e.g. synchronisation of remote encryption devices, calendar time synchronisation is required.

Performance

Conditions : FDDI local area network in ring topology with 40 nodes and 2 500 m circumference.

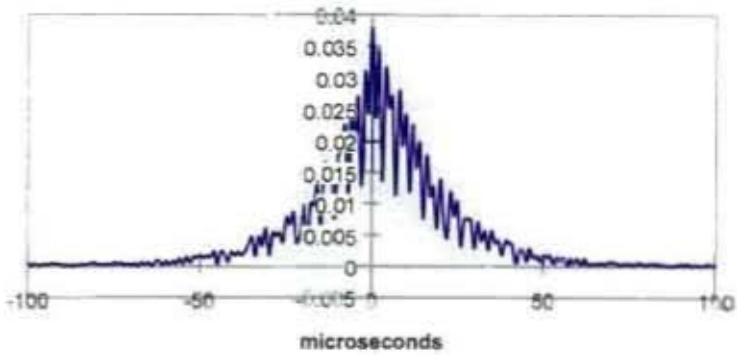


Figure 2 : Probability Density Function of the True Clock Offset

Note : True clock offset between the software clocks of two machines synchronized over an FDDI network (14 691 samples at one sample every 16 seconds)

NTS/Application Interface

The Interface to NTS is defined by the following service requests :

- NTS Get Time
- NTS Set Time

File Transfer Services

Description

File Transfer Services (FTS) is an extended profile network protocol providing reliable data transfers by emulating connection-oriented byte stream transport between applications executing on different host processors on a local area network. Two host processors that share an FTS connection may be located either on the same backplane or within sub-systems that are connected by different Network Interface Cards (NICs).

Principles of Operation

FTS uses an addressing scheme where every user thereof chooses a unique 64-bit name for the user's transport endpoint. The FTS transport addressing scheme is completely separate from the network addresses used by the NICs. This is because the NICs use dynamically-assigned network addresses, whereas communicating FTS entities use globally-administered transport endpoint names.

The message interface to FTS provides a simplified OSI-like connection-oriented transport service between these transport endpoints. The data transfer over FTS connections is based on reliable byte streams. Multiple connections can be made between two transport endpoints. Each connection is unidirectional.

FTS provides services to perform two types of data transfer, namely read requests and write requests. FTS supports a Go-Back-N flow control scheme that permits high bandwidth bulk data transfers.

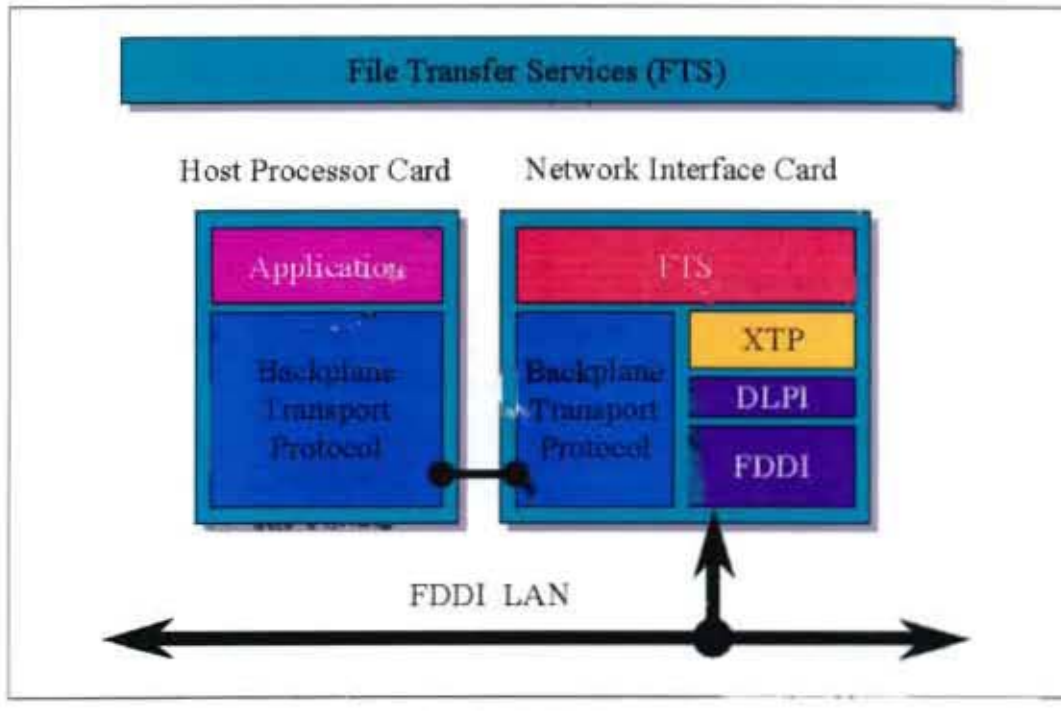


Figure 1 : FTS Protocol Stack

File Transfer Services

FTS/Application Interface

The interface to FTS is defined by the following service requests :

- FTS Register
- FTS Solicited Reply to Registration
- FTS Unregister
- FTS Solicited Reply to Unregistration
- FTS Write Request
- FTS Read Request
- FTS Write Request Data
- FTS Read Request Data
- FTS Write Request Acknowledge
- FTS Read Request Data Acknowledge
- FTS Error

Performance

- > 15 $Mbits^{-1}$ end-to-end throughput (for > 4 000 byte messages)

C⁄⁄ Systems (Pty) Ltd • P.O. Box 171 • Rondebosch 7701 • South Africa • Telephone : (+27 21) 683 5490 • Facsimile : (+27 21) 683 5435 • URL : http://www.cci.co.za/

Built-in Test Services

Description

The Built-in Test Services (BITS) in conjunction with the Network Management Station (NMS) provides real-time monitoring (integrity checking) and dynamic configuration control of the entire networked system, event and alarm reporting, statistical performance measurement, comprehensive diagnostic facilites, computer-assisted trouble shooting and maintenance.

Network Management consists of two levels; the lower level implementing network management functions and the higher level implementing system health monitoring. Lower level network management functions consist of Managing Applications and Managing Agents.

Principles of Operation

The managing application manages managed objects by means of management agents. It derives data from the managed objects by means of a LAN-based network management protocol (i.e. SNMP) and stores this data in a Management Information Base (MIB). The data in the MIBs is then processed to provide user level management information by means of the NMS man-machine interface.

The management agent performs management operations on managed objects and produces notification of events on behalf of managed objects within the node in which it resides. It acts as an intermediary between the managing application and the entities which have managed objects within the node.

The Nework Management Station provides higher level system health monitoring and provides online monitoring and control of the entire network system, reconfiguration management and high-level diagnostics facilities.

For maintenance crew manned systems the NMS provides extensive man-machine interface functionality. It supports an operator's console which provides graphics-based, diagrammatic visualisation of the system and its network components. The display provides high resolution colour graphics and interacts with pointing devices such as mouse, trackball or joystick. The NMS consists of hardware and computer software components. The hardware component provides physical connectivity to the LAN as well as providing a processing platform for the NMS software component.

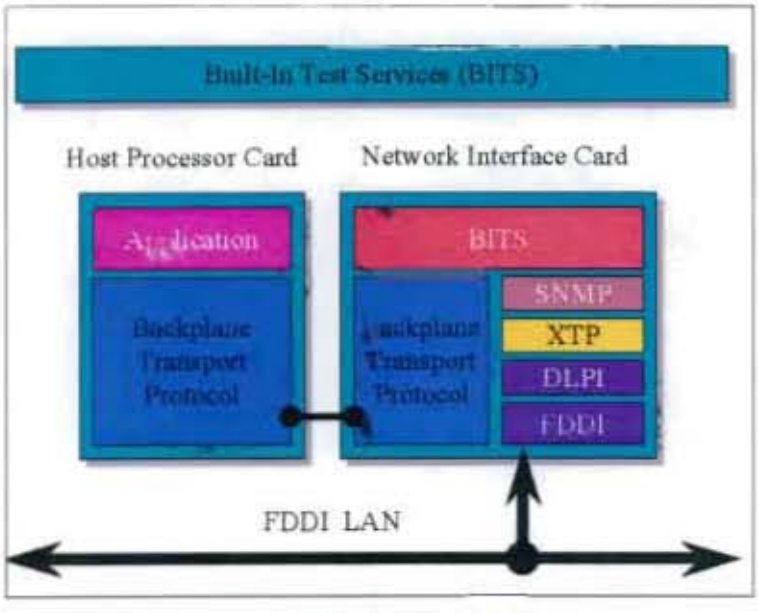


Figure 1 : BITS Protocol Stack

Built-in Test Services

BITS/Application Interface

The following is a subset of messages for the BITS user application interface :

- BITS Get Producer List
- BITS Get Consumer List
- BITS Get Produced Messages
- BITS Get Consumed Messages
- BITS Get ASAPs
- BITS Get Status Message
- BITS Get Status APID

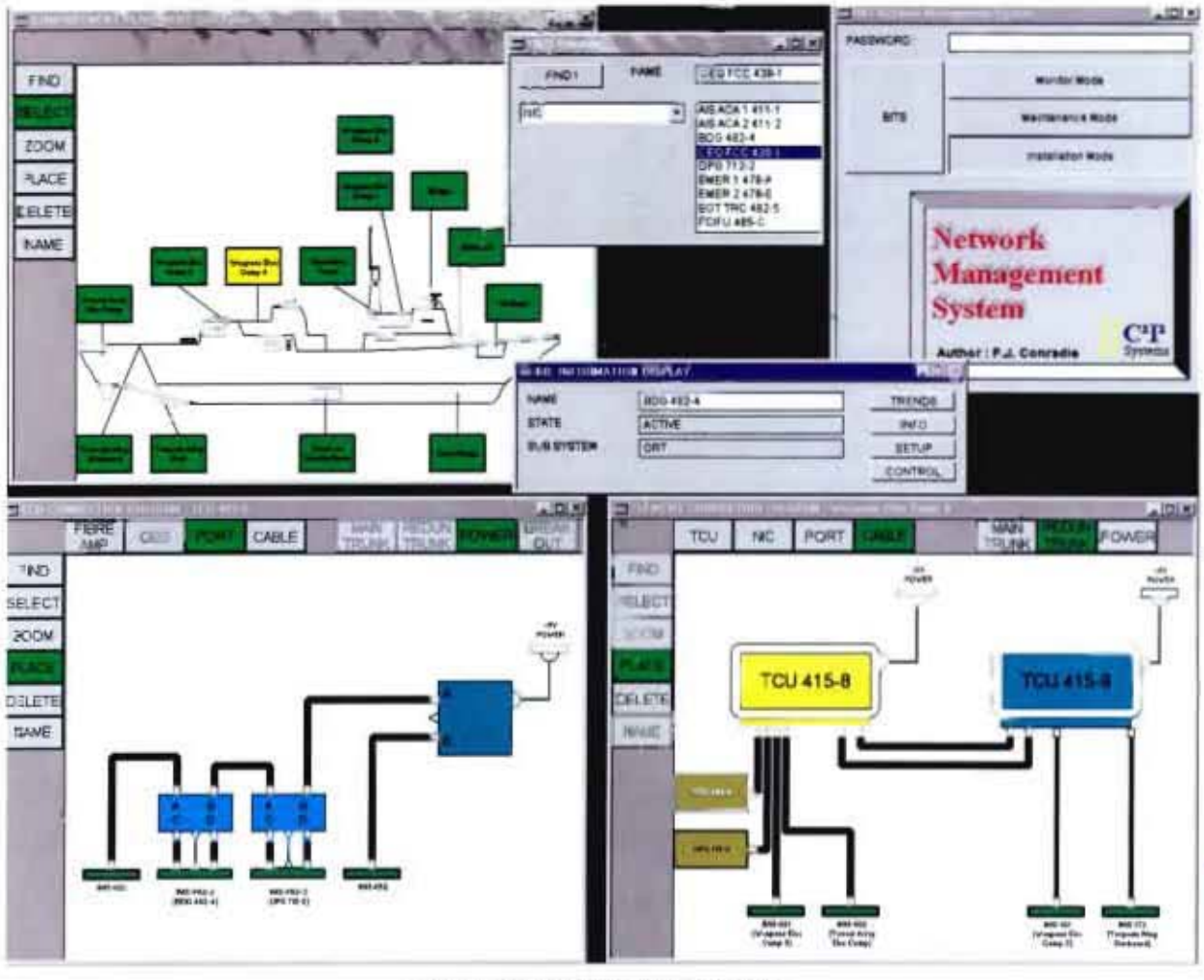


Figure 2 : Typical Network Management MMI

APPENDIX B FDDI ADAPTOR SPECIFICATIONS

PMC FDDI Adapter

C²I² Systems has developed a PMC FDDI Network Interface Card (NIC), which is ideally suited to embedded platforms. This NIC has been developed from SysKonnect's PCI FDDI NIC and as such features a wide range of compatible and qualified software drivers.

Seen from a physical point of view, an FDDI network consists of a dual-redundant fibre-optic ring. Only one of the two rings is used during normal operation. The second ring merely serves as a back-up medium. Stations in the ring are classed as Class A or Class B stations. Class A stations connect directly to the ring while Class B stations connect via a concentrator.

FDDI Features

FDDI meets the increasing demands of sophisticated networks :

- High data transmission rate (100 Mbps), efficiency and cost-effectiveness make FDDI the optimum solution for real-time and multimedia networks, as well as corporate backbones
- Up to 2 km between nodes
- Wide geographic range (100 km maximum circumference in dual ring)
- Large number of supported nodes (up to 500 dual-attached) in the ring
- Deterministic and bandwidth-efficient timed token passing access method
- Fault-tolerant operation provides for ring re-configuration in cases of problems such as media disruption or node failure
- A high level of protection against tapping and malfunction when using fibre optic cabling
- Optimum data protection and system availability by supporting SFT III server mirroring with Mirrored Server Link (MSL) driver
- Synchronous Bandwidth Allocator (SBA)
- High-performance PCI data transfer
- Two connector types available, SC or ST connectors
- Optical Bypass Switch Control
- Fully software configurable
- SMT Version 7.3

The PMC FDDI DAS NIC provides a Class A connection and integrates directly into the dual FDDI ring. NICs with ST connectors have the same optical characteristics as NICs with SC connectors.

Applications

- Digital telephony
- SCADA applications
- Distributed real-time applications
- Vetronics applications
- Mission-critical applications
- SAFENET applications

PMC FDDI Adapter

PMC FDDI Adapter Specifications

Designation	Connector	Grade	Attachment
CCII/FDDI/PMC/DAS/ST/COM	ST	Commercial	Dual
CCII/FDDI/PMC/SAS/ST/IND	ST	Industrial	Single
CCII/FDDI/PMC/DAS/SC/COM	SC	Commercial	Dual
CCII/FDDI/PMC/DAS/SC/IND	SC	Industrial	Dual
CCII/FDDI/PMC/DAS/ST/COM	ST	Commercial	Dual
Bus Interface	IEEE P1386.1 D2.0 32-bit PCI-Bus electrical and CMC formfactor		
Network Interface (Fibre)	ANSI X3T9.5 and X3T12 compatible		
LAN Controller	AMD Supernet 3		
RAM	128 kBytes CMOS static		
Flash EPROM	128 kBytes		
I/O Addresses	Automatic by PCI V2.1 Plug-and-Play assigned to the slot		
Interrupts	PCI INT A		
DMA	Automatic depending on PCI slot		
Timer	3 channels @ 6,25 MHz max.		
Dimensions	149 mm x 74 mm x 13,5 mm		
Power Consumption	< 1,45 A @ 5 V		
MTBF	Figures according to MIL-HDBK-217F, Parts Count Method: Ground Mobile T_j = 65 C, T_a = 45 C 19 000 hrs Naval, Sheltered T_j = 60 C, T_a = 40 C 26 000 hrs Airborne, Inhabited Cargo T_j = 75 C, T_a = 55 C 20 000 hrs		
Envirionmental Specifications	**Temperature**	**Commercial**	**Industrial**
	Operating	0 C to +55 C	-15 C to +75 C
	Non-Operating	-40 C to +85 C	-50 C to +85 C
	Humidity	0% to 95%	
	Shock	20 g for 11 ms	
	Vibration	5 to 500 Hz at 2 g	
Drivers	• VxWorks (4.3BSD + END) • DOS • NetWare • Windows • OS/2	• SCO UnixWare • Novell • AIX Unix • Linux • SUN Solaris	
Supporting Tools	• Remote Boot Software • FDDI SMT Diagnostic Program • Hardware Diagnostic Program for DOS • Hardware Diagnostic Program for PPC, Windows NT 3.5.1 and above PPC (NDIS 3) • Hardware Diagnostics Program for VxWorks-supported host carrier cards		

© C²I² Systems (Pty) Ltd

CCII Systems (Pty) Ltd • Registration No. 90/05058/07

COMMUNICATIONS
COMPUTER INTELLIGENCE
INTEGRATION

Interface Control Document

for the

Information Management System

Application Interface Services

	C²I² Systems	External
Document No.	CCII/A500/IMS/6-ICD/1	
Issue	1.1	
Issue Date	1999-02-19	
Print Date	2001-01-04	
Copy No.		

© CCII Systems (Pty) Ltd

The copyright of this document is the property of CCII System (Pty) Ltd. The document is issued for the sole purpose for which it is supplied, on the express terms that it may not be copied in whole or in part, used by or disclosed to others except as authorised in writing by CCII Systems (Pty) Ltd.

