The Hong Kong Polytechnic University

Department of Electrical Engineering

A Study of the IEEE 1394 Cluster and Its Application in Parallel Transportation Simulation Systems

YU Yong

A thesis submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

2006

CERFIFICATE OF ORIGINALITY

I herby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously or neither written nor material which has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in text.

_____ (Signed)

YU YONG _____ (Name of student)

Abstract

The design of modern transportation systems leads to many challenges and it makes great impact on societies. The foremost issues concern safety, delays due to congestion, and vulnerability due to natural and man-made disasters. Computer simulations of transportation system provide a mean to study the system performance or to evaluate different strategies in order to determine the most effective solution for a problem. While large-scale computer simulations may require a long time to complete, decisions often must be made promptly and therefore, parallel and distributed simulation methods are often used to reduce the execution-time of such large-scale simulations. High-end supercomputers had been dominating this field in the past decades, but the high prices of high-end supercomputers made the common applications impossible. There are remarkable progresses in the parallel and distributed simulation field recently and network of workstations or clusters which composed of many workstations interconnected through a network become a popular method to speedup large-scale simulation systems.

Cost-effectiveness is the major factor that drives the use of computer clusters in simulation systems. High-end computer clusters are still not popular due to the very costly interconnection medium. In this thesis, a computer cluster is proposed, which employs a new interconnection device - the IEEE 1394, or Firewire, as the interconnection media. The IEEE 1394 is relatively low in price, while good in performance comparing to other interconnection medium. Such

cluster is employed to implement two typical microscopic transportation simulation systems, the performance improvement of the two systems are observed, and therefore, these systems can accommodate large-scale simulations in shorter time. This allows the simulation systems to be applied in different aspects including decision making or personnel training of the different transportation systems.

The Firewire system is originally designed for interconnected digital audio/video equipments to a computer and in order to utilize the device as a general purpose networking media, a custom-designed communication protocol is devised. The protocol is tailored for implementing parallel application over the IEEE 1394 device. The new protocol reduces the communication overhead and an improvement of around 25% is shown when comparing to the traditional TCP/IP protocol over the IEEE 1394 device. In order to fully utilize the transmission capacity of the two operation modes of the IEEE 1394 device, and to support basic QoS (Quality of Service) feature, a fuzzy control mechanism is applied to the communication protocol. With this fuzzy control mechanism, the protocol can maximize the overall performance according to the data transmission format of different applications. The fuzzy controlled communication protocol shows a speedup factor of up to 20% when comparing to the ordinary mechanism.

The performance of the IEEE 1394 cluster based on our new protocol is also compared to the clusters using other interconnection media such as the Ethernet

to illustrate the differences in terms of cost-effectiveness. Case studies of two typical transportation simulation applications, namely the Railway Operation Simulation System and the Urban Traffic Simulation System, which run on the IEEE 1394 cluster were carried out, the results substantiated that these applications benefit significantly from such a cluster. The communication protocol is suitable for applications with different data transmission characteristics and this gives a good prospect in utilizing parallel computing in transportation simulations. Case studies in the thesis are carried out on a cluster, which consists of only four workstations. A statistical model is developed to estimate the performance of such cluster in a larger scale. The model shows that with a cluster that consists of 64 workstations, the performance is still acceptable.

Acknowledgements

I am very grateful to Dr. Yu-fai Fung for his continual support and guidance throughout the three years. His observations and comments helped me to establish the overall direction of the research and move forward with investigation in depth. I thank him for providing me the opportunity to do this research.

I would like to thank Dr. W. L. Cheung who has provided much useful advice and encouragement.

I would like to thank Dr. P. T. Chan for his valuable direction and help.

I would also like to thank my family for standing by me, supporting me to accomplish my research work.

Contents

1 Introduction

Computer simulation methods are widely used in transportation planning, forecasting, and decision-making systems, very often in such a system like transportation forecasting, the time required to execute the simulation program is a crucial factor for the design of such simulation system. A short term transportation forecast may be required to predict the transportation condition for the next hour, or for the next 20 minutes [92], [16], [74], [65], [27]. If users want to make decision based on the simulation results, they may want to do the simulation dozens of times with different parameters that simulate different traffic conditions. Long-term transportation simulation is often used for the planning of metropolitan layout design etc. [76], [88], simulation over a very long period of time in terms of years is often required, while the simulation does not necessarily to be completed within hours, it will consume too much computing time when running on a general purpose computer and therefore, render the system not suitable for practical use.

High performance supercomputers are often used to fulfill the computing requirements demanded by such Microscopic Transportation Simulation systems, but they are often quite expensive, computer clusters are a cheaper alternative for supercomputers, but high-end clusters are still expensive and make them difficult to be widely deployed for simulating a microscopic transportation system.

The networking system is a major element in the implementation of a com-

puting cluster, and the cost and performance of the cluster are greatly affected by different networking mechanisms. Currently, the most commonly used networking technology for a cluster is the Fast Ethernet, which gives a transfer rate of 100 Mbps. There are also other high-end technologies such as the Gigabit Ethernet and the Myrinet [1], [8], [64]. Both Gigabit Ethernet and the Myrinet system can sustain higher bandwidth, 1 Gigabits per second but they are rather expensive. The IEEE 1394 bus, or the FireWire [37], (through out this thesis, the term IEEE 1394 and Firewire are used inter-changeably) which has a transfer rate of 400 Mbps but costs much less than the Gigabit Ethernet, is a suitable candidate for building a cost-effective computer cluster. Currently, many workstations already include an IEEE 1394 link, so networking with the 1394 does not require additional hardware. Moreover, the new standard, IEEE 1394b [39] has been defined and the maximum transfer rate for the new standard is up to 3.2Gbps. Currently, the 800 Mbps 1394 device is also available and by comparing the transfer rate, the IEEE 1394 bus is a significant improvement over the conventional Fast Ethernet so it could be a cost-effective solution for developing cluster systems. In the near future, when devices based on the IEEE 1394b standard emerge, then its speed will be compatible to, or even out-perform, the gigabit transfer-rate devices and therefore, it could become a popular networking device. Studies carried out in this project can be regarded as an initial stage of a continuous research of the IEEE 1394 devices and will pave the way for future development and applications of, the more advanced Firewire devices.

