An Implementation of a Real-Time Message-Oriented Middleware

Manie Steyn, C²I² Systems (Pty) Ltd, Cape Town

***Abstract*--**
This paper presents details of implementing a Real-Time Message-Oriented Middleware (MOM) using commercial of-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 implementing a real-time message distribution application. It also indicates 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.

***Index Terms*--**
Distributed Systems and Protocols, Real-Time Protocols, Message-Oriented Middleware, Application Interface Services.

I. INTRODUCTION

Interconnecting mission-critical, real-time distributed systems is a complex task in that many nodes interact with each other, thus 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.
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 of 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 [1]. The IMS realises the network backbone and network interface cards (NICs) as a sub-system responsible for timely delivery of messages and seamless integration of all sub-systems.

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

The Application Interface Services (APIS) is one of the services provided by the IMS. It uses the principles of Message-Oriented Middleware (MOM) to distribute messages timely across the network without the need for 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 uncoupling is the most important distinction between APIS and other connection-oriented middleware.

This paper sets out to establish the requirements set by the transfer of real-time distributed information on a MOM such as APIS. It then details the specific issues pertaining to the implementation of APIS and presents subsequent performance measurements.

In addition, the software designs for two generations of the APIS MOM are presented.
The first generation uses the multicast features of the Xpress Transport Protocol (XTP), a Transport Layer (OSI Layer 4) protocol, implemented on a Fibre Distributed Data Interface (FDDI) Physical Layer (OSI Layer 1). The characteristics and suitability of both XTP and FDDI in a real-time environment are discussed and an analysis is done to obtain optimal values for setting the performance parameters of FDDI.
Due to some current limitations in XTP, a second generation of APIS was implemented directly on the Data Link Layer (OSI Layer 2) and the FDDI Physical Layer (OSI Layer 1). The advanced features of FDDI

such as synchronous bandwidth allocation and the very low Bit Error Rate (BER) of optical fibre, combined with the coding schemes of FDDI to reduce packet errors, are some of the factors considered for this implementation of APIS.

Both versions of APIS were implemented on a Pentium-based single board computer (SBC) using message passing on a Multibus II backplane as the interface to the user. The user's application was implemented on a second SBC. This *off-host architecture* provides a heterogeneous distributed computing environment allowing the user's application to execute on its own processor under an operating system of their choice. 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.

Section II of this paper describes various characteristics of real-time information and the network service models used to deal with them. Section III discusses the architecture of APIS, its real-time tasks and the use of multicast addressing. Section IV details the implementation of APIS using XTP and FDDI. Section V explains why a second implementation of APIS, without XTP, was considered. Some performance results are presented in Section VI and the paper is concluded in section VII.

II. Real-Time Information Characteristics

A. Information Types

A real-time system handles several types of information flow, each with vastly different characteristics and requirements. Pardo-Castellote, et al [2] classify a few common examples summarised below:

1) Signals

Measurements of quantities that change over time are usually updated repetitively and may require delivery to more than one destination in the system.

Some properties of signals are:
1. **Time-critical --** Signals have a well defined *time-to-live* and are useless if the data is old.
2. **Idempotent --** Repeated updates are acceptable.
3. **Last-is-best --** Latest information is more important than retrying missed samples.
4. **High bandwidth --** Due to the repetitive nature of signals.

2) Commands

Commands are used to effect change in a system.

Some properties of commands are:
1. **Reliable --** The system cannot lose any commands.
2. **Sequential --** The order of a sequence of commands must be preserved.
3. Often **not time-critical**.
4. **Atomic Commitment --** Must be delivered reliably once and only once.

3) Status

Status information reflects the overall state or goals of the system.

Some properties of status information are:
1. **Persistent --** Status information usually persist for some time.
2. **Idempotent --** Repeated updates are acceptable.
3. Sometimes **time-critical**.
4. Sometimes **reliable**.

4) Requests

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

Some properties of requests are:
1. **Reliable --** The system cannot lose any requests.
2. Often **time-critical**.
3. **Synchronous --** If the client waits until the request is fulfilled.
4. **Asynchronous --** If the client does not wait until the request is fulfilled.

B. Network Service Models

Each of the Information Types discussed above can be assigned to a *Network Service Model* for delivery. The following traditional models have been summarised from [1]:

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. TCP is an example of this service model.

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, of which UDP is an example.

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. RPC and database applications using SQL are examples of this service model.

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.

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.

C. Message-Oriented Middleware

In 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, focus on the message rather than the channel set-up to deliver the message. 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.