Document prepared by C²I² Systems (Pty) Ltd, Cape Town

Contents

1. Scope 1
1.1 Introduction 1
1.2 Overview 1
2. Applicable Documents 2
2.1 Specifications 2
2.2 Standards 2
2.3 Reference Documents 2
3. Application Interface Services 3
3.1 Application Interface Services (APIS) 3
3.1.1 APIS_INIT() 5
3.1.2 APIS_OPEN() 5
3.1.3 APIS_CLOSE() 5
3.1.4 APIS_PRODUCE() 5
3.1.5 APIS_DEMAND() 5
3.1.6 APIS_REMOVE_PRODUCE() 5
3.1.7 APIS_REMOVE_DEMAND() 5
3.1.8 APIS_SEND_MSG() 5
3.1.9 APIS_RECEIVE_MSG() 5
3.2 APIS Multibus II message passing 6
3.2.1 Multibus II message passing protocol packet 6
3.2.2 Multibus II Transport Sockets 6
3.3 Parameter Representation 6
3.3.1 Scalar Data Types 6
3.3.2 Address Ordering 6
3.4 Message Acknowledgements 7
3.4.1 Application Layer Integrity Control 7
3.4.2 Reply Timing 7
4. Multibus II APIS Implementation 8
4.1 APIS_INIT() 8
4.1.1 Message Description 8
4.1.2 Unsolicited Request 8
4.1.2.1 Parameters 8
4.1.2.2 Parameter Offsets 9
4.1.3 Unsolicited Reply 9
4.1.3.1 Parameters 9
4.1.3.2 Parameter Offsets 9
4.2 APIS_OPEN() 10
4.2.1 Message Description 10
4.2.2 Unsolicited Request 10
4.2.2.1 Parameters 10
4.2.2.2 Parameter Offsets 11
4.2.3 Unsolicited Reply 11
4.2.3.1 Parameters 11
4.2.3.2 Parameter Offsets 12
4.3 APIS_CLOSE() 13
4.3.1 Message Description 13
4.3.2 Unsolicited Request 13
4.3.2.1 Parameters 13
4.3.2.2 Parameter Offsets 14
4.3.3 Unsolicited Reply 14
4.3.3.1 Parameters 14
4.3.3.2 Parameter Offsets 14

4.4 APIS_PRODUCE() ... 15
4.4.1 Message Description ... 15
4.4.2 Unsolicited Request ... 15
4.4.2.1 Parameters ... 15
4.4.2.2 Parameter Offsets ... 16
4.4.3 Unsolicited Reply ... 17
4.4.3.1 Parameters ... 17
4.4.3.2 Parameter Offsets ... 18
4.5 APIS_DEMAND() ... 19
4.5.1 Message Description ... 19
4.5.2 Unsolicited Request ... 19
4.5.2.1 Parameters ... 19
4.5.2.2 Parameter Offsets ... 20
4.5.3 Unsolicited Reply ... 21
4.5.3.1 Parameters ... 21
4.5.3.2 Parameter Offsets ... 21
4.6 APIS_REMOVE_PRODUCE() ... 22
4.6.1 Message Description ... 22
4.6.2 Unsolicited Request ... 22
4.6.2.1 Parameters ... 22
4.6.2.2 Parameter Offsets ... 23
4.6.3 Unsolicited Reply ... 23
4.6.3.1 Parameters ... 23
4.6.3.2 Parameter Offsets ... 24
4.7 APIS_REMOVE_DEMAND() ... 25
4.7.1 Message Description ... 25
4.7.2 Unsolicited Request ... 25
4.7.2.1 Parameters ... 25
4.7.2.2 Parameter Offsets ... 26
4.7.3 Unsolicited Reply ... 26
4.7.3.1 Parameters ... 26
4.7.3.2 Parameter Offsets ... 27
4.8 APIS_SEND_MSG() ... 28
4.8.1 Message Description ... 28
4.8.2 Solicited Message ... 28
4.8.2.1 Parameters ... 28
4.8.2.2 Parameter Offsets ... 29
4.8.3 Unsolicited Reply ... 30
4.8.3.1 Parameters ... 30
4.8.3.2 Parameter Offsets ... 30
4.9 APIS_RECEIVE_MSG() ... 31
4.9.1 Message Description ... 31
4.9.2 Solicited Message ... 31
4.9.2.1 Parameters ... 31
4.9.2.2 Parameter Offsets ... 32
5. PC-Compatible (VxWorks) APIS Implementation ... 33
5.1 APIS_DEMAND() ... 33
5.1.1 Message Description ... 33
5.1.2 Unsolicited Request ... 34
5.1.2.1 Parameters ... 34
5.1.2.2 Parameter Offsets ... 35
5.1.3 Unsolicited Reply ... 35
5.1.3.1 Parameters ... 35
5.1.3.2 Parameter Offsets ... 36
6. Returned Values ... 37

Abbreviations

ASAP	Application Services Access Point
APID	Application Identifier
APIS	Application Interface Services
ASU	Application Service User
BITS	Built-in-Test Services
CPU	Central Processing Unit
CS	Integrated Combat System
FDDI	Fibre Data Distributed Interface
IMS	Information Management System
LAN	Local Area Network
LSB	Least Significant Byte
MBII	Multibus II
MSB	Most Significant Byte
NIC	Network Interface Card
PC	Personal Computer
PDU	Protocol Data Unit
SAFENET	Survivable Adaptable Fibre Optic Embedded Network
TBD	To Be Determined
TP	Transport Protocol

1. Scope

1.1 Introduction

The sub-systems interfacing to each other via the Information Management System (IMS) shall interface to the Local Area Network (LAN) via the Network Interface Card (NIC). IMS shall implement protocol processing, flow, rate and error control as well as local network management.

The NIC may be either a Multibus II-compatible FDDI card, or PC/PCI- compatible card. In all cases, the application software shall interface to the LAN via the Application Interface Services (APIS) software. In the case of the Multibus II NIC interface, the Application Interface Services software shall run on the NIC and in the PC/PCI card case, the application software and the protocol software shall share the PC CPU processing resources. So as not to affect the performance of the host station in the PC case, the host processor should be at least a 66 MHZ 486DX2 CPU employing a PCI parallel backplane bus.

The Application Interface Services shall be a flexible, real-time network interface employing an open architecture.

1.2 Overview

This document describes Application Interface Services software which provides the interface between the application program and the IMS LAN for both the Multibus II and PC Message Queue interfaces.

Paragraph 3 is an overview of the Application Interface Services provided by IMS.

Paragraph 4 describes the implementation of the Application Interface Services for a Multibus II system. It provides the messages that flow between the application and APIS to bit level.

Paragraph 5 describes the message queue (VxWorks) interface for a PC-compatible APIS

Paragraph 6 describes the returned values from the services provided.

2. Applicable Documents

2.1 Specifications

The following documents, to the exact issue, form part of this document. In the event of a conflict between this document and the documents referenced here, this document shall take precedence.

a. C²I² Systems Document No. CCII/A500/IMS/6-PIDS, entitled *Prime Item Development Specification for the Patrol Corvette Combat Suite Information Management System.*

2.2 Standards

The following documents, to the exact issue, form part of this document. In the event of a conflict between this document and the documents referenced here, this document shall take precedence.

a. 453508-001 Multibus II Transport Protocol Specification and Designer's Guide, Rev. 001.

b. ANSI/IEEE STD1296-1987 IEEE Standard for a High-Performance Synchronous 34-Bit Bus: Multibus II.

2.3 Reference Documents

The following documents do not form part of this document. They are listed as a reference to clarify any points addressed in this document.

a. 469157-001 iRMX System Call Reference, Rev. 001.

b. 469150-001 iRMX System Configuration and Administration, Rev.001.

c. 469160-001 iRMX Programming Techniques and Examples, Rev. 001.

d. 500 0017 Technical Reference Manual for CL386/DAS High-Performance CPU/FDDI Controller, Rev. 004.

3. Application Interface Services

3.1 Application Interface Services (APIS)

Application Interface Services (APIS) allow the Application Service User (ASU) to communicate on the network without having any knowledge of the message sources or destinations. This allows for a degree of dataflow abstraction. The sources and destinations of messages are determined by the transport and underlying layers.

Following a cold or warm start of a Multibus II sub-system processor host, a once only APIS_INIT() command is issued to APIS. This allows APIS to clear it's tables in respect of this Multibus II host.

An ASU registers with APIS utilising the APIS_OPEN() command. It shall also inform APIS which messages it can source and which messages it requires, to perform its functions, with the APIS_PRODUCE() and APIS_DEMAND() commands respectively. The ASU does not specify the source or destination of any of these messages. The Application Interface Services will determine the destination and source addresses for the messages. This will ensure that if a message changes, only the application code has to change.

To transmit a message to another registered ASU, the source ASU utilises the APIS_SEND_MSG() command.

An ASU can dynamically add and remove messages with the APIS_PRODUCE(), APIS_DEMAND(), APIS_REMOVE_PRODUCE() and APIS_REMOVE_DEMAND() commands.

When an ASU issues the APIS_CLOSE() command, all the messages transferred to and from it will be removed from the network.

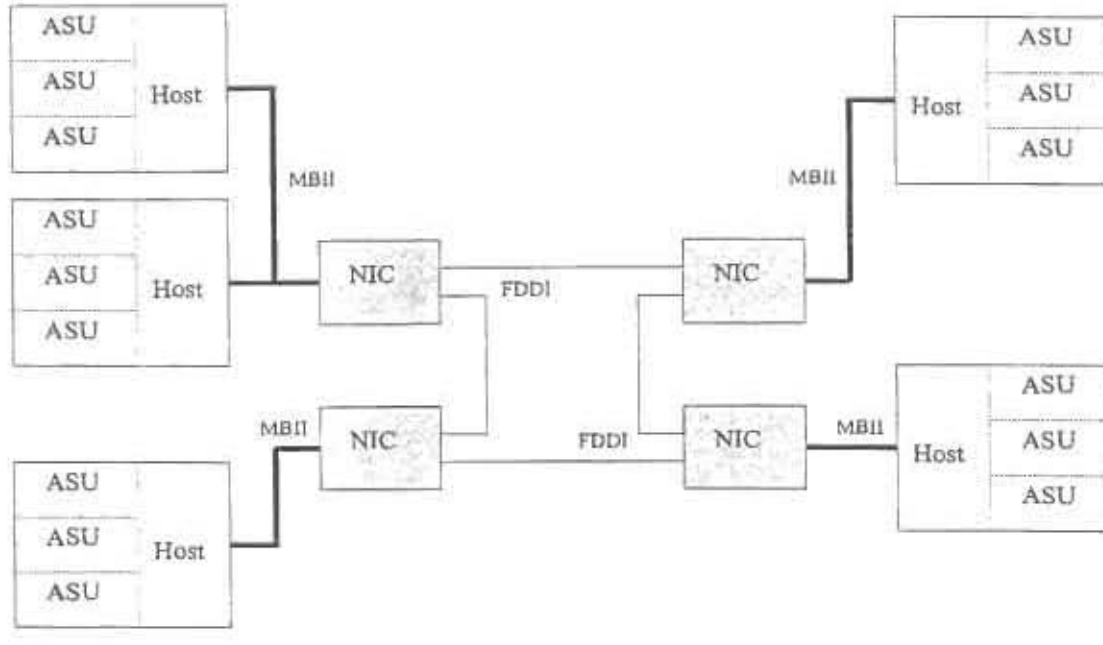


Figure 1. Typical LAN Configuration utilising Multibus II based NICs

A message on the LAN is identified by a unique message ID number. APIS interprets the Message ID as a numerical number consisting of four 16 bit fields. The System Contractor can allocate meaning to any field in the Message ID without effecting the operation of the APIS.

The only APIS requirement is that the Message ID uniquely identifies a message on the LAN.

The following Application Interface Services shall be provided :

- APIS_INIT()
- APIS_OPEN()
- APIS_CLOSE()
- APIS_PRODUCE()
- APIS_DEMAND()
- APIS_REMOVE_PRODUCE()
- APIS_REMOVE_DEMAND()
- APIS_SEND_MSG()
- APIS_RECEIVE_MSG()

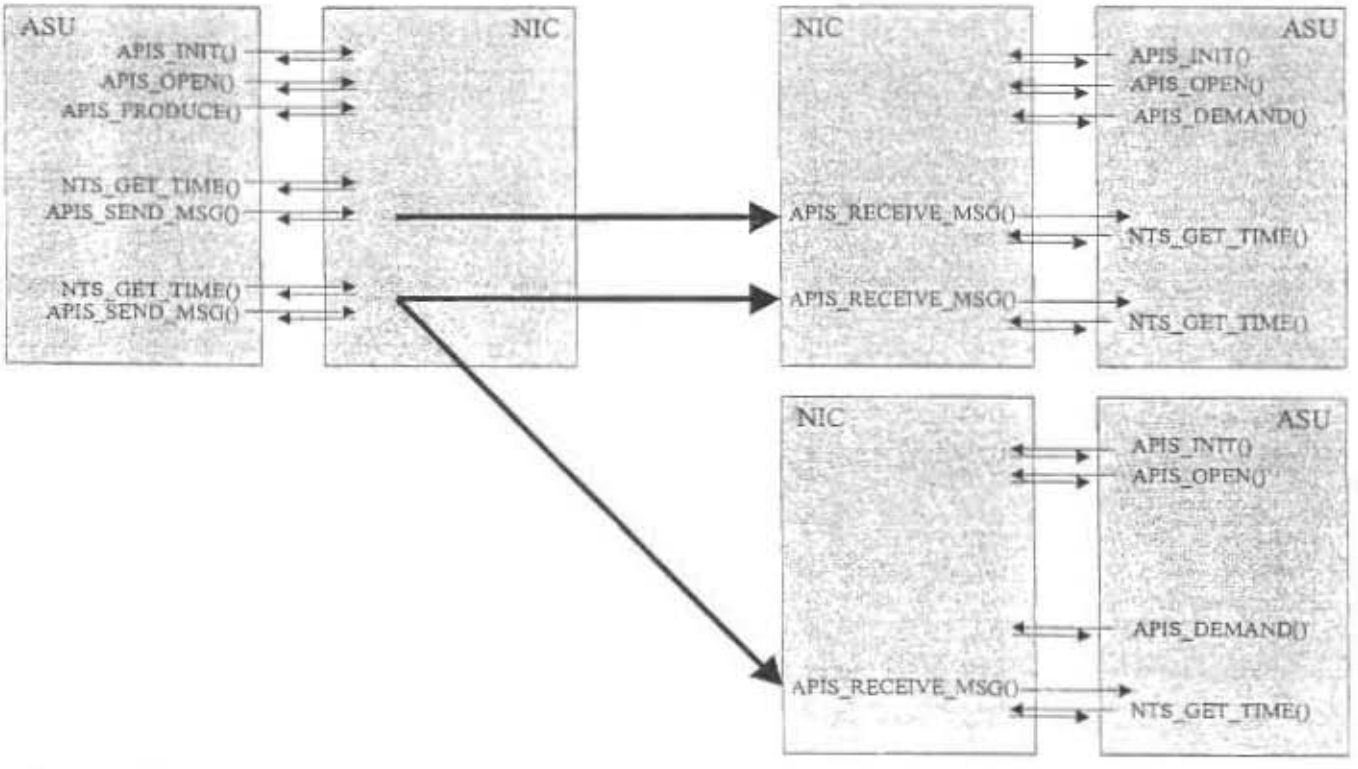


Figure 2. Transition diagram example

3.1.1 APIS_INIT()

This service command assists the Application Interface Services with table administration. It allows for the removal of all ASU information linked to the processor host card that issued the command as well as freeing of associated unused memory buffers. This command should only be issued once per processor host card after start-up.

3.1.2 APIS_OPEN()

This service command provides for the registration of the application with the Application Interface Services as a potential Application Service User (ASU). This allows for the bi-directional identification of ASUs with APIS, through the exchanging of Application IDs and Application Service Access Point (ASAP) descriptors.

3.1.3 APIS_CLOSE()

This service command provides for the closure of the ASAP.

3.1.4 APIS_PRODUCE()

This service command registers a data message with the Application User Interface for transmission.

3.1.5 APIS_DEMAND()

This service command registers a data message with the Application User Interface for reception.

3.1.6 APIS_REMOVE_PRODUCE()

This service command provides for the removal of a stream data message from the Application User Interface.

3.1.7 APIS_REMOVE_DEMAND()

This service command provides for the removal of a stream data message from the Application User Interface.

3.1.8 APIS_SEND_MSG()

This service command allows for the transmission of a stream data message from the CS application to other application(s) on the network that are registered as requiring that particular message.

3.1.9 APIS_RECEIVE_MSG()

This service command allows for the passing on to the application the received stream data message from another application. This is not a request made by the ASU, but an event generated by the APIS.

3.2 APIS Multibus II message passing

The APIS Interface requires that remote applications use the Multibus Transport Protocol or at least a Multibus Transport Protocol Implementation, which interfaces with the Multibus II datalink protocol.

3.2.1 Multibus II message passing protocol packet

APIS makes use of the following Multibus II Transport protocol packet for unsolicited packets.

		Source	Destination	Herdware defined (4 bytes)
		Request ID	Type	
Destination Port ID		Transmission control	Protocol ID	Transport Defined (8 bytes)
Transaction control	Transaction ID	Source Port ID		
				Used by APIS for parameter passing (20 bytes)

3.2.2 Multibus II Transport Sockets

APIS makes use of the Multibus II Transport Protocol concept of Sockets to identify source and destination software endpoints. A Socket consists of a 16 bit Host_ID and a 16 bit Port_ID as defined in [2.2.a]. The Host_ID identifies the MPC agent and normally reflects the slot_nr of the Processor board in the Multibus II backplane. Within a host (Processor Board in the Multibus II backplane), the Port_ID identifies a software endpoint (mailbox in case of destination endpoint).

