UNIVERSITY OF CALGARY

A Network Protocol for Real Time Applications

by

DONALD JOSEPH MOLARO

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA
MAY, 1999

Abstract

This thesis describes a local area network (LAN) protocol for real-time distributed systems that provides guaranteed bandwidth, maximum latency, and reliable data delivery. The protocol takes advantage of current networking protocols and hardware to provide an inexpensive solution that is suitable for embedded systems. This thesis describes the protocol and its implementation and compares the actual performance of the protocol with theoretical predictions. The results show that this protocol is appropriate for systems using real-time data over LANs, including applications ranging from air traffic control and factory automation to data collection in laboratories and intraship communications. The experiments measure the parameters needed for others to design systems using this protocol. This work provides pertinent information for researchers with interests in real-time networks and for designers of embedded systems that depend on real-time communications. The major contribution of this work is a simple and well-designed network protocol that meets real-time requirements, as well as a functioning implementation on cost-effective hardware that is suitable for embedded systems.

Acknowledgments

Jim Parker gave me the opportunity to start this.
Cullen Jennings was instrumental in making me work on - and finish this work.

Dedication

Dad.

Table of Contents

List of Tables

List of Figures

Chapter 1 - Introduction

The world happens in real time. Computers, however, are not particularly well suited to dealing with the real world; they are at their most effective when working with internal data, and performing calculations and reporting results when the calculations have been completed. It is only when required to interact with either a user or other aspects of the physical world that time becomes a fundamental issue.

Computer systems that deal with real-time inputs and outputs are found in automobiles, consumer electronics, and industrial control systems. Computer systems that deal with real-world objects, rather than abstract objects, must be concerned with the constraint of time.

It is difficult to program real-time computer systems and to verify that they can correctly perform the tasks assigned to them. Many advances in computer programming, such as object oriented programming techniques, have ignored real-time systems. Indeed, some techniques tend to avoid the entire time-efficiency issue in the name of simplifying the software design. Real-time systems have traditionally been the domain of engineering rather than of computer science, and this trend has unfortunately meant that many advances made in computer science have not been incorporated into the real-time domain.

Traditionally, systems that address real-time aspects of computer science and engineering have been limited to either single processor systems or closely coupled multiprocessor systems. The ability to build effective distributed real-time systems has been limited by networking constraints. Normally, computer network protocols are only concerned with managing the available bandwidth on the medium; this thesis, on the other hand, proposes a method of using traditional networking hardware and applying it to real-time problems. The solution to these types of problems is also applied to embedded systems and discussed for its use in safety-critical applications. The problem approached here is determining how to network embedded real-time systems for high-performance applications so that they can be used for multimedia applications, local area networks, and some aspects of wide area computer networks. As part of fully

understanding where this work fits in the larger context, this thesis also examines current developments in local area computer networks, embedded computer systems, and real-time computer systems.

Computers are being used in telephones, televisions, radios, and other devices that have not traditionally utilized computer systems, and industry observers expect these multiple devices to merge into a single multifunction device. These systems deal with time-sensitive data streams that must meet specific requirements: the data must arrive at the right time and be intact, the transmitted data must arrive within a specific interval, and it must arrive without errors. These devices have, until recently, used analog communications exclusively. Analog data is, by its very nature, time and order dependent. Signal loss in these applications results in a graceful loss in communications. By comparison, the loss of data in a digital stream can result in garbled and incorrect calculations, or even the complete failure of the data transmission.

Analog communication networks are very cost effective for one-to-many communication applications but are prohibitively expensive for many-to-many communication applications. Traditional computer networks are proficient at detecting errors and recovering data, but have not as yet properly addressed the issue of timely data delivery. Moreover, traditional computer networking protocols have dealt mostly with "burst" data and seldom with continuous data streams. Indeed, the newest and most advanced computer networks, such as asynchronous transfer mode (ATM), are not so much concerned with providing the maximum amount of bandwidth, but rather with providing a consistent quality of service to the nodes on the network. Bandwidth reservation protocols such as RSVP are being retrofitted to existing networking protocols and devices. The placing of voice and other time-critical communications on computer networks is currently one of the most active areas of focus in networking. An additional motivation for this work is therefore the need to build such systems within real-world constraints such as cost, ease of implementation, and the ability to upgrade the system to new technology as it appears.

In the context of this thesis, data falls into one of two fundamental categories: data for which time is a relevant dimension and data for which time is not relevant. Traditional computer networks such as 802.3 Ethernet do not consider time to

be of paramount importance. However, in real-time applications, many temporal and quality restrictions are placed on the data stream. In real-time applications, for example, data must be guaranteed to transmit at the correct time, data must arrive intact, and the transmission process must not garble or damage the data being transmitted. Data must also arrive by a certain deadline; data that arrives after or before the correct time window cannot be used by the system. The data is also constrained by order: each datum has a place in the stream and data cannot be transposed without destroying the integrity of the stream.

1.1 Terminology

Terms used in this thesis have specific meanings that should be clarified in the context of the above discussion.

Real-time computer systems perform calculations that must be completed before a specific time deadline or else the results of the calculation are useless and the system has failed. In general, real-time systems are non-terminating; that is, they have a set of tasks to perform in each time period and have been designed to perform that set of tasks continuously.

An examples of this type of system is a continuous data collection and analysis system such as SCADA, used in nuclear plant monitoring. These systems must run continuously and must continually perform a set of tasks within a series of time intervals. The time constraints in the "real world" are the important aspect of the tasks that have been assigned to real-time systems.

It is common to distinguish between *soft* and *hard* real-time systems. Soft real-time systems must perform a variable list of tasks by a specific deadline. One example of this is a system running a control application that occasionally inserts a task to perform system maintenance without violating the real-time aspects of any of the running tasks. Hard real-time systems must perform a set of invariant tasks, and must meet stricter timing deadlines than those found in soft systems [Kopetz:93].

Embedded computer systems are not general purpose computer systems; rather, they are computer systems suited to one or a small number of predefined tasks.

Embedded systems, as their name implies, are usually part of a larger device, such as control systems inside consumer electronic products. The computers inside a cellular phone, or those that control anti-lock braking in an automobile, are examples of embedded computers. Embedded systems also usually deal with activities in the "real world" in that they interact with people and objects that have a physical presence. Because embedded computer systems are often placed in consumer electronic devices and other devices that have high production volumes, expense is a very relevant concern for the designers and producers of these systems.

Many embedded systems are designed to work unattended for an extended period of time, and as such, they need to have failure recovery built into their design. Graceful failure is a design plan whereby if one component of the system fails, other components can take over responsibility for the functions of the failed component without user intervention. To avoid a single point of failure, it is common to add redundant, idle components to a system to take over from any failed parts.

Network protocols are the rules expressed in a mixture of software and hardware that allow computers to communicate with each other. Latency, bandwidth, reliability, and robustness are all aspects of computer networks that are of particular interest to this project. *Latency* is a measurement of the time (from start to completion) of the transmission of a data packet. Latency can be variable or constant, bounded or unbounded. *Bandwidth* is the amount of data that can be sent on the computer network in total and the amount of the network that can be used by individual stations in a unit of time. *Reliability* is the guarantee or probability that a datum that has been sent from a node will arrive at its destination. *Robustness* is the probability that corrupted data will be detected and recovered. Another important issue is the need for data to arrive at its destination in the same order that it was transmitted.

Safety-critical systems are systems that put peoples' lives or well-being at risk if they fail. Some examples of safety-critical systems are air traffic control systems, anti-lock brakes in automobiles, nuclear plant control systems, and medical applications. Such systems are normally very carefully constructed so that they do not fail. In the event of failure, they have been designed to fail gracefully, meaning that redundant components or other "non-critical" systems can take over for the critical system

[Budhiraja:93]. Designers of safety-critical systems must prepare for the failure of components, or even the complete system, through either a manual backup or by building redundancies into the system. Any failed part of the system should recover automatically, without user intervention.

1.2 Goals

The primary goal of this thesis is to develop a suitable protocol for implementing distributed real-time systems that provides precise, predictable, and constant values for network bandwidth, latency, reliability, and robustness. While meeting the requirements as defined, the design should be as simple as possible. In this thesis, I offer a demonstration of the network protocol and describe an implementation of the protocol that can be tested and experimented with on an embedded system. Because the target applications of this work are embedded systems, this project has also considered the need to minimize both the development effort and the cost in deploying the functioning system. As part of this effort, the design attempts to use as much existing off-the-shelf hardware and software as possible to minimize development time and cost.

The example use of this protocol comes from the domain of air traffic control (ATC). The requirements for ATC systems are extremely precise. Each controller requires a "position" from where he or she may listen and talk to a variable number of inputs and outputs. For example, each controller may simultaneously listen to adjacent controller zones, several aircraft radios, and one or more ground lines (telephones). Moreover, each audio source may be listened to at different volumes and through a variety of speakers at the position. The combination of audio sources and mixing parameters is also highly dynamic; the controller can change the parameters of what he or she is listening to and to whom he or she is talking by using a graphical user interface (GUI) that is part of the system. This very complex situation requires the transmission and reception of large amounts of time-sensitive data that needs to be managed effectively between a large number of systems. ATC systems are also safety critical: their failure could put people in harm's way.

1.3 Structure

In the following chapter, I survey relevant literature in the field and describe the areas of computer science used to address the problems encountered in the development process. In Chapter 3, I examine current developments in computer systems networking, with a particular emphasis on real-time networks. Chapter 4 describes the networking and embedded systems requirements of the protocol that has been developed. In Chapter 5, I provide the details of the implementation that was done to demonstrate the protocol and to determine experimental values. Chapter 6 describes the results of the performed experiments and provides an analysis of the collected values. Finally, in Chapter 7, I present the conclusions of the thesis.

Chapter 2 - The State of the Art

This thesis synthesizes several different areas of computer science, including network protocols, time synchronization, and embedded, safety-critical, and real-time systems. In Chapter 3, I discuss networking protocols and systems, with an emphasis on real-time attributes. In this chapter, I focus on the state of the art in various areas of computer science that are relevant to the goal of this project, and I describe some recent developments of interest.

In complex network computer systems, such as those being suggested by this thesis, accurate and precise time synchronization between nodes is very important. The proposed application requires that the system work with a high degree of synchronization. This chapter examines the way that time is manipulated by computers and discusses the ability to synchronize computer systems within a local area network.

This chapter also summarizes the hardware, operating systems, and software techniques in use for embedded and real-time systems. Embedded, real-time, and safety-critical systems are an increasingly important area of study as the world becomes a more digital place. Embedded computer systems are part of larger devices and are usually small, inexpensive, and limited in their function. Real-time systems are computer systems that have the concept of time in the real world embedded in their calculations. Fault-tolerant or safety-critical systems are taking on greater importance as computer systems are relied on to perform functions that, in the event of failure, will place people in danger. Embedded, real-time, and safety-critical systems are all active areas of research and development work; this chapter discusses some of the relevant literature and projects that are being undertaken in these areas.

2.1 Time Synchronization.

The concept of a universal, synchronized time is important in distributed systems in general and is of fundamental interest in distributed real-time systems. Time synchronization is the process that distributed computer systems use to maintain their

clock progressions at the same rate as that of the other clocks in the network. The relative amount of synchronization that can be achieved is called the granularity of the clock system. There are two main classes of synchronization algorithms: those of network agreement (or averaging) and those of a client-server design where synchronization is to a master clock. Advanced systems such as CesiumSpray are hybrid designs that use features of both network agreement and master clock synchronization [Verissimo:97].

2.1.1 Time and Real-time Systems

Real-time computer systems depend upon knowing the time, or at the least, knowing the time interval between events. Time can be defined in terms of the relative frequency of recurring events [Fuller:75]. In the case of an occurrence of a real world event, the real time system may need to take action by a specific deadline. Events in real-time systems may be generated by external stimuli or by time itself. The granularity of the local clock and the synchronization of the distributed time represent how closely events in the system can be scheduled next to each other. Variances in the time between synchronized systems is known as jitter. Jitter happens when clocks of poor quality are resynchronized often, causing the time on a system to "jump" back and forth in a seemingly random fashion. Large jitter is a condition that is to be avoided, and distributed real-time systems work to minimize it or its effects. One effect of jitter is that time-triggered events can go off before or after they have been intended.

2.1.2 Universal Time

Clocks in computer systems are of notoriously poor quality and normally can drift as much as a few percent in the space of a month [Motorola:96b]. After the clocks in a distributed system are synchronized, each of the systems begins to drift; occasionally, regular resynchronization can bring them into agreement with each other. However, a collective drift in the distributed system can result in the overall system

drifting from universal time. Universal (or reference) time was originally provided as radio time signals such as those of the NIST (National Institute of Standards and Technology) in the U.S., CHU in Canada, and RBU in Russia [Lichtenecker:97]. The accuracy of these signals as related to the Coordinated Universal Time (UTC) is a few microseconds. Time synchronization is also available over the Internet from a number of sources, such as NASA, the NIST, and the U.S. Navy. By using the network time protocol (NTP), it is possible for a computer system to be synchronized within a few tens of milliseconds of universal time [Mills:92].

For synchronization of even greater accuracy a Global Positioning System (GPS) receiver can be used. GPS is a satellite-based location system that uses very precise and accurate clocks onboard satellites orbiting the earth. With this system it is possible to obtain a clock reading that is within a few hundred nanoseconds of UTC. GPS time synchronization is considered to be the current "state of the art" in synchronizing time in distributed computer systems [Schmid:97].

Table 1: Relative Accuracy of Time Systems

Synchronization Method	Accuracy	Relative measure
Internet	milliseconds	0.001
Radio	microseconds	0.000001
GPS	nanoseconds	0.000000001

2.2 Network time synchronization systems

It is possible to build complex distributed systems that do not have synchronized time. The Internet is a good example of this. Using operating system concepts such as critical sections, semaphores, monitors, and time-outs, one can design and build systems that are highly coordinated. However, these systems cannot be real-time systems because there is a possibility that a resource will be deadlocked or unavailable, making it impossible to meet real-time constraints. In real-time systems, the

system must have an intimate knowledge of the local time, and in distributed real-time systems, each system must use the same time. Real-time systems need to know what time it is, when events happen, and by what time they must respond to an event. The same constraints are true for distributed real-time systems.

The most commonly used network time synchronization system, NTP, can typically achieve synchronization in local area networks of a few milliseconds and of a few tens of milliseconds in wide area networks [Mills:92 and Mills:85]. NTP is now a de facto standard in use around the world; client and server software is freely available and can provide highly accurate synchronization. For the greatest possible accuracy, a GPS synchronization system such as CesiumSpray can achieve clock synchronization in the range of a few hundred microseconds from UTC [Verissimo:97].

Two fundamental aspects of time synchronization are relevant to real-time systems: synchronization of the system to absolute time and synchronization of events within a system to a relative time from other events in the system. Many distributed systems only require that the relative time between the systems be synchronized, such as with distributed file systems or internal mail transactions. So long as the system is independent of outside influences, the absolute time is not relevant. If the system is interacting with other systems outside its local domain, the simplest approach is to synchronize both systems to a third reference time. There are a number of public domain, freeware, shareware, and commercial packages capable of providing this service. NIST, for example, provides a list of sixteen current vendors [NIST:98].

2.2.1 GPS Time synchronization

Time synchronization is currently an active area of research. As a case in point, "The Journal of Real Time Systems" recently dedicated a full issue to this subject [Schmid:97]. The editor, Ulrich Schmid, indicates that the advent of GPS-based time synchronization systems has, by an order of magnitude, increased the accuracy that time synchronization systems can achieve. GPS can be used to synchronize systems on a scale never before possible.

Schmid sees it as interesting to attempt to revise and improve on existing

"old" time synchronization systems that were initially proposed in the 1980s [Halpern:84, Lundeliu:84, Lamport:85, Mahaney:85, and Schneider:85]; such as reported by Alari [Alari:97] and Fetzer [Fetzer:97]. Schmid sees the availability of highly accurate and inexpensive GPS receivers as an enabling technology that will allow systems that are separated by vast distances to take action in a highly coordinated and synchronized fashion. This order-of-magnitude increase in the accuracy and precision of time synchronization ability has spawned new research in this topic. Schmid also identifies the failings of GPS, and external time synchronization in general, for safety-critical systems.

Navigation has always depended on knowing the precise time: the development of accurate mechanical clocks allowed for precise worldwide navigation [Quill:66]. The GPS is a satellite-based planet-wide location system that can also be used to determine universal time. An objective of this system is to allow people using an inexpensive GPS receiver to accurately locate their position anywhere on the planet (recent prices for "consumer grade" systems have fallen below $200.00 (U.S.)). The system uses highly accurate time signals from several satellites that are in the receiver's line of sight. By noting which satellites provided the signals and the relative differences in the times provided, it is possible to calculate the location of the receiver. The satellites carry highly accurate cesium and rubidium atomic clocks, and as a side effect of the location algorithm it is also possible to very accurately determine the universal time to within a few hundred nanoseconds. Peter Dana provides an overview of GPS technology [Dana:97].