The IEEE 1394 interface is a peer-to-peer high-speed serial bus standard. It was first developed in the mid 1980s by the Apple Computer. As other manufacturers gained interest in FireWire, a working committee was formed to create a formal standard on the architecture. The resulting specification was IEEE 1394-1995 [37]. There was a higher speed 1394 serial bus specification called the 1394b, which was adopted in the year 2002 and defined serial bus extensions for running the serial bus at speeds into the gigabit per second range.

The Firewire was originally designed to interface digital audio/video equipments to a computer. In order to support digital audio/video data stream, the device supports two kinds of transfer modes, namely isochronous and asynchronous. In isochronous transfer, data delivery should be at constant intervals, suitable for video signal. These transfers provide for applications where a guaranteed quality of service is required, but do not include the retransmission of data when transmission errors occur. For asynchronous transfers, the delivery of data does not need to be at a constant rate. Data is retransmitted when an error in its transmission occurs in the link layer. Therefore, packets are not lost.

The IEEE 1394 draws much attention since its emergence mainly due to its high throughput of 400 Mbps. There are already studies on utilizing the device for general networking. However, as the Firewire was not designed to work as a general purpose networking device therefore, new communication protocol and control mechanism are required in order to optimize the physical features of the device. These are the major research directives of this project.

In order to examine the overall performance of a cluster based on the 1394 interface, we implemented two different types of microscopic transportation simulation systems based on the cluster. Microscopic transportation simulation represents a computational intensive problem and therefore, is a suitable candidate for parallel implementation. Moreover, in simulating the flow of traffic, it will generate different kinds of data flow, which may be benefit from the two transfer modes of the Firewire and this is also crucial to our investigation.

The major objectives of this thesis are:

a) To study the mechanism of connecting a cluster of commodity PCs with the IEEE 1394 interface.

b) To examine the technique of fully employing the data transmission capacity of the IEEE 1394 interface in parallel computing.

c) To explore the utilization of the IEEE 1394 cluster in transportation simulation systems to improve their performance.

This thesis includes seven chapters. In the next Chapter, we will first introduce the basic architecture and specification of the IEEE 1394 interface. The features of the device will be compared with other networking equipments in order to examine the cost-effectiveness of the interface. In Chapter 3, we will concentrate on the discussion of the communication mechanism and protocol of the device. The new communication protocol that we developed can utilize the full capacity of the Firewire will be discussed in details.

The 1394 interface supports two kinds of communication modes, namely

asynchronous and isochronous. Based on our studies in Chapter 2, and 3, the overall communication performance can be improved if we can utilize the communication modes properly with respect to the data structure. Therefore, a control mechanism is implemented to switch between the modes according to the data structure. Chapter 4 describes the Fuzzy control that we implemented for the switching mechanism and the overall communication mechanism is also discussed.

In Chapter 5, we describe the implementation of a computer cluster based on the IEEE 1394 interface for solving two typical transportation simulation systems to evaluate how the cluster facilitates the simulation systems and improves the performance. As our test bed is only based on four computers, therefore, it is desirable that a model can be developed so that performance of a cluster with a larger scale can be gauged. Chapter 6 discusses issues related to the performance modeling and predicted performances of a cluster in different scales are presented. Finally, in Chapter 7, conclusion of this investigation is given and the future direction for the development of the IEEE 1394 interface is discussed.

2 The IEEE 1394 interface

2.1 Introduction

The IEEE 1394 interface provides a high transfer-rate of 400 Mbps at a very low cost; therefore, it could be a cost-effective solution for the implementation of a high-speed computer cluster. In this Chapter, the specification of the IEEE 1394 will first be described. In Section 2.3, features of different networking devices, including the Gigabit Ethernet [30] and Myrinet [9], will be compared so that cost-effectiveness of the FireWire can be evaluated.

2.2 The IEEE 1394 specifications

The IEEE 1394 (also called FireWire) is a peer-to-peer high-speed serial bus standard [37], [38], [39], [86]. It was first developed in the mid 1980s by the Apple Computer. As other manufactures gained interest in FireWire, a working committee was formed to create a formal standard on the architecture. The resulting specification was the IEEE 1394-1995 standard [37]. Different interpretations of the 1995 specification have led to interoperability problems, to clarify the specification, a supplement to the 1995 specification was developed in 2000, and it is called 1394a [37]. This supplement included additional features and made improvements intended to increase performance or usability. In 2002, a higher speed 1394 serial bus specification called the 1394b [39] was adopted, which defined serial bus extensions for running the serial bus at speeds into the

gigabit per second range.

The IEEE 1394 specification is similar to the USB standard [96] but it is capable of higher speeds. The USB standard defines a transfer rate of only 12 Mbps. The original objective of the IEEE 1394 was to interconnect various peripherals using the same cable. Interconnection between multimedia peripherals and a computer is one of the many challenges which the specification has to account for. Hence two transfer modes with different QoS are supported by the specification to accommodate multimedia data transmission.

IEEE 1394 is a low-cost (issues related to the cost will be discussed in Section 2.3) high-performance serial bus standard, it provides low-latency and high bandwidth communications to cater for multimedia data types [89]. The IEEE 1394 and IEEE 1394a specification defines a maximum data transfer rate at 400 Mbps, and other speed mode, 100 Mbps and 200 Mbps are also allowed. In the new standard, the IEEE 1394b standard, 800 Mbps, 1.6 Gbps and 3.2 Gbps data transfer rates are catered. The PCI IEEE 1394 interface cards which follow the 1394b specification and offer a transfer rate of 800 Mbps, are already available in the commodity market.

The serial bus protocols are described as a set of three stacked layers. As compared to the OSI 7-layer model [41], these three layers deal with communications between serial bus devices, as the functions of the lower four layers of the OSI model. The protocol stack of IEEE 1394 is shown in Figure 2-1.