Status Sockets specified for each command are only valid for that particular command and any exceptions will be stated. The Host_ID / Port_ID socket information in the APIS packet parameter field, is to be considered the only valid specified software endpoint, even if this proves to be a duplication of information included in the packet header.

3.3 Parameter Representation

3.3.1 Scalar Data Types

All parameters used are unsigned except for the signed parameter 'Return Value' which is formatted as a two's-complement 16 bit value.

3.3.2 Address Ordering

Little endian ordering are used for parameter storing. Multi-byte objects are stored starting with the low-order byte at the lowest address.

4. Multibus II APIS Implementation

4.1 APIS_INIT()

4.1.1 Message Description

This message is used to inform an Application Interface Service of the start-up of an application processor card host. It must be called once by every host, before any applications on this host issues any other APIS request. It allows for the removal of all ASU information linked to the processor host card that issued the command, as well as for freeing associated unused memory buffers. It is a Multibus II TP Transaction Message of type **unsolicited request with unsolicited reply**. The unsolicited request will come from the ASU and the unsolicited reply from APIS.

This message is passed through the APIS Command Socket which consists of the combination [Host_Id.Port_Id] or [nic_slot_nr:0x0E01].

If the APIS_INIT request indicates that a reply is required (Port ID > 0), this request will be acknowledged (reply sent) within 5000ms.

4.1.2 Unsolicited Request

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x00	Identifies open message
Reserved	8 bits	0x00	
Reserved	16 bits	0x0000	
ASU Host ID	16 bits	0x0000-0x13	APIS will combine this ASU Host ID with Status Port ID to form the socket address.
Status Port ID	16 bits	0x0000-0xFFFF	APIS will use this Port ID with the ASU Host ID to form the Status Socket.

4.1.2.1 Parameters

Command ID The 8 bit unsigned value **0x00** identifying this protocol data unit (PDU) as an APIS_INIT command.

ASU Host ID A 16 bit unsigned value identifying the ASU Multibus II agent host.

Status Port ID A 16 bit unsigned value identifying the ASU status software endpoint within a host. This Port ID, combined with the ASU Host ID forms a full MB II transport socket address. This Status Socket formed by the ASU Host ID: Status Port ID pair, will be used by APIS for returning the reply to this request. A Port ID value of 0x0000 indicates that no reply must be generated by APIS for this request.

4.1.2.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Command ID	Reserved Field	Reserved Field		LSB	MSB	LSB	MSB
				ASU Host ID		Status Port ID	

4.1.3 Unsolicited Reply

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x80	Identifies message
Reserved	8 bits	0x00	
Return Value	16 bits	Valid return values	Refer to list of return values

4.1.3.1 Parameters

Command ID The 8 bit unsigned value **0x80** identifying this protocol data unit (PDU) as an APIS_INIT_REPLY command.

Return Value APIS_INIT returns a 16-bit signed value representing the status of the call. Allowed values are listed below. (Refer to paragraph 6. Return Values.)

NO_ERROR no error occurred

4.1.3.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3
Command ID	Reserved Field	LSB	MSB
		Return Value	

4.2 APIS_OPEN()

4.2.1 Message Description

This message is used to open an Application Interface Service for an application. It is a Multibus II TP Transaction Message of type **unsolicited request with unsolicited reply**. The unsolicited request will come from the ASU and the unsolicited reply from APIS.

This message is passed through the APIS Command Socket which consists of the combination [Host_id:Port_id] or [nic_slot_nr:0x0E01].

4.2.2 Unsolicited Request

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x01	Identifies open message
Application ID	8 bits	0x00-0xFF	Identifies ASU with Application ID
Reserved	16 bits	0x0000	
ASU Host ID	16 bits	0x0000-0x13	APIS will combine this ASU Host ID with Status Port ID to form the socket address.
Status Port ID	16 bits	0x0001-0xFFFF	APIS will use this Port ID with the ASU Host ID to form the Status Socket.
ASU Text String	32 bits	0x20:0x20:0x20:0x20 - 0x7E:0x7E:0x7E:0x7E	Used to identify ASUs in a more user-friendly readable format.

4.2.2.1 Parameters

Command ID The 8 bit unsigned value **0x01** identifying this protocol data unit (PDU) as an APIS_OPEN command.

Application ID An 8 bit unsigned value identifying a local ASU to APIS.

ASU Host ID A 16 bit unsigned value identifying the ASU Multibus II agent host.

Status Port ID A 16 bit unsigned value identifying the ASU status software endpoint within a host. This Port ID, combined with the ASU Host ID forms a full MB II transport socket address. This Status Socket formed by the ASU Host ID: Status Port ID pair, will be used by APIS for returning the reply to this request. This Status socket is also required by Build-in-Test Services (BITS) for polling the status of this ASU under BITS (see BITS_GET_STATUS_APID under BITS).

A Port ID value of 0x0000 is not allowed for the APIS_OPEN command.

ASU Text String Four 8 bit alpha-numeric characters (0x20 - 0x7E) identifying the ASU in a more user-friendly readable format. This String is not used by APIS, but only returned in conjunction with the ASAP handle, by BITS, to provide a more readable link to the ASAP handle.

4.2.2.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Command ID	Application ID	Reserved Field		LSB	MSB	LSB	MSB
				ASU Host ID		Status Port ID	

Byte 8	Byte 9	Byte 10	Byte 11
Text 0	Text 1	Text 2	Text 3
ASU Text String			

4.2.3 Unsolicited Reply

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x81	Identifies message
Reserved	8 bits	0x00	
Return Value	16 bits	valid return values	Refer to list of return values
ASAP	16 bits	0x0000-0xFFFF	Generated by APIS. Used by the ASU (and other ASUs) to identify itself.

4.2.3.1 Parameters

Command ID The 8 bit unsigned value **0x81** identifying this protocol data unit (PDU) as an APIS_OPEN_REPLY command.

Return Value APIS_OPEN returns a 16-bit signed value representing the status of the call. Allowed values are listed below. (Refer to paragraph 6. Return Values.)

NO_ERROR	no error occurred
E_APID_IN_USE	specified APID has already been used on this NIC
E_STRING_INVALID	specified ASU TEXT STRING is invalid

ASAP A 16 bit unsigned (APIS network wide globally unique) handle identifying the ASU. If the Return Value indicates NO_ERROR, this handle is valid and must be used by this ASU in all subsequent commands.

4.2.3.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Command ID	Reserved Field	LSB	MSB	LSB	MSB
		Return Value		ASAP	

4.3 APIS_CLOSE()

4.3.1 Message Description

This message is used to close an Application Service Access Point (ASAP). All messages sent via this ASAP, before this command is issued and successfully acknowledged, will be transferred to consumers. All messages previously registered under this ASAP will be removed. It is a Multibus II TP Transaction Message of type **unsolicited request with unsolicited reply**. The unsolicited request will come from the ASU and the unsolicited reply from APIS. If an ASAP is closed all the messages transmitted and received will be removed from the APIS memory.

This message is passed through the APIS Command Socket which consists of the combination [Host_id:Port_id] or [nic_slot_nr:0x0E01].

4.3.2 Unsolicited Request

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x02	Identifies message
Reserved	8 bits	0x00	
ASAP	16 bits	0x0000-0xFFFF	ASAP to close
ASU Host ID	16 bits	0x0000-0x13	APIS will combine this ASU Host ID with Status Port ID to form the socket address
Status Port ID	16 bits	0x0000-0xFFFF	APIS will use this Port ID with the ASU Host ID to form the Status Socket.

4.3.2.1 Parameters

Command ID The 8 bit unsigned value **0x02** identifying this protocol data unit (PDU) as an APIS_CLOSE command.

ASAP A 16 bit unsigned (APIS network wide globally unique) handle identifying the ASU ASAP that must be closed.

ASU Host ID A 16 bit unsigned value identifying the ASU Multibus II agent host.

Status Port ID A 16 bit unsigned value identifying the ASU status software endpoint within a host. This Port ID, combined with the ASU Host ID forms a full MB II transport socket address, which is used by APIS for returning the reply message.

A Port ID value of 0x0000 indicates that no reply must be generated by APIS for this request.

4.3.2.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Command ID	Reserved Field	LSB	MSB	LSB	MSB	LSB	MSB
		ASAP		ASU Host ID		Status Port ID	

4.3.3 Unsolicited Reply

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x82	Identifies message
Reserved	8 bits	0x00	
Return Value	16 bits	valid return values	Refer to list of return values

4.3.3.1 Parameters

Command ID The 8 bit unsigned value **0x82** identifying this protocol data unit (PDU) as an APIS_CLOSE_REPLY command.

Return Value APIS_CLOSE returns a 16-bit signed value representing the status of the call. Allowed values are listed below. (Refer to paragraph 6. Return Values.)

NO_ERROR	no error occurred
E_ASAP_NOT_OPEN	specified ASAP not locally registered

4.3.3.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3
Command ID	Reserved Field	LSB	MSB
		Return Value	

4.4 APIS_PRODUCE()

4.4.1 Message Description

This message is used to add a message for transmission to the message list. It is a Multibus II TP Transaction Message of type **unsolicited request with unsolicited reply**. The unsolicited request will come from the ASU and the unsolicited reply from APIS.

This message is passed through the APIS Command Socket which consists of the combination [Host_id:Port_id] or [nic_slot_nr:0x0E01].

4.4.2 Unsolicited Request

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x03	Identifies message
Priority	8 bits	1-8	Priority (decimal) of message to be transmitted
ASAP	16 bits	0x0000-0xFFFF	Identifies sender.
ASU Host ID	16 bits	0x0000-0x13	APIS will combine this ASU Host ID with Status Port ID to form the socket address.
Status Port ID	16 bits	0x0000-0xFFFF	APIS will use this Port ID with the ASU Host ID to form the Status Socket.
Message ID	64 bits	0x0001:0x0001: 0x0001:0x0001 - 0xFFFF:0xFFFF: 0xFFFF:0xFFFF	Message ID of message to be transmitted
Data Length	16 bits	1-4000	Size (decimal) of message to be transmitted.
Committed Repetition Interval	16 bits	0-65535	Repetition interval (decimal) between message transmissions (ms).

4.4.2.1 Parameters

Command ID The 8 bit unsigned value **0x03** identifying this protocol data unit (PDU) as an APIS_PRODUCE command.

Priority An 8 bit unsigned value specifying the priority of the message to be produced.

Value	Meaning	
0,9-255	Invalid	
1 - 8	Priority:	
	1	highest priority synchronous data,
	2-4	high priority synchronous data,
	5-7	low priority asynchronous data,
	8	lowest priority asynchronous data.

ASAP A 16 bit unsigned (APIS network wide globally unique) handle identifying the ASU ASAP that is to be used for producing this Message ID.

ASU Host ID A 16 bit unsigned value identifying the ASU Multibus II agent host.

Status Port ID A 16 bit unsigned value identifying the ASU status software endpoint within a host. This Port ID, combined with the ASU Host ID forms a full MB II transport socket address, which is used by APIS for returning the reply message.

A Port ID value of 0x0000 indicates that no reply must be generated by APIS for this request.

Message ID Four 16 bit unsigned value fields specifying the Message ID of a message to be produced. Wildcard values (0x0000) are not allowed in any of the Message ID fields for APIS_PRODUCE.

Data Length Specifies the length (1- 4000 decimal) in bytes of the data message.

Committed Repetition Interval Specifies the minimum producing repetition interval in ms.

Value	Meaning
0	This is not a repetitive message and therefore requires no specific bandwidth allocation
1-65535	Specifies repetition interval between transmissions for specific bandwidth allocation using Data Length

4.4.2.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Command ID	Priority	LSB	MSB	LSB	MSB	LSB	MSB
		ASAP		ASU Host ID		Status Port ID	

Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB
Message ID 0		Message ID 1		Message ID 2		Message ID 3	
Message ID							

Byte 16	Byte 17	Byte 18	Byte 19
LSB	MSB	LSB	MSB
Data Length		Committed Repetition Interval	

4.4.3 Unsolicited Reply

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x83	Identifies message
Reserved	8 bits	0x00	
Return Value	16 bits	valid return values	Refer to a list of the return values
Host ID	16 bits	0x0000-0x13	ASU will combine this Host ID with Producer Port ID to form the Producer Socket
Producer Port ID	16 bits	0x0000-0xFFFF	If return value is NO_ERROR then this Port ID combined with the Host ID forms the Producer Socket that the ASU will use to transfer data to APIS. If the return value signals an error, then this will be 0.

4.4.3.1 Parameters

Command ID The 8 bit unsigned value **0x83** identifying this protocol data unit (PDU) as an APIS_PRODUCE_REPLY command.

Return Value APIS_PRODUCE returns a 16-bit signed value representing the status of the call. Possible values are listed below. (Refer to paragraph 6. Return Values.)

NO_ERROR	no error occurred
E_ASAP_NOT_OPEN	specified ASAP not locally registered
E_MSG_ID_INVALID	specified Msg ID is invalid
E_LENGTH_INVALID	specified Length is invalid
E_PRIORITY_INVALID	specified Priority is invalid
E_INTERVAL_INVALID	specified Repetition Interval is invalid
E_PROD_EXIST_FOR_ASAP	specified Msg ID is already registered for this Priority and ASAP
E_PRODUCER_LIMIT	maximum number of producers has registered this Msg ID
E_PROD_BW_CHANGE	specified Length and Repetition Interval values do not correspond with previously registered values. (Only valid if specified Msg ID is already registered under a different ASAP).
E_BW_LIMIT	maximum bandwidth allocation limit exceeded

4.4.3.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Command ID	Reserved Field	LSB	MSB	LSB	MSB	LSB	MSB
		Return Value		Host ID		Producer Port ID	

4.5 APIS_DEMAND()

4.5.1 Message Description

This message is used to add a message for reception to the message list. It is a Multibus II TP Transaction Message of type **unsolicited request with unsolicited reply**. The unsolicited request will come from the ASU and the unsolicited reply from APIS.

This message is passed through the APIS Command Socket which consists of the combination [Host_id:Port_id] or [nic_slot_nr:0x0E01].

4.5.2 Unsolicited Request

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x04	Identifies message
Reserved	8 bits	0x00	
ASAP	16 bits	0x0000-0xFFFF	Identifies receiver
ASU Host ID	16 bits	0x0000-0x13	APIS will combine this ASU Host ID with the relevant Port IDs to form a socket address
Status Port ID	16 bits	0x0000-0xFFFF	APIS will use this Port ID with the ASU Host ID to form the Status Socket.
Message ID	64 bits	0x0000:0x0000: 0x0000:0x0000 - 0xFFFF:0xFFFF: 0xFFFF:0xFFFF	Message ID of message to be received.
Consumer Port ID	16 bits	0x0000-0xFFFF	APIS will transfer any received data for the ASU to the Consumer Socket consisting of the ASU Host ID and this Port ID.
Repetition Interval	16 bits	0x0000-0xFFFF	Minimum repetition interval for message delivery (ms).

4.5.2.1 Parameters

Command ID The 8 bit unsigned value **0x04** identifying this protocol data unit (PDU) as an APIS_DEMAND command.

ASAP A 16 bit unsigned (APIS network wide globally unique) handle identifying the ASU ASAP that is the consumer of this Message ID.

ASU Host ID A 16 bit unsigned value identifying the ASU Multibus II agent host.

Status Port ID A 16 bit unsigned value identifying the ASU status software endpoint within a host. This Port ID, combined with the ASU Host ID forms a full

MB II transport socket address, which is used by APIS for returning the reply message.

A Port ID value of 0x0000 indicates that no reply must be generated by APIS for this request.

Message ID Four 16 bit unsigned value fields specifying the Message ID of a message to be consumed. Wildcard values (0x0000) are allowed in any of the Message ID fields.

Consumer Port ID A 16 bit unsigned value identifying the ASU consumer software endpoint within the ASU host. This Port ID, combined with the ASU Host ID forms a full MB II transport socket address, which is used by APIS for delivering the requested message.

Repetition Interval Specifies the minimum delivery repetition interval in ms. APIS will deliver only the last received data message after expiry of the specified minimum time interval since last delivery. Thus, messages received by APIS during the indicated time interval after the last delivery, will not be passed on to the consumer.

Value	Meaning
0	Immediate delivery of messages on reception by APIS.
1-65535	Specifies immediate delivery of latest message to ASU if this specified time interval since last delivery to the ASU has expired.

4.5.2.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Command ID	Reserved Field	LSB	MSB	LSB	MSB	LSB	MSB
		ASAP		ASU Host ID		Status Port ID	

Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB
Message ID 0		Message ID 1		Message ID 2		Message ID 3	
Message ID							

Byte 16	Byte 17	Byte 18	Byte 19
LSB	MSB	LSB	MSB
Consumer Port ID		Repetition Interval	

4.5.3 Unsolicited Reply

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x84	Identifies message
Reserved	8 bits	0x00	
Return Value	16 bits	valid return values	Refer to a list of the return values

4.5.3.1 Parameters

Command ID The 8 bit unsigned value **0x84** identifying this protocol data unit (PDU) as an APIS_DEMAND_REPLY command.

Return Value APIS_DEMAND returns a 16-bit signed value representing the status of the call. Allowed values are listed below. (Refer to paragraph 6. Return Values.)