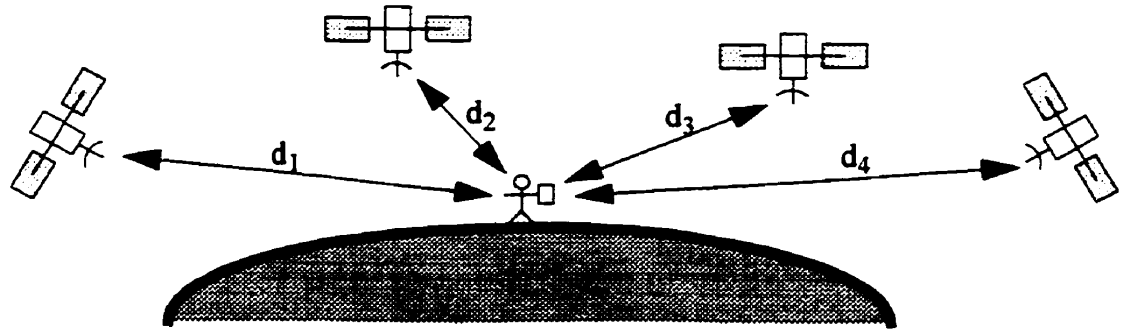


Figure 1: GPS and satellite Interaction

GPSs for both time synchronization and location require a clear line of sight to at least four, and preferably five, satellites. There are many places on earth where it may take several minutes for enough satellites to be "over the horizon" for an accurate fix to be obtained. Other factors can affect the usefulness of GPS as a means to synchronize time in computer systems; for example, GPS signals can be affected by buildings and other landform interference. There is also a hardware cost to implementing GPS time synchronization. While the minimum cost of a GPS may only be a few hundred dollars, for embedded and other systems this extra cost may make them prohibitively expensive. The basic unreliable nature of GPS does not make it suitable for safety-critical systems that require a consistently up-to-date time.

The GPS is a system provided essentially for and by U.S. military organizations. There have been historical cases where the system has been "degraded" worldwide and in other cases turned off because of military or political considerations. The use of GPS for location or time synchronization is essentially at the discretion of the U.S. Many uses of advanced time synchronization cannot allow their time source to be controlled by outsiders. However, for many other applications, the advantages of GPS outweigh the disadvantages.

CesiumSpray represents the archetype of a GPS-based time synchronization system that is suitable for worldwide area networks. CesiumSpray [Verissimo:97] is a hybrid system that uses GPS receivers to synchronize master clock

systems within local areas in a global network.

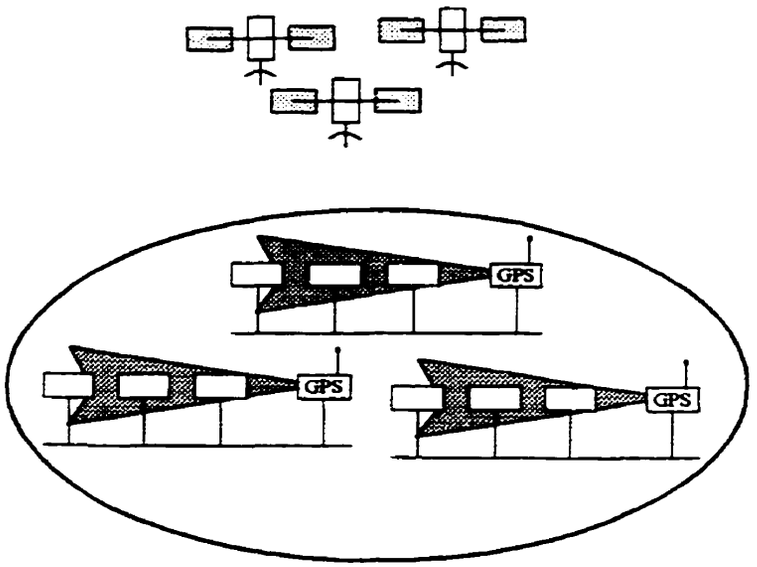


Figure 2: CESIUM Spray

The essence of the CesiumSpray system is that selected nodes in a local area network collect the universal time from their GPS time receivers and spray that time information out to other nodes on the local area network. In this way, accurate time information is transmitted to all nodes of the global network as quickly as possible. The creators of CesiumSpray claim the accuracy and precision of the system to be in the order of a few hundred microseconds across the entire system.

Another hardware-based time synchronization system called UTCSU, suggested by Schossmaier, Schmid, Horauer, and Loy [Schossmaier:97], is a very precise and accurate system for disseminating time in a computer network. Its goal is also to disseminate a highly accurate time source, such as a GPS receiver, across a network. This system represents an effective method of synchronizing time in a computer network, but it also requires custom hardware and represents a complex approach.

Halang and Wannemacher [Halang:97] propose a series of hardware improvements to allow computer systems to more effectively deal with the constraints placed upon them in hard real-time systems. The basis of their approach is to exploit the underlying parallelism that is inherent in these systems by assigning a piece of dedicated hardware to the timekeeping tasks. Normally, the single CPU in the hardware system

would address these tasks in a time-sharing fashion, but the use of dedicated hardware (as they describe) allows for a more precise modeling of the real world within the system.

To achieve high-precision timekeeping within their system they employ a GPS receiver which allows for a synchronization of 100 microseconds to UTC. The GPS unit they use has a retail price of $100 (U.S.) for the receiver and $70 (U.S.) for the antenna. The GPS antenna, however, must remain in the line of sight of the GPS satellites, reducing the number of general purpose applications that this device may be used for.

2.2.2 NTP

The most common system used for synchronizing time in distributed systems, and that which is claimed to be the most successful distributed system ever, is NTP (network time protocol) [Mills:92, Mills:85, and Mills:81]. NTP is credited to David Mills of the University of Maryland, who has published about a dozen papers on this topic. NTP is a point-to-point protocol where each client contacts a server to ascertain the current time. Mills believes that there are now in excess of 100,000 time servers available on the Internet. NTP is a software-only system that provides synchronization in local area networks of a millisecond or two and of a few tens of milliseconds in global wide area networks. Using servers and clients available on the Internet, it is possible to achieve synchronization to within a few tens of microseconds of "global" time across the Internet from the NIST or NASA. NTP is currently a version 3 revision and is enshrined in Internet RFC 1305. Mills is planning enhancements for NTP version 4, such as secure time servers and the ability to encrypt time messages. NTP is a robust and elegant protocol suitable for a wide variety of environments, and it has allowed for the inexpensive and reasonably precise synchronization of systems around the world.

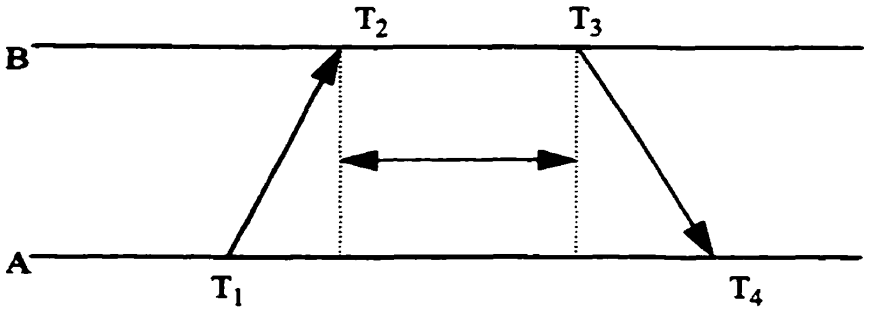


Figure 3: NTP Client/Server Interaction

The close synchronization of time in many computers located on a local area network is not only fundamental to making them work together effectively in a traditional network operating system such as UNIX or Windows NT, it is also an important step in making the system work as a coordinated cluster of systems.

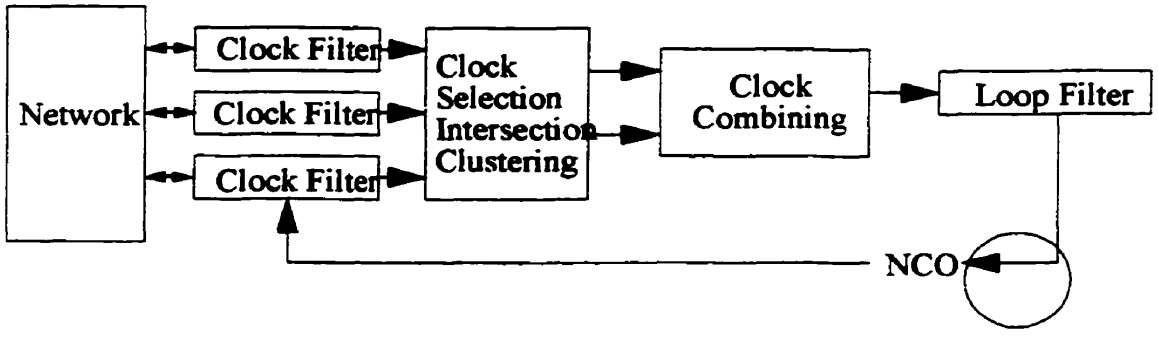


Figure 4: NTP Feedback loop

While NTP is suitable for making real-time systems with modest synchronization requirements; such as to coordinate systems to exhibit behavior within a second or two of each other. For finer-grained synchronization it is not a viable mechanism. IP (Internet Protocol) is typically the underlying protocol of the system on which NTP is used, and IP does not guarantee that delivery will occur within a specified period or even guarantee delivery. For highly coordinated, sub-millisecond real-time systems, the synchronization of NTP is not sufficient.

2.2.3 DCE/DTS

DCE, the Distributed Computing Environment, is a system that allows a number of computer systems to behave in consort with one another [Rossenberry:92]. This system is based upon RPCs (Remote Procedure Calls) which are typically carried across TCP/IP computer networks. DCE was incorporated into the OSF/1 operating system to function as its method of distributed and remote computing. It is as much a product as a methodology for writing distributed programs.

DCE is an OSF (Open Software Foundation) standard and it has been adopted by several vendors, such as IBM, DEC, and HP. It is used in Microsoft Windows NT as a tool for deploying distributed systems. A complex distributed system that supports network authentication and resource replication, DCE can also provide for a highly managed computing environment. DCE itself is implemented with secure RPC calls and it allows large, complex, and highly distributed systems to effectively appear as a single computer system. DCE can also support the replication of resources such as disks so that in the event of hardware or communications failure, the system can continue to provide service.

Highly distributed, reliable systems such as those provided for by DCE have a notion of a single reference time. Without highly synchronized time it is impossible for distributed systems to effectively replicate transactions. DCE includes a time synchronization system called DTS, the Distributed Time System.

NTP and DTS have some fundamental similarities, as well as differences. The following table compares the two systems:

Table 2: Comparison of NTP & DTS

Feature	NTP	DTS
Accuracy	10s of milliseconds	Configurable
Internet Servers Available	Yes	No
Basic Transport	UDP	RPC
Co-operative with each other	No	Yes

DTS is a commercial product supported by large companies, and as such, it is considered to be more of an industrial standard than its public domain relative, NTP. For distributed real-time systems, the same features and drawbacks can be noted for DTS as for NTP. For highly coordinated systems, the unreliable nature of the underlying protocol make either of these two schemes inappropriate.

Distributed computer time synchronization is currently achieved in a networking environment with a number of different architectures. It is possible in a complex computer network to synchronize all the times on the systems without synchronizing to an external reference time. It is also possible to include a number of reference time servers in a computer network to allow for both accurate and precise time. Using current technology, time synchronization in a computer network can be within a few milliseconds.

2.3 Embedded Computer Systems

Embedded systems are computer systems that have been incorporated into a larger device. They can vary from simple feedback-control circuits to complex systems incorporating long-term storage and networking facilities. Embedded systems share many similar components with standard computer systems, such as a CPU and memory [Terry:96]. However, they do not usually include disk or networking capabilities and often have rudimentary, if any, human interactive displays. The design of the embedded system is usually a function of the overall device's design and as such is incorporated

directly (or embedded) in the device of which it is a part. Common sense states that for a device to be inexpensive and robust, the number of parts used to make that device should be kept to a minimum.

2.3.1 Microcontrollers

To keep the number of components to a minimum, but still retain full functionality, embedded systems are commonly implemented on microcontrollers. A microcontroller is a chip that integrates a CPU, memory, and interface hardware into a single package. This results in a system with a single major component, the microcontroller, and for most applications, a minimum of required additional hardware. It is possible, however, to construct complex computer systems around such devices, but they are constrained by the limitations built into many microcontroller designs. Typically, microcontrollers have a smaller address range and the technology used to implement them may be "a step behind" that which is currently state of the art [Intel:97]. Virtually all CPU manufacturers have a line of microcontrollers designed for embedded applications. Devices can range from the relatively simple, such as the Motorola MC68HC11, which sells for a few dollars at the time of this writing, to the IBM/ Motorola PowerPC line which costs several hundred dollars. The following table summarizes the capabilities of some commonly available units.

Table 3: Comparison of Available Embedded CPUs

Name & Manufacturer	Type	Features	Cost (1998)
PowerPC - IBM/ Motorola [IBM:98]	32/64 bit RISC	High performance Power PC processor Floating Point Unit Large cache memory	$ 250.00
StrongARM - Digital [DEC:96]	32/64 bit RISC	Good Performance Processor Cache Memory	$ 50.00
Coldfire -Motorola [Motorola:97]	32 bit CISC	68030 CPU Core Small cache Memory On Chip Timers Serial Interface built in.	$ 25.00
68HC11 - Motorola [Motorola:96]	8 bit CISC	Small amount of RAM, EEPROM and ROM on board Serial & Parallel interfaces Real-Time timers Digital IO Ports	$ 8.00

Depending upon the required application, a systems designer can choose a device with the right combination of processing power and features.

2.3.2 Embedded Systems Design

Most consumer devices that use electricity now incorporate some electronic components. Everything from the smallest devices, such as watches and children's toys, to large items like automobiles incorporate microprocessor control. While these control units are usually physically small and require only a modest amount of power, they have dramatically increased the utility of some types of electronic devices. In order to be useful, embedded systems need to be incorporated into a larger device; they are not usually a useful device on their own.

Embedded systems must be engineered and designed carefully, because many of the applications that they are to be put into will require them to operate for an extended period without outside maintenance or monitoring. They must be robust and, in

the event of failure, they must fail in a manner that does not endanger people or property. If an embedded system is to be used in a consumer electronics application, the design of the system must support mass production and be a cost-effective solution to the problem at hand.

One issue in embedded systems design that does not normally occur in general purpose computer systems is deciding whether to use dedicated hardware or implementing the required functionality in software. Some work of embedded systems can be carried out by hardware and other aspects can be carried out in software. The design of the embedded system must account for the work best done in each of the paradigms. The introduction of programmable hardware, such as Programmable Gate Arrays (FPGAs), has further blurred this distinction [Olukotun:94]. Embedded systems are available in a variety of designs, including single processors, closely coupled multiprocessors, or distributed systems. The embedded system may be built to handle a specific problem domain or it may be designed for general purpose applications. The hardware architecture of the embedded system will depend on the application in which it is being used.

2.3.3 Embedded Systems Applications

Consumer electronics is an area that has benefited from the development of embedded computer systems: televisions, kitchen appliances, and household goods all use these types of systems. The development of small and powerful computer systems has also led to new classes of products, such as active noise control systems, that would simply be impossible or too expensive to build without embedded systems.

Embedded systems have also allowed for the development of "smart" devices. A smart device is capable of making decisions about the environment it is in, and altering its behavior based upon those inputs. Streetlights are a simple example of a "smart" device. They take into account the time of day, the level of darkness, and the cost of electricity and then decide how and when to turn themselves on and off. We have recently seen the dramatic use of military "smart" weapons such as missiles, bombs, and other munitions that can identify an enemy target, attack, and destroy it with a minimum

of outside intervention [Clancy:97]. Whether one agrees or disagrees with the use of such devices for military applications, they do demonstrate just how effective self-controlled devices can be. The use of powerful embedded systems has also permitted the development of autonomous and semiautonomous robotic systems. These robotic systems can explore areas that are dangerous, difficult, or inconvenient to get to, allowing humans and machines to work together effectively. Embedded systems that can make decisions about their environment need to be able to gather data from that environment. Such systems are called sensor-enabled. A wide variety of electronic sensors are available for such stimuli as light, radiation, acceleration, heat, pressure, etc.

Embedded systems are usually part of a device in the "real world," and the devices that incorporate them are interacting with other devices and aspects that occur in real time. Many embedded systems are real-time systems, which means that calculations must occur by specific deadlines imposed by factors outside of the computer system.

2.4 Survey of Real Time Computer Systems

Real-time systems are a subclass of general purpose computer systems. While a real-time system can perform general purpose tasks, a general purpose computer system may not necessarily be suited to real-time computing. Real-time computer systems perform computing tasks that have time completion deadlines in the real world. These types of systems are used for control, data acquisition, or any application in which time is a parameter of the computation. Real-time systems can be applied to any computational situation that involves doing something where time is a relevant metric of success. Examples of real-time systems that most people are familiar with are the computational parts of automobiles such as anti-lock brakes, plant automation, and timekeeping. However, much of the "advanced" technology that has recently been produced, such as speech and handwriting recognition, also has real-time aspects. It would not be particularly useful or interesting, for example, if a recognition system took a long time to interpret the input. Other systems, such as advanced machine control, computer vision recognition systems, and other human computer interface systems, all

have to work in real time with either other computer systems or humans. Real-time systems can be loop or event based, and the fundamental structure of the software is a list of tasks that must periodically be run or events that must be serviced within a defined time scale [Artesyn:98].

Real-time systems are divided into hard and soft systems [Kopetz:93]. Hard real-time systems must complete computations within a set period of time. The computational load is invariant and the system must guarantee it to be completed; the failure of the system to complete those tasks within the specified interval is deemed to be a complete failure of the system. These types of real-time systems are also called deterministic. Soft real-time systems often have a variable task list that must be serviced. If the computer system cannot service the tasks within the specified interval, the tasks are deemed to have failed but the overall system is still considered to be functioning. These types of real-time systems are called non-deterministic. Hard real-time systems are used in safety-critical applications or whenever failure is not an option. Soft real-time systems can be used in environments where flexibility is preferred over determinism.