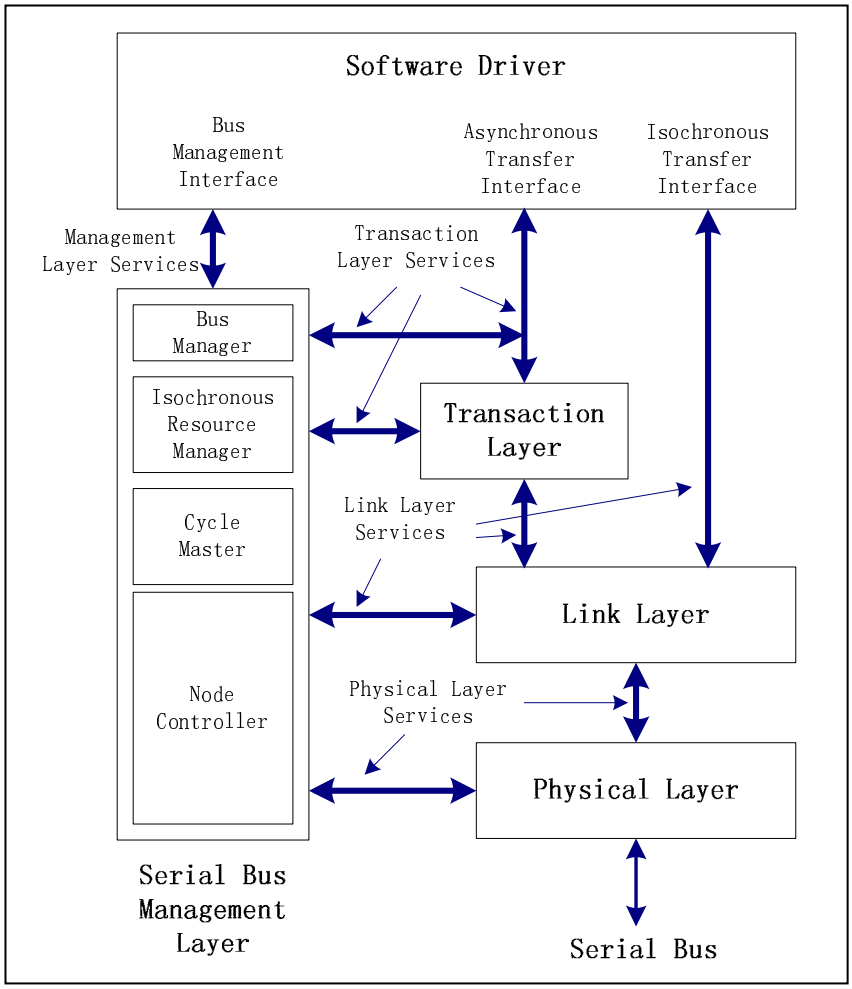


Figure 2-1: The 1394 Protocol Stack

a) The transaction layer defines a complete request-response protocol to perform the bus transactions required to support the Control and Status Registers (CSR) [35], [36] Architecture (the operations of read, write, and lock). Note that the transaction layer does not add any services for isochronous data, although it does provide a path for isochronous management data to get to the Serial Bus management via reads from and compare-swaps with the isochronous control CSRs.

b) The link layer provides an acknowledged datagram (a one-way data transfer with confirmation of request) service to the transaction layer. It

provides addressing, data checking, and data framing for packet transmission and reception. The link layer also provides an isochronous data transfer service directly to the application, including the generation of a "cycle" signal used for timing and synchronization. One link layer transfer is called a "subaction."

c) The physical layer has three major functions:

   1) It translates the logical symbols used by the link layer into electrical signals on the different Serial Bus media.

   2) It guarantees that only one node at a time is sending data by providing an arbitration service.

   3) It defines the mechanical interfaces for the Serial Bus. There is a different physical layer for each environment: cable and backplane. The cable physical layer also provides a data resync and repeat service and automatic bus initialization.

The Serial Bus protocols also include Serial Bus management, which provides the basic control functions and standard CSRs needed to control nodes or to manage bus resources. The bus manager component is only active at a single node that exercises management responsibilities over the entire bus. At the nodes being managed (all those that are not the bus manager), the Serial Bus management consists solely of the node controller component. An additional component, the isochronous resource manager, centralizes the services needed to allocate bandwidth and other isochronous resources.

In the link layer and transaction layer, the serial bus offers two data transfer types:

- Isochronous transfers that require data delivery at constant intervals. These transfers provide for applications where a guaranteed quality of service (QoS) is required, but do not include the retransmission of data when transmission errors occur.
- Asynchronous transfers, where the delivery of data at a constant rate is not required. Data is retransmitted when an error in its transmission occurs in the link layer. Therefore, packets are not lost.

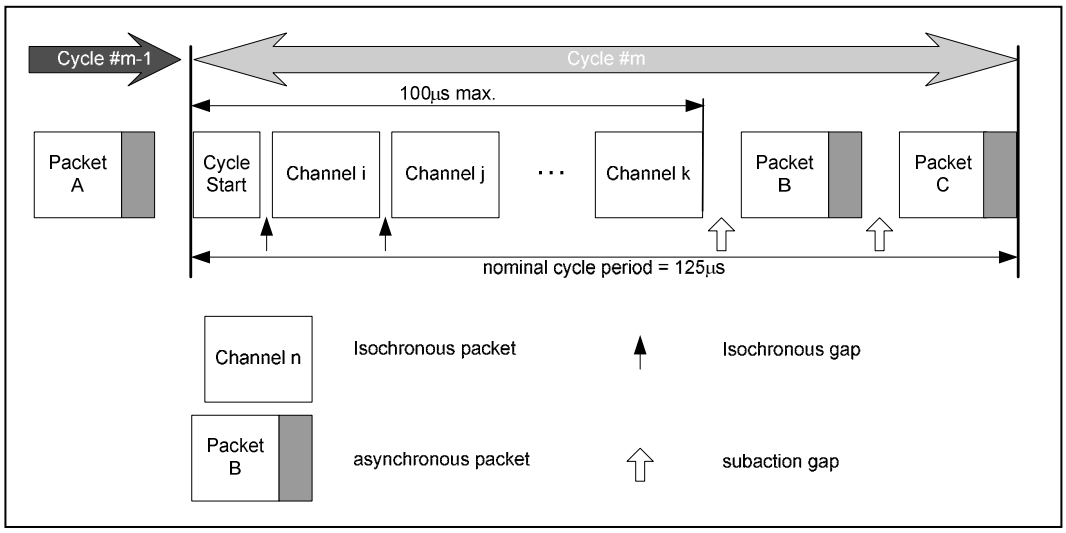


Figure 2-2: The IEEE 1394 cycle structure

The IEEE 1394 bus operates in a cycle (isochronous cycle) mode in the Link layer (Figure 2-2). An isochronous cycle begins after a cycle start signal is sent, and ends when a subaction gap is detected. During an isochronous cycle, only isochronous subactions may occur. An isochronous cycle begins every 125μs,

on average. The isochronous packets can use up to 100μs of the total 125μs cycle time. Asynchronous packets can transfer on the bus only after a subaction gap is detected. That is, the isochronous transfer mode has absolutely higher priority than the asynchronous transfer mode.