NO_ERROR	no error occurred
E_ASAP_NOT_OPEN	specified ASAP not locally registered
E_INTERVAL_INVALID	specified Repetition Interval is invalid
E_CONS_SKT_INVALID	specified MBII socket is invalid
E_DEMAND_EXIST_FOR_ASAP	specified Msg ID is already registered for this ASAP

4.5.3.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3
Command ID	Reserved Field	LSB	MSB
		Return Value	

4.6 APIS_REMOVE_PRODUCE()

4.6.1 Message Description

This message is used to delete a message for transmission from the message list. All messages matching the specified message ID, sent via this ASAP before this command is issued and successfully acknowledged, will be transferred to consumers. It is a Multibus II TP Transaction Message of **type unsolicited request with unsolicited reply**. The unsolicited request will come from the ASU and the unsolicited reply from APIS.

This message is passed through the APIS Command Socket which consists of the combination [Host_id:Port_id] or [nic_slot_nr:0x0E01].

4.6.2 Unsolicited Request

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x05	Identifies message
Reserved	8 bits	0x00	
ASAP	16 bits	0x0000-0xFFFF	Identifies sender
ASU Host ID	16 bits	0x0000-0x13	APIS will combine this ASU Host ID with Status Port ID to form the socket address.
Status Port ID	16 bits	0x0000-0xFFFF	APIS will use this Port ID with the ASU Host ID to form the Status Socket.
Message ID	64 bits	0x0000:0x0000: 0x0000:0x0000 - 0xFFFF:0xFFFF: 0xFFFF:0xFFFF	Message ID of produced message to be removed

4.6.2.1 Parameters

Command ID The 8 bit unsigned value **0x05** identifying this protocol data unit (PDU) as an APIS_REMOVE_PRODUCE command.

ASAP A 16 bit unsigned (APIS network wide globally unique) handle identifying the ASU ASAP that will no longer be the producer of this Message ID.

ASU Host ID A 16 bit unsigned value identifying the ASU Multibus II agent host.

Status Port ID A 16 bit unsigned value identifying the ASU status software endpoint within a host. This Port ID, combined with the ASU Host ID forms a full MB II transport socket address, which is used by APIS for returning the reply message.

A Port ID value of 0x0000 indicates that no reply must be generated by APIS for this request.

Message ID Four 16 bit unsigned value fields specifying the Message ID of a produced message previously registered. Wildcard values (0x0000) are allowed in any of the Message ID fields. All Message IDs currently being produced under the specified ASAP, matching this wildcard value, will be removed.

4.6.2.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Command ID	Reserved Field	LSB	MSB	LSB	MSB	LSB	MSB
		ASAP		ASU Host ID		Status Port ID	

Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB
Message ID 0		Message ID 1		Message ID 2		Message ID 3	
Message ID							

4.6.3 Unsolicited Reply

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x85	Identifies message
Reserved	8 bits	0x00	
Return Value	16 bits	valid return values	Refer to list of return values.

4.6.3.1 Parameters

Command ID The 8 bit unsigned value **0x85** identifying this protocol data unit (PDU) as an APIS_REMOVE_PRODUCE_REPLY command.

Return Value APIS_REMOVE_PRODUCE returns a 16-bit signed value representing the status of the call. Allowed values are listed below. (Refer to paragraph 6. Return Values.)

NO_ERROR	no error occurred
E_ASAP_NOT_OPEN	specified ASAP not locally registered
E_MSG_NOT_REGISTERED	specified Msg ID is not registered for this ASAP

4.6.3.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3
Command ID	Reserved Field	LSB	MSB
		Return Value	

4.7 APIS_REMOVE_DEMAND()

4.7.1 Message Description

This message is used to delete a message for reception from the message list. No messages, as specified by the Message ID parameter, received after this command is issued and successfully acknowledged, will be transferred to the ASU. It is a Multibus II TP Transaction Message of **type unsolicited request with unsolicited reply**. The unsolicited request will come from the ASU and the unsolicited reply from APIS.

This message is passed through the APIS Command Socket which consists of the combination [Host_id:Port_id] or [nic_slot_nr:0x0E01].

4.7.2 Unsolicited Request

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x06	Identifies message
Reserved	8 bits	0x00	
ASAP	16 bits	0x0000-0xFFFF	Identifies sender
ASU Host ID	16 bits	0x0000-0x13	APIS will combine this ASU Host ID with Status Port ID to form the socket address.
Status Port ID	16 bits	0x0000-0xFFFF	APIS will use this Port ID with the ASU Host ID to form the Status Socket.
Message ID	64 bits	0x0000:0x0000: 0x0000:0x0000 - 0xFFFF:0xFFFF: 0xFFFF:0xFFFF	Message ID of message request to be removed

4.7.2.1 Parameters

Command ID The 8 bit unsigned value **0x06** identifying this protocol data unit (PDU) as an APIS_REMOVE_DEMAND command.

ASAP A 16 bit unsigned (APIS network wide globally unique) handle identifying the ASU ASAP that will no longer a consumer of the specified Message.

ASU Host ID A 16 bit unsigned value identifying the ASU Multibus II agent host.

Status Port ID A 16 bit unsigned value identifying the ASU status software endpoint within a host. This Port ID, combined with the ASU Host ID forms a full MB II transport socket address, which is used by APIS for returning the reply message.

A Port ID value of 0x0000 indicates that no reply must be generated by APIS for this request.

Message ID Four 16 bit unsigned value fields specifying the Message ID of a demanded message previously registered. Wildcard values (0x0000) are allowed in any of the Message ID fields. A Message ID containing wildcard fields will only remove a demand if it was demanded with this exact Message ID. In other words issuing an APIS_REMOVE_DEMAND with a wildcard Message ID will not remove demands that are subsets of the specified Message ID. The special case of using a Message ID of 0.0.0.0 however, will remove all demands previously registered for this ASAP.

4.7.2.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Command ID	Reserved Field	LSB	MSB	LSB	MSB	LSB	MSB
		ASAP		ASU Host ID		Status Port ID	

Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB
Message ID 0		Message ID 1		Message ID 2		Message ID 3	
Message ID							

4.7.3 Unsolicited Reply

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x86	Identifies message
Reserved	8 bits	0x00	
Return Value	16 bits	valid return values	Refer to list of return values.

4.7.3.1 Parameters

Command ID The 8 bit unsigned value **0x86** identifying this protocol data unit (PDU) as an APIS_REMOVE_DEMAND_REPLY command.

4.8 APIS_SEND_MSG()

4.8.1 Message Description

This message is used to send a stream data message from an application. It is a Multibus II TP Transaction Message of **type solicited request with unsolicited reply**. The solicited request will come from the ASU and the unsolicited reply from APIS.

This message is passed through the APIS stream data socket (Producer Socket) as returned with the APIS_PRODUCE() command.

4.8.2 Solicited Message

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x07	Identifies message
Reserved	8 bits	0x00	
Reserved	16 bits	0x0000	
ASU Host ID	16 bits	0x0000-0x13	APIS will combine this ASU Host ID with Status Port ID to form the socket address.
Status Port ID	16 bits	0x0000-0xFFFF	APIS will use this Port ID with the ASU Host ID to form the Status Socket.
Message ID	64 bits	0x0001:0x0001: 0x0001:0x0001 - 0xFFFF:0xFFFF: 0xFFFF:0xFFFF	Message identifier
ASAP	16 bits	0x0000-0xFFFF	Identifies sender.
Message Length	16 bits	1-4000	Length (decimal) of message to be transmitted.
Message Content	8 bits * Message Length	0x00-0xFF	Data

4.8.2.1 Parameters

Command ID The 8 bit unsigned value **0x07** identifying this protocol data unit (PDU) as an APIS_SEND_MSG command.

ASU Host ID A 16 bit unsigned value identifying the ASU Multibus II agent host.

Status Port ID A 16 bit unsigned value identifying the ASU status software endpoint within a host. This Port ID, combined with the ASU Host ID forms a full

MB II transport socket address, which is used by APIS for returning the reply message.

A Port ID value of 0x0000 indicates that no reply must be generated by the APIS for this request.

Message ID Four 16 bit unsigned value fields specifying the Message ID of a message to be transferred. Wildcard values (0x0000) are not allowed in any of the Message ID fields.

ASAP A 16 bit unsigned (APIS network wide globally unique) handle identifying the ASU ASAP producing the specified Message.

Message Length A 16 bit value specifying the length of the data message in bytes. The maximum size of the message data is 4000 (decimal) bytes.

Message Content A contiguous number of bytes as specified by the Message Length. Sequencing of data will be guaranteed.

4.8.2.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Command ID	Reserved Field	Reserved Field		LSB	MSB	LSB	MSB
				ASU Host ID		Status Port ID	

Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB
Message ID 0		Message ID 1		Message ID 2		Message ID 3	
Message ID							

Byte 16	Byte 17	Byte 18	Byte 19	Byte 20			Byte N
LSB	MSB	LSB	MSB	First Byte of Data			Last Byte of Data
ASAP		Message Length					

4.8.3 Unsolicited Reply

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x87	Identifies message
Reserved	8 bits	0x00	
Return Value	16 bits	valid return values	Refer to list of return values.
Reserved	32 bits	0x00000000	
Message ID	64 bits	0x0000:0x0000: 0x0000:0x0000 - 0xFFFF:0xFFFF: 0xFFFF:0xFFFF	Message Identifier

4.8.3.1 Parameters

Command ID The 8 bit unsigned value **0x87** identifying this protocol data unit (PDU) as an APIS_SEND_MSG_REPLY command.

Return Value APIS_SEND_MSG returns a 16-bit signed value representing the status of the call. Allowed values are listed below. (Refer to paragraph 6. Return Values.)

NO_ERROR	no error occurred
E_ASAP_NOT_OPEN	specified ASAP not locally registered
E_MSG_ID_INVALID	specified Msg ID is invalid
E_ASAP_INVALID	specified ASAP is not registered as a producer for this Msg ID
E_LENGTH_INVALID	specified Length is invalid
E_NO_CONSUMER	no consumers are registered for this Msg ID

4.8.3.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Command ID	Reserved Field	LSB	MSB	Reserved Field			
		Return Value					

Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB
Message ID 0		Message ID 1		Message ID 2		Message ID 3	
Message ID							

4.9 APIS_RECEIVE_MSG()

4.9.1 Message Description

This message is used to send a stream data message received by APIS to the application data stream socket. It is a Multibus II TP Transaction Message of type **solicited message**. The solicited message will come from APIS. No reply is required for this data transfer event.

This message is passed through the stream data socket (Consumer Socket) as specified in the APIS_DEMAND() command.

4.9.2 Solicited Message

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x08	Identifies message
Reserved	8 bits	0x00	
Reserved	16 bits	0x0000	
Reserved	32 bits	0x0000	
Message ID	64 bits	0x0000:0x0000: 0x0000:0x0000 - 0xFFFF:0xFFFF: 0xFFFF:0xFFFF	Message identifier
ASAP	16 bits	0x0000-0xFFFF	Identifies source
Message Length	16 bits	1-4000	Length(decimal) of message in bytes
Message Content	8 bits * message length	0x00-0xFF	Data

4.9.2.1 Parameters

Command ID — The 8 bit unsigned value **0x08** identifying this protocol data unit (PDU) as an APIS_RECEIVE_MSG command.

Message ID — Four 16 bit unsigned value fields specifying the Message ID of a message being transferred. No Wildcard values (0x0000) will be used in any of the Message ID fields.

ASAP — A 16 bit unsigned (APIS network wide globally unique) handle identifying the ASU ASAP that is the producer of the specified Message.

Message Length — A 16 bit value specifying the length of the data message in bytes. The maximum size of the message data is 4000 (decimal) bytes.

Message Content A contiguous number of bytes as specified by the Message Length. Sequencing of data will be guaranteed.

4.9.2.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Command ID	Reserved Field	Reserved Field		Reserved Field			

Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB
Message ID 0		Message ID 1		Message ID 2		Message ID 3	
Message ID							

Byte 16	Byte 17	Byte 18	Byte 19	Byte 20			Byte N
LSB	MSB	LSB	MSB	First Byte of Data			Last Byte of Data
ASAP		Message Length					

5. PC-Compatible (VxWorks) APIS Implementation

This section describes the interface between the ASU and APIS when both run as separate tasks under VxWorks, sharing the PC CPU processing resources.

The PC-Compatible interface between the ASU and APIS is essentially the same as for the MULTIBUS II implementation. The parameter descriptions differ in that VxWorks message queues will be used to pass messages, instead of MULTIBUS II sockets. Message queue ID's will be passed in place of the Host ID and Port ID socket fields, in messages between the ASU and APIS. The combination of Host ID and Port ID fields is 32 bits wide, which is the same size as a VxWorks message queue ID.

The APIS Command Socket will be identified by a global variable containing the message queue ID which simulates this socket. The APIS Command Socket queue can only be identified at run-time, since message queue ID's cannot be set to specific values, unlike MULTIBUS II mailboxes, which can be attached to fixed port numbers. The name of the APIS Command Socket queue global variable is apis_cmd_socket.

An example of the APIS_DEMAND() message is shown.

5.1 APIS_DEMAND()

5.1.1 Message Description

This message is used to add a message for reception to the message list. The request will come from the ASU and the reply from APIS.

This message is passed through the APIS Command Socket, which will be identified by a global variable containing the message queue ID. The name of the APIS Command Socket queue global variable is apis_cmd_socket.

5.1.2 Unsolicited Request

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x04	Identifies message
Reserved	8 bits	0x00	
ASAP	16 bits	0x0000-0xFFFF	Identifies receiver
Status Queue ID	32 bits	0x00000000 - 0xFFFFFFFF	This Queue ID will identify the Status socket.
Message ID	64 bits	0x0000:0x0000: 0x0000:0x0000 - 0xFFFF:0xFFFF: 0xFFFF:0xFFFF	Message ID of message to be received.
Consumer Queue ID	32 bits	0x00000000 - 0xFFFFFFFF	APIS will transfer any received data for the ASU to the Consumer Socket as identified by the Consumer Queue ID.
Repetition Interval	16 bits	0x0000-0xFFFF	Minimum repetition interval for message delivery (ms).

5.1.2.1 Parameters

Command ID The 8 bit unsigned value **0x04** identifying this protocol data unit (PDU) as an APIS_DEMAND command.

ASAP A 16 bit unsigned (APIS network wide globally unique) handle identifying the ASU ASAP that is the consumer of this Message ID.

Status Queue ID A 32 bit unsigned value identifying the ASU status software endpoint within a host. This Queue ID is used by APIS for returning the reply message.

A Queue ID value of 0x0 indicates that no reply must be generated by APIS for this request.

Message ID Four 16 bit unsigned value fields specifying the Message ID of a message to be consumed. Wildcard values (0x0000) are allowed in any of the Message ID fields.

Consumer Queue ID A 32 bit unsigned value identifying the ASU consumer software endpoint within the ASU host. This Queue ID is used by APIS for delivering the requested message.

Repetition Interval Specifies the minimum delivery repetition interval in ms. APIS will deliver only the last received data message after expiry of the specified minimum time interval since last delivery. Thus, messages received by APIS during the indicated time interval after the last delivery, will not be passed on to the consumer.

Value Meaning

0	Immediate delivery of messages on reception by APIS.
1-65535	Specifies immediate delivery of latest message to ASU if this specified time interval since last delivery to the ASU has expired.

5.1.2.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Command ID	Reserved Field	LSB	MSB	LSB			MSB
		ASAP		Status Queue ID			

Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB
Message ID 0		Message ID 1		Message ID 2		Message ID 3	
Message ID							

Byte 16	Byte 17	Byte 18	Byte 19	Byte 20	Byte 21
LSB			MSB	LSB	MSB
Consumer Queue ID				Repetition Interval	

5.1.3 Unsolicited Reply

Parameters	Size	Allowed Values	Description
Command ID	8 bits	0x84	Identifies message
Reserved	8 bits	0x00	
Return Value	16 bits	valid return values	Refer to a list of the return values

5.1.3.1 Parameters

Command ID The 8 bit unsigned value **0x84** identifying this protocol data unit (PDU) as an APIS_DEMAND_REPLY command.

Return Value APIS_DEMAND returns a 16-bit signed value representing the status of the call. Allowed values are listed below. (Refer to paragraph 6. Return Values.)

NO_ERROR	no error occurred
E_ASAP_NOT_OPEN	specified ASAP not locally registered
E_MSG_ID_INVALID	specified Msg ID is invalid
E_INTERVAL_INVALID	specified Repetition Interval is invalid
E_CONS_SKT_INVALID	specified socket is invalid
E_DEMAND_EXIST_FOR_ASAP	specified Msg ID is already registered for this ASAP

5.1.3.2 Parameter Offsets

Byte 0	Byte 1	Byte 2	Byte 3
Command ID	Reserved Field	LSB	MSB
		Return Value	

6. Returned Values

Return Value	Decimal Value	Status Description
NO_ERROR	0	no error occurred
E_ASAP_NOT_OPEN	-101	specified ASAP not locally registered
E_APID_IN_USE	-102	specified Application ID has already been used on this NIC
E_MSG_ID_INVALID	-201	specified Msg ID is invalid
E_STRING_INVALID	-202	specified ASU TEXT STRING is invalid
E_LENGTH_INVALID	-203	specified Length is invalid
E_PRIORITY_INVALID	-204	specified Priority is invalid
E_INTERVAL_INVALID	-205	specified Repetition Interval is invalid
E_CONS_SKT_INVALID	-206	specified MBII socket is invalid
E_PROD_EXIST_FOR_ASAP	-301	specified Msg ID is already registered for this Priority and ASAP
E_PRODUCER_LIMIT	-302	maximum number of producers has registered this Msg ID
E_PROD_BW_CHANGE	-303	specified Length and Repetition Interval values do not correspond with previously registered values. (Only valid if specified Msg ID is already registered under a different ASAP).
E_BW_LIMIT	-304	maximum bandwidth allocation limit exceeded
E_DEMAND_EXIST_FOR_ASAP	-401	specified Msg ID is already registered for this ASAP
E_MSG_NOT_REGISTERED	-501	specified Msg ID is not registered for this ASAP
E_ASAP_INVALID	-601	specified ASAP is not registered as a Producer for this Msg ID
E_NO_CONSUMER	-602	no consumers are registered for this Msg ID

APPENDIX D APIS SOFTWARE DESIGN DOCUMENT

CCII Systems (Pty) Ltd Registration No. 90/05058/07

COMMUNICATIONS
COMPUTER INTELLIGENCE
INTEGRATION

Software Design Document

for the

Information Management System

Application Interface Services

	C²I² Systems	External
Document No.	CCII/A500/IMS/6-SWDD/1	
Doc Issue	0.3	
Issue Date	1999-04-28	
Print Date	2001-01-04	
File Name	E:\TEMP\CDIASD01.W70	
Copy No.		