In their Frequently Asked Questions (FAQ) [4], 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:

1. One-way
2. Request-Reply
3. Store-and-Forward
4. Publish-Subscribe

Publish-Subscribe MOM, also referred to as *dissemination architecture*, is of particular interest when distributing signals to distributed systems. 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.

III. APIS ARCHITECTURE

APIS is an implementation of a real-time Publish-Subscribe MOM.

Three of the main objectives of the APIS design are:

1. Data (signal) distribution in real-time –
   Multiple producers can (for the sake of redundancy and reliability) distribute the same information to one or many consumers, all within a guaranteed maximum latency.

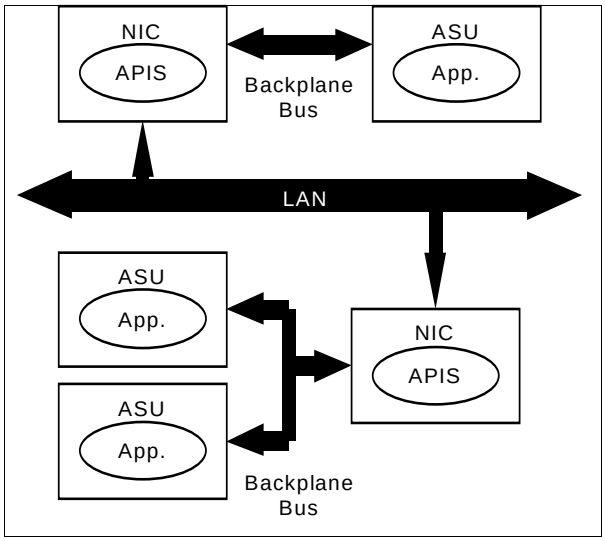


Figure 1. *Off-Host Architecture*

2. Off-Host Communication Architecture –
   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. Figure 1 shows several APIS Service User (ASU) applications communicating via Network Interface Cards (NICs) using an Off-Host architecture.
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.

3. Use of COTS technology –
In order to be affordable, components of the system should be commercial off-the-shelf (COTS). This also implies that an *open systems* architecture be adopted.

A. APIS Nomenclature

The following terms are used to describe the APIS architecture:

1. APIS Message
Applications communicate by sending user-defined APIS Messages, identified only by a user-provided *APIS Message ID*. Therefore 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[1].

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.
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.

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.

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.

B. Multicast Addressing

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.

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 to Class A, B, or C unicast addresses are. 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. An IEEE 802 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 [10].

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} APIS Message IDs.

[1] 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.

C. Real-Time Scheduled Tasks

APIS is implemented as a collection of tasks executing simultaneously to achieve maximum efficiency. An operating system that employs a pre-emptive, hard real-time scheduler is essential for the implementation of APIS. Refer to Figure 2 for an architectural overview.

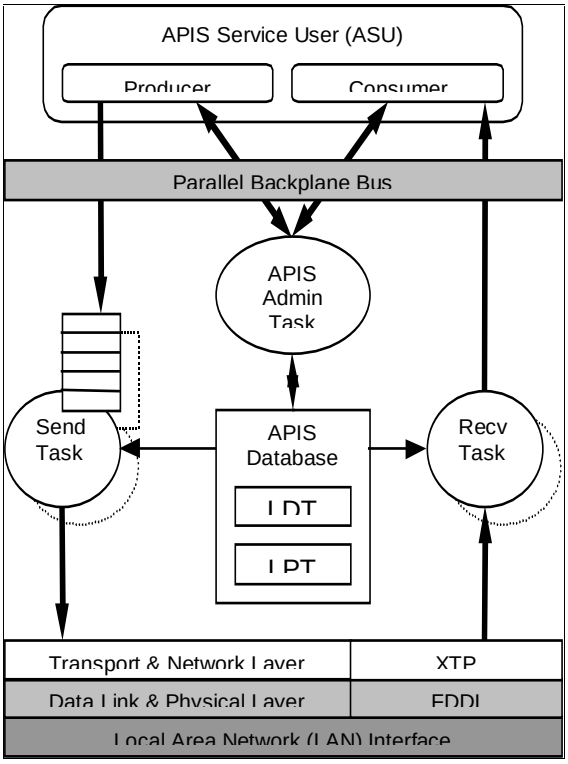


Figure 2. *APIS Architecture*

Two tables are used as a database for all messages registered on the NIC. The first is the *List of Produced Messages*, or *Local Produced Table* (LPT), containing information about producers registered on this NIC and the messages they produce. The second is the *List of Demanded Messages*, or *Local Demand Table* (LDT), containing the information of the consumers registered on this NIC and the messages they demand.