This cycle structure of the serial bus makes the performance characteristic of the two transmission modes totally different. In contrast to conventional I/O-based communication, asynchronous transfer is based on memory read/write communication architecture. Addressing of the IEEE 1394 serial bus follows the CSR architecture for 64-bit fixed addressing [35], [36]. The 64-bit node ID is further divided into a 10-bit wide bus ID and a 6-bit wide physical ID. Therefore up to 1023 busses, each can have up to 63 nodes, may be interconnected (Figure 2-3). The remaining 48 bits are used for the spaces of node-memory and registers. The Open Host Controller Interface (OHCI) Specification for IEEE 1394, which specifies the link layer implementation of the IEEE 1394 architecture, is available. This specification provides a feature for writing to remote memory that is called Physical Write. This feature automatically becomes effective when the address in the packet header indicates the physical memory space of a target node. The Physical Write is carried out by a DMA on the IEEE 1394 host adapter and thus does not interrupting the host processor. This feature can be used to achieve lower transmitting latency. For an IEEE 1394 device which is compliant with the IEEE 1394a standard, the nominal maximum throughput is 400 Mbps, and minimum latency is around 50μs.

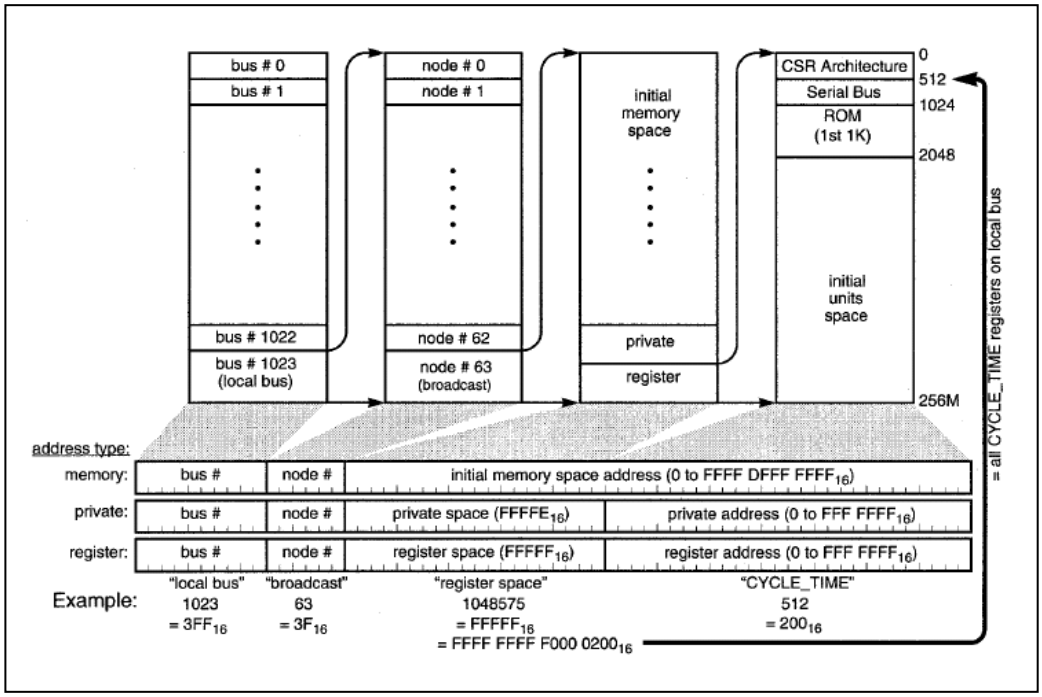


Figure 2-3: The IEEE 1394 addressing

2.3 Cost-effectiveness of the IEEE 1394 cluster

As discussed in section 2.2, although the IEEE 1394 has a nominal bandwidth of 400 Mbps, it is shared on the whole bus, if there are many nodes on the IEEE 1394 bus, the bandwidth that one node can utilize is fairly low. So if using one single bus to interconnect all the nodes of the cluster, the cluster performance may be no better than a 100 Mbps Ethernet interconnected cluster [70]. In the next Chapter, a study on the performance of the IEEE 1394 when using as a generic networking device will be discussed and we will show that when the number of nodes in the network is less than 8, then the overall performance of the IEEE 1394 is still better than a common Fast Ethernet system. In addition, the IEEE 1394 bus is especially suitable for some specific parallel applications; take

the railway simulation system for example.

Figure 2-4 depicts the cluster structure of the railway simulation system. Since this system is for training purposes, the system runs in the same time scale as in the real-world environment, and it employs 3D animations to display the simulated railway system in real-time. Thus between the central simulator and the graphical workstations that runs the virtual reality display, there are strong data communications. Such data communications are in constant time interval and almost in a steady throughput.