© C²I² Systems. The copyright of this document is the property of C²I² Systems. The document is issued for the sole purpose for which it is supplied, on the express terms that it may not be copied in whole or part, used by or disclosed to others except as authorised in writing by C²I² Systems.

Document prepared by C²I² Systems (Pty) Ltd, Cape Town

TABLE OF CONTENTS

1. Scope 1
   1.1 Identification 1
   1.2 System overview 1
   1.3 Document overview 1
2. Referenced documents 2
   2.1 Standards 2
   2.2 Navy Documents 2
   2.3 Program Documents 2
   2.4 Project Documents 2
3. CSCI design decisions 3
4. CSCI architectural design 3
   4.1 CSCI components 4
      4.1.1 Send Task 4
      4.1.2 Receive Task 5
      4.1.3 Admin Task 6
      4.1.4 List of Demanded Messages 7
      4.1.5 List of Produces Messages 8
      4.1.6 List of Application Service Users 9
   4.2 Concept of execution 9
   4.3 Interface design 10
      4.3.1 Interface identification and diagrams 10
      4.3.2 XTP Interface 10
      4.3.3 Multibus II Interface 10
      4.3.4 Message Queue Interface 10
      4.3.5 Fast Ethernet Interface 10
      4.3.6 Synchronous Bandwidth Requester Interface 11
5. CSCI detailed design 12
   5.1 APIS INIT 12
   5.2 APIS OPEN 13
   5.3 APIS CLOSE 14
   5.4 APIS PRODUCE 15
   5.5 APIS DEMAND 16
   5.6 APIS REMOVE PRODUCE 16
   5.7 APIS REMOVE DEMAND 18
   5.8 APIS SEND 19
   5.9 APIS RECEIVE 20
   5.10 PDU event - Ready To Produce 21
   5.11 PDU event - Ready To Connect 22
   5.12 PDU event - Broadcast New Produce 23
   5.13 PDU event - Broadcast New Demand 24
6. Notes 25
Appendix A 26

ABBREVIATIONS AND ACRONYMS

ACDM	Architecture Concept Demonstration Model
ANSI	American National Standards Institute
APIS	Application Interface Services
ASCII	American Standard Code for Information Interchange
BIT	Built-In Test
BITS	Built In Test Services
CS	Combat Suite
DID	Data Item Description
FDDI	Fiber Distributed Data Interface
FOM	Figure Of Merit
FTS	File Transfer Services
ICD	Interface Control Document
ISO	International Standards Organisation
IMS	Information Management System
IT	Information Technology
LRU	Line Replaceable Unit
LAN	Local Area Network
LSS	Lightweight Support Services
MIPS	Mega-Instruction Per Second
MOM	Message Orientated Middleware
MMI	Man-Machine Interface
NIC	Network Interface Card
NTS	Network Time Services
OS	Operating System
OSI	Open Systems Interconnect
PCB	Printed Circuit Board
PFM	Platform
PC	Personal Computer
PCI	Peripheral Component Interconnect
PMC	PCI Mezzanine Card
POSIX	Portable Operating System Interface Extension
RISC	Reduced Instruction Set Computer
RAM	Random Access Memory
SCA	Standard Console Assembly
SCS	Standard Computing Segment
SMT	Station Management
SNC	Standard Naval Console
TBD	To Be Determined
UDP	User Datagram Protocol
WBS	Work Breakdown Structure
XTP	Xpress Transport Protocol

1. Scope

1.1 Identification

This document addresses the design for the Information Management System (IMS) Application Interface Services (APIS).

1.2 System overview

The sub-systems interfacing to each other via the Information Management System (IMS) shall interface to the Local Area Network (LAN) via the Network Interface Card (NIC). The NIC shall implement protocol processing, flow, rate and error control as well as local network management.

The NIC shall provide the following services to the sub-system:

- Application Interface Services (APIS) for realtime message passing.
- Network Time Services (NTS) for synchronization.
- File Transfer Services (FTS) for passing large amounts of data (files).
- Built In Test Services (BITS) for obtaining status information regarding the LAN.

APIS is Message Orientated Middleware (MOM) which abstracts the connection oriented Transport Layer from the user on the sub-system. It allows the sender of a message (known as a *Producer* in APIS) to be unaware of the receiver of the message (known as the *Consumer*). Each message that is passed through APIS has a unique message identifier that is allocated by SINC. A Producer of a specific message shall register to APIS on the its sub-system NIC with the message identifier of that message. A Consumer on another sub-system wishing to receive this message shall register to APIS on its NIC with the same message identifier. APIS establishes a virtual connection between the Producer and Consumer and delivers the message to the Consumer every time it is sent by the Producer. Multiple Producers and Consumers can be set up for a specific message in this way.

The NIC may be either a Multibus II compatible, or a PC/PCI-PMC compatible FDDI card. In the case of the Multibus II NIC interface, APIS shall run on the NIC and the user's application shall run on a separate Multibus II compatible host. The user shall interface to APIS via Multibus II messages as defined in the APIS ICD. In the case of the PC/PCI-PMC FDDI card, APIS and the user's application shall share the PC CPU processing resources. The user shall interface to APIS via message queues provided by the operating system, using messages as defined in the APIS ICD.

1.3 Document overview

This document shall discuss the overall design of the IMS APIS as well as decisions taken regarding the architecture of the software.

2. Referenced documents

2.1 Standards

a. C²I² Systems Document No. CCII/110/COMP15, *C²I² Systems Development Methodology*

2.2 Navy Documents

2.3 Program Documents

2.4 Project Documents

a. C²I² Systems Document No. CCII/A500/IMS/6-PIDS, *Prime Item Development Specification for the Patrol Corvette Combat Suite Information Management System.*

b. C²I² Systems Document No. CCII/A500/IMS/6-ICD/1, *Interface Control Document for the Information Management System Application Interface Services*

c. C²I² Systems Document No. CCII/A500/IMS/6-ICD/7, *Interface Control Document for the Information Management System Application Interface Services for the NAV NIC*

d. C²I² Systems Document No. CCII/A500/IMS/6-SRS/1, *Software Requirement Specification for the Information Management System Application Interface Services*

e. C²I² Systems Document No. CCII/A500/IMS/6-SWDD/6, *Software Design Description for the Synchronous Bandwidth Requester of the Information Management System*

3. CSCI design decisions

APIS shall execute under a realtime multitasking operating system. Interface to the user shall be via Multibus II in the case of a Multibus II NIC card or via interprocess communication mechanisms provided by the operating system ie. Message Queues (mailboxes). APIS shall gain access to the IMS FDDI LAN via a STREAMS interface to the XTP module.

4. CSCI architectural design

APIS is implemented as a collection of processes executing simultaneously to achieve maximum efficiency. Refer to **Figure 1** for an architectural overview. Two static tables exist to act as a database for all messages registered on the NIC. Three different classes of processes can operate on these tables as follows:

- **Receive Task** A process that receives a data message via the XTP protocol and dispatches it to all the users that are registered on this NIC to receive this message. The List of Demanded Messages on this NIC will have an entry for this message listing all the users that APIS should deliver this message to. Several of these Receive Task may operate simultaneously.

- **Send Task** A process that receives a data message from the user via the Multibus II or Mail Queue interface, validates the user as a registered Producer and passes it to the CS FDDI LAN via XTP. The List of Produced Messages will have an entry for this message containing the valid Producers as well as the XTP context information. Several Receive Tasks may operate simultaneously.

- **Admin Task** A process that receives control messages from the user via the Multibus II or Mail Queue interface. Control messages are used to register/deregister users as Producers for sending a particular message or as Consumers to receive a particular message. It is possible for multiple Admin Tasks to operate simultaneously although one process handling the administration is sufficient.

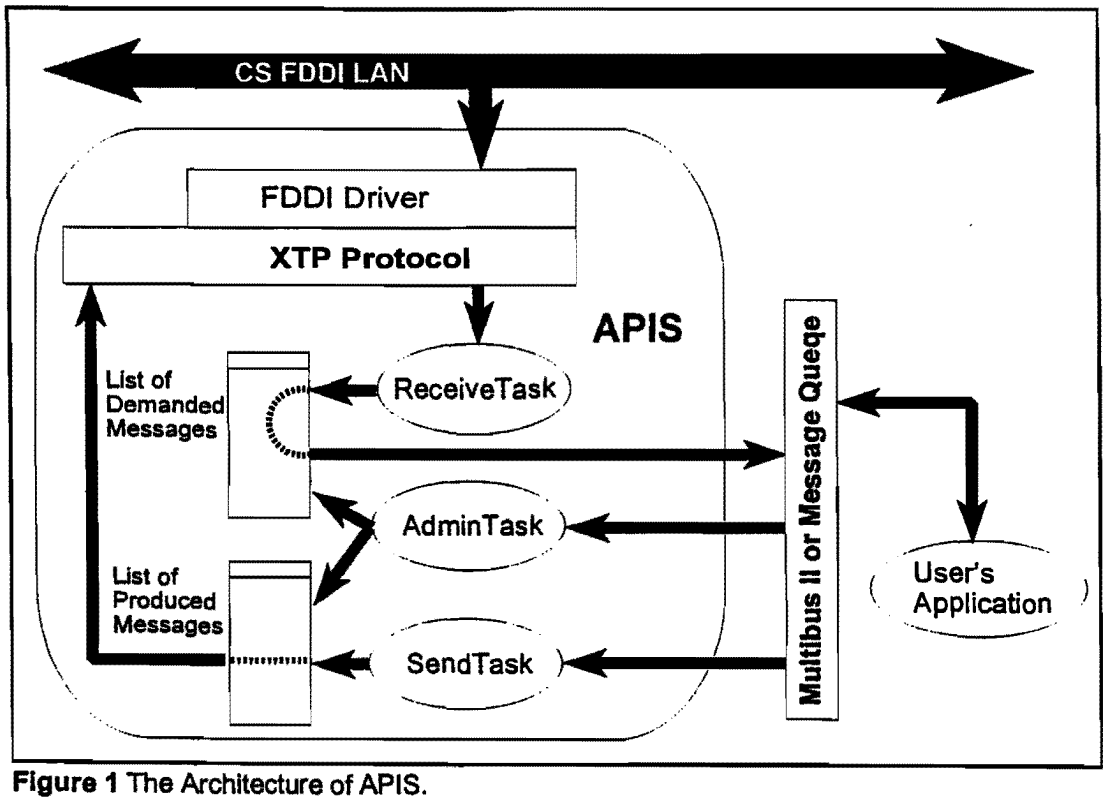


Figure 1 The Architecture of APIS.

4.1 CSCI components

4.1.1 Send Task

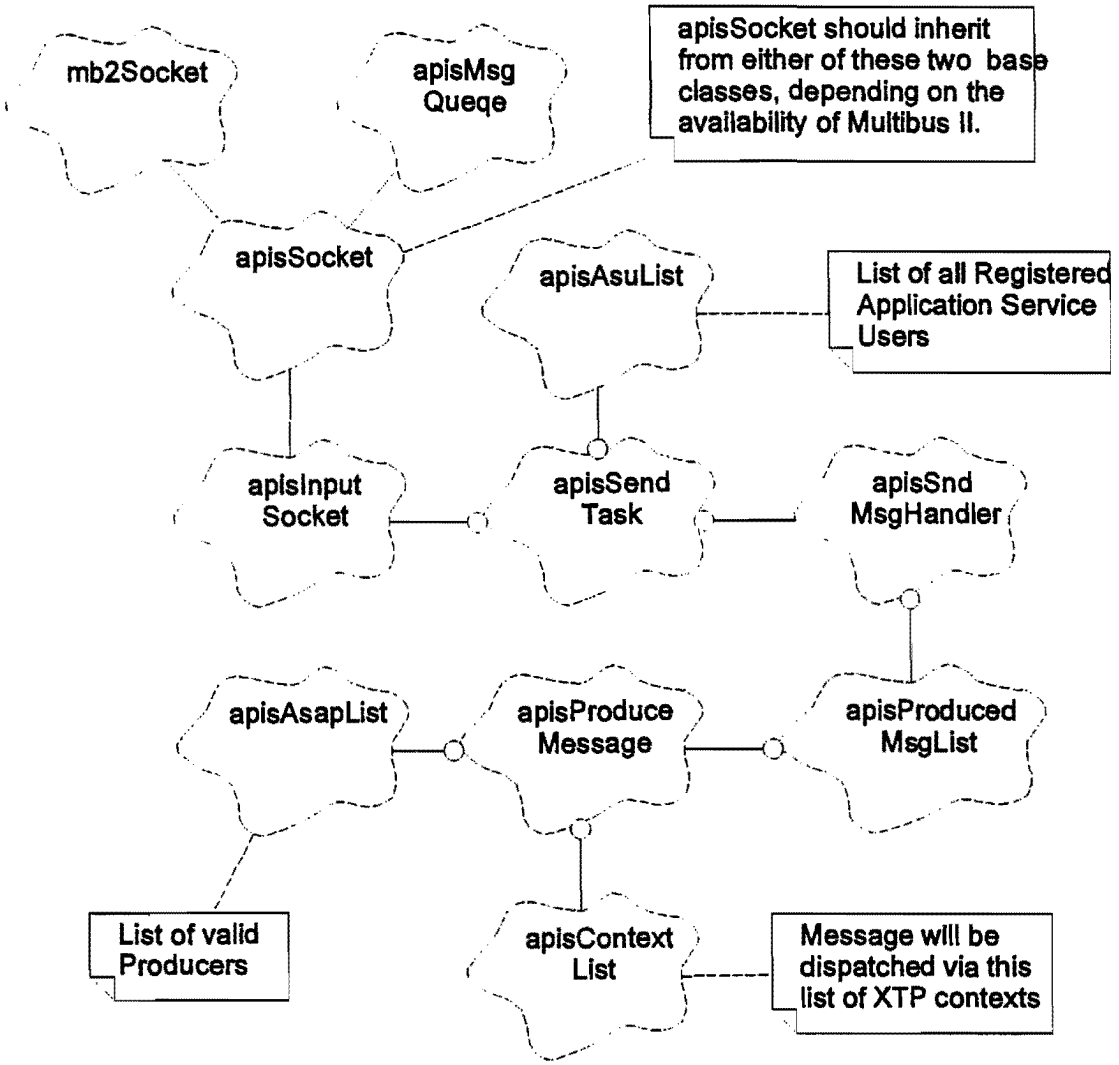


Figure 2 Architecture of the Send Task.

4.1.2 Receive Task

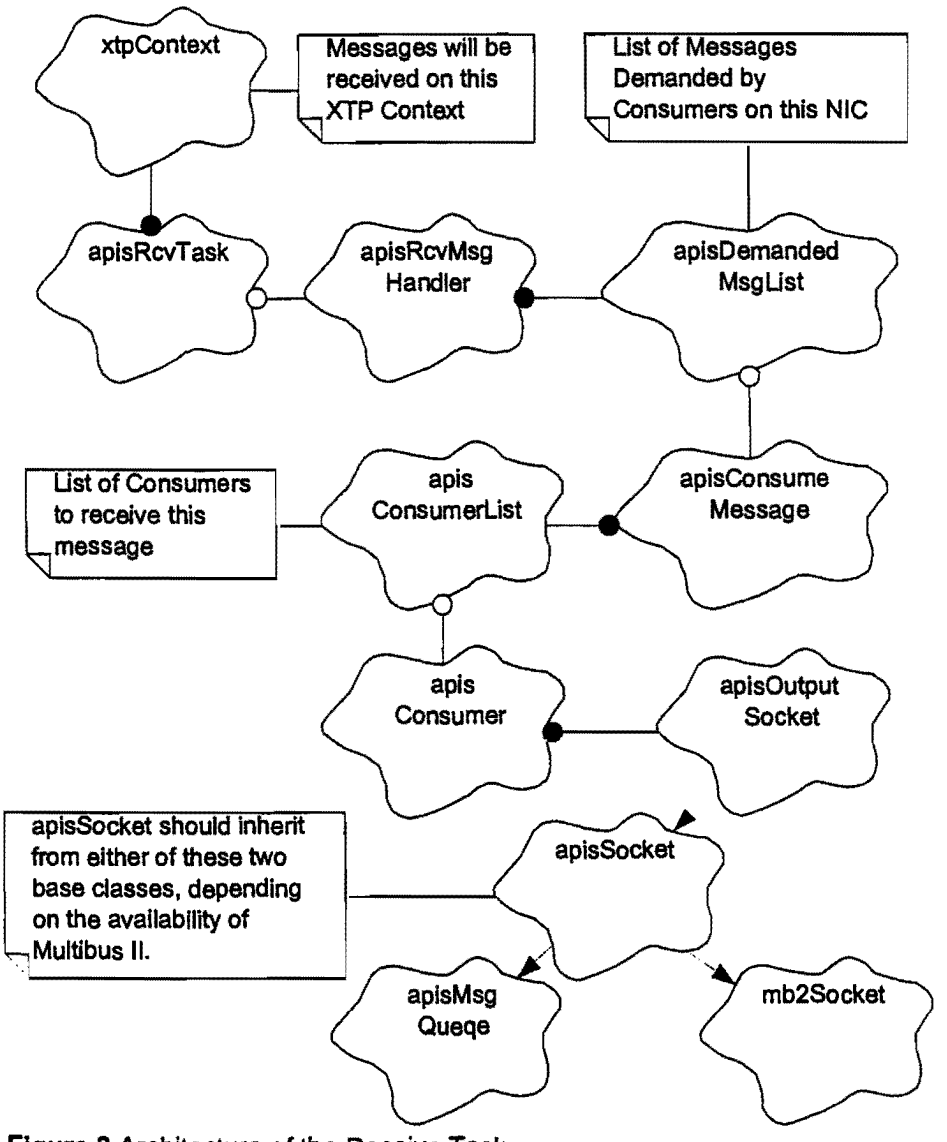


Figure 3 Architecture of the Receive Task.