The developer of real-time systems can choose to implement all of the functionality required from the hardware up, without the support of an operating system. However, it is more common for the hardware to support a real-time operating system that will assist in the development of the final system. Several variants of real-time operating systems are available with certain characteristics that make them more or less suitable for any given project. The operating systems considered in this survey have been selected to highlight their range and to demonstrate how the systems are used and made available to the development community.

The following brief overview of some common real-time systems outlines the options that are available to systems designers.

- **Rtems - Real-time executive for missile systems**

Rtems is a publicly available real-time operating system developed by the U.S. Department of Defense. The primary objective of this system was to be a small, highly robust operating system suitable for embedded applications such as autonomous missile control systems. Rtems has proven to be a good multipurpose real-time operating

system. It is small, highly portable, and does not introduce a high overhead onto the system. It features a kernel that is capable of performing both time-sharing and priority task queuing, with high priority tasks being given preference in the execution cycle. Rtems does not normally include support for devices such as displays, long-term storage such as a disk, or networking interfaces. Since Rtems is not a "supported" system, designers will have to either find or implement their own device support as required. Because of its small "footprint" and high portability, Rtems is suitable for embedded applications [www.rtems.army.mil]. Development environments for Rtems are also publicly available in the form of a GNU cross-compiler for the architecture of the target system. Rtems is suitable for both soft and hard real-time applications.

•OS-9

OS-9 is a commercial and highly portable operating system suitable for embedded systems. It is available for a variety of processors including PowerPC, Motorola 68000, Intel X86, and the Digital ARM. OS-9 supports common devices such as networking interfaces, LCD screens, and other devices that are found in embedded computer systems. Microware of Iowa implements, supports, and sells OS-9. OS-9 was designed as a real-time operating system specifically for use in embedded applications. A suite of development tools (such as Java, C, and C++) for developing applications for OS-9 is also available from Microware [www.microware.com]. OS-9 is suitable for both soft and hard real-time applications.

• QNX

QNX is a commercial, PC-based real-time operating system that is also currently available on PowerPC and MIPS platforms, although this is a relatively recent development in its history. QNX is available from the QNX Corporation of Ottawa, Ontario. Using standard PC hardware, QNX implements a real-time operating system and supports standard PC devices such as video cards, networking interfaces, and storage devices. QNX is suitable for soft real-time applications and some hard real-time applications. Typical applications for QNX include data acquisition and automated control systems, like those found in a factory automation environment. QNX Corporation provides development tools such as editors and language compilers, as well

as general purpose tools such as window systems [www.qnx.com].

• **Real-time Linux**

Real-time Linux is based upon the popular and freely available Linux operating system. While Linux is available on a variety of hardware platforms, it is normally found on Intel-based personal computers. Linux was developed by the Internet community as an "alternative" operating system to commercial systems, and is largely the work of a single individual, Linus Torvalds. Real-time extensions, such as the ability to pre-empt the kernel, have been added to a real-time variant of Linux. Similar in abilities to QNX, it is suitable for soft real-time systems and some large-grained hard real-time systems. Real-time Linux benefits from the many features of Linux that it has inherited, such as device support and a suite of public domain development tools. Linux is available from a large number of sources but is easily obtained from the Red Hat Corporation [www.redhat.com].

• **Real-Time Windows NT**

Real-Time Windows NT is based upon the standard Windows NT system. Incorporating similar features as real-time Linux, such as timers and a kernel that can be pre-empted, Real-Time Windows NT can be used to implement soft real-time systems. Real-Time Windows NT is a product of Microsoft Corporation of Redmond, Washington. It includes support for a wide variety of devices and many features. A large amount of overhead in the system is given to supporting system tasks. Because of the complexity of this system, Real-Time Windows NT should only be considered for soft real-time applications [Epplin:98].

Table 4: Comparison of Real-Time Operating Systems

Characteristic	Operating system	Comments
General purpose operating system	Windows NT (RT) RT Linux QNX	Operating systems that in addition to supporting real time features are also capable of supporting general purpose computing demands.
Suitable for embedded systems	Rtems OS-9	Operating systems that have the required size and portability to be placed in the smaller memory size typical of embedded systems.
Commercial	Windows NT QNX OS-9	If the operating system is available from a commercial source, as well as being supported.
Public Domain	RT-Linux Rtems	Operating systems that are available to the public at large, these systems, may be of very high quality but will not normally be supported.
Soft Real time	RT - Linux Windows NT (RT)	Operating system is suitable for soft real time applications only.
Hard Real time	Rtems QNX OS-9	Operating system is suitable for both soft and hard real time applications.

2.4.1 Programming Real-time Systems.

There is essentially no consensus on how to develop embedded real-time applications. The traditional model of a structured approach is still popular [Bohannon:98], although there are a significant number of researchers working within the object oriented paradigm [Clohessy:97, Gullekson:96, and Kim:97]. There is also a large community of researchers who propose system development shells and other techniques that are not strictly object oriented or structured [Ghose:97, Gomaa:86, and Schneider:95]. Kopetz proposes a more rigid engineering approach [Kopetz:91] and Sridhar provides a useful checklist for designing real-time communications systems

[Sridhar:98].

The motivation for moving into an object oriented paradigm for embedded systems is precisely the same as the motivation used to justify object oriented design in other applications: namely, the advantages of abstraction, reuse, and more effective modeling of the problem. However, some of the traditional advantages of object oriented design can certainly affect the fundamental aspect of real-time programming. The most significant constraint in real-time systems programming is knowing and fully understanding the execution time model of the program. The use of inheritance, method chaining, and polymorphic methods can result in programs that have a complex execution timing model. The issue of memory garbage collection in languages such as Java and Smalltalk can result in the system spending a large amount of time, at what appear to be random intervals to the system, away from dealing with the critical real-time tasks - these types of issues can be dealt with but the developer must be aware of many things that the system normally handles. One can still use object oriented languages to implement real time systems; the developer must be aware of a more complex timing model.

Structured techniques may be more effective at modeling the computer system and responding effectively to its needs. However, this approach precludes taking advantage of the many recent developments in software techniques. The essential issue in developing embedded real-time systems is the need to apply the correct technique to the situation at hand. There does not seem to be, at this time, a magic bullet for all real-time applications [Cox:86].

No matter which programming paradigm is chosen for a real-time system, the architecture of the system can be further classified into different aspects. The system may poll its sources of input continuously, it may wait for an interrupt to be generated externally, or it may introduce an interrupt internally from a timer. Real-time systems may interface with the world using any, or all, of these techniques

Chapter 3 - Computer Networks

Networking protocols allow computer systems to exchange and share data. A great deal of effort has been made to make these systems fast, reliable, and sensitive to timing constraints. Some computer networks are for general purpose applications and others have been specifically designed to solve particular problems.

In this chapter, I describe the history and background of computer networks. I discuss networking protocols and describe and evaluate how well they can perform in a real-time environment. Then, I examine the protocols that have been specifically created by others to address real-time and safety-critical applications.

Several types of local area computer networks are in common use, and these are generally differentiated by their topology. Topology is the manner in which the computers are connected with each other; the usual configurations are fully connected, star, bus, or ring. In *fully connected networks*, each computer has a dedicated link to all other computers in the network; some closely coupled networks used in high-performance systems employ this type of connection. In a *star network*, each of the nodes is connected to a central node or switch that performs traffic management. *Bus networks* share a common communications medium and each computer is capable of transmitting and receiving at the same time; the most common computer network implemented with this is Ethernet. Finally *ring networks* occur when each computer in the system is only connected to its neighbors on the "left" and "right".

Arguably the first practical local area network was invented at Xerox PARC. Ethernet was originally developed by Xerox at the Palo Alto Research Center in the late 1970s as a 2.94-megabit per second CSMA/CD designed to connect approximately 100 workstations [Metcalfe:76]. Ethernet was a success at Xerox. An IEEE standard, 802.3, was drawn up by Xerox, Intel, and DEC. The result of this work was the development of 10-megabit per second Ethernet. Ethernet is now in very widespread use and the standard has been moved from an implementation on coaxial cable to an implementation on twisted-pair wires. Recent advances have increased the bandwidth from 10 megabits to 100 megabits, and the 1-gigabit per second version is

currently in the final stages of standardization. To address some of the performance limitations of standard Ethernet it is now possible to implement an Ethernet network as a star using a device called an Ethernet switch hub of the star. At the same time that Xerox was developing Ethernet, other companies were investing resources and research into other networking systems such as token ring [Tanenbaum:96] and ARCnet [Contemporary:98].

Switched networks such as switched Ethernet, switched token ring, and ATM use a central node for traffic management. This central node ensures that there are no collisions and a minimum amount of bandwidth is allocated to each connected node. Switched networks are typically used in modern high-performance LANs.

Network protocols are combinations of software and hardware that allow computer systems to communicate with each other. The goal of network protocols is to communicate with other systems with a known amount of bandwidth, latency, reliability, and cost. Some types of computer networks have made design trade-offs for these factors. For example, a protocol that has a very low latency transmits small packets very often, which will result in a loss of available bandwidth on the medium. Figure 1 shows those aspects of network protocol design that can be traded in various ways to achieve specific design goals.

The seven-layer OSI model is often used to describe network protocols. OSI has developed a standardized method for classifying different aspects of network protocols ranging from a physical description of how data is transmitted to describing how applications behave and communicate with each other. An excellent reference work in this area is Andrew Tanenbaum's "Computer Networks" [Tanenbaum:96]. Networking protocols and systems are commonly identified by which layers in the protocol model they cover.

- **Application** - Control of attributes of the data stream specific to the application; an understanding of what the data means.
- **Presentation** - Control of the format of data being sent on the network; abstract data management.

- **Session** - Enhanced services for the transport layer, such as synchronization and dialog control.
- **Transport** - Acceptance of data from the session layer, and if necessary, the reformatting **of** that data for the network layer.
- **Network** - Control of the operation of the local subnet; congestion control.
- **Datalink** - Recognition of frames, boundaries, and error conditions and error detection.
- **Physical** - How the mechanical, electrical, or other physical parts of the network interact with one another.

Typically when a protocol is referred to it is referring to handling one or more of the layers of the OSI model.

3.1 Theoretical basis

Computer networks deal with channels of data. A channel is an abstraction of two nodes communicating. There are two important theoretical considerations when dealing with computer networks, the maximum amount of data that may be sent across the channel and the manner in which the channels are allocated.

The maximum amount of data that may be sent across a channel is a well understood phenomenon [Tanenbaum:96d]. H. Nyquist in 1924 developed an equation that places a strict upper limit on the amount of data that may be sent across a channel. Shannon later refined Inquests work to include noise. Shannon's result is the maximum data rate of a channel whose bandwidth is H Hz, and whose signal to noise ratio is S/N is:

$$maximimdatarate = HLog_2(1 + S/N)$$

This result imposes a strict upper limit on the amount of data that may be sent over a channel.

Static channel allocation schemes such as time division multiplexing divide the available channel into discrete parts. A perfect channel, such as queue has the

following time model. With a mean time delay, T, a channel of capacity C bps, with an arrival rate of λ, each frame having a length drawn from an exponential probability density function with mean $1/\mu$ bits/frame [Tanenbaum:96e]

$$T = \frac{1}{\mu C - \lambda}$$

If this channel is further divided into N independent subchannels, such as would be found in time division multiplexing, each with a capacity of C/N bps. The mean input rate on each of the subchannels will now be λ/N. If we recompute T we now get:

$$T_{TDM} = \frac{1}{\mu(C/N) - (\lambda/N)} = \frac{N}{\mu C - T} = NT$$

In essence we have now reduced the capacity of the channel by a factor of N. For bursty traffic that would normally be found in computer networks this can be a limitation - the full capacity of the network is never available to any station. As we will see this limitation will become an asset for real time traffic.

3.2 General Purpose Computer Networks

General purpose computer networks are available in two basic variants: those that are part of the 802 standard and those that are not. The 802 series of standards makes up the majority of local area networks in use. They cover Ethernet, token ring, and FDDI. Other networks that do not fall under the 802 standard include ARCnet and a number of networking systems developed for telecommunications, such as ISDN and ATM.

3.2.1 802 Standard Networks

Aloha is the starting point for the development of all the 802 protocols, of which Ethernet, token ring, and token bus are a part. Aloha is a simple protocol

developed at the University of Hawaii by Abramson [Abramson:85 and Tanenbaum:96b]. A transmit and collision detect system that uses a common medium, Aloha's original purpose was to allow remote data collection. The fundamental issue in Aloha, and in subsequent shared media systems, is the allocation of the transmission channel. The solution used by Aloha is to let all users transmit at any time, detect any destroyed or garbled frames, and then wait a random amount of time before retransmitting.

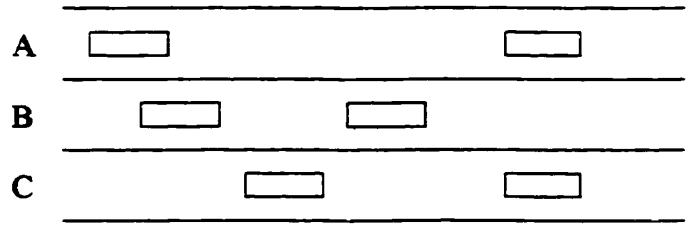


Figure 5: ALOHA systems are permitted to transmit at arbitrary times

One result of the work done on Aloha was the development of Ethernet at Xerox PARC in 1972. [Metcalfe:76]. This development was followed by the standardization of Ethernet with Xerox, Intel, and DEC developing Ethernet hardware. 802.3 Ethernet was the standard eventually developed. Using a media access and the collision detection protocol CSMA/CD, it is possible to use Ethernet to deploy local area networks of several systems. Subsequent advances in Ethernet technology have seen the development of 100- and 1000-megabit versions as well as the development of switches that reduce or eliminate many collision issues.

In some settings, Ethernet is nearly as ubiquitous as telephone wiring. While Ethernet has many positive attributes, it is generally not considered to be suitable for real-time applications. Ethernet makes absolutely no quality of service guarantees, and because the collision avoidance system is open, a station may never be given the opportunity to transmit. With the coming convergence of telephony and computer systems; and the importance of the reliable delivery of time-sensitive data, attempts have been made to update or advance Ethernet so that it can compete with systems such as

ATM. Isochronous Ethernet and the development of the RSVP protocol are two efforts that share this goal. The Ethernet model has proven to be a highly flexible base to build on.

Ethernet uses differential Manchester encoding at the physical layer to encode data. When it was first developed, Ethernet required the use of either "thick" or "thin" coaxial cabling and supported only 10-megabit transmission. This type of cabling was found to be expensive and somewhat prone to failure. As the technology matured new cabling systems for Ethernet have been developed. Modern Ethernet networks use simpler, less costly, and much more robust twisted-pair cabling.

Token ring networks, specifically the 802 token ring developed by IBM and later incorporated into the 802 standard, have good real-time properties [Kamat:95]. Token ring is a collision-free protocol where access to the network medium is arbitrated by the protocol so that it is impossible for more than one station to transmit at a time. A complete description of the protocol is contained in the 802 standard, and Tanenbaum provides a high-level description [Tanenbaum:96a].

In a token ring network, a special frame (called the token) circulates around the ring. The station that has last received the token is permitted to transmit, and after the transmission, the station passes the token to the next station in the ring. Token ring does have some very interesting characteristics, but under a low "bursty" load, token ring performs poorly. Stations that have data to transmit may wait a significant amount of time before they possess the token, and stations with nothing to transmit may be given the token unnecessarily. However, in high load situations, where every station has data to transmit, the round robin nature of the system allows for virtually full utilization without the dangers of thrashing or overloading the network.

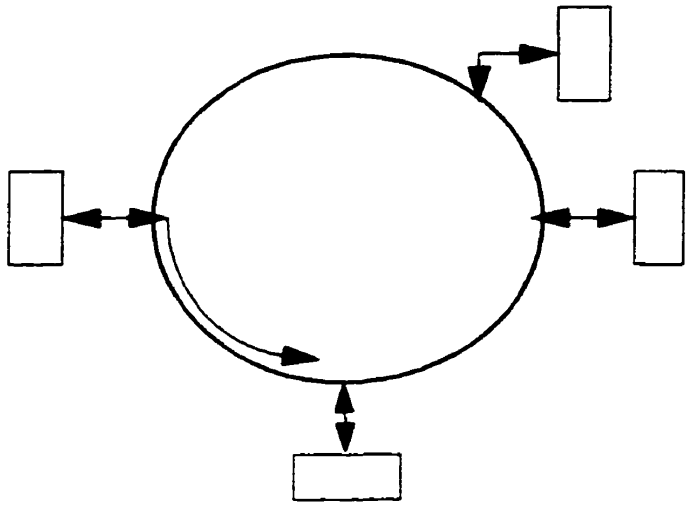


Figure 6: Token Ring Network

A station that loses or drops the token can compromise the reliability of token ring networks. If the station in possession of the token "crashes," the token will be lost and no station on the network will be able to transmit.

The real-time properties of token ring networks are in general shared between all types of token networks. Bandwidth on the network is effectively partitioned between the stations and as the token is passed between stations, each is given access to the network. While the token ring protocol guarantees that each station will eventually be given the opportunity to transmit, it does not guarantee the interval between opportunities to access the network. This may be unacceptable for certain types of real-time systems.

802 token ring networks are medium independent; they have been implemented on a variety of mediums including IBM Type-1, Category 3, and Category 5 UTP cabling, and they can be physically deployed using a ring or a star cabling pattern. 802 token ring is available in 4-megabit and 16-megabit per second implementations.