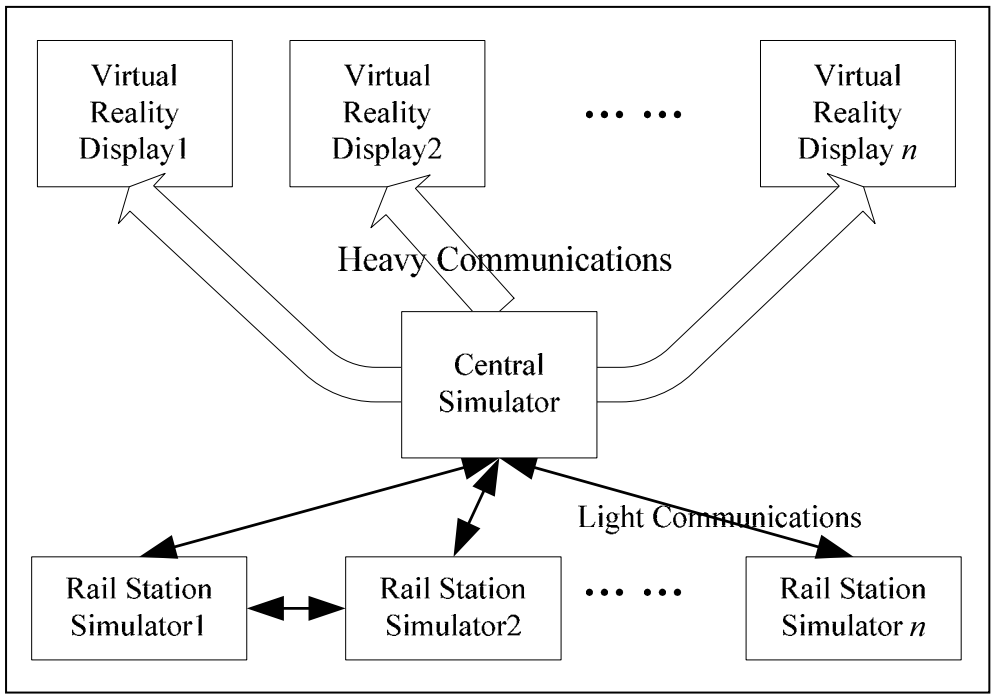


Figure 2-4: The railway simulation system structure

The performance of such a system implemented on an Ethernet cluster is barely acceptable; the rendering rate of the 3D animations averaged about 15 frames per second (fps). Even on a 100M Fast Ethernet cluster, when there are more

than five virtual reality displays connected, the average rendering rate will become fewer than 20 fps, since many copies of the same data have to be transferred repeatedly on the network. The 3D animation display observed is sometimes not very smooth, because of the unsteady delivery time of the data packets.

Since IEEE 1394 is designed to carry multimedia data, with the isochronous mode of the IEEE 1394 bus, the above problems can be resolved. The guaranteed constant delivery time of the Isochronous mode makes the 3D animation display steady, and the broadcasting nature, if utilized properly, can enable the system to accommodate much more virtual reality displays, since only one copy of each frame has to be sent on the bus. In Chapter 5, detail description of the implementation of the railway simulation system with the IEEE 1394 will be given.

In the cluster as shown in Figure 2-4, there are a huge volume of data interchanging between the center simulator and the virtual reality displays, while data interchange between other nodes is limited, a single IEEE 1394 bus connects all nodes of the cluster will be a suitable network structure. So each node must include a IEEE 1394 card, currently an IEEE 1394 interface card costs about USD 25, a cluster that consists of 20 nodes will cost USD 500 on the interconnection. While the price is somewhat expensive than the 100M Ethernet, the performance will be much better (details given in Chapter 5), especially in terms of the 3D animations.

Urban traffic simulation is another popular parallel application that runs on a cluster. In an urban traffic system, vehicles travel on the roads just like network data transfer on the network links. A common parallelization of such system is to divide an urban area into sub-areas, and one program modules simulate a member of the sub-areas, and runs on one node of a cluster. The architecture of the cluster for implementing such a system is shown in Figure 2-5.

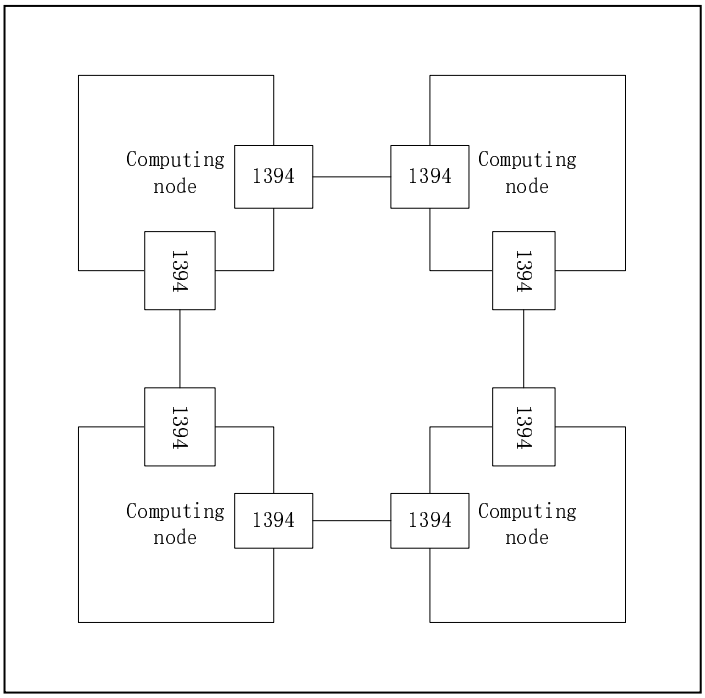


Figure 2-5: the cluster structure using IEEE 1394

Vehicles in the urban traffic system exchange frequently between adjacent sub-areas, so on the corresponding cluster, there will be heavy data communications between one node and all its neighbors, while low data communications rate between the node and other non-neighboring nodes, this makes a point-to-point network topology an ideal solution for such simulation architec-

ture. While a switched network like Ethernet is not suitable for this kind of point-to-point topology, the cheap IEEE 1394 device is an ideal option for constructing such a cluster.

Figure 2-6 shows a cluster of 4 * 4 nodes, each node has direct links to all its neighbors, when applying the point-to-point network topology, each link must have two IEEE 1394 cards on its two ends, thus for a 4 * 4 nodes cluster, 48 IEEE 1394 cards are needed. So the cost for the interconnection of such a cluster is 48 * USD25, which is USD1200.