4.1.3 Admin Task

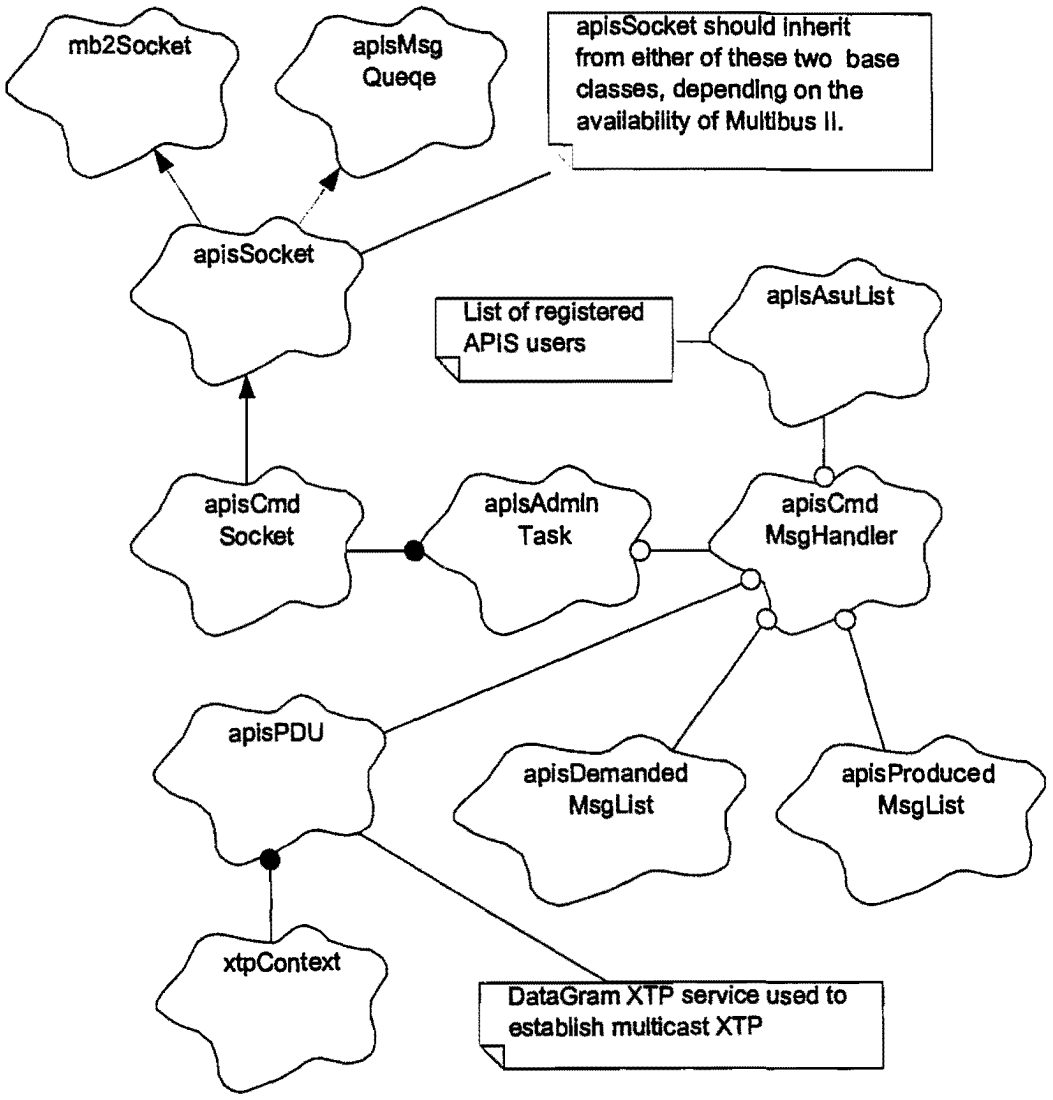


Figure 4 Architecture of the Admin Task.

4.1.4 List of Demanded Messages

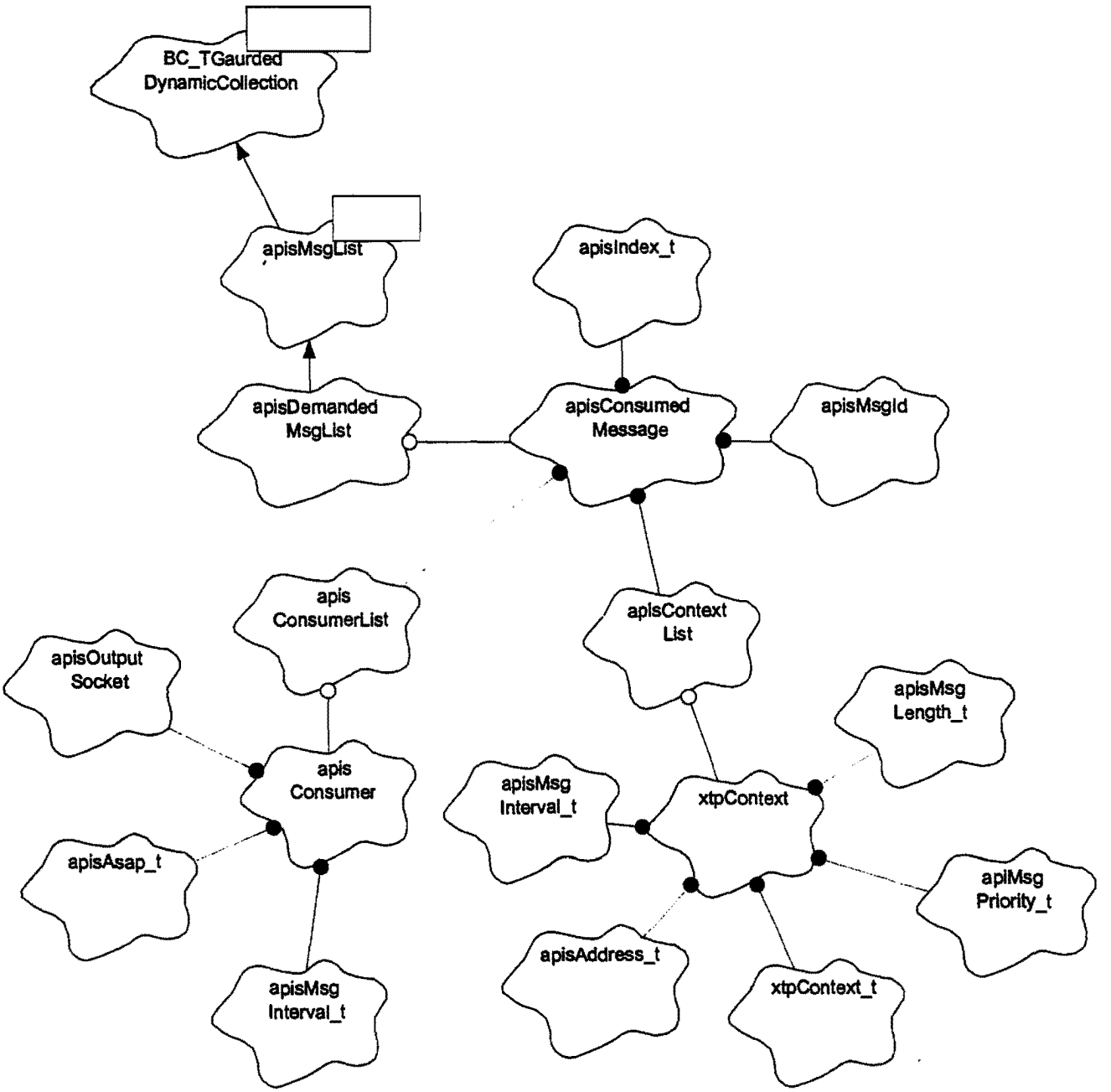


Figure 5 Architecture of the Demanded Message List.

4.1.5 List of Produces Messages

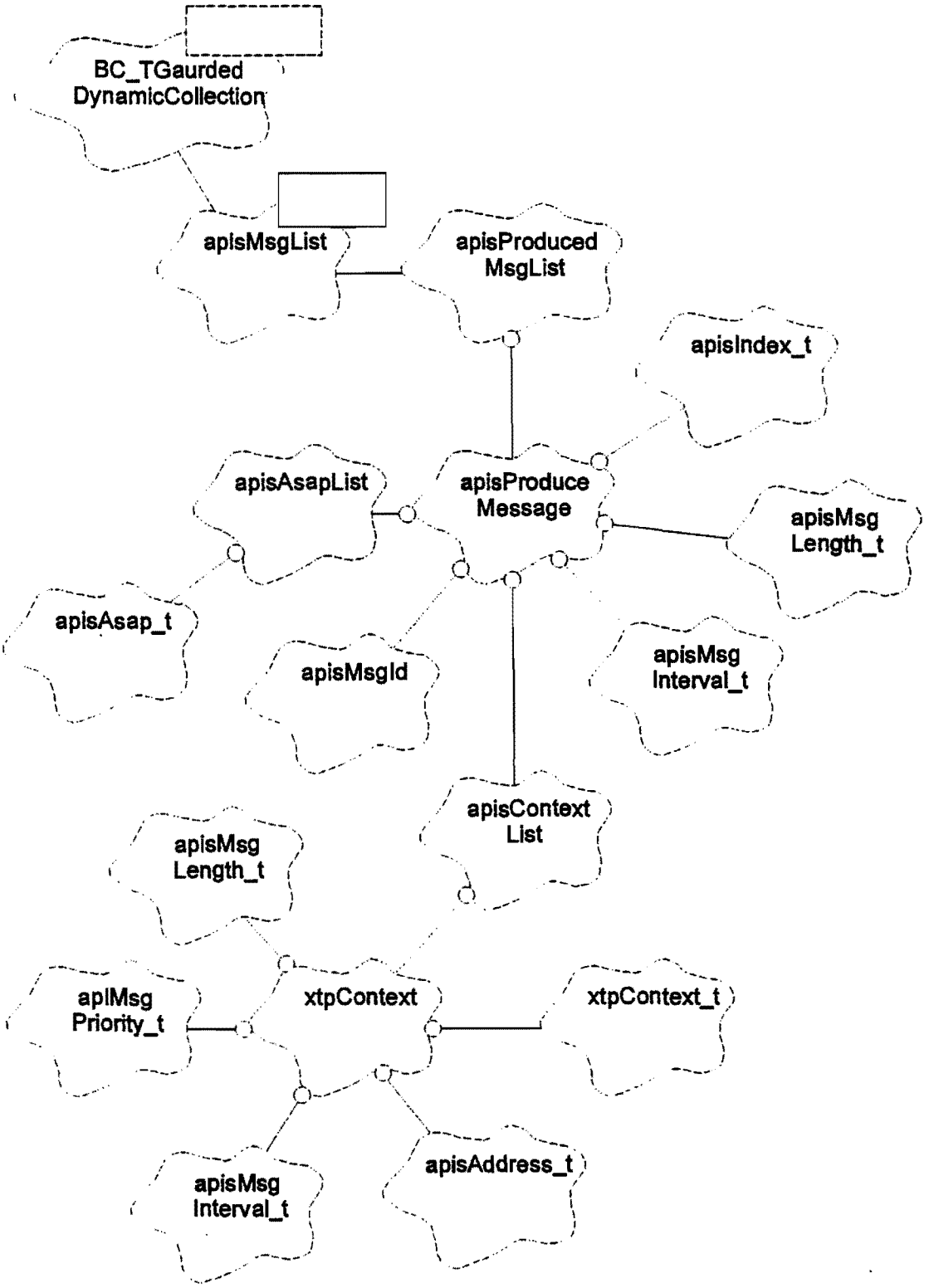


Figure 6 Architecture of the Produced Message List.

4.1.6 List of Application Service Users

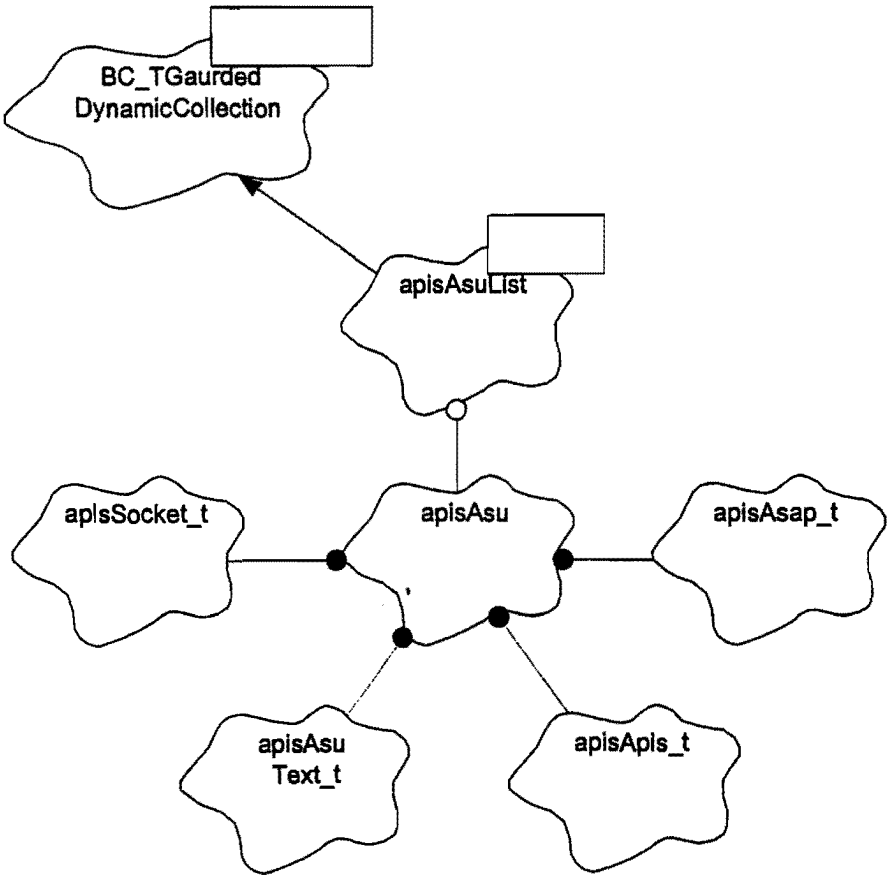


Figure 7 Architecture of the Application Services Users List.

4.2 Concept of execution

All the processes that is needed by APIS will start when the program is invoked and will run simultaneously until APIS is terminated.

Detailed process and object interaction is deferred to Appendix A which contains the CASE (Rational Rose - Booch) design of the software.

5.5 APIS DEMAND

The Admin Task receives this control message from a registered user wishing to receive a particular message.

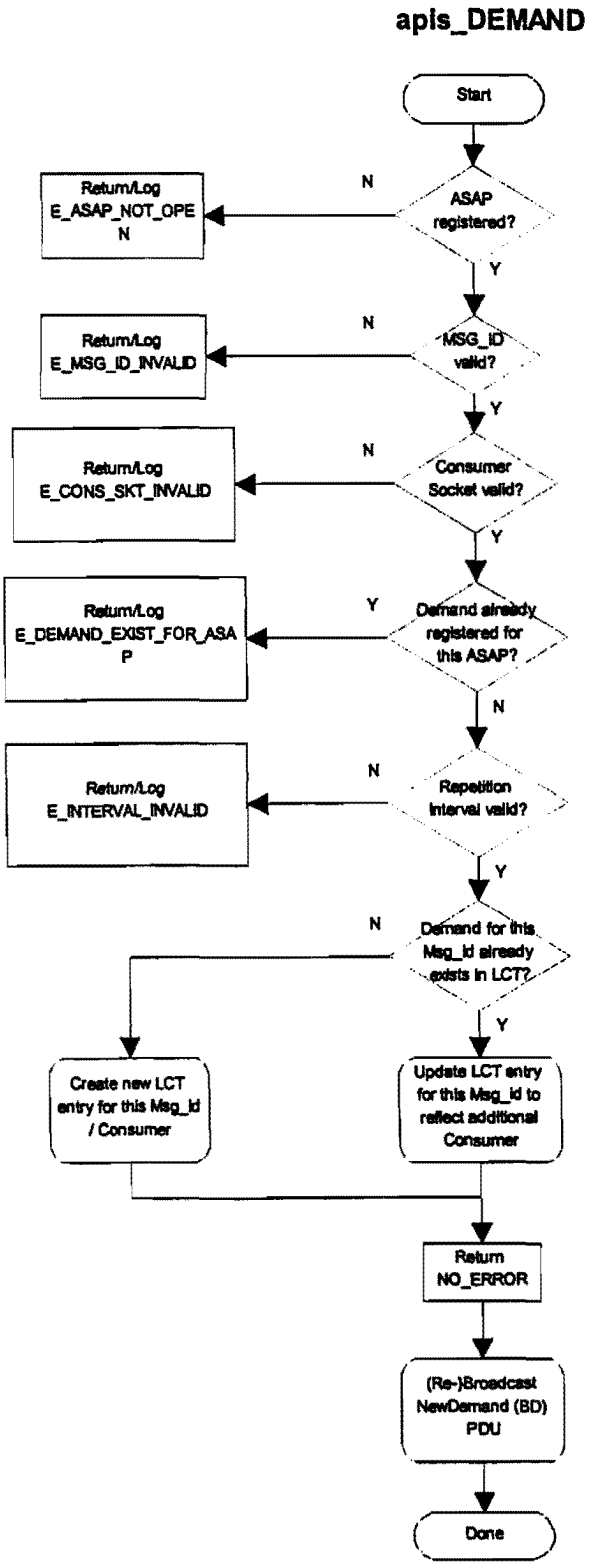


Figure 13 Add a Consumer for this message.