The 802.4 token bus system has also been used in real-time environments such as machine shop automation. Developed in the 1980s by General Motors, the basic premise of token bus is that the network physically uses a bus topology, but logically,

each node has an upstream and downstream neighbor. Again, a token is passed around the nodes of the network and the station in possession of the token has exclusive access to the network. Token bus is a complex protocol that has not achieved widespread use.

3.2.2 FDDI Fibre Distributed Data Interface

FDDI is a token ring protocol that is implemented on a fiber optic infrastructure, providing a theoretical bandwidth of 100-megabits per second. Fiber optic networks can span great distances (200 km in the case of FDDI) and are invulnerable to electromagnetic interference. It has been shown that FDDI networks can be used for real-time traffic [Feng:96, Feng:95, Chen:95, and Malcolm:94].

FDDI, like the 802 token ring, is a collision-free protocol. Perhaps FDDI is best understood as a series of point-to-point networks in which each station in the ring is attached to its neighbors. The FDDI token is timed and each station holds the token for a specific period of time. The advantage is that by injecting multiple tokens into the ring, more than one station can transmit at a time and station-to-station latency is bound to a specific value according to the size of the ring. The FDDI ring can be viewed as a delay line where stations are responsible for removing their own data from the ring. Designed to be consistently under full load, FDDI is suitable as backbone architecture, but not as a station-to-station burst protocol.

Although single-attached FDDI is vulnerable to the failure of a single node in the ring, two practical solutions can address this problem. The first approach is an optical bypass: simply, when a node fails, the data remains as bits of light and is passed over the failed node. Secondly, and more commonly, FDDI can be configured as two concentric and counter-rotating rings. For example, an FDDI ring operating under normal circumstances may look like the following:

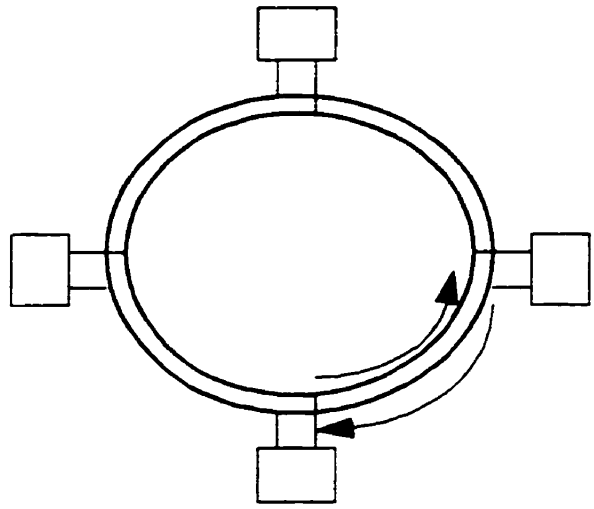


Figure 7: FDDI Ring

However, in the event of the failure of a node or link in the network, the ring automatically degrades into a single-attached ring, avoiding the failure.

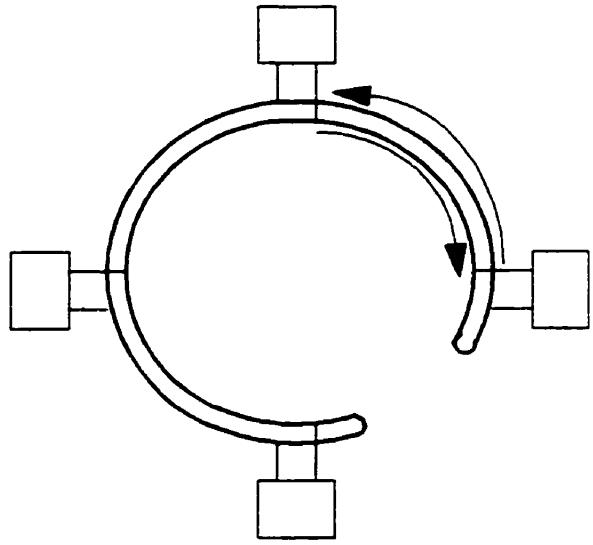


Figure 8: FDDI Ring with failed link

The practical issues with FDDI involve high latency and the partitioning of bandwidth when deployed in a large system. A typical problem in a large FDDI network is the relatively high latency and low bandwidth available to each node. Available FDDI bandwidth decreases linearly with the number of stations present on the

ring. Digital Equipment has developed an FDDI switching unit, marketed as the Gigaswitch, which is targeted at solving this problem [DEC:97]. This technology has been successful in addressing the shortcomings of FDDI.

Many people view FDDI as a technology in decline because it is expensive, complex, and fragile when compared with competing systems that are cheaper, faster, and more robust. However, FDDI is a proven technology that has known real-time properties, and it has been successfully deployed in mission- and safety-critical environments.

3.2.3 ATM

Asynchronous transfer mode (ATM) is a relatively recent development in computer systems networking. It was developed for the telephone industry, which saw the need for a digital system to carry voice, real-time data, and video signals. The fundamental unit of data in ATM is a cell which is a 53-byte data packet (of which 5 bytes are the header and the remaining 48 bytes are the payload or data). ATM provides a connection-oriented transport layer; each station that wishes to transmit requests its ATM switch to set up a virtual circuit between itself and the receiving station. This technique is called bandwidth reservation. The 5-byte header in the ATM cell contains the routing information for the cell across these virtual links. The ATM switch only has to maintain a list of input connections to a list of output connections. While ATM was designed for constant bit rate applications such as telephony and video transmission, the technology has also proven itself adaptable in computer networking systems [Larson:95].

ATM systems are available from a variety of vendors for both switch devices and interface cards for computer systems. In comparison to existing competitive systems, such as 100 megabit Ethernet, ATM is expensive. However, there is growing acceptance of ATM for use as a guaranteed service computer network [Yang:97]. ATM is a modern guaranteed quality of service network and is suitable for real-time applications [Raha:96]. Past attempts to develop computer network systems with similar properties have resulted in real-time applications such as data gathering and machine

control. In guaranteeing a quality of service, the network provides a minimum amount of bandwidth to each end of the connection and guarantees that the amount of time taken for data to traverse the network is limited. Some earlier systems that attempted to achieve this are ARCnet and token bus, and other systems, such as isochronous Ethernet and Ethernet using a reservation protocol (RSVP) have attempted to re-engineer existing common technologies.

3.2.4 ARCnet

ARCnet (Attached Resource Computer) is a networking protocol that has been used for real-time systems. Developed by Datapoint in 1968, it is suitable for small- to medium-sized networks [Contemporary:98]. Each station in an ARCnet is allocated a station number ranging from 1 to 255. A token-passing mechanism controls access to the network and the receiver acknowledges each transmission by stations on the network. The QNX operating system is an example of a network operating system that utilizes the abilities of ARCnet. QNX is a UNIX-like operating system with very good real-time properties. The addition of ARCnet to QNX has allowed the development of complex real-time network systems. ARCnet can be implemented as either a bus or star topology, or as a hybrid of both designs. There are two versions of ARCnet: a relatively slow 2.5-megabit per second system that can support up to 256 stations on a network and a faster version of 20-megabits per second that can support 2048 stations in a single network. While it is possible to use ARCnet to implement and deploy general purpose computer networks, the relatively slow speed and expensive cabling requirements mean that it is not used for general purpose applications. However, for distributed real-time systems requiring complex processing at the stations, QNX and ARCnet are an excellent combination

3.2.5 ISDN

ISDN is a networking technology that was developed to address the

increase in digital traffic on telephone networks. It has had some limited success in use for that market. For high bandwidth applications, it is superseded by ATM and for lower bandwidth applications, it is being replaced by other technologies such as ADSL (Asymmetric Digital Subsriber Line).

3.3 Computer Networks -designed for real time performance

Real-time protocols are of particular interest to this work and there are networks that have been designed to provide real time performance. The primary issues in real-time computer networks are to provide reliable data service and to ensure that data is delivered on a timely basis. There are specific purpose protocols, such as SAFENET [MIL-HNBK-818A], MIL-STD-1553 [MIL-STD-1553], and 802.4 token bus [Tanenbaum:96c], that have been designed to address real-time and safety-issues. As we have seen, general purpose protocols like token ring and FDDI have some real-time properties. There have been additional attempts to extend other protocols such as Ethernet and provide them with real-time properties.

A real-time network must provide a guaranteed amount of bandwidth and a guaranteed maximum delay for access to the network. These two aspects are referred to as the quality of service and the latency of the network.

Quality of service refers to the guaranteed performance that the network can offer its users. The reliable delivery of data is obviously fundamental to the quality of service provided by a network, but the time required for the network to transmit the data from point to point is also fundamental. This delay is called the latency of the network. Particular types of data transmissions are very sensitive to long or variable latency in networks. The simplest and most commonly affected type of data is voice. Human conversation requires that the round-trip time not exceed a delay of 200 milliseconds. For interactive media applications, it is generally accepted that the maximum latency in the communications channel can be 200 milliseconds. Addressing these issues is of fundamental importance to interactive media-carrying systems (such as satellite-based phone systems produced by Iridium, Globalstar, and Teledisc) [Teledisc:98].

Latency and bandwidth are closely related concepts. In static conditions, changes to the requirements for one will directly affect the other.

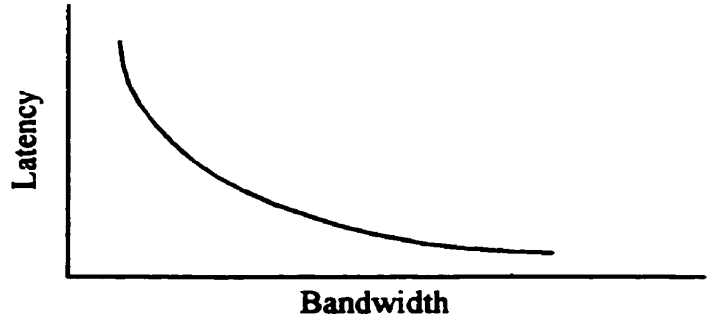


Figure 9: Latency vs. Bandwidth

Bandwidth is simply the number of bits that two stations using the network can transmit between them per unit time. There are physical and protocol-based limits on how effectively the medium may be used. In a guaranteed bandwidth network, each station is given a separate section of the bandwidth to transmit on. Time and frequency multiplexing are the two fundamental bandwidth assignment methods.

Time division multiplexing divides the time that is available to transmit into discrete units; each station is given at least one unit. Frequency division multiplexing divides a portion of the transmission spectrum into many channels and each transmitter in the network is then assigned a unique channel.

The fundamental characteristic of guaranteed quality of service networks is that bandwidth is allocated and guaranteed to the transmitter before it is needed.

Computer networks that have good real-time properties must be able to guarantee timely delivery of data within the network. The applications of these types of technologies are for the control of complex machines and the gathering of data within those machines. Many of the most complex machines ever built are for military purposes, and the technology used in these systems is useful for understanding computer networks designed for real-time systems because they were developed with cost as no object. Since the applications of these systems also tend to be quite narrow, there is little subtlety in their design and implementation. In the following section, I examine technologies that are specifically designed as networking protocols for real-time and

reliable delivery systems, and the applications of such systems

3.3.1 MIL-STD-1553

Modern military hardware such as tanks, ships, and aircraft need to transmit data around their respective systems. For example, the USAF uses MIL-STD-1553 in F-15 and F-18 fighter aircraft. The standards used for these systems are rigidly defined by the military so that multiple vendors can supply equipment that operates seamlessly together [MIL-STD-1553]. This standard has been developed for military avionics but has seen diverse applications including use in the London Underground. MIL-STD-1553 networks are centrally controlled data bus systems. Control of the data bus is handled by the bus controller module, which initiates all transmissions on the network. With this central control mechanism, bus contention is avoided. In comparison, the ARINC 429 - which is used in commercial passenger aircraft - specification calls on each transmitter in the system to have a dedicated data bus allocated to it, making bus contention a moot point [SBS:98]. The disadvantage of this system is that once in place it is difficult to modify and expensive to deploy. MIL-STD-1553 is a time division multiplexing system. The bus master device allows each station in turn to transmit information and provides command information to nodes in turn. The maximum number of stations on a bus is 30, with the 0 and 31 positions reserved for the bus master device and broadcast address.

3.3.2 FBRN - FDDI Based Reconfigurable Network

Zhao has proposed FBRN based upon FDDI for fault-tolerant real-time communications [Zhao:94]. This system uses multiple redundant FDDI connections between nodes and an advanced message routing protocol.

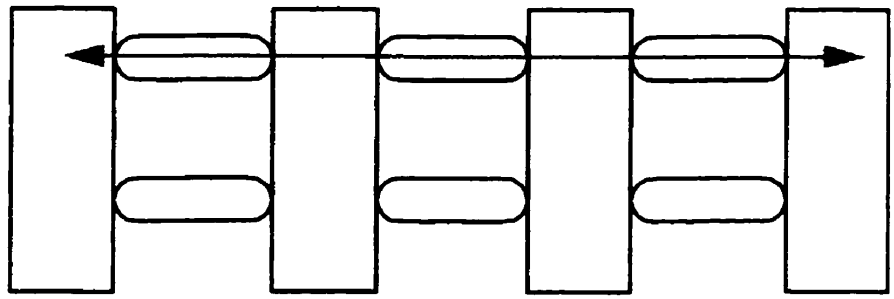


Figure 10: FBRN Network

The failure of single or multiple links in an FBRN network can automatically be re-routed around.

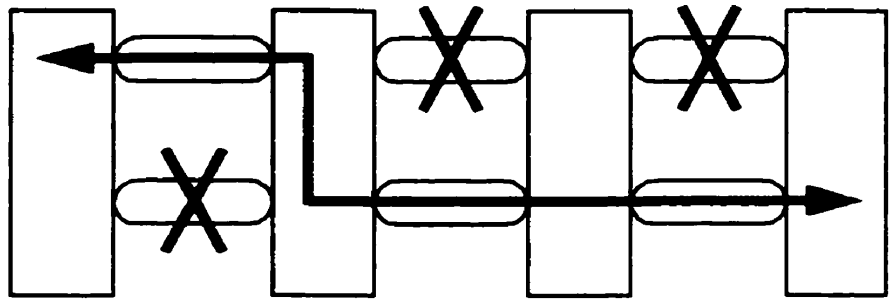


Figure 11: FBRN with failed links

The routing protocol that Zhao proposes uses fault detection and retransmission, and multiple message transmission, to meet real-time constraints. FBRN is the precursor to reliable computer networks for applications in military real-time applications.

3.3.3 SAFENET

Systems such as those found onboard navy ships require higher performance than would normally be found in an office environment. Currently, the U.S. military is using MIL-STD-2204, or SAFENET (Survivable Adaptable Fiber Optic Embedded Network), in this role [MIL-HDBK-818A]. SAFENET is physically implemented using FDDI hardware and other aspects of the system are specified as interface layers. The aspects of SAFENET that make it appealing for its applications are:

- **Flexibility** - The only interface restriction in SAFENET is the use of FDDI, and there are no restrictions on the implementation of higher levels of the protocols. The interface can be implemented as part of a larger computer system or as part of an embedded single-purpose system.
- **Profile Options** - There are many options available to systems in a SAFENET network because the protocol is designed so that devices or systems only have to meet a minimum requirement, but at their option may implement "higher" parts of the protocol.
- **Survivability** - Building on the automatic fault isolation and avoidance inherent in FDDI, SAFENET adds additional redundancy by using bypass units, which allow the network to bypass multiple failed stations.
- **Fiber Optics** - Fiber optics are lightweight, carry a high bandwidth, and are insensitive to electromagnetic interference
- **Multiple Vendors** - Because SAFENET is based upon commercial standards, there is limited propriety hardware and technology in the system. This allows components to be sourced from many vendors.
- **Co-existence** - Legacy systems that use point-to-point interfaces can be used in conjunction with SAFENET systems with no interference from SAFENET.
- **Performance** - The underlying technology of FDDI has a bandwidth of 100-megabits per second, which provides adequate performance for most applications.

The typical application of SAFENET is in warship computer support, such as that found on large naval vessels (including frigates, aircraft carriers, and submarines). Yet even with the performance derived from using 100 megabit FDDI hardware, there are some applications where performance levels are not sufficient.

SAFENET has been superseded in these areas by HPN (High Performance Network), which is implemented using ATM technology [Irey:98]

3.3.4 Isochronous Ethernet

Isochronous Ethernet is an attempt to use Ethernet infrastructure to address some real-time issues [Worsley:97]. Isochronous Ethernet incorporates a standard 10-megabit per second Ethernet channel and 96 64-kilobit per second ISDN B channels. The design supports the use of the 10 megabit channel for data communications as a regular Ethernet segment. Other services that require a guaranteed quality of service would utilize the ISDN circuits. There are many advantages of using Isochronous Ethernet as it can inter-operate with existing Ethernet equipment and cabling as well as current ISDN and telephone equipment. Isochronous Ethernet has not been well received by the commercial community; it is an example of a technology that has not interested users

3.3.5 RSVP

Not strictly an addition to Ethernet, RSVP is a protocol used in conjunction with IP that instructs the networking systems to sustain a particular quality of service between two points. It works in the following manner:

The **source** sends a Quality of Service (QoS) requirement suggestion through the network to the receiver.

If the **receiver** accepts the request, it generates the QoS parameters for the network and sends these back to the sender.

The **networking** components allocate sufficient resources to guarantee the required quality of service [Gouda:98].

3.4 Research Networks

Nakajima and Tokuda have developed a real-time networking system suitable for microkernel-based real-time systems [Nakajima:97]. The applications that they envision for their system are those of real-time multimedia systems such as the delivery of video and voice. However, this system has several inherent limitations that prevent its use as an effective real-time operating system; in particular, the networking protocol cannot guarantee a specific quality of service to the nodes in the network.