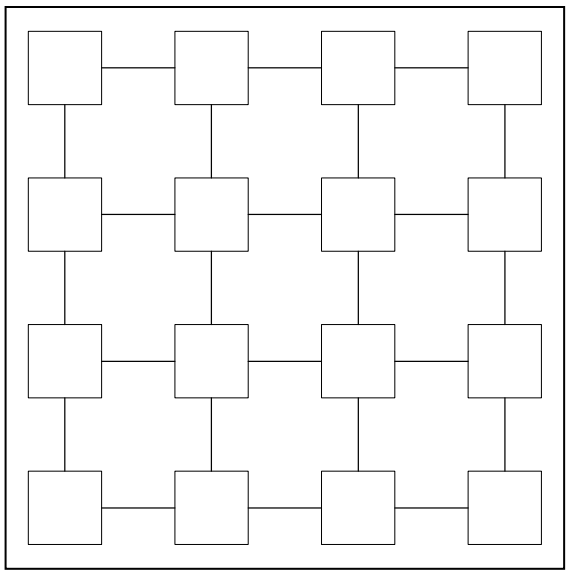


Figure 2-6: A sample cluster of 4 * 4 nodes

It is understood that a cluster computing platform will provide a cheaper alternative to high cost and high performance computers. As an example, the IBM RS/6000 F80 high performance computer with 6 RISC 450 MHz CPUs, 8 G memory, costs around 50,000 USD, while a cluster with 9 PCs, each PC with a

3.0 GHz Intel P4 CPU, 1 G memory, and a total of 24 IEEE 1394 interface cards, costs about 7,800 USD. The IBM RISC CPU is claimed to be 4 times faster than Pentium CPU, so this total frequency of the IBM high performance computer should be compatible to 450 * 6 * 4 = 10.8 GHz Intel CPU. For the PC cluster, the total frequency is 3.0 * 9 = 27 GHz. The total CPU power of the PC cluster is much higher than the IBM high performance computer, even through the latency introduced by the interconnection network will diminish the performance of the cluster, it is reasonable to guess that the cluster can offer comparable performance to the IBM high performance computer, but at a much lower price. While the cost of cluster is much lower than the cost of high performance computer, it also varies significantly mainly depending on the choice of the interconnection network. In this thesis, the cost-effectiveness comparison between clusters which are constructed by different interconnection networks is studied.

Gigabit Ethernet is commonly used for interconnection of clusters [111], [60], [17], [12], [52]. As the descendent of Fast Ethernet, its nominal bandwidth is increased to 1 Gbps. If Gigabit Ethernet is employed for the interconnection of the cluster as shown in Figure 2-5, connecting such 16 nodes cluster will need 16 Gigabit Ethernet cards and 1 Gigabit Ethernet switch with 16 ports. A Gigabit Ethernet cards cost about USD80, and a 16 ports Gigabit Ethernet switch cost around USD500 at the time of writing this, so the overall interconnection would cost around USD1780. The Gigabit Ethernet is more expensive than the IEEE 1394. While some motherboards may already have a build-in Gigabit

Ethernet interface, such motherboards are often more expensive than others, so using Gigabit Ethernet is still an expensive solution. Although when connected by the Gigabit Ethernet, each node can have a nominal bandwidth of 1Gbps, the bandwidth has to be shared by data transfers to all the directions (there can be 2, 3 and 4 directions). While using the IEEE 1394 point-to-point topology, transfers to each direction can have a bandwidth of 400 Mbps exclusively. Since data transfer distribution between nodes in a traffic simulation application is quite even, the application can benefit a lot from the IEEE 1394 cluster, and the performances are expected to be comparable with the Gigabit Ethernet cluster, details given in Chapter 5. Moreover the IEEE 1394 of 800 Mbps is now available in the market, and it will offer even higher data transfer rate of 1.6 Gbps and 3.2 Gbps in the near future, so applying the IEEE 1394 for cluster interconnection is a promising option.

Myrinet [8], [64] is a high performance packet communication and switching technology that is widely used to interconnect clusters of workstations, PCs, servers, or single board computers. The Myrinet 2000 provides full-duplex 2 Gbps data rate links, switch ports, and interface ports. As a network technology that is designed specifically for the requirements of high performance or high availability clustering, Myrinet provides excellent performance and features for cluster interconnection. It provides flow control, error control, and low latency. The Myrinet switch networks can scale to tens of thousands of hosts, and can also provide alternative communication paths between hosts. The host interfaces

of Myrinet can execute a control program to interact directly with host processes and bypassing the operating system for low-latency communication, and directly with the network to send, receive and buffer packets [46]. Figure 2-7 depicts the performance of Myrinet 2000. With message size greater than 1000 Bytes, Myrinet 2000 can sustain a data rate approaching 250 MByte/s (2 Gbit/s).

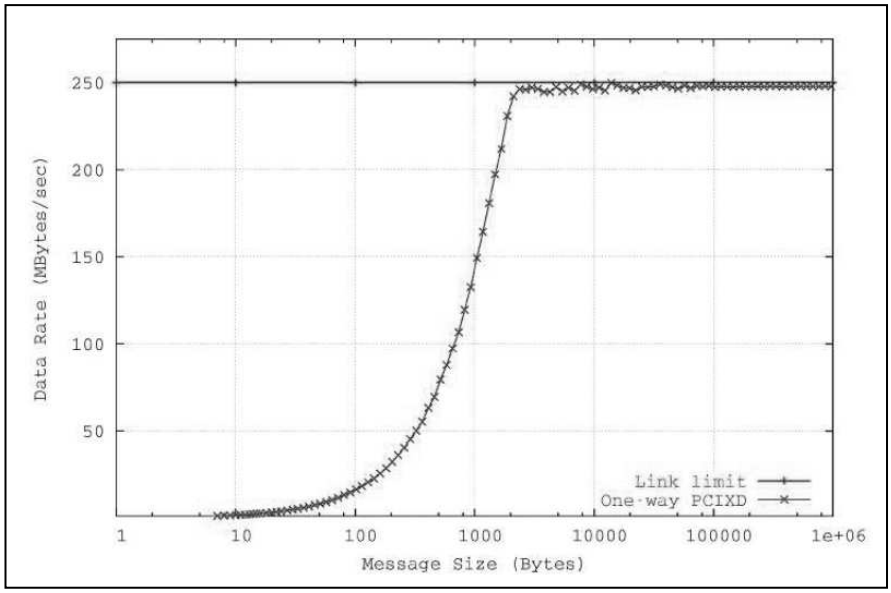


Figure 2-7: Sustained one-way throughput of Myrinet 2000

The very high performance of Myrinet technology also results in very high cost of the devices. The Myrinet components are implemented with full-custom-VLSI CMOS chips technology, a PCI network interface card with one fiber port costs about 600 USD. The CLOS type of switch is adopted in Myrinet to support direct n-to-n nodes communication; one of such switch with 8 fiber ports costs about 4000 USD. Even a 1m long fiber cable costs 70 USD.