5.6 APIS REMOVE PRODUCE

The Admin Task receives this control message from a registered Producer who no longer wants to send a message he has registered before.

apis_REMOVE_PRODUCE

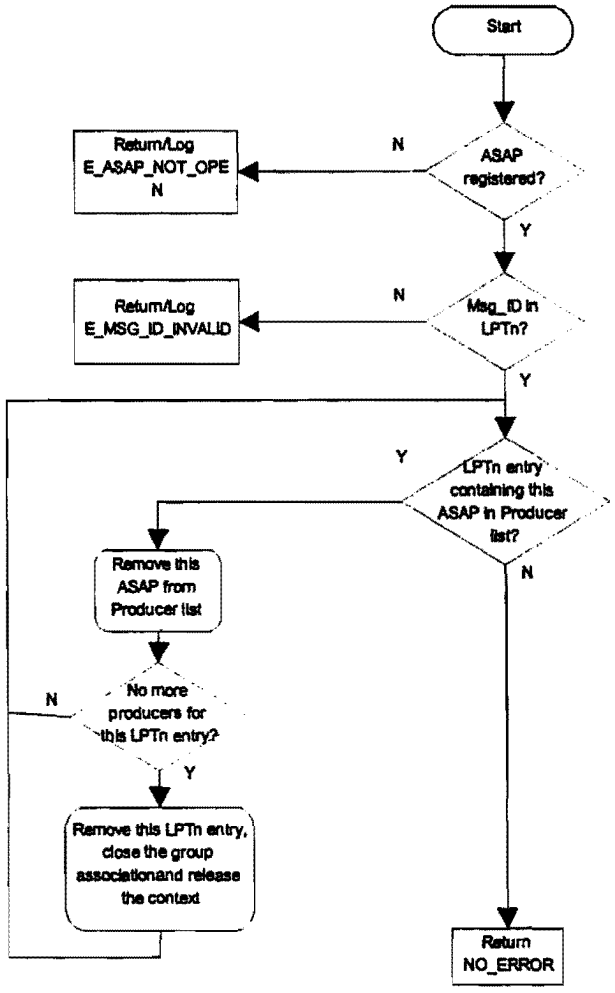


Figure 14 Remove a Producer for this message.

5.7 APIS REMOVE DEMAND

The Admin Task receives this control message from a registered Consumer who no longer wants to receive a message it has registered before.

apis_REMOVE_DEMAND

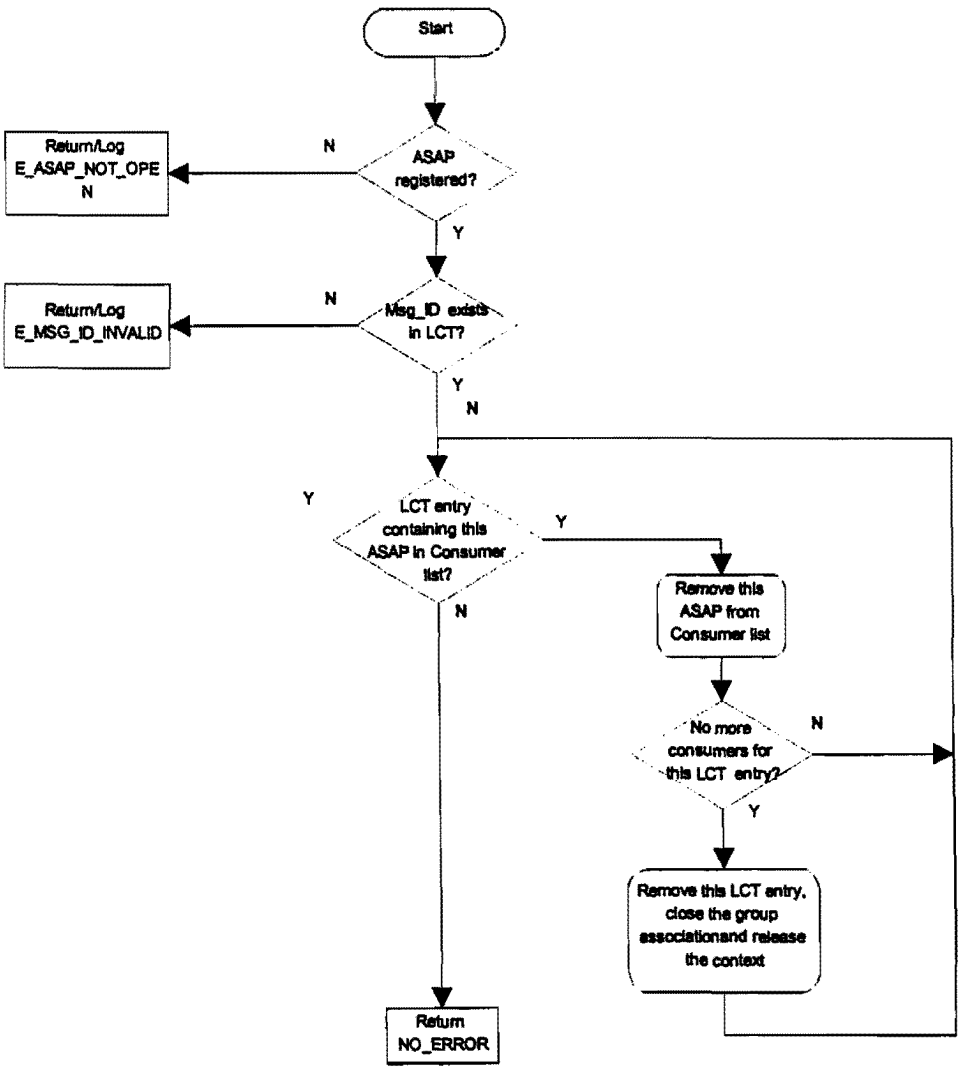


Figure 15 Remove a Consumer for this message.

5.8 APIS SEND

The Receive Task receives this data message from a registered Producer and passes it on via the XTP protocol to the CS FDDI LAN .

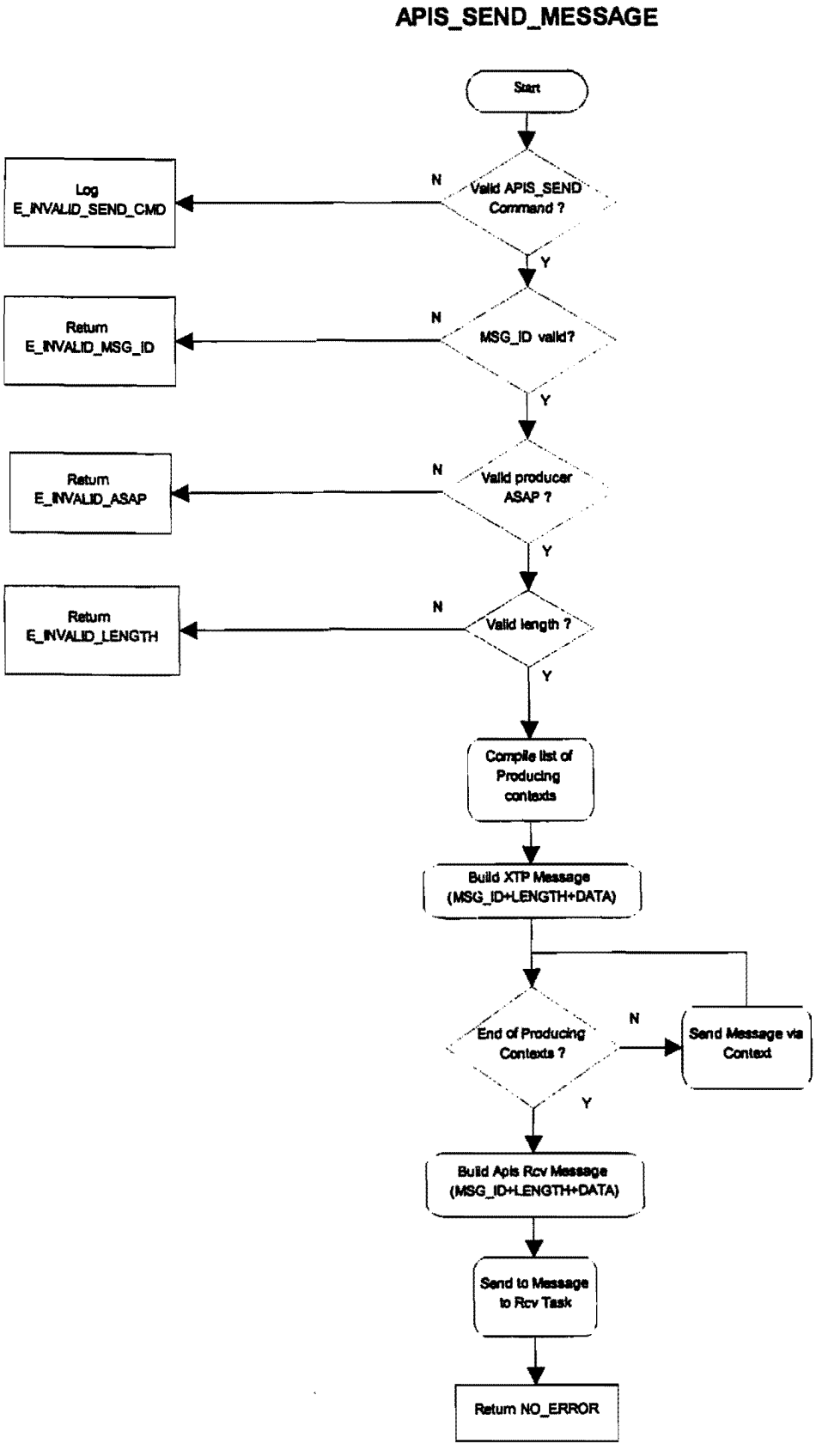


Figure 16 Producer sends a message.

5.9 APIS RECEIVE

The Receive Task receives this data message via XTP and passes it on to all the registered users that have also registered as a Consumer for this message.

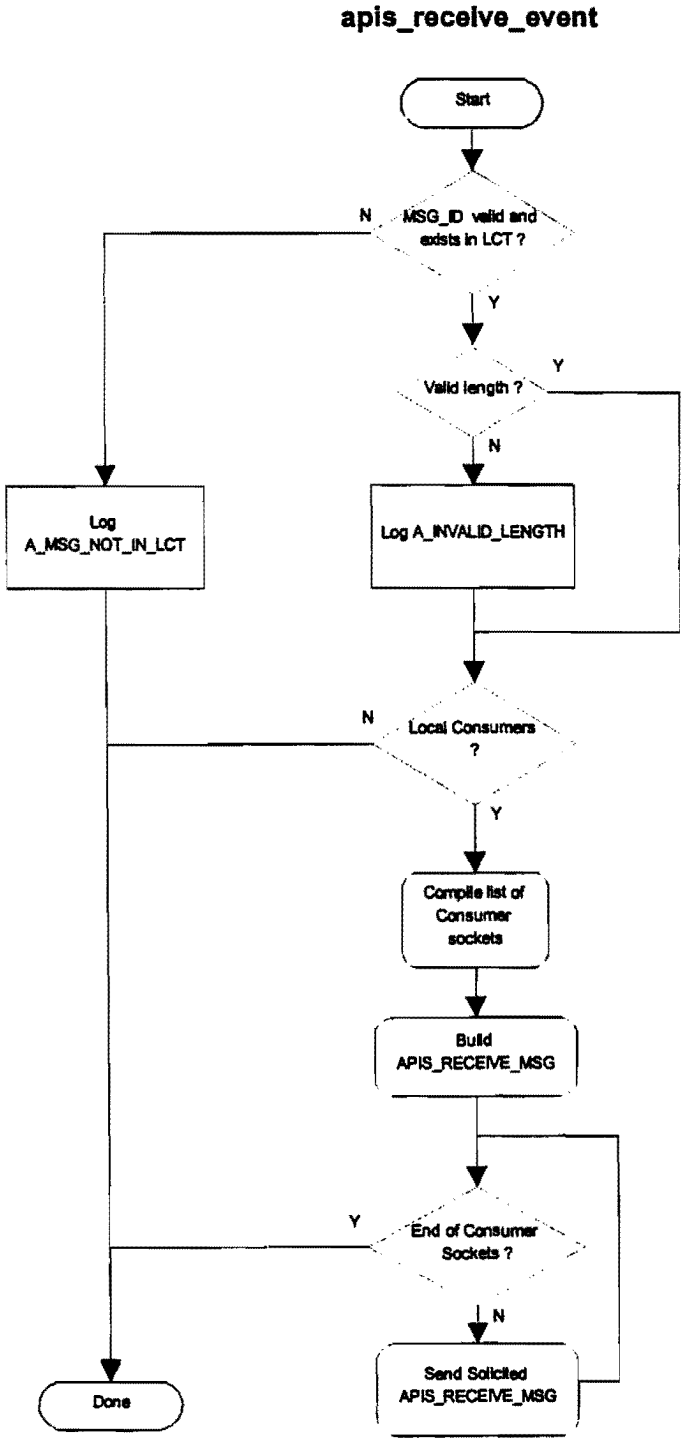


Figure 17 Consumer receives a message.

5.10 PDU event - Ready To Produce

The Admin Task receives this internal PDU message when a remote NIC is ready to establish a new multicast connection for a marticular message.

Unicast ReadyToProduce (UP PDU) event

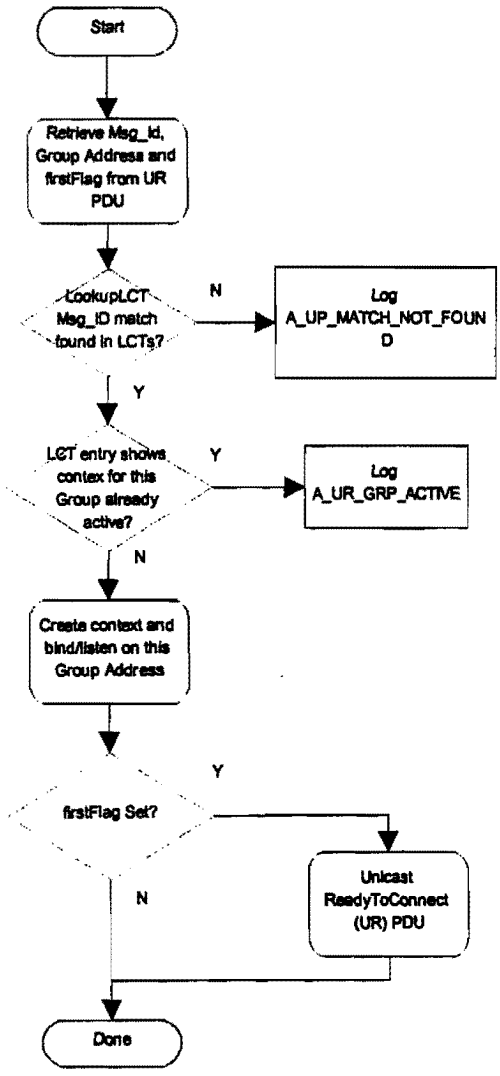


Figure 18 Ready to Produce event.

5.11 PDU event - Ready To Connect

The Admin Task receives this internal PDU message from a remote NIC when it is ready to connect to a multicast association.

Unicast ReadyToConnect (UR PDU) event

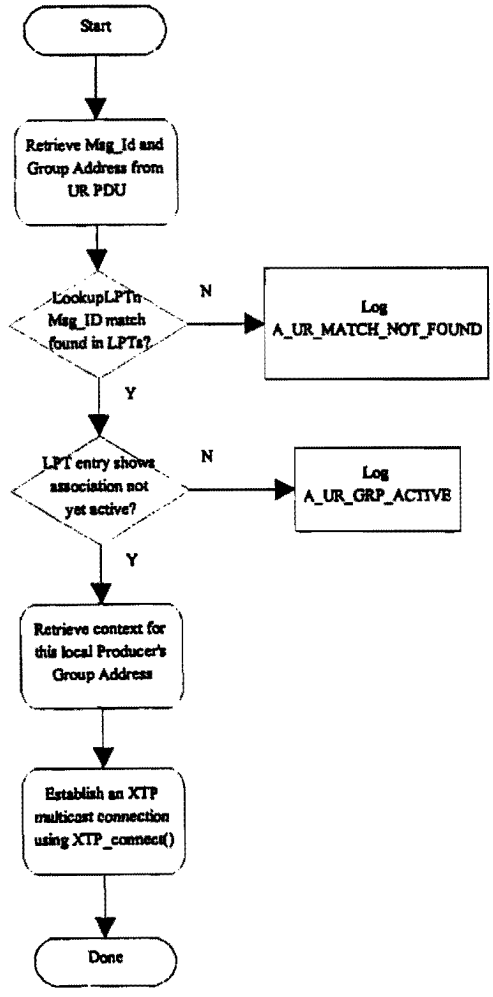


Figure 19 Ready to Connect event.