Three different types of tasks can operate on these tables as follows:

1. **Receive Task -** A task that receives a data message via the XTP protocol and dispatches it to all the consumers 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, with subsequent entries for all the consumers to which APIS should deliver this message. Several of these Receive Tasks may operate simultaneously.
2. **Send Task -** A task that receives a data message from the producer via the Multibus II interface, validates the producer as registered and passes it to the 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 Send Tasks may operate simultaneously. In the present design there are eight Send Tasks, one for each message priority that APIS supports.
3. **Admin Task -** A task that receives control messages from the producers and consumers via the Multibus II 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 exist on each NIC.

IV. APIS Implementation

The key components for the APIS implementation were selected as follows (each is discussed below):

1. **VxWorks** as RTOS for high-performance real-time multitasking kernel, STREAMS networking support and pre-emptive task scheduling.
2. **FDDI** as physical medium for timed token protocol, guaranteed latencies and bandwidth allocation.
3. **XTP** as transport protocol for multicast services.
4. The target hardware is an Intel Pentium-based SBC featuring a Multibus II Parallel Backplane Bus (PBB) and a PMC slot to host an FDDI card.

Although this paper focuses on the design and implementation of the APIS MOM, it is worthwhile recording the effort required to establish a real-time network platform supporting the implementation of 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 was bought 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 was acquired and re-engineered to the PMC standard.
3. XTP was not available for VxWorks at the time. The XTP source code for the Sun Solaris operating system was acquired and ported to VxWorks.

The suitability of the selected components to the design of APIS is discussed below.

A. Real-Time Operating Systems (RTOS)

APIS was implemented on the VxWorks RTOS, which provides pre-emptive scheduling as well as fast, deterministic context switching[2].

As previously indicated, a pre-emptive, hard real-time task scheduler is paramount to the implementation of APIS. APIS provides eight levels of priority to the ASU for sending messages. Eight message queues, each with a *Send Task* to service the queue, are used to implement the prioritisation of messages in APIS. This implementation effectively removes the need to implement a separate 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*.

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, a chance exists that *priority inversion* might occur. Priority inversion is the condition where a task of lower priority can, under a certain set of circumstances, deny a task of higher priority access to a shared resource indefinitely. VxWorks provides a special set of mutual exclusion semaphores, with priority inheritance protocol, to prevent such an occurrence.

B. Off-Host Communication Architecture

Multibus II (MBII) Parallel Backplane Bus (PBB) technology was used to implement the current ASU interface to APIS.

An advantage of MBII 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 bus.

C. Xpress Transport Protocol (XTP)

XTP [8][9] 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 [1]. 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.*" [8]

D. Fibre Distributed Data Interface

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 [5].

FDDI uses a *timed token* access method to share the medium amongst stations. This access method is different from the traditional MAC 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.

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.

The Target Token Rotation Time (TTRT) is the key network parameter that affects performance [6]; all stations on the network must negotiate and agree on its value. 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 [6] as follows:

The TRT can be as long as $2 \times \text{TTRT}$ [6]. Thus synchronous stations should set TTRT to half of the required repetition interval.

[2] By implementing a flat memory model, as well as executing tasks as threads in a globally shared memory environment, VxWorks can guarantee very low and deterministic context switching times.

TTRT should allow at least one maximum size frame along with the synchronous time allocation, if any [6]. That is:

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

Where: D_{max}, the Ring Latency[3], equals 0,06017 ms
F_{max}, the Maximum Frame Time[4] is 0,360 ms
Token Time, for 11 bytes, is 0,00088 ms
$\sum SA$ is total Synchronous Allocations

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

To achieve a latency of less than 5 ms, TTRT was chosen as 2 ms. This value allows $\sum SA$ to be at least 1,5 ms or 18 750 bytes during each TRT.

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

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

At the chosen TTRT the efficiency and maximum access delay can be calculated from the formulas derived in [6] as follows:

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 this case.

V. Observations and Limitations

During the implementation of APIS, the following observations were made:

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 [8]. Unfortunately the implementation of XTP we were using only provided the *Reliable Multicast Stream Service*.

Although we were implementing APIS on a very low BER ($2{,}5 \times 10^{-10}$ [6]) fibre LAN, the chance did exist that a lost message would be retransmitted, thereby missing its deadline.