> "The limitation of the current design is that NPS provides UDP as a transport protocol. UDP/IP enables packets to be scheduled adequately, but does not provide quality of service support for applications. Thus, the system cannot control the bandwidth of respective applications." [Nakajima:98]

The networking protocol implemented results in introduced latencies of 1200+ milliseconds. For interactive multimedia, a latency of this length results in unnatural and unacceptable delays.

3.4.1 MARS

MARS (Maintainable Real-Time System) is a systems model, programming environment, and operating system suitable for implementing distributed real-time applications [Kopetz:89]. The stated primary goal of the system is to provide predictable performance under peak loading conditions. MARS components are connected to each other via a common bus and access to this bus is controlled by a TDMA, time division multiple access, algorithm. The synchronization between nodes is achieved with a clock-averaging algorithm [Kopetz:87]. MARS provides actively redundant nodes in the network to support fault tolerance, and if the system fails, it will fail silently. The network and task management in MARS systems is scheduled off-line. Kopetz describes a variant of Modula/2, known as Modula/R, that is suitable for programming systems running MARS [Kopetz:92].

3.5 Networking Technology Summary

This chapter has examined a number of different networking techniques, topologies, and technologies. As we have seen, there are trade-offs in network protocol and hardware design, and important issues must be considered when dealing with time-sensitive data.

Chapter 4 - Protocol Requirements and Design

In light of what has been discussed about real-time systems, the next step is to sketch a protocol that is characterized by bounded latency, guaranteed bandwidth, robustness, and reliability. The protocol will meet both networking and embedded systems requirements. In this chapter, I first describe the requirements for the required real-time network protocol and examine the requirements of the embedded systems. I then evaluate possible design alternatives with respect to these requirements and select an overall design idea.

4.1 Networking Requirements

The protocol needs to support time-synchronized distributed real-time multimedia applications and must provide an inexpensive and effective real-time networking protocol for embedded systems. It must also support bidirectional multimedia data. There are therefore many constraints that limit design decisions. The network protocol must guarantee that all stations in the network have sufficient network access, transmitted data must arrive reliably at its destination, and the network must handle data so that time constraints can be met. An ideal solution would allow the implementation to "tune" for desirable attributes. For example, in some applications it may be advantageous to trade bandwidth for a decrease in the latency of the system. A flexible solution that allows for these types of changes is therefore preferable to one that does not.

Multimedia applications contribute a large steady stream of data (such as audio) to the network, and delays in transmission or the loss of data result in "clicks" and "pops" being introduced into the playback. In computer networks, the data stream is normally modeled as a "burst"; each station does not generally require access to the network, but from time to time it requires virtually exclusive access. For example, in a file-serving application, the workstation node will request a file, process it, and return the file to the server. This type of transaction will result in a burst of read requests, a

long period of no network activity, and finally, a burst of write requests.

The types of applications I am addressing here do not fall into this pattern. Media applications such as contiguous audio and video have constant data requirements. For non-interactive systems, using the "burst" model will suffice; the media file is requested from the server and then played at the station. Introducing a long latency into the playing of the media can create the illusion of a contiguous data stream. For interactive media applications, however, this model does not work. In interactive media, such as in a telephone or videophone, the end-to-end latency in the system must be kept very small and large buffering of data is not an acceptable option [Teledisc:97]. To achieve acceptable interactive performance in multimedia applications, the latency in the system must be kept to a minimum.

In a media application such as a telephone, each station must also gain sufficient access to the network or risk losing data or falling out of sync with the remote station. As each node in the network collects data, it must transmit that data to the receiving node. If access to the network is blocked for some reason; or if the network medium is incapable of carrying all of the data, the buffering on the node will either result in an unacceptable delay being introduced or data loss. Neither of these situations is acceptable. Nodes in an interactive real-time network must also have contiguous and sufficient bandwidth on the network so that they can transmit the required data when necessary.

If any of the minimum conditions of the networking protocol are not met, unwanted artifacts occur in the data stream. Artifacts such as "clicks" and "pops," missing or late data, or the unsynchronized playback of data must be avoided. These types of artifacts result in unacceptable audio quality. In the case of safety-critical applications, they may be disastrous. Users of safety-critical systems are expected to interact with each other over the system using natural conversation, and interaction happens very quickly. Humans have a very low tolerance for delays introduced into conversation, as anyone who has experienced a poor-quality long-distance line can attest.

In contrast to traditional computer applications, where, due to the data bursts, stations have random high data requirements, media applications have continuous and steady data flows. They also have various (but wide) bandwidth requirements.

Acceptable audio may require as little as a few tens of kilobits per second, whereas high-quality video may require several hundred megabits per second of bandwidth. The networking protocol design should be able to support the bandwidth requirements for a complete range of applications. The latency of the protocol should be low enough that there is no perceived echo or delay in the system, since humans can perceive latencies of a few hundred milliseconds as an undesirable echo (in applications where the participants interact with each other, this leads to broken speech and disorientation). For safety-critical applications, the loss of natural speech rhythm is a serious issue.

The protocol design should also address safety-critical applications. Guaranteeing service in the event of a node failure is a necessary requirement in designing the protocol. It should be possible to detect the failure of a node in the network, and to replace the failed node without operator intervention. The creation of safety-critical distributed real-time systems will require this type of functionality before they can be used in situations where peoples' lives may be at risk.

In summary, this network protocol must meet numerous requirements. It must have low latency, and the streams of data coming to and from positions must be steady and of good quality. The bandwidth provided to each station has to be sufficient to support multimedia applications at data rates of a few kilobits per second to several megabits per second. The ability to send and deliver data on time must be guaranteed. If possible, this protocol should meet the requirements of safety-critical applications. The protocol must be able to detect the failure of nodes and provide a means for other nodes to take over; avoidance of single points of failure should be built into the design.

4.2 Embedded systems requirements

This protocol is to be applied to embedded systems where cost is often an issue. Cost in embedded systems can be measured in many ways. The cost of design and implementation is represented by the complexity of the design and the amount of time required to implement it. The choice of hardware and software is also important; for example, in systems that use off-the-shelf components, the cost of the design and implementation may be quite small, but the purchase of sufficient licenses to produce

large quantities may make the software cost too high.

Quite often, the most expensive component of an embedded system is the processor. If the CPU requirements can be kept low enough, designers can use a lower performance unit and reduce the overall cost of the system.

Finally, the cost of maintaining the system over the long term includes the availability of components and the ability to upgrade components used in the implementation. Both software and hardware must be maintained. Good software design will allow the software to be maintained over an extended period, and the design should be flexible enough that designers can select specific hardware technology as late as possible in the design process, enabling them to choose the most cost-effective alternative.

To summarize, in order to be useful for embedded systems, the protocol must be simple, cost effective, and effective in the use of resources.

4.3 Networking protocol design alternatives

Many protocols have been designed specifically for real-time networking. In this section, I examine the competing paradigms in order to choose a design paradigm for the embedded system and real-time networking protocol to be implemented.

When designing a protocol, you need to examine competing paradigms to determine where the requirements of the system are most closely aligned. The basic families of computer networks are collision detection multiple access (CDMA), frequency division multiple access (FDMA), time division multiple access (TDMA), and star networks [Tanenbaum:96]. These possible paradigms need to be examined in relation to the real-time networking problem.

Collision detection multiple access networks are common. Ethernet is probably the most commonly used local area networking technology. With Ethernet, each of the computers shares a common medium and transmits whenever it has data, and specialized circuitry in the network hardware detects when two stations have transmitted at the same time. These systems have the merits of being simple and inexpensive.

Frequency division multiple access is a system in which each of the nodes in the network is assigned a particular part of the frequency spectrum for its exclusive use. This type of multiplexing is widely used for the distribution of analog media such as radio and television. Computer systems do not normally use FDMA because it does not support the burst data model well. FDMA does have an advantage over CDMA and TDMA in that more than one station can transmit at a time.

Time division multiplexing is a networking standard in which only one station on the network is allowed to transmit at a time. TDMA systems impose an order on the systems in the network and each station is given explicit permission to transmit and explicitly releases the resource of the network. TDMA has many advantages for real-time applications, and most real-time networking systems in use are TDMA based.

Star networks are also commonly used for time-critical applications. These networks give a dedicated channel to each node in the network and employ a complex switch in the center of the network to manage the traffic. Examples of this type of application are ATM and switched Ethernet.

ARINC 429 is a star networking system used in embedded systems such as those onboard aircraft. Each transmitter has a dedicated bus that it may transmit on with up to twenty stations listing on that bus. Obviously, the amount of wires and cabling required to achieve this can be massive.

4.3.1 Properties of CDMA

The most common example of CDMA technology, as has been mentioned, is Ethernet. Ethernet will remain a viable method for interconnecting computers on a LAN for the foreseeable future. With some care, it is possible to build Ethernet networks so that they are robust in the event of the failure of one or more components. The Ethernet standard also supports the concepts of broadcast and multicast.

Ethernet has not been a success for real-time networking; many of the properties that make it ideally suited as a networking standard for computers are precisely the reasons that it is poorly suited as a networking standard for real-time

systems. Ethernet handles bursts of data very well, but is not very successful at handling continuous data streams. The Ethernet standard allows data to be lost and corrupted, with the recourse being retransmission or letting the higher levels of the protocol deal with data recovery. This means that data may never arrive (or arrive with an unknown latency from the source) or that data in a stream arrives out of order. These types of problems will pose fatal problems for real-time applications.

In an Ethernet network, each workstation is capable of using the complete bandwidth of the network; however, this also means that none of the workstations have a guaranteed bandwidth. One interesting property of Ethernet is that it supports the concept of multicast, a broadcast that is only processed by a select set of nodes on the network.

Ethernet, and CDMA networks in general, are an ideal standard for allowing a relatively small number of computers to share data over a LAN. However, for supporting real-time applications, it fails on all of the selection criteria. Each transmitting station in an Ethernet network is responsible for properly arbitrating the shared network.

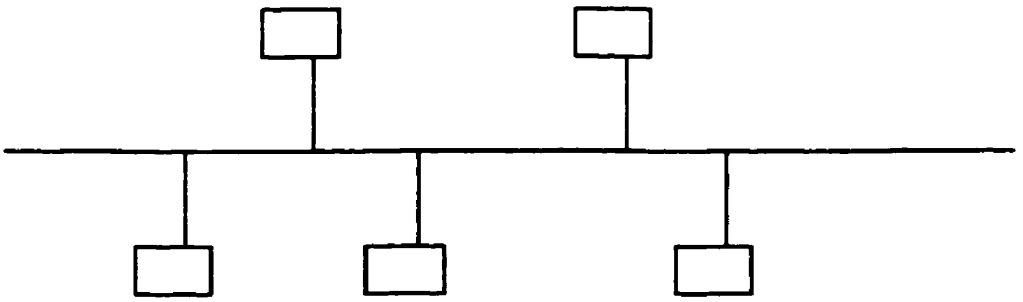


Figure 12: Shared Bus Ethernet Network

Ethernet has some properties that make it good for embedded systems development. Ethernet is ubiquitous and is available for virtually every platform. Support for Ethernet is part of many operating systems, including many real-time and embedded operating systems. The hardware and software required to implement it is very well understood, and the hardware required to implement Ethernet is very inexpensive for the potential performance it offers. Ethernet can provide a variety of data

transmission levels in the form of 10, 100, or 1000-megabit interfaces, allowing the embedded systems designer to choose appropriate cost and performance levels for the specific task at hand. Bus networks, such as Ethernet, are very robust in the event of the failure of a single node in the system.

4.3.2 Properties of FDMA

Frequency division multiplexing uses a common medium to carry multiple signals at the same time with a specific frequency range allocated to each signal. This bandwidth reservation guarantees quality of service connections. A highly effective means for sending large amounts of analog data from a central host to a number of stations, this system is used for television and radio. FDMA technology has not been exploited into a LAN environment. While it is suitable for real-time applications in that the signals are continuous and have low latency, it is difficult to recover from a failed node. There is little or no digital hardware available to exploit this technique, so building an embedded system with it would be an extremely expensive and time-consuming enterprise. In the FDMA paradigm, the receiver controls the arbitration to the medium.

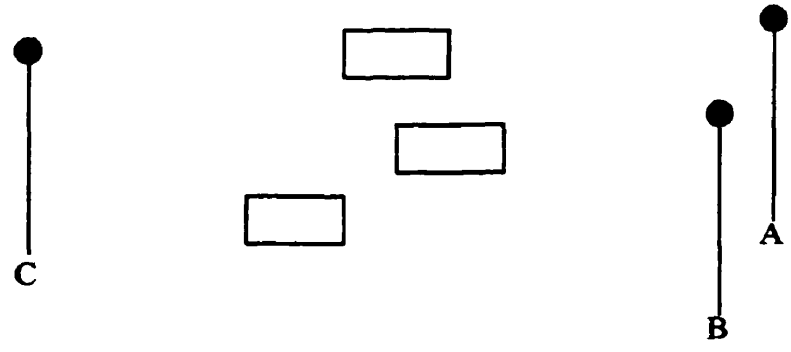


Figure 13: FDMA Multiplexing

FDMA is not considered to be a viable option for connecting systems transmitting digital data over a local area network. Because of the static nature of the bandwidth allocation, it performs very poorly in an environment that has burst data requirements. In a bandwidth switching scheme, such as that used in digital cellular telephones, each station is not guaranteed timely access to the network.

4.3.3 Properties of TDMA

Time division multiple access techniques are also bandwidth reservation techniques, in which each station on the network is given an interval during which it may transmit. Each node must have a means to ensure that it does not transmit at the same time as other nodes in the system. One possible method of solving this problem is to have the stations pass a "token" around. For example, in an 802.5 token ring, each station is only allowed to possess the token for a specific period of time, and only stations that possess the token are permitted to transmit. These types of networks have very good real-time properties. Latency is bounded and since each station gets to transmit in turn, each station can utilize the full bandwidth of the network. As well, since only one station can transmit at a time, data delivery is guaranteed. However, the failure of nodes and links can be a problem in token ring networks, and if the token is lost, its regeneration can be time consuming. The hardware to implement token ring networks, in comparison to Ethernet, is quite expensive. Token ring networks have good high load bandwidth capacity, but it is at the expense of imposing a comparatively high latency on the network.

The other common implementation of the token ring method of TDMA is FDDI. FDDI has some of the advantages of token ring and has been used for real-time networks. However, it is expensive and latency is variable between nodes. Because FDDI is actually a series of point-to-point connections and not truly a ring, the latency between any two nodes depends on the number of nodes between them. The nature of token ring systems prevents broadcast and multicast from being implemented since each station is aware only of its neighbors and not of the complete network. Ring networks are also susceptible to a single node failure, and in this event, the ring may break and the network may fail. The transmitting station controls access to a TDMA network.

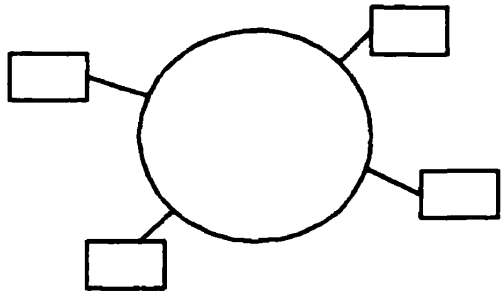


Figure 14: TDMA Network - Token Ring

It is possible to implement TDMA on other topologies such as a bus. In these cases, as in an 802.4 token bus, the systems have been implemented primarily for real-time machine shop control, and the resulting standard is very complex. The token bus standard is over 200 pages long and the implementation of the protocol is very complicated; further, the physical layer implementation is incompatible with Ethernet. Token bus is reliable in the event of node failure, although a significant amount of time may be required for token regeneration

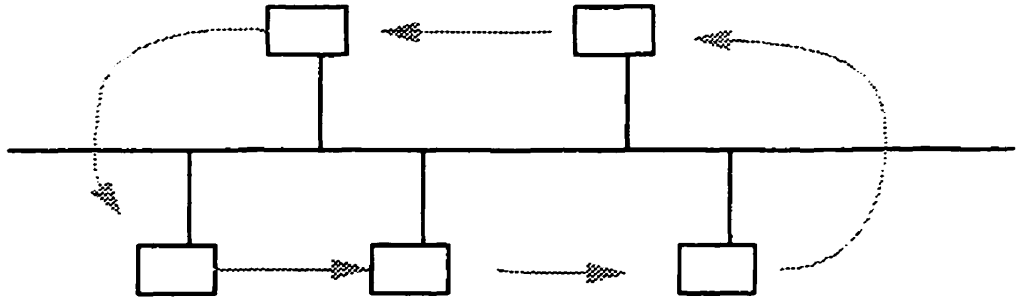


Figure 15: TDMA Network - Token Bus

4.3.4 Properties of Star Networks

Star networking is a reservation method accomplished by brute force. Each system in the network has a dedicated cable providing the full bandwidth of the

network, and the center of the network consists of a large, powerful, and dedicated system that manages the traffic between nodes. Systems such as this, like ATM or fully switched Ethernet, have many advantages for real-time applications including low latency, guaranteed bandwidth, and very high reliability of packet delivery. The central hub station controls arbitration and access to a star network.

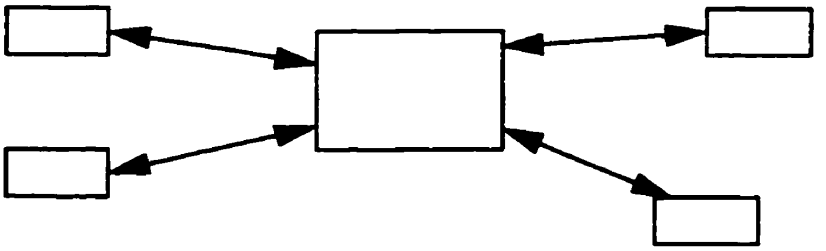


Figure 16: Star Network - Switched Ethernet

A disadvantage to star systems or cable multiplexing is that if the central node fails, the entire system fails with it; the only way to achieve automatic failure recovery in the system is to provide redundant links to each system in the network. Moreover, it would also be difficult for any station, other than the central one, to passively detect a station's failure and replace it automatically.