5.12 PDU event - Broadcast New Produce

The Admin Task receives this internal PDU message from a remote NIC when a new Producer for a particular message was added.

Broadcast NewProduce (BP PDU) event

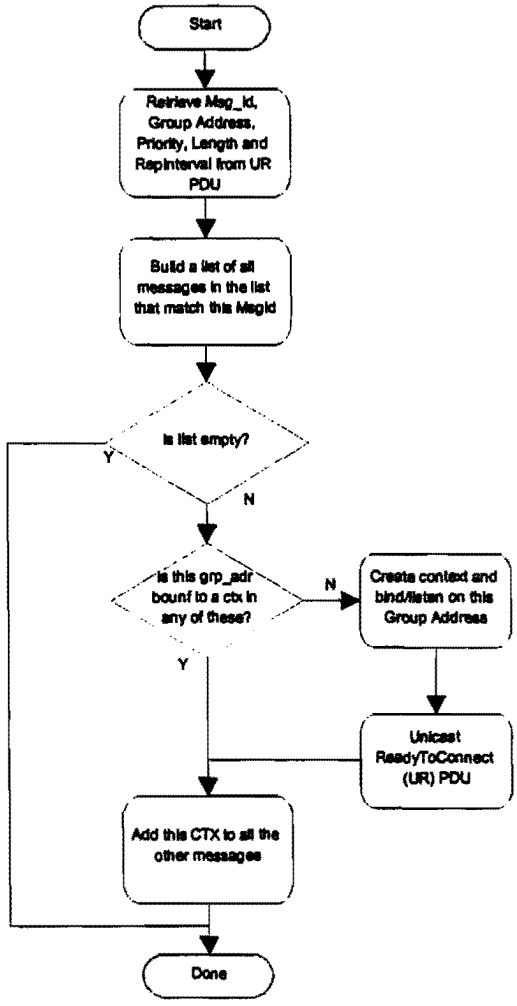


Figure 20 Broadcast New Produce event.

5.13 PDU event - Broadcast New Demand

The Admin Task receives this internal PDU message from a remote NIC when a new Consumer for a particular message was added.

Broadcast NewDemand (BD PDU) event

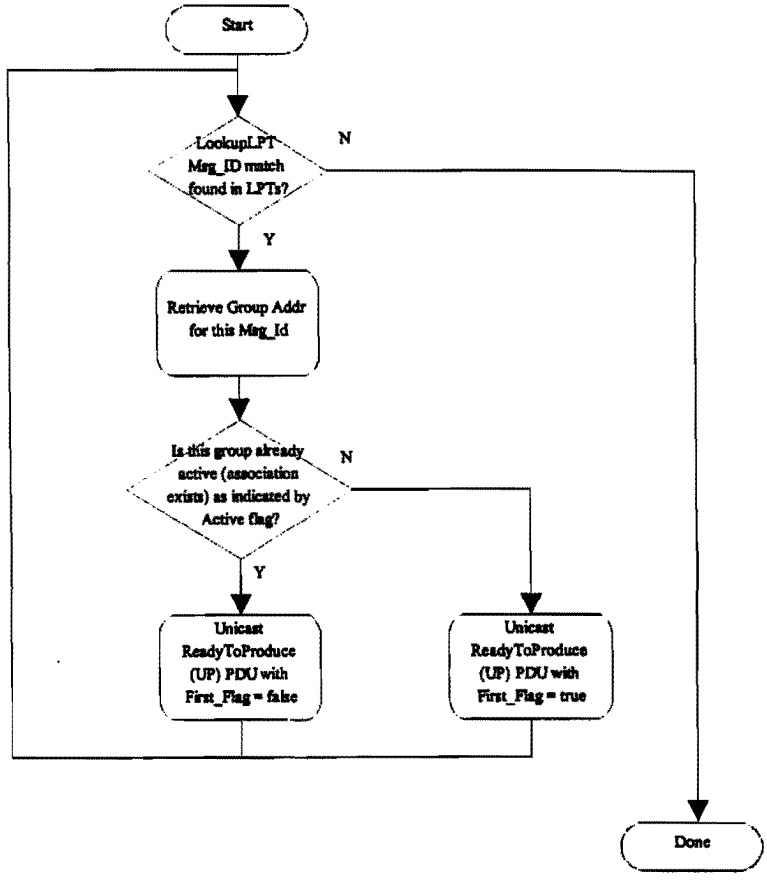


Figure 21 Broadcast New Demand event.

6. Notes

APPENDIX E APIS MESSAGE LATENCY PROFILE

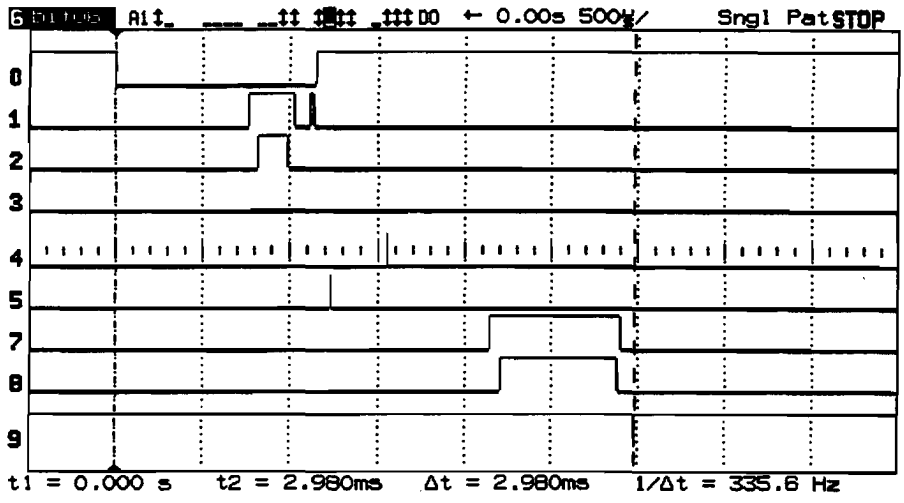


Figure E1. The application-to-application time to deliver a 4 000 byte message in APIS. (2,980 ms)

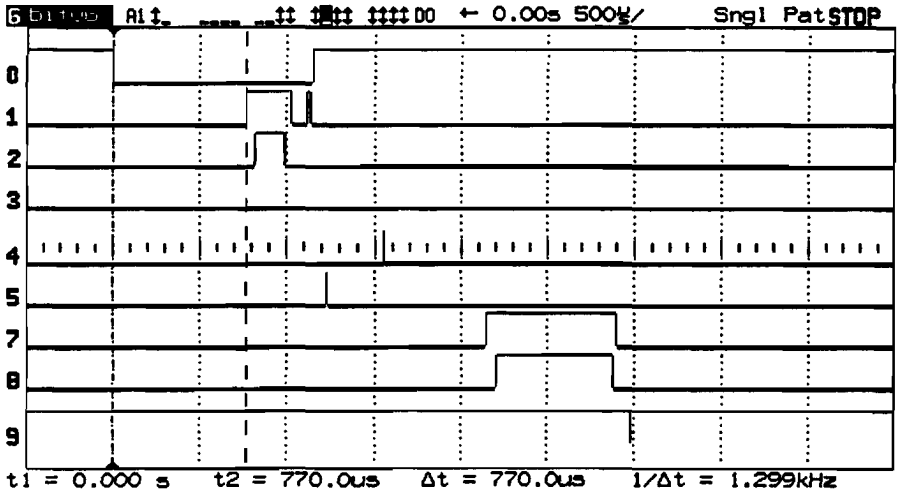


Figure E2. The time required to send a 4 000 byte message from the Producer ASU to the NIC. (770 µs or 25,8% of total)

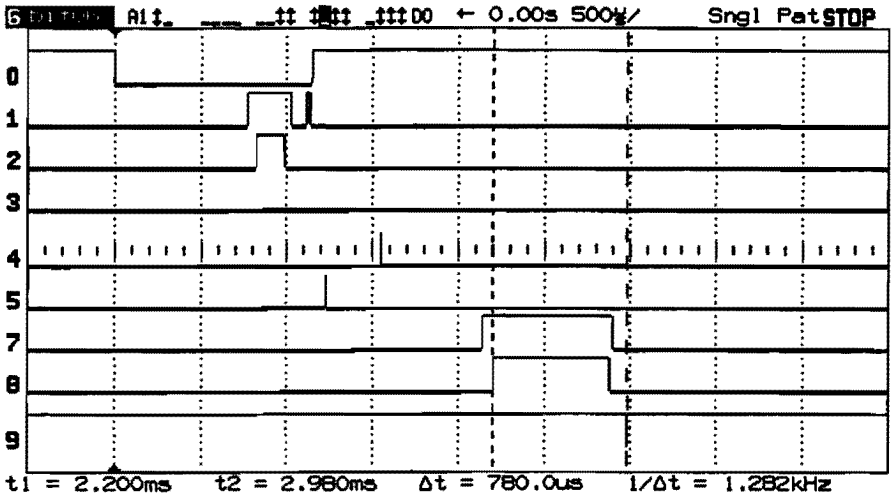


Figure E3. The time required to send a 4 000 byte message from the NIC to the Consumer ASU. (780 µs or 26,2% of total) The combined time spent in Multibus is 1,550 ms or 52% of the total.

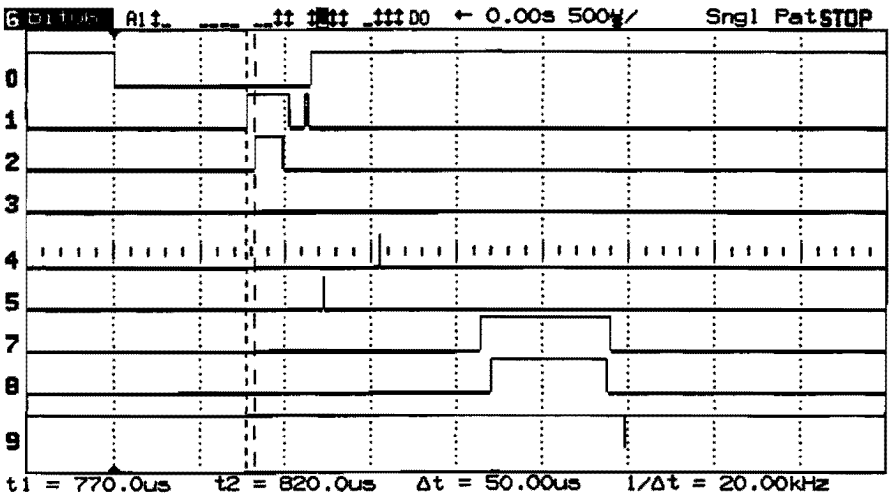


Figure E4. The delta between APIS receiving the message via Multibus and copying it to the FDDI driver is 50 µs. This time includes an indexed lookup in the data base and ASU verification.

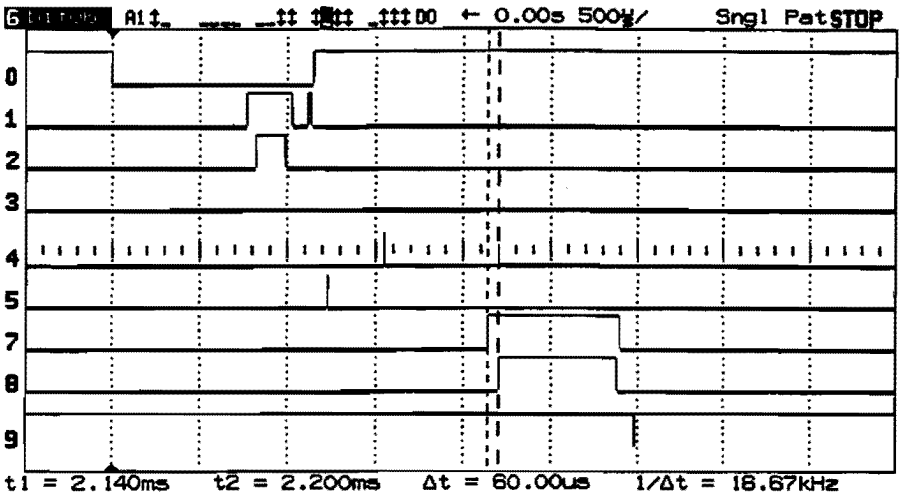


Figure E5. The delta between APIS receiving the message from the FDDI driver and initiating the Multibus transfer to the Consumer ASU is 60 μs.

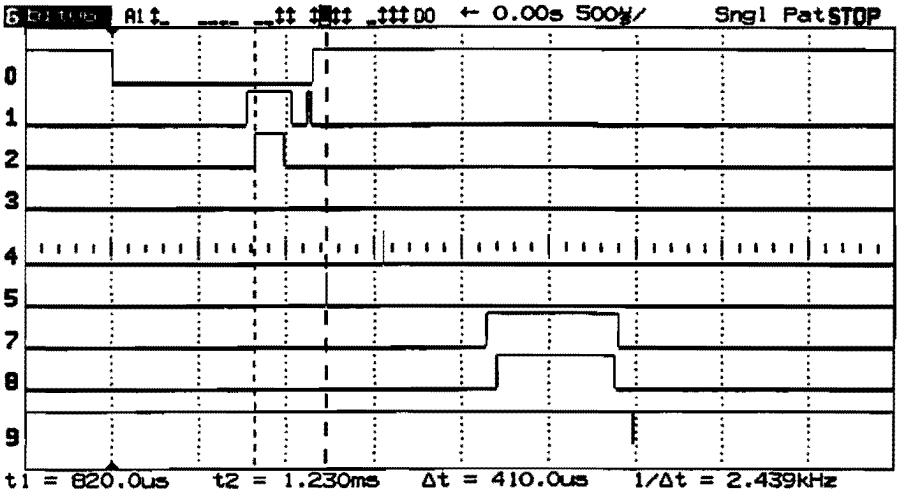


Figure E6. The time since APIS copied the data to the FDDI driver until the arrival of the token was 410 μs.

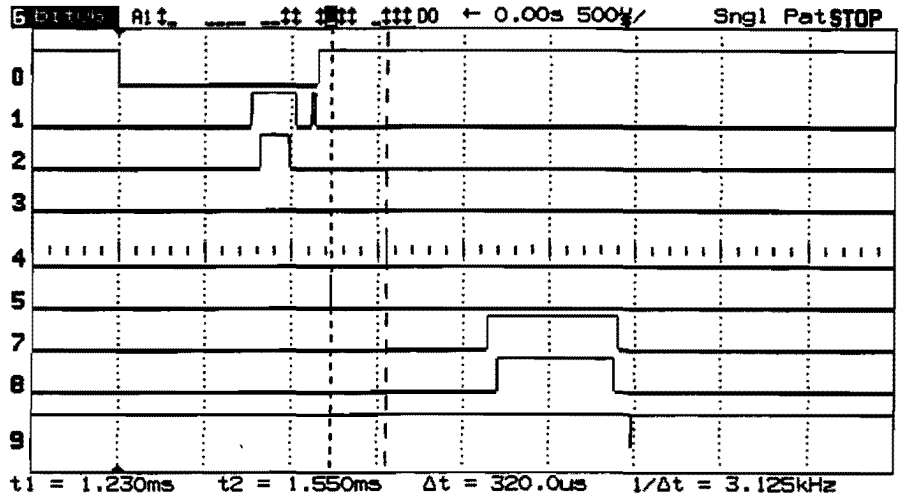


Figure E7. The time needed to transmit 4 000 bytes onto the FDDI media is 320 μs.

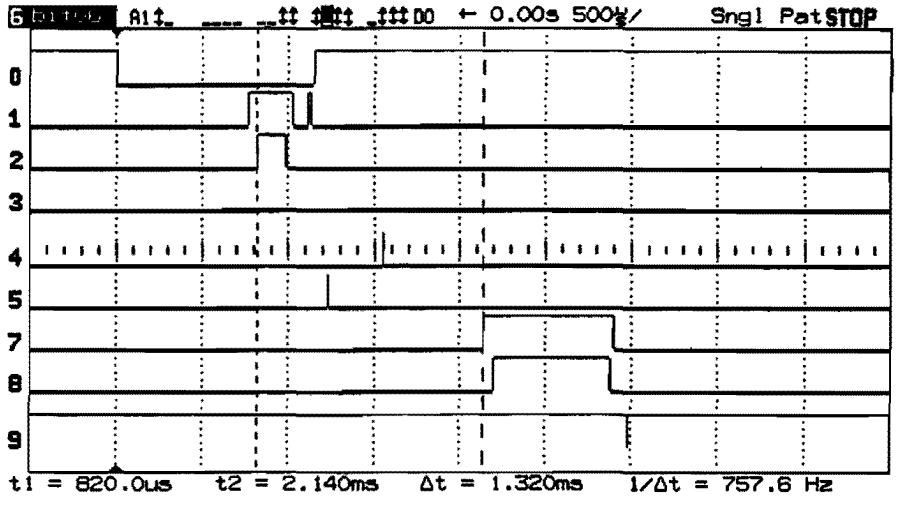


Figure E8. If we ignore the latencies of the Multibus transfers and APIS processing, the latency of the FDDI driver is only 1,32 ms or 44,3 % of the total.