2) File Descriptor Limit

A more serious trait of the XTP implementation pertaining to file descriptors was observed. Since this implementation of XTP was geared towards reliable multicasting, a separate STREAM was allocated for every multicast group that was joined, 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 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. It is also argued that the 4B/5B encoding scheme and the *Frame Check Sequence* (FCS) used by FDDI, eliminate the need for additional error checking by an advanced transport layer.

VI. Performance Measurements

APIS 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 SBC referred to as the Host.

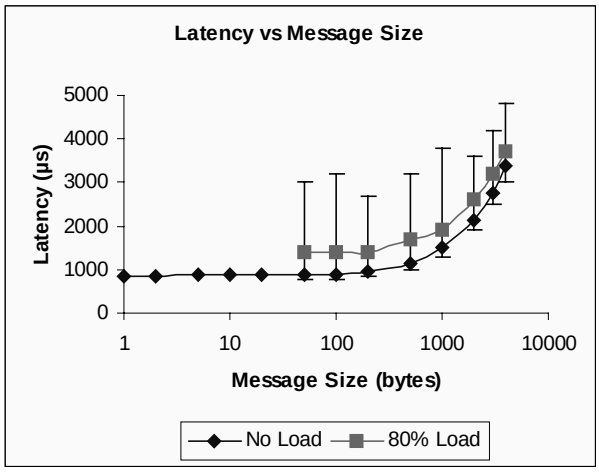
The relative times between events occurring on a single NIC were measured using a timer provided by the Pentium architecture. The timer increments at 200 MHz and can be read with a few simple assembler language instructions with little impact on the executing code.

Events that occurred on separate NICs, i.e. the start of a message transmission on one NIC and the reception of it on another, were measured by asserting signals on the serial ports of the NICs at various points in the software code. VxWorks allows direct access to the registers of the serial port devices, enabling us to effect change of the RTS and DTR signals with simple read and write instructions. The signals were monitored on a digital oscilloscope and logic analyser to calculate offsets.

The latencies depicted in Figure 3 represent the end-to-end message transfer latencies between two test

[3] For 2 km of optical fibre and 50 stations, D_{max} = (2 km) × (5,085 µs) + (50 stations) × (1 µs/stations) = 60,17 µs
[4] Maximum Frame Time = (4 500 bytes) / (125 MHz) = 0,360 ms

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 3. *APIS latency vs Message Size*

The 80% network load consisted of a 12 Mbit/s (12%) data transfer between the applications to load the network stack of each NIC, as well as an additional 68% load created by an independent network loading station.

Figure 4 shows a breakdown of the latencies that comprise a single APIS message transfer under no load conditions. 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 and the overall latency of the transfer. The contribution of the driver to the latency is 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 dependant on the size of the message, it only involves Message and Producer verification to the database.

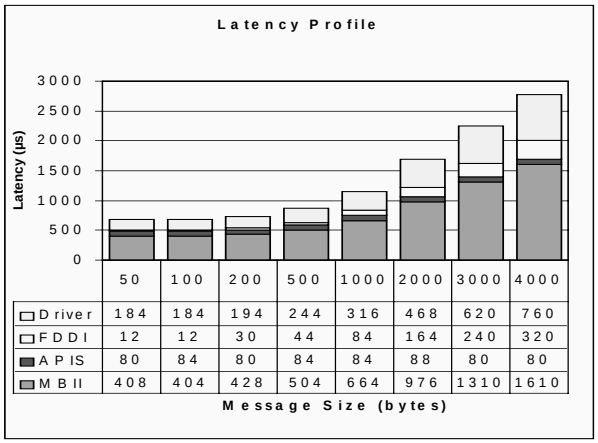


Figure 4. *Profile of a typical APIS Message transfer*

VII. CONCLUSIONS

It is concluded that the properties of Publish-Subscribe MOM are well suited to distributing signals within a real-time distributed system. These signals are latency-critical and therefore not necessarily reliable; it is better to lose one message and wait for its next update than to receive stale data.

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.

VIII. REFERENCES

[1] R.M. Young, “Real-Time Protocol Strategies for Mission-Critical, Distributed Systems”, PhD Dissertation, University of Witwatersrand, July 1996.

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

[3] 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.

[4] International Middleware Association, “Frequently Asked Questions”, http://www.imwa.org/utilities/faq.html.

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

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

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

[8] XTP Forum, “Xpress Transport Protocol Specification”, XTP 95-20, Revision 4.0, 1 March 1995

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

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