4.4 Design Decisions

It is now possible to describe the requirements and desirable properties of the real-time network protocol that is to be implemented. The requirements call for a system that has low latency, is reliable and robust, has sufficient bandwidth for each station to support multimedia applications, and can guarantee the ability to send and deliver data on time. The protocol should also address safety-critical applications and failure recovery.

Frequency division multiplexing would be an appropriate real-time networking paradigm, since it allows for low latency between sender and receiver. As each station has been assigned a part of the spectrum, the bandwidth is guaranteed and it

is possible to passively detect when a node has failed. However, this paradigm has not been implemented as a local area network and new technology and hardware would have to be developed, resulting in a very high cost.

Most of the system's real-time attributes could be met with a conventional TDMA system such as token ring, with the very important exception that latency in a token ring network is not constant. Stations that are far apart on the ring have large latency as the transmission of data must wait for, and in some cases pass through, all intermediate stations. The one constraint that could be met with a conventional TDMA system is the equality of stations, where each station would have sufficient bandwidth using a 16-megabit per second token ring or 100-megabit per second FDDI. But TDMA fails to meet other constraints, notably cost, robustness in the event of failure, and keeping the latency of the system low enough for interactive media applications. The other method of TDMA, token bus, is too expensive and too complex to be considered for the application here.

CDMA systems such as Ethernet do not, in general, meet the real-time requirements of guaranteed access to each system on the network and reliable data delivery. However, CDMA systems can meet many of the other requirements, such as low cost, low transmission latency, and high bandwidth. They can also detect a node that has ceased to transmit.

Star networks have many real-time advantages and many modern telecommunications and local area networks are built in this fashion. They allow each station equal bandwidth and have very low latency from station to station. However, the failure of the central switch node would result in the collapse of the network, and because it is not possible to passively determine that a particular node has failed, node replacement is difficult. The cost of the central switch node also makes this type of network prohibitively expensive for small systems.

The following table compares the most important features of the competing networking paradigms for this project:

Table 5: Comparison of Networking Paradigms

	Real Time	Safety Critical	Low Cost
FDMA	Yes	Yes	No
TDMA	Yes	No	No
CDMA	No	Yes	Yes
Star	Yes	No	No

From this analysis of possible options, we can conclude that none of the existing networking paradigms meets all of the constraints proposed for our networking system. Given this situation, we could relax the constraints until one of the competing paradigms is able to meet our requirements. The first constraint that could be relaxed is the safety-critical requirement, since this attribute is not necessary for the initial system. Yet even with this constraint removed there is still no clear winner.

An alternative paradigm approach to the problem is to attempt to implement a hybrid solution. The only system that has low enough hardware costs is CDMA (in the form of Ethernet) so we are compelled to choose it as the hardware platform. Ethernet at its lowest form is merely a method of putting information onto a wire and retrieving that information. It should be possible to use Ethernet as a low-level means to move data around and to implement a TDMA protocol on top of it. The selected solution is therefore a hybrid design that uses the low-cost hardware of Ethernet, but abandons the CDMA paradigm in favor of TDMA. This will result in a low-cost solution that has the appropriate real-time and safety-critical characteristics.

Several advantages are gained by using Ethernet hardware, such as the ability to choose 10-, 100-, or 1000-megabit mediums and the ability to broadcast or multicast on the medium.

4.5 High Level Protocol Design

The primary challenges for this network protocol are to ensure that no two stations transmit at the same time and that each station gets a timely opportunity to transmit. There are two basic methods for achieving these goals.

First, a token may be passed around the network and whichever station possesses the token is given exclusive access to the network. This is an effective system for sharing network resources and is, for example, the system used by 802 token ring and FDDI networks. The drawback is that token passing introduces an unbounded latency into the system. The initial generation of a token, and the regeneration of a lost token, can take a long time and can be complex. Since a stated goal of the desired system is that it have bounded latency, this solution is not acceptable.

The second approach to the transmission issue is to divide the network into a series of time slots, with each station being given a set of slots in which it may transmit. The system can be controlled by interrupt generating timers, which are commonly found in computer systems. This does not result in system overhead in processing a token, or in the additional complexity of token generation or regeneration.

While the token-passing method does guarantee that each station will get the opportunity to use the network, there is no guarantee concerning how long a node will have to wait before it has access. The windowing method, on the other hand, does guarantee that every station on the network will get a specific amount of bandwidth within a latency. However, there is a limit on the number of stations that can be present on the network. In the windowing method, the individual systems in the network do need to have precisely synchronized clocks. The clock synchronization algorithms discussed in Chapter 3 are not appropriate for this type of environment because the class of broadcast and average algorithms do not provide sufficient accuracy to synchronize the transmission of packets in the microsecond range. Systems that can synchronize clocks to the required precision, such as those based upon using a GPS receiver, would require expensive hardware at each node and are not reliable enough to be used in safety-critical applications. The solution to this dilemma is that since the clocks in the stations only

need to be in sync relative to one another, the absolute time scale, as is provided by a GPS-based system, is not relevant for the application of these systems. A single node on the network that broadcasts a synchronization or "mark" pulse on the network. This will cause all stations to reset their internal timers to a relative time of zero is all that is required.

Given that the interval between mark pulses is the latency of the network, it is now possible to construct a model of the network. There is a relationship between the latency, bandwidth, and maximum number of time slots available, and therefore the number of stations.

Notation:

S - Slots available on network

L - Latency (frequency of mark pulse)

$\mathbf{B_T}$ - Total Bandwidth of network (bits per second)

C - Sample per slot

D - Bit length of samples

$$S = \left\lfloor B_T / \left(\frac{(C \times D)}{L} \right) \right\rfloor$$

A central or master node that sends out a synchronization or "mark" pulse coordinates the nodes. Each node in the network is given a specific and fixed time period (after the mark) in which it may transmit.

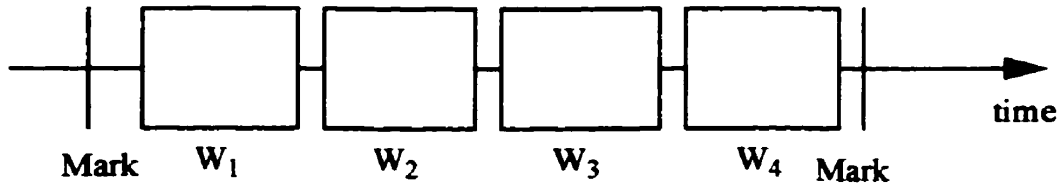


Figure 17: Example Mark Pulse and Assigned Windows

Each node accesses the network by being given exclusive access to the shared medium. Since the total amount of bandwidth is limited, and because each node

in the network is pre-assigned a specific amount of bandwidth, the total number of transmitters in the network is also limited. By changing the amount of delay between mark pulses and the amount of time that each node is permitted to transmit, the latency, bandwidth, and number of stations that can participate in the network can be controlled.

The desired properties of the protocol are that it have bounded latency, guaranteed bandwidth, robustness, and reliability. The protocol described here meets these criteria.

• **Bounded Latency**

The frequency of the mark pulse describes a window in which each station has an opportunity to transmit. This frequency also represents the maximum delay that a station will have to wait to access the network.

• **Guaranteed Bandwidth**

Each station in the network is given a window of time (between each of the mark pulses) in which it may transmit, and no other station is permitted to transmit in that window. Thus the station is guaranteed a specific bandwidth on the network.

• **Robustness**

Because there is no opportunity for stations to interfere with one another, the transmitted data cannot be corrupted by collision.

• **Reliability**

The data transmitted on the network will reliably arrive at the destination.

Two types of stations participate in the network: a master station that is responsible for periodically issuing a mark pulse, and the nodes that participate in the communication. The control loops of each of these types of nodes can be described in pseudo-code.

Master Node - p code.

```
Task 1.
On the event of the mark timer going off.
Reset mark pulse timer.
Send mark pulse
Repeat
```

The client node can similarly be described:

Participating Node - p code.

```
Task 1.
Wait for mark pulse to arrive
From the local node number calculate time of transmission
window
Set transmission timer for window.
Task 2. On the event of the transmission timer going off
Send data.
```

By giving a central controller responsibility for coordinating the participating nodes we have altered the access control model of standard broadcast mediums such as Ethernet. This centralized control allows us to impose the necessary real-time properties on the protocol.

Chapter 5 - Design of Required System

This protocol is suitable for the development of a complex audio system to be used as an air traffic control (ATC) simulation system, with the expectation that the paradigms developed here could also be used as a template for an operational ATC system and other interactive multimedia systems. Air traffic controllers have very demanding roles. Even under relatively simple circumstances, they need to listen to and communicate with a wide variety of sources. For example, is not uncommon for a controller to listen to several air traffic radios and to communicate with other air traffic centers over ground lines. Each audio source needs to be presented to the controller clearly and in a timely fashion. Moreover, there can be no delay and there must be a minimum of interference over the audio channels.

The protocol that is to be implemented is meant to support ATC voice terminals. ATC voice terminals must support a user interface of some kind and must be able to transmit and receive audio data from other terminals and sources. ATC systems need to provide telephone-quality audio to each station. There can be no perceivable delay in communication between any combination of systems on the network and the system must not introduce artifacts into the audio. For a minimal system, each station must be able to handle at least eight audio sources. In addition, the system must:

- Support bidirectional 8 kHz audio (56 8-bit samples, each of 7 milliseconds) at each position;
- Have a maximum latency of less than 200 milliseconds;
- Provide for the guaranteed delivery of data as well as guaranteeing that each node is serviced frequently enough so that the latency requirements are met; and,
- Be deterministic, in that the calculated values must not change in the face of system load.

This is a real-world problem that this work is attempting to address. As with many real-world problems, cost is also an issue that needs to be considered. In addition to the cost of the time required to implement and integrate the solution, there is the cost of maintaining and improving the solution and, in the case of commercial

devices, there is the cost of building and deploying the solution. These cost factors must all be taken into account in the design phase.

To summarize, these are the networking requirements of an ATC audio network:

- **Bandwidth per station**

  8 kHz per station bidirectional constant data

- **Maximum system latency**

  Below 50 milliseconds end-to-end.

- **Reliability**

  Data must not be lost or corrupted in the transmission process.

- **Robustness**

  Must be possible to detect the failure of either the position node or the central master node.

$$S = \left\lfloor B_T / \left(\frac{(C \times D)}{L} \right) \right\rfloor$$

$$S = \left\lfloor (10000000) / \left(\frac{(56 \times 8)}{0.007} \right) \right\rfloor$$

$$S = 156$$

Time windows available on a 10-megabit Ethernet
with a 7ms mark & 128 byte packets

The system design uses a central node to provide the synchronization pulse that is broadcast on the network; this node is also responsible for combining or mixing the required output audio signals. Excluding cabling delays, which are negligible, the relative time on each of the position nodes will then be precisely synchronized. Each station will be allocated a selection of time slots during which it is expected to transmit,

and series of time slots that data for it will be available from the network. The initial synchronization pulse is called the "mark" pulse, which indicates the beginning of a new time window and the termination of the previous one. The window of time between two successive mark pulses is the latency for a packet in the system; for example, a mark pulse generated at 8 kHz will result in latency of 7 milliseconds for a station to have the opportunity to both transmit and receive. Each transmission and receive window is guaranteed bandwidth to each of the stations. Thus both latency and bandwidth become a function of the frequency of the mark pulse.

A significant advantage of this protocol for real-time systems in general, and for safety-critical systems in particular, is that the failure of individual nodes can be passively detected, even if it is the failure of the central node providing the mark pulse. Merely by collecting a profile of the network, an extra redundant station can detect the failure of a station and take over for it without user intervention.

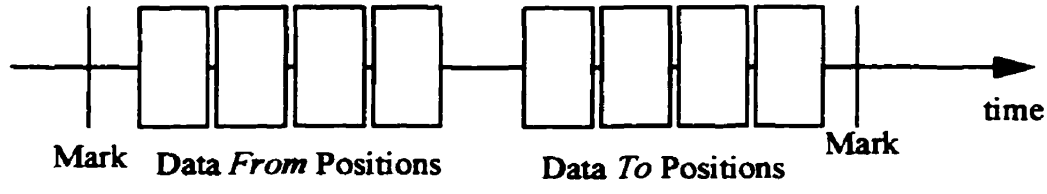


Figure 18: Network Trace

Several practical issues emerge when looking at this protocol design. There are two distinct types of nodes in the network: a master node that provides the synchronization "mark" pulse to all other nodes in the network, and nodes that receive the pulse and then select an appropriate window in which to transmit. The arbitration of which node transmits in what time slot is predetermined, with each node in the system having a position in the network indicated by a small integer value. For example, the station given the third position in the network would receive the third set of transmission windows. The best way to determine how this protocol will work in practice is to imagine frames being put onto the Ethernet wire over an interval between mark pulses.

The theoretical model outlined above does not, however, allow for some

real-world concerns. Ethernet hardware normally detects the occurrence of a collision, and in order for it to do this, there does need to be a brief period of network silence at the end of each packet. This value will be determined experimentally in Chapter 6. The additional practical factor is jitter. Jitter is the delay in processing an interrupt on the system, and affects how closely the packets can be placed together on the network. The jitter in the network protocol is the summation of a number of factors, including the drift in the timers, whether the interrupt handlers are in cache memory, the length of cable between nodes, and the design of the code. Jitter in the system does significantly affect the performance of the protocol.

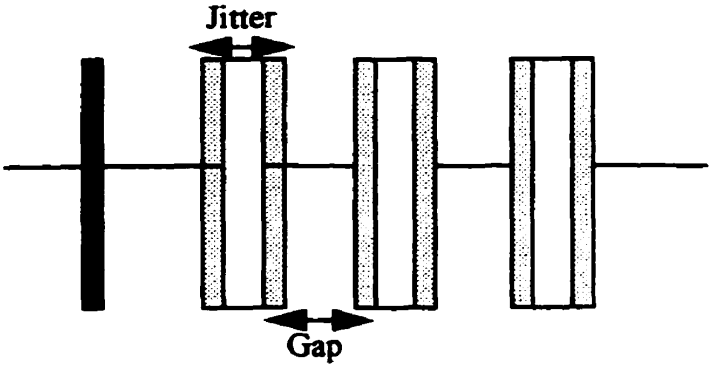


Figure 19: Inter packet Gap and Jitter

The theoretical calculations have included the practical matters of inter-packet gap and jitter.

Notation:

S - Slots available on the network

L - Latency: time between start of mark pulses (seconds)

B -Total bandwidth of network (bits per second)

C - Payload per slot (bits)

J - Jitter in system (seconds)

G - Required gap between packets (seconds)

M - Mark pulse length (bits)

O - Overhead length per slot (bits)

Developed Model of Real Time Network

$$X_B = B_T \times X_D$$

$$S = \left\lfloor (B_T - M_B - G_B - O_B) / \left(\frac{(C \times D) + J_B + G_B + O_B}{L} \right) \right\rfloor$$

Notation

- **S** - Slots available on network
- **L** - Latency (Frequency of Mark Pulse)
- **B_T** -Total Bandwidth of Network (Bits Per Second)
- **C** - Sample Per Slot
- **D** - Bit length of Samples
- **J_D** -Jitter in system (Seconds)
- **J_B** - Jitter in system (Bit Length)
- **G_D** - Required Gap between packets (Seconds)
- **G_B** - Required Gap between packets (Bit Length)
- **M_D** - Mark Pulse Duration (Seconds)
- **M_B** - Mark Pulse (Bit Length)
- **O_B** - Bit Length of Overhead per Packet

The safety-critical features of the system also need to be designed at this point. The primary type of failure that the safety-critical aspects of the system are attempting to cover is the complete failure of a node. Because of the broadcast nature of Ethernet and the highly structured nature of the network protocol it is possible to passively detect when a station has gone off-line. In the same manner, it is also possible to detect the failure of the master node. To ensure the safety-critical features of a system implemented with this protocol, it is necessary to design nodes that merely listen and wait for a node of a particular type to go off-line and then step in and take its place.

In an ATC system, each of the position nodes collects audio data and sends it to a central node where it is processed and mixed together. The central node then sends out all of the resulting audio to the position nodes, where the audio is played on a combination of speakers and headsets.

In the network model we have designed here, there is no direct data transmission between position nodes; data is only transferred between the central mixing node and the position nodes. To that end, the time window has been divided in half, with the first half for the position nodes to transmit data to the mixer, and the second half for the mixer to send the output data back to the positions to be played.