Table 2-1 gives a further cost-effectiveness comparison of various network interfaces. Since it is difficult to examine the performances of all these networks

interfaces, the performance figure given here are all derived from the nominal performance, and the latencies listed here are regarding to the hardware layers only.

Table 2-1: The cost-effectiveness comparison of network interfaces

	Fast Ethernet	Gigabit Ethernet	Myrinet	IEEE 1394a	IEEE 1394b
Max. throughput	100 Mbps	1 Gbps	2 Gbps	400 Mbps	800 Mbps
Network structure	Bus	Bus	Switched	Bus	Bus
Latency	20µs	20µs	5µs	7.5µs	n/a
Cost/Link	20USD	200USD	1170USD	25USD	100USD
Cost/Mbps	0.2USD	0.2USD	0.6USD	0.06USD	0.12USD

In general a Fast Ethernet PCI card costs 10 to 30 USD, a typical 16 ports Fast Ethernet switch's price is around 120 USD, so the cost per link for Fast Ethernet is around 20 USD. With the appearance of non-optical technology, the price for Gigabit Ethernet drops dramatically. While 32-bit PCI Gigabit Ethernet card is low in price, it is reported that it is hard to achieve gigabit performance on such devices [17]. The 64-bit PCI card has to be employed to pursue the nominal 1 gigabit throughput of the Gigabit Ethernet. A typical copper Gigabit Ethernet 64-bit PCI Card costs about 150 USD, and a 8 ports copper Gigabit Ethernet switch costs about 300 USD, so the per link cost is around 200 USD. The Giga-bit Ethernet architecture is still an expensive option. As for the Myrinet, though its price dropped a little recently, it is still very expensive comparing to other devices. As we have discussed, a Myrinet-Fiber/PCI interface card cost 600 USD, an 8-port switch with Fiber ports costs 4,000 USD, and a 1m long Myri-

net-2000 fiber cables costs 70USD, so the per link cost for Myrinet is at least 1,170USD. While for Firewire, a 400 Mbps PCI Firewire card costs around 20-30 USD. The 800 Mbps IEEE 1394b PCI card is just coming forth recently, a bundle of the card and an 800 Mbps cable cost about 100USD.

It is obvious that Myrinet is the most expensive one considering the per link cost, and Fast Ethernet and IEEE 1394a (400Mbps) is among the most inexpensive ones. The per Mbps cost can be further derived from the per link cost, and it can be used as a major factor for the cost-effectiveness feature of the various interconnection network interfaces. IEEE 1394a is observed to be the most inexpensive one among all the networks interfaces, even lower than the Fast Ethernet. Myrinet, although supports much better throughput, is still the most expensive interface. Gigabit Ethernet seems offer good performance with fairly low cost per Mbps cost, but in practice, it is difficult to achieve its nominal throughput, as shown in Figure 3-1, the TCP throughput is only 30% higher than that of the IEEE 1394a. The cost per Mbps of IEEE 1394b interface is higher than that of the IEEE 1394a interface, but it is better than all other interfaces. So the IEEE 1394 family interfaces are much better in cost-effectiveness than other network interfaces.

2.4 Concluding remarks

In this Chapter, the performance and cost issues of several interconnection methods for building computer clusters are studied. Myrinet can provide very high

level of performance, but is very expensive in comparison with other network interfaces. Gigabit Ethernet provides high bandwidth, but its latency time is not ideal. Moreover, the complementary hardware, including the switching hub, makes it an expensive device. Fast Ethernet is readily available at a low cost. However, as a network for connecting nodes of a cluster, its performance is also not very good.

IEEE 1394, however, provides good performance at low cost. It provides less than half of the latency of Fast Ethernet and a 400 Mbps maximal link speed, yet it is as inexpensive as Fast Ethernet, if considering the cost per Mbps, IEEE 1394 is the most cost-effective one among all the network systems (Table 2-1). Certainly IEEE 1394 is inferior to Myrinet in terms of performance, and particularly in terms of link speed. It does, however, have some features that are useful for a cluster system, such as remote memory write and reliability of the link layer. The IEEE 1394 is thus a good solution for implementing low-cost cluster systems. In addition, its performance will become even better in the near future, because transfer rates of 800 Mbps, 1.6 and 3.2 Gbps are described in the next generation specification of the IEEE 1394b, and PCI card that support 800 Mbps data rate has emerged, at a rather low price of only 100 USD. We can expect that the price will be even lower as the Fast Ethernet device and the 400 Mbps IEEE 1394 interface card had experienced. So IEEE 1394 is definitely a promising device for interconnecting a computer cluster.

3 A communication protocol on IEEE 1394

3.1 Introduction

In Chapter 2, general features and properties of the IEEE 1394 device have been discussed. The Firewire (IEEE 1394) was originally designed for multimedia data transfer between a PC and other digital AV devices; it attracted immediate attention as a generic network transmission media because of its high performance/cost ratio. The most straightforward approach of employing the Firewire device for network transmission is to implement the TCP/IP protocol stack over it directly, and this is often called IP-over-Firewire [81]. By enabling IP-over-FireWire, the Firewire device becomes a general physical network interface, and then the legacy applications based on the TCP/IP protocol can run on it directly. However, in our studies, we found that such method is not very effective and therefore, a novel communication protocol was devised in order to fully utilize the device's capacity. This Chapter presents an in-depth evaluation of the IEEE 1394 basic data transmission features, and discusses the implementation of a communication protocol over IEEE 1394, which is designed to meet the cluster computing demand.

3.2 Generic networking protocol over the IEEE 1394

IP-over-FireWire is a generic networking protocol for IEEE 1394 and

IP-over-FireWire provides many useful features. One can use IP-over-FireWire to connect two or more computer systems by FireWire for file sharing, Internet sharing, or for the use of other IP-based services. Many commodity computer products have build-in 10/100 Ethernet and FireWire 400 interfaces. The nominal 400 Mbps throughput of Firewire, which is much higher than that of Fast Ethernet, makes FireWire a competing option for local area network.

There are several IP-over-Firewire implementations on different operating systems. The Windows 2000 offers a build-in Ethernet emulator over the Firewire [55], so the TCP/IP protocol stack can use the Firewire device as an Ethernet device, all legacy TCP/IP applications can use the Firewire as a standard networking interface without any modifications in programming. Unibrain offers a software driver called the FireNet, a commercial product, on the Windows and Macintosh operating systems [95], which is also one of such implementations. An internet RFC archive, RFC 2734, defines an implementation of IPv4 over IEEE 1394 [81], this document specifies how to use IEEE Std 1394-1995 Standard for the transport of Internet Protocol Version 4 (IPv4) datagram. It defines the necessary methods, data structures and codes for such purpose. It includes not only packet formats, encapsulation methods for datagram, but also an Address Resolution Protocol (1394 ARP) and a Multicast Channel Allocation Protocol (MCAP). Both 1394 ARP and MCAP are specific to a Serial Bus; the latter permits management of Serial Bus resources when used by IP multicast groups, however, the above two IP-over-Firewire implementations do

not follow this RFC document. There is a Linux driver project for IEEE 1394 [33], and it offers two modules namely: ip1394 and eth1394. The ip1394 module is the first attempt to implement an IP-over-Firewie under Linux. It partly follows the RFC 2734. The eth1394 module is more recent when comparing to the ip1394 module, while it does not follow the RFC 2734, it emulates the IEEE 1394 device as an Ethernet device, so that the TCP/IP protocol stack can utilize the IEEE 1394 device just like an Ethernet NIC. In the following sections, we will examine the different implementations of the IP-over-Firewire methodology.

3.3 Performance of IP-over-Firewire

The FireNet from Unibrain supports only the Windows and Macintosh platform, it is a full Ethernet emulation network and works seamlessly with all existing Ethernet compatible software and hardware. As claimed by Unibrain, when running at 400 Mbps, FireNet over TCP/IP is significantly faster than 100 baseT Ethernet and comparable to the Gigabit Ethernet (see Figure 3-1 and Table 3-1 [95]) [54]. The results depicted in Figure 3-1 were obtained by initiating FTP downloading and uploading of a 1.17GBytes file between two Pentium 4 workstations (1.7GHz, 512 RAM, and 60 GB HDD - 7200rpm).

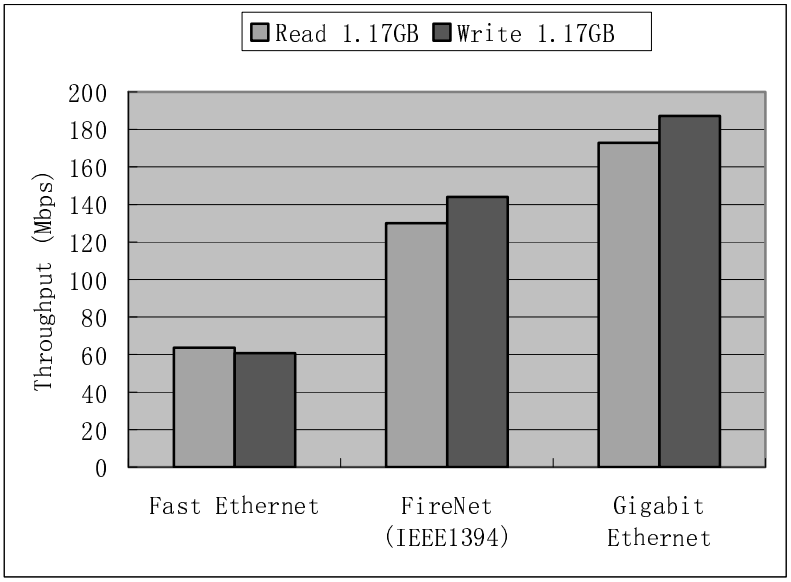


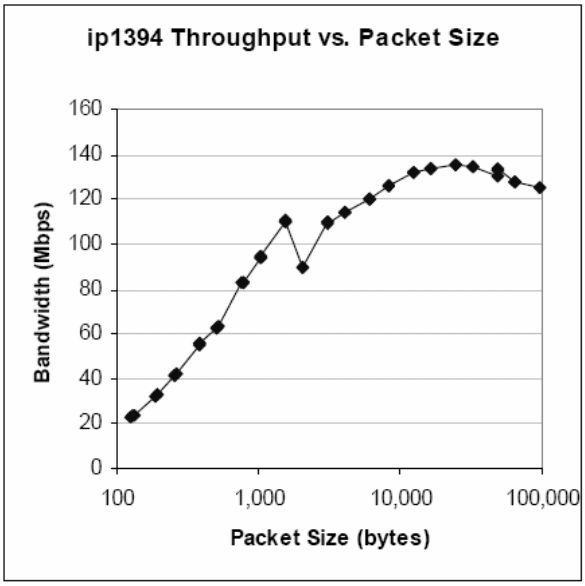
Figure 3-1: Performance of FireNet

Table 3-1: Performance of FireNet

	Read 1.17GB	Write 1.17GB
Fast Ethernet	63.68 Mbps	60.8 Mbps
FireNet(IEEE 1394)	130 Mbps	144 Mbps
Gigabit Ethernet	172.8 Mbps	187.2 Mbps

The Linux 1394 project provides a more complete device driver set to support the 1394 device. It supports both OHCI and PCILynx type hardware, both are implementations of the IEEE 1394 specification, and most new IEEE 1394 devices are based on the OHCI type only. The Linux 1394 implements IP-over-Firewire by two modules: ip1394 and eth1394. In [48], bandwidth of both ip1394 and eth1394 against the packet size were provided (Figure 3-2). The data were collected by initiating TCP write() method between two computers,

and the resulting data is the average of 4096 consecutive write() operations. As shown in Figure 3-2, the peak performances of the two drivers are almost the same as that of the FireNet. By comparing the performance data of FireNet, ip1394 and eth1394, we can observe that although they can achieve a good performance, by comparing to the standard Ethernet, they all cannot exceed a throughput limit which is around 140 Mbps. The 140 Mbps limit is far lower than the nominal 400 Mbps bandwidth of the Firewire device which is defined by IEEE 1394-1995 standard [37]. As defined in the IEEE 1394, the bandwidth of the asynchronous transfer mode is around 130 Mbps, which is close to the maximum bandwidth obtained in FireNet, ip1394 and eth1394. This may imply that only the asynchronous transfer mode is utilized in these IP-over-Firewire implementations.