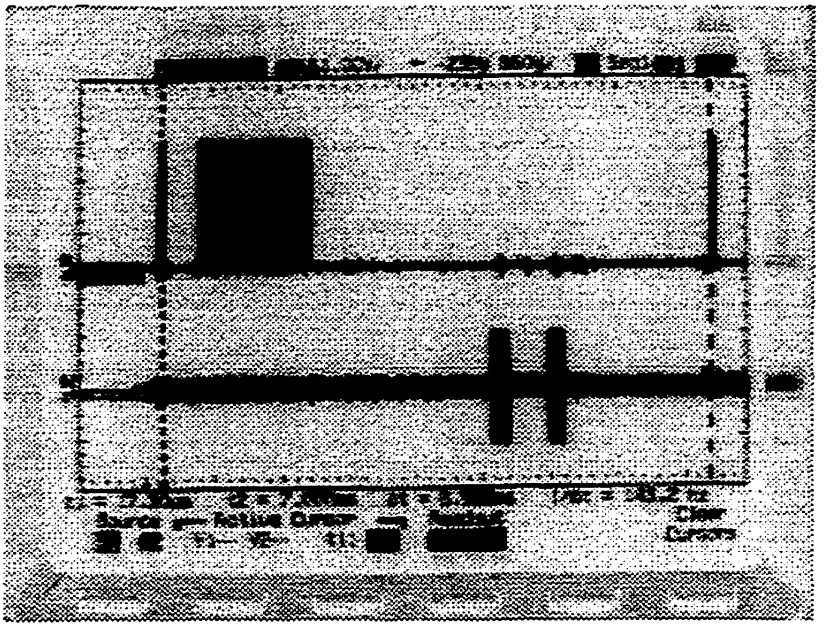


Figure 20: Network Trace of ATC system - Photograph

From a high level, the design of the ATC voice network looks like this:

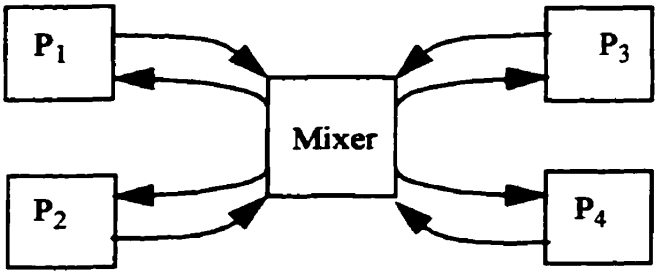


Figure 21: Data Flow in ATC system

The central master/mixing node coordinates the actions to the position nodes, controlling when they are permitted to transmit. The position nodes each receive an individual data packet for each interval from the central node.

The detailed design of the system is presented in the form of pseudo-code. There are several different tasks outlined and they will all run on a variety of nodes in the system. There are detailed designs for the tasks on the master and position nodes, as well as designs for the backup master and position nodes. The design of the actions of

the nodes in the system can best be understood in the tasks that they have to perform.

ATC Master Node - p code.

Task 1.

On the event of the mark timer going off.
Reset mark pulse timer.
Send mark pulse
Repeat

Task 2.

On the event of the mark pulse going off
Set transmission timer for 1/2 of the mark period
Repeat

Task 3.

On the event of the transmission timer going off
Send all frames to position nodes.

Task 4.

Do any necessary housekeeping functions.

ATC Position Node - p code.

Task 1.

Wait for mark pulse to arrive
From the local node number calculate time of transmission window
Set transmission timer for window.

Task 2.

On the event of the transmission timer going off
Send frame to audio mixer node

Task 3.

Perform any necessary housekeeping functions.

ATC Backup Master Node - p code

Task 1.

Wait for mark pulse to arrive
Re-set watchdog timer for slightly longer than mark period.

Task 2.

On the event of the watchdog timer going off
Restart node as master node.

ATC Backup Position Node - p code

Task 1.

Wait for mark pulse to arrive
Examine node mask for any missing nodes
If any are missing, restart self as that node.
Set node mask to clear.

Task 2.

On the event that data is received from any node
Set mask bit for node to on.

Each node will be allowed to meet the real-time constraints that have been laid out for it. The primary challenge for this type of system is to ensure that no two stations transmit at the same time and that each station has an opportunity to transmit. The design presented here meets this challenge. It has also been seen that the broadcast nature of Ethernet can be an advantage for the purposes of synchronization and redundancy. This design requires that at least two different types of nodes be used in the network. The master node is responsible for providing the synchronization or mark pulse to the network. The position nodes use the reception of the mark pulse to synchronize the relative times between them and to select unique transmission windows. The tasks required for each type of node to implement the network protocol have been outlined. Each type of node can be backed up by listening for the failure of a running node. The design of the backup nodes also addresses the safety-critical aspects of the system.

Chapter 6 - Details and Results of implementation

In the previous chapters we have seen the evolution of this system from needs specification and paradigm selection to system design. This chapter presents the details of the implementation. The task for this implementation is to transmit and receive 56 bytes of audio data every 7 milliseconds. Each of the position nodes is to send the data to a central node, where it is processed, and then sent back to each node in the system for it to play. As seen in Figure 22, the pipeline of three layers in the central node results in a total latency of 3 x 7 milliseconds (21 milliseconds) in the system. The human "threshold" for detecting noticeable echo in a communications channel is in the order of a few hundred milliseconds. If the system meets these performance requirements, the audio will be deemed to be without delay and the system will be a success.

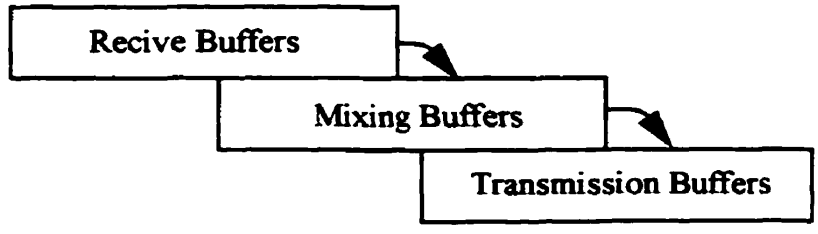


Figure 22: Data Pipeline in Mixer

The computational tasks on each node, in addition to the networking requirements, are complex. The position stations run a graphical user interface. The mixer node is required to combine multiple audio sources, at different volumes, and to provide the results in a timely fashion.

6.1 Previous Attempts

The system described here was not the first attempt at such a system. The initial attempts at designing this system were to implement it on a standard PC using standard audio hardware. Linux was chosen as the operating system platform because of

its excellent programming interface and the ability to get very close to the hardware and still maintain a good high-level interface for the user. This first attempt did demonstrate the proof of concept of the implementation and it initially appeared that the audio quality would be acceptable. However, with further testing it became apparent that the network was chaotic and that traffic could not be effectively managed with more than a trivial number (2) of transmitting nodes on a network.

In an attempt to deal with the traffic management problem, the system was re-deployed as a star connected network. This solved the traffic management issue, although the processing time imposed by the central node increased the latency to an unacceptable level. It was considered that a more advanced central node, such as a dedicated Ethernet switch, could be an effective solution. The investigation of the capabilities of such devices was illuminating. The highest-performance unit examined, the Cisco Catalyst 5000 [Mandeville:96], had a worst case switching time on the order of a few tens of milliseconds (70), which induced an unacceptable additional 140 milliseconds round-trip latency into the system. In addition, the cost of the Ethernet switch would make the system unacceptably expensive. The realization was that the 802.3 Ethernet protocol was designed for carrying a significant amount of data that did not have specific timing requirements. In parallel to this, it was determined that the PC platform was not appropriate for this application. While it is possible to implement real-time systems on PC platform systems, the granularity involved in this system drove the hardware too far. The complexity of the modern PC was the limiting factor: many of the actions in a PC are taken by the hardware directly and cannot be effectively disabled. These actions may only require a few milliseconds, but taken together, they were enough to violate the real-time requirements of the system.

From these initial attempts, it was concluded that general purpose computers have too many variables in them to be effective fine-grained real-time systems, and that for the application outlined above, the general purpose platform would not be cost effective. Conventional networking technologies, while able to carry the bandwidth, could not meet all of the real-time requirements.

6.2 Implemented Hardware

It was decided that custom hardware was needed to achieve the necessary real-time hardware requirements and that a custom communications medium was needed to achieve the real-time software requirements. Several people undertook this project, with the author of this work assigned the task of designing and implementing the real-time network system. Others in the group were responsible for the high-level software and the design and implementation of the hardware. The high-level software is not relevant to the description of the implementation, so it is not included in this discussion. Since the real-time aspects of the software are closely tied to the hardware, a brief outline of the implemented hardware is warranted.

The design requirements for the hardware in this project were those of simplicity, cost, and performance. The design had to be simple enough that it could be implemented and easily understandable. It also had be a low-cost solution, so the cost of the components was a major concern. Finally, the computer system had to be powerful enough to accomplish the task set out for it.

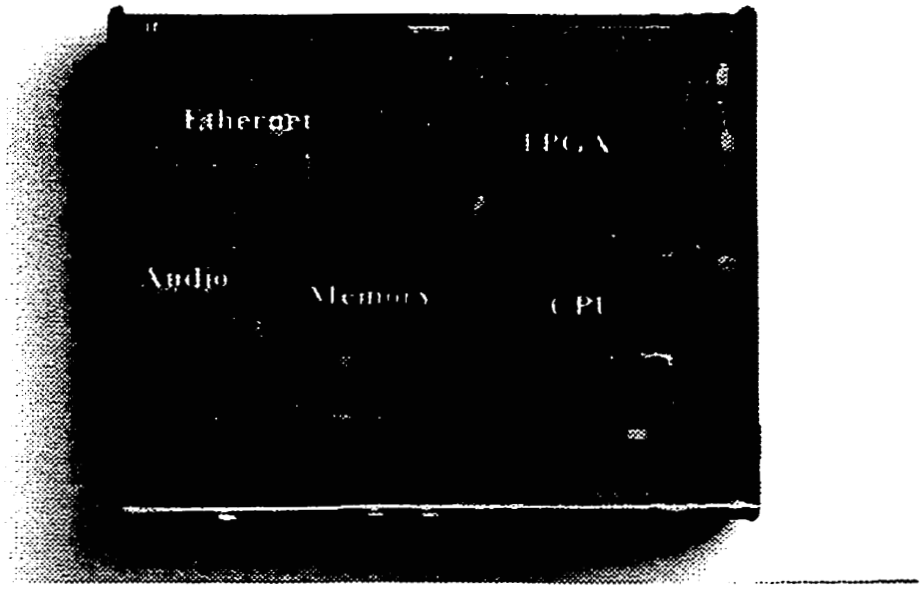


Figure 23: Annotated Photo of Hardware

The core of the computer is the microcontroller, a Motorola Coldfire. It consists of a 68020 CPU core, two serial ports, several timers, and a small amount of memory. In addition, much of the required hardware for interfacing to memory and computer buses are built into the package. With this type of device, computer systems can be implemented with a minimal amount of additional hardware. The computer system did have to carry out some computationally difficult tasks, such as mixing multiple audio channels on the central node. It is possible to implement these tasks in software, but the required computational power would have made the system impractical. The design therefore uses a Programmable Gate Array (FPGA).

An FPGA is a hardware device that is programmed once it is installed in a computer system. FPGAs in development systems replace custom Application Specific Integrated Circuits (ASICs). Carefully programmed, the FPGA can do complex calculations very quickly, and in this design, allowed us to off-load the complex computational tasks from the CPU.

The position nodes are responsible for collecting audio data and playing

audio data on either of two speakers. To that end, a pair of audio codecs is included in the design. These are memory-mapped devices, which means that they can be accessed by either writing to or reading from specific memory locations. The codecs chosen for this system can record and play back audio from 8 kHz (AM radio quality) to 44 kHz (CD quality). The design was to be for a completely embedded system that did not require a complex user interface. A touchscreen interface was therefore included in the design instead of the traditional screen and keyboard interface. While the hardware supports the addition of a touchscreen interface, it is not currently implemented.

The most important element of the hardware for the purposes of this work is the communications interface. As described above, we chose to implement a TDMA interface on top of Ethernet hardware. The hardware chosen for this is the AMD-79C940BKC-MACE Ethernet Controller. This particular chip offers several advantages: it is a fully functional and very inexpensive Ethernet interface chip, and it can drive the Ethernet interface in an interrupt-based fashion, which results in highly time-efficient processing that places minimal CPU load on the system.

In implementing the network protocol, we used 802.3 (Ethernet) compliant packets. However, some simplifications were applied to the design. Packet sizes were fixed to carry 56 bytes of data, and no additional protocol information was included except for the minimum that was required. As described above, the features of Ethernet such as collision detection and retransmission were not used. However, the design also had to avoid the error conditions generated by the Ethernet hardware.

Two hardware errors are of concern: the detection of collision and jabber conditions. The collision resolution method used by standard Ethernet is to have each station retransmit the corrupted data after waiting a random and arbitrary amount of time. Obviously, this is not appropriate behavior for real-time systems. The protocol, by design, avoids collision conditions. Collisions, if they do happen, cause data to be lost and there is no recovery built into the protocol. Jabber conditions occur when stations on the network transmit for too long. Ethernet packets have a maximum length; they can at most carry 1500 bits of payload data. Packets that exceed this maximum are considered to have "jabbered" on the network. An improperly implemented Ethernet driver, a hardware fault, or more likely, the packing too close of packets on the network medium

can cause jabber. Between each of the packets there must be a minimum period of rest, as packets placed too close together will cause the packets to blend together and the network to jabber.

In the protocol designed here, the advanced features of Ethernet are not used. In many ways, the features that Ethernet provides to "simplify" the transmission and reception of data over the network would almost certainly violate the real-time constraints that we are attempting to meet. This protocol design attempts to be as hardware independent as possible and views the network merely as a method of getting data between points.

6.3 Development Environment

This description of the protocol implementation would not be complete without describing the development environment. The goal of the development system was to provide a quick turnaround of the edit-compile-run cycle commonly used in incremental software development. Linux is a free variant of the UNIX operating system that provides a full suite of software development tools, editors, and code revision management systems. It runs on low-cost hardware and has proven to be a well-supported development platform. In addition to Linux, GCC and G++ are free, and C and C++ compilers are available for a multitude of platforms. A cross-compiler based on these technologies supports the Coldfire processor.

The development of this type of time-dependent software required debugging tools that were unusual for standard software development. The development of time-critical systems requires monitoring tools that have a notion of time built in, and subtle timing issues cannot be debugged with tools that instrument a running program and therefore change its timing properties. The use of a digital oscilloscope and logic analyzer allowed the joint monitoring of the network and parts of the computer system and provided the means to do careful, passive measurement of the running systems. The oscilloscope proved to be an invaluable tool in developing the hardware and in debugging the time-dependent portions of the software.

While the oscilloscope's ability to determine subtle timing issues with the network system was important, it did not allow us to see the data on the network and debug such issues as buffer overruns. For that task, an Ethernet "sniffer" was used. This allowed us to plot and reconstruct the transactions on the network in a post-mortem fashion. The "sniffer" is a very useful and required tool when developing software that is accessing the network because it allows the actual data to be examined for errors. The final tool, and perhaps the most effective, is the human sense of hearing: many of the most subtle errors in the system were detectable only by listening very carefully to the other voices and synthetic signals placed into the system.

The advantage of using Ethernet-compliant packets was that standard Ethernet "sniffing" tools could be used as effective debugging aids. The packet design also allows for the addition of higher-level protocol information, such as IP addresses, that should become a design requirement later on.

The implementation of the software consists of three different levels. At the lowest level there is a core of software that deals with sending and receiving data to and from the Ethernet hardware. This part is called the Ethernet core. The core includes interrupt handlers for data reception and routines to classify the data as it comes in off the wire. There are also handlers for error conditions that in normal circumstances are not called. Finally, there are routines to handle sending data to other nodes.

At a slightly higher level, there is the code that describes how the higher level tasks in the master and position nodes run and interact. The code for these parts of the system is directly based upon the pseudo-code described in the previous chapter.

The layout of the code for development purposes also represents the logical relationships between the different elements. The communications core is stored in and maintained as a library, and as much as possible, time and space efficiency are considered to be critical design requirements. A great deal of care was taken in implementing the system so that it would be simple to understand and sensitive to performance issues. For the initial implementation, it was decided that the safety-critical aspects were not going to be implemented in this version of the system.

6.4 Measured Values

In the calculations outlined in the previous chapter, values had been assumed for the required inter-packet gap and the jitter in the system. Given that we now have an implementation of this system we can measure these values directly.

6.4.1 Methodology

The measurements of the real-world values discussed here were taken on the prototype system. The method used to collect the measurements was similar to methods used when debugging the system. The system was set to run, monitoring equipment was attached, and measurements were taken directly off of the hardware. The challenge in making the measurements was to do so passively, without affecting the state of the running system.

Successive versions of the program used smaller and smaller delay loops to transmit packets; eventually, the minimum value was found. Jitter in the system was measured by letting the system run for a period of time and tracking how much variance there was in the transmission time of packets from the position stations. After it became clear that the system would not jitter beyond the current value, that value was declared the jitter.

6.4.2 Results

Using the oscilloscope, the minimum inter-packet gap was measured to be zero. Within the Ethernet standard, the inter-packet gap is optional [AMD:97]. The application controls timing carefully enough that it was possible to set this value to zero. This figure represents the closest that two packets may be safely placed next to one another on the network before they are detected as a collision.

The jitter in the system is the variance in time that the system shows over a number of iterations. This was measured as 0.023 milliseconds. In the system as

implemented, this is the amount of variability that needs to be allowed for so that the systems can stay in synchronization. If this allowance is violated, there is a possibility that two stations may transmit at the same time, which could result in a fatal network collision.

We may now revisit the calculations concerning jitter and minimum inter-packet gap.

$$S = \left\lfloor \frac{(L - J - G)B - M}{(J + G)B + O + C} \right\rfloor$$

$$S = \left\lfloor \frac{(0.007 - 0.00023 - 0)10000000 - 576}{(0.00023 + 0)10000000 + 208 + 448} \right\rfloor$$

$$S = 29$$

The safety-critical features of the system were not implemented. It should be apparent from the scope trace that it would be possible to detect the failure of a node. From this, we can conclude that the planned design for implementing node replacement is possible.

The result of this design and development effort is a reliable, simple, and inexpensive protocol for dealing with time-critical data streams. The protocol guarantees bandwidth for individual nodes, latency, and the reliable delivery of data. We can surmise from the design that the protocol is also robust in the event of a node's failure.

The protocol is also extendable. Several enhancements could be made to the design that would significantly improve its utility and not violate the desirable real-time properties. Enhancements fall into two basic classifications. First, there is the improvement of the existing features of the protocol. The extensions to the system could support faster networking mediums such as 100-megabit and 1000-megabit Ethernet standards. This would allow for a significant reduction in latency or an increase in bandwidth or both. The protocol is not constrained to Ethernet; any bidirectional broadcast medium will do. The extension into a wireless or non-Ethernet optical system is also reasonable. We have seen that the design supports the dynamic replacement of

failed nodes, although this was not implemented as part of this project. Second, the protocol could be integrated into more standard systems. TCP/IP could be implemented on top of the Ethernet layer, providing the ability to participate with standard TCP/IP devices. At this time, systems implemented with this protocol cannot co-exist with "regular" 802.3 Ethernet; however, it would be advantageous in many ways to integrate these real-time systems into non-real-time networks. A suggested implementation would add an additional Ethernet port to the master node and implement an 802.3/TCP/IP stack on that interface. This would allow the system to be controlled and monitored using an industry-standard protocol.

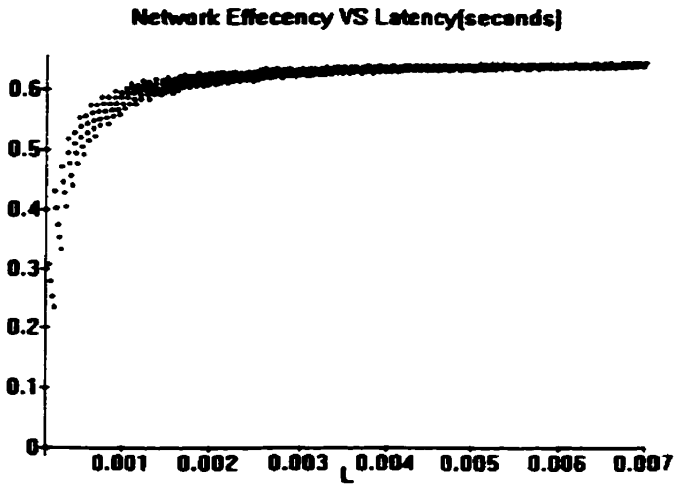
Chapter 7 - Conclusion

This thesis describes the successful design and implementation of a protocol with good real-time properties. The protocol reliably delivers data with a known latency and provides service to all nodes in the network. The protocol developed here is adjustable, in that it is possible to change the parameters of the protocol and trade one positive aspect for another. For example, by increasing the latency in the protocol you can increase the data bandwidth available to each node. We can also increase the latency or decrease the bandwidth to increase the number of permitted nodes on the network.

The experimental data collected in this work provides real-world values for the constants in the equations used to describe the protocol. We can now, with a degree of confidence, enter the desired features of the protocol and get a believable model of the real-world results. A simple robust design can support the structures placed upon it. A fundamental characteristic of good design is that it should be no more complicated than it needs to be. An unnecessarily complex design is much more likely to fail simply because there are more parts that can fail. The protocol detailed here is an example of a design that is sufficiently complex to solve the task laid out for it, but does not suffer from features that could interfere with its operation.

7.1 Additional Results

The networking scheme presented here can achieve much higher average utilization than standard Ethernet. It can approach Ethernet's theoretical maximum; normally, in estimations of "standard" Ethernet, the expected utilization is only 30% [Molaro:93]. Here, the scheme allows the utilization to reach 65%. The values used were: B=10 Megabits , C=56 bytes, J=2.9 milliseconds, G=0 milliseconds, M=84 bytes, and O=26 bytes. M and O are large because we pass some control information for the GUI in this portion of the packet. This results in a network utilization of 65% when L=7 milliseconds. The graph of network utilization versus latency is shown.



This work develops a real-time networking scheme and provides a sample implementation. Measured real performance was compared to theoretical values for the particular network hardware that was used, confirming the implementation of many aspects of the hardware.

The contribution of this work is a simple, well-designed, largely media-independent network protocol that meets real-time requirements, and a functioning implementation on cost-effective hardware suitable for embedded systems. The protocol allows distributed computer systems to participate in real-time.

References

[Abramson:85] Abramson, N. 1985. Development of the ALOHANET. IEEE Transactions on Information Theory 31.2: 119-123.

[Alari:97] Alari, Gianluigi and Augusto Ciuffoletti. 1997. Implementing a Probabilistic Clock Synchronization Algorithm. Journal of Real Time Systems 13: 25-46.

[AMD:97] Advanced Micro Devices. 1997. AM79C940 Media Access Controller for Ethernet. Publication number 16235: 36-37.

[Artesyn:98] Artesyn Corporation. 1998. Choosing an Operating System for Embedded Real-Time Applications. Technical white paper.

[Bohanon:98] Bohannon, Aiden and Donal Hefferman. 1998. A Structured Approach to Real-Time Application Software Design for the CAL Environment. Fifth International CAN Conference.

[Budhiraja:93] Budhiraja, Navin, Keith Marzullo, Fred B. Schneider, and Sam Toug. 1993. The Primary Backup Approach. In Distributed Systems, ed. Sape Mullender (2nd edition), 199-216. Addison Wesley: ACM Press.

[Chen:95] Chen, Biao, Sanjay Kamat, and Wei Zhao. 1995. Fault-Tolerant Real-Time Communication in FDDI Based Networks. Proceedings of the 16th IEEE Real-Time Systems Symposium held in Pisa, Italy, December 1995, 141-150.

[Clancy:97] Clancy, Tom. 1997. Airborne: A Guided Tour of an Airborne Task Force. New York: The Berkley Publishing Group.

[Clohessy:97] Clohessy, Kim, Brian Berry, and Peter Tanner. 1997. New Complexities in the Embedded World: The OTI Approach. Position Paper for ECOOP Workshop on Object Oriented Real-Time Systems, held in June 1997. Object Technology International Inc.

[Contemp:88] Contemporary Controls. 1988. ARCnet Tutorial & Product Guide.

[Cox:86] Cox, Brad and Bill Hunt. 1986. Objects, Icons and Software ICs. Byte

Magazine (August): 161-176.

[Dana:97] Dana, Peter H. 1997. Global Positioning System (GPS) Time Dissemination for Real-Time Applications. Journal of Real Time Systems 12: 9-40.

[DEC:97] Digital Equipment Corporation. 1997. Gigaswitch/FDDI Product Brief. Document Number EC-F7805-42.

[DEC:96] Digital Equipment Corporation. 1996. Digital Semiconductor SA-110 Microprocessor, Technical Reference Manual. Document Number EC-QPWLC-TE.

[Epplin:98] Epplin, Jerry. 1998. Adapting Windows NT to Embedded Systems. Embedded Systems Programming (June): 44-61.

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

[Feng:96] Feng, Fang, Wei Zhao, and Amit Kumar. 1996. Bounding Application to Application Delays for Multimedia Traffic in FDDI-Based Communications Systems. Multimedia Computer Networking (January 29-31, 1996): 174-185.

[Fetzer:97] Christof Fetzer, and Flaviu Christan. 1997. Integrating External and Internal Clock Synchronization. Journal of Real-Time Systems 12: 123-171.

[Fuller:75] Fuller, R. Buckminster. 1982. Synergetics: Explorations in the Geometry of Thinking. London, New York: Macmillan Publishing Company. 276-277.

[Ghose:97] Ghose, K., S. Aggarwal, P. Vasek, S. Chandra, A. Raghav, A. Ghosh, and D.R. Vogel. 1997. ASSERTS: A Toolkit for Real Time Software Design. Proceedings of the Ninth Euromicro Workshop on Real Time Systems, 224-232.

[Gomaa:86] Gomaa, H. 1986. Software Development of Real Time Systems. Communications of the ACM 29.7 (July): 657-668.

[Gouda:98] Gouda, Mohamed G. 1998. Elements of Network Protocol Design. New

York: John Wiley & Sons Inc. 266-268.

[Gullekson:96]Gullekson, Garth and Bran Selic. 1996. Design Patterns for Real Time Software. In Embedded Systems Conference West held in San Jose, California from September 17-19, 1996.

[Halang:97] Halang, Wolfgang and Markus Wannemacher. 1997. High Accuracy Concurrent Event Processing in Hard Real Time Systems. Journal of Real Time Systems 12: 77-94.

[Halpern:84] Halpern, J., B. Simons, R. Strong, and D. Dolev. 1984. Fault Tolerant Clock Synchronization. In Proceedings of the 3rd Annual ACM Symposium on Principals of Distributed Computing held in Vancouver, BC from August 27-29, 1984, 89-102.

[IBM:98] International Business Machines. 1998. PowerPC 740 and PowerPC 750 Embedded Microprocessor Datasheet.

[Intel:97] Intel. 1997. Embedded Write-Back Enhanced Intel DX4 Processor. Document Order Number 272771-002.

[Irey:98] Irey, Philip, Robert Harrison, and David Marlow. 1998. Techniques for LAN Performance Analysis in a Real Time Environment. Journal of Real-Time Systems 14: 21-44.

[Kamat:95] Kamat, Sanjay and Wei Zhao. Real Time Performance of Two Token Ring Protocols. In Principals of Real-Time Systems, ed. S. Son. : Prentice-Hall.

[Kim:97] Kim, Kwang Hae(Kane) .1997. Toward New-Generation Object Oriented Real-Time Software and System Engineering. SERI Journal 1.1 (January): 1-23.

[Kopetz:87] Kopetz, H. and W. Ochsenreiter. 1987. Clock Synchronization in Distributed Real-Time Systems. IEEE Transactions on Computers 36.8 (August): 933-940.

[Kopetz:89] Kopetz, Hermann, Andrwas Damm, Christian Koza, Marco Mulazzani, Wolfgang Schwabl, Christoph Senft, and Ralph Zainlinger. 1989. Distributed Fault-Tolerant Real-Time Systems: The MARS Approach.

IEEE Micro (February): 25-40.

[Kopetz:91] Kopetz, H., R. Zainlinger, G. Fohler, H. Kantz, P. Puschner, and W. Schutz. 1991. An Engineering Approach to Hard Real-Time Systems Design. In Proceedings of the 3rd European Software Engineering Conference, ESEC 91, held in Milano, Italy in October 1991, 166-188.

[Kopetz:92] Kopetz, H., G. Fohler, G. Grunsteidl, H. Kantz, G. Popischil, P. Puschner, J. Reisinger, R. Schlatterbeck, W. Schutz, A. Vrchoticky, and R. Zainlinger. 1992. The Programmer's View of MARS. In Proceedings of the 13th Real-Time Systems Symposium held in December 1992, 223-226. Phoneix AZ.

[Kopetz:93] Kopetz, Hermann and Paulo Verissimo. Real Time and Dependabilty Concepts 1993. In Distributed Systems, ed. Sape Mullender (2nd edition), 413. Addison Wesley: ACM Press.

[Lamport:85] Lamport, L. and P.M. Melliar-Smith. 1985. Synchronizing Clocks in the Presence of Faults. Journal of ACM 32.1 (January): 52-78.

[Larson:95] Larson, Robert E. 1995. Methods for Implementing IP on ATM Networks. Odyssea Systems Corporation. (December).

[Lichtenecker:97]Lichtenecker, Reiner. 1997. Terrestrial Time Signal Dissemination. Journal of Real Time Systems 12: 41-61.

[Lundelius:84]Lundelius, J. and N. Lynch. 1984. A New Fault Tolerant Algorithm for Clock Synchronization. In Proceedings of the Third ACM Symposium on Principals of Distributed Computing, held in Vancouver, Canada, 75-88.

[Mahaney:85] Mahaney, S. and F.B. Schneider. 1985. Inexact Agreement: Accuracy, Precision and Graceful Degradation. In Proceedings of the Fourth Annual ACM Symposium on Distributed Computing held in Minaki, Ontario, August 1985, 237-249.

[Malcolm:94] Malcolm, Nicholas, Sanjay Kamat, and Wei Zhao. 1996. Real-Time Communications in FDDI Networks. Journal of Real-Time Systems 10.1 (January): .

[Mandeville:96]Mandeville, Robert and David Newman. 1996. Canned Heat. Data

Communications Magazine (February): .

[Metcalfe:76] Metcalfe, R.M. and D.R. Boggs. 1976. Ethernet: Distributed Packet Switching for Local Computer Networks. Communications of the ACM 19 (July): 395-404.

[MIL-HNBK-818A] U.S Military. 1996. Draft Military Handbook 818A: Survivable Adaptable Fiber Optic Embedded Network. 30 September 1996.

[MIL-STD-1553] Military Standard 1553. 1998. An Interpretation of MIL-STD-1553. 26 May 1998.

[Mills:81] Mills, D.L. 1981. RFC 778, DCNET Internet Clock Service. Network Working Group. (April).

[Mills:85] Mills, David. 1985. RFC 958, Network Time Protocol (NTP). Network Working Group. (September).

[Mills:92] Mills, David L. 1992. Network Time Protocol (Version 3): Specification, Implementation and Analysis. Network Working Group. (March).

[Molaro:93] Molaro, Donald and Cullen Jennings. 1993. A Simple Interface to Distributed Programming. In Proceedings of SS'93: High Performance Computing, held June 6-9 1993 in Calgary, Alberta, 271-278.

[Motorola:96] Motorola - Semiconductor Technical Data. 1996. Technical Summary MC68HC11 Series 8-Bit Microcontroller, MC68HC11KA4TS/D. Motorola Inc.

[Motorola:96b]Motorola - Semiconductor Technical Data. 1996. Real Time Clock Plus RAM with Serial Interface-MCHC68T1. Motorola Inc.

[Motorola:97] Motorola - Consumer Systems Group. 1997. Product Brief MFF5307 Integrated Coldfire Microprocessor. Motorola Document MCF5307/D.

[Nakajima:98] Nakajima, Tatsuo and Hideyuki Tokuda. 1998. User Level Real-Time Network System on Microkernel-based Operating Systems. Journal of Real Time Systems 14: 45-60.

[NIST:98] National Institute of Standards and Technology. 1998. Publishers of Computer Time Synchronization Software. www.nist.gov

[Olukotun:94] Olukotun, Kunle, Rachid Helaihel, Jeremy Levit, and Ricardo Ramirez. 1994. A Software-Hardware Cosynthesis Approach to Digital Systems

Simulation. IEEE Micro: 48-58.

[Quill:66] Quill, Humphrey. 1966. John Harrison: The Man Who Found Longitude. London, England: John Baker.

[Raha:96] Raha, Amitava, Sanjay Kamat, and Wei Zhao. 1996. Admission Control for Hard Real-Time Connections in ATM LANs. IEEE Infocomm '96.

[Rosenberry:92]Rosenberry, Ward, David Kenney, and Gerry Fischer. 1992. Understanding DCE. O'Reilly and Associates. Sebastopol CA.

[SBS:98] SBS Avionics Technologies. 1998. ARINC 429 Commentary. SBS Avionics Technologies (August).

[Schmid:97] Schmid, Ulrich. Editorial. Journal of Real Time Systems 12: 119-122.

[Schneider:95]Schneider, Stanley, Vincent Chen, and Gerardo Pardo-Castellote. 1995. The Control Shell Component-Based Real-Time Programming System. In 1995 IEEE Conference on Robotics and Automation,

[Schossmaier:97]Schossmaier, Klaus, Ulrich Schmid, Martin Horauer, and Dietmar Loy. 1997. Specification and Implementation of the Universal Time Coordinated Synchronization Unit (UTCSU). Journal of Real Time Systems 12: 295-327.

[Sridhar:98] Sridhar, Thayumanavan. 1998. Strategies for Communications System Software Design. Embedded Systems Programing (June): 34-42.

[Tanenbaum:96]Tanenbaum, Andrew S. 1996. Computer Networks. New Jersey: Prentice Hall.

[Tanenbaum:96a]Tanenbaum, Andrew S. "Computer Networks." Pp. 292

[Tanenbaum:96b]Tanenbaum, Andrew S. "Computer Networks." Pp. 246

[Tanenbaum:96c]Tanenbaum, Andrew S. "Computer Networks." Pp. 287

[Tanenbaum:96d]Tanenbaum, Andrew S. "Computer Networks." Pp. 81

[Tanenbaum:96e]Tanenbaum, Andrew S. "Computer Networks." Pp. 244

[Teledisc:98] Teledisc Corporation. 1998. Technical Overview of the Teledisc Network.

[Teledisc:97] Teledisc Corporation. 1997. Does Latency Matter. (December).

[Terry:96] Terry, Dennis. 1996. Choosing a Processor for Embedded Real-Time Applications. Heurikon Corporation, White Paper.

[Verissimo:97]Verissimo, Paulo, Luis Rodrigues, and Antonio Casimiro. 1997.

CesiumSpray: A Precise and Accurate Global Time Service for Large Scale Systems. Journal of Real Time Systems 12: 243-294.

[Worsley:97] Worsley, Debra and Tokunbo Ogunfunmi. 1997. Isochronous Ethernet: An ATM Bridge for Multimedia Networking. IEEE Multimedia 4.1 (January-March): 58-67

[Yang:97] Yang, Tao. 1997. Gigabit Ethernet and ATM. www.atm.cs.ndsu.nodak.edu/~tyang/gigabit.html

[Zhao:94] Zhao, Wei, Amit Kumar, Gopal Agrawal, Sanjay Kamat, Nicholas Malcom, and Biao Chen. 1994. Real Time Communications in FDDI Based Reconfigurable Networks. Proceedings of the IEEE Workshop on Real-Time Operating Systems, May 1994, 49-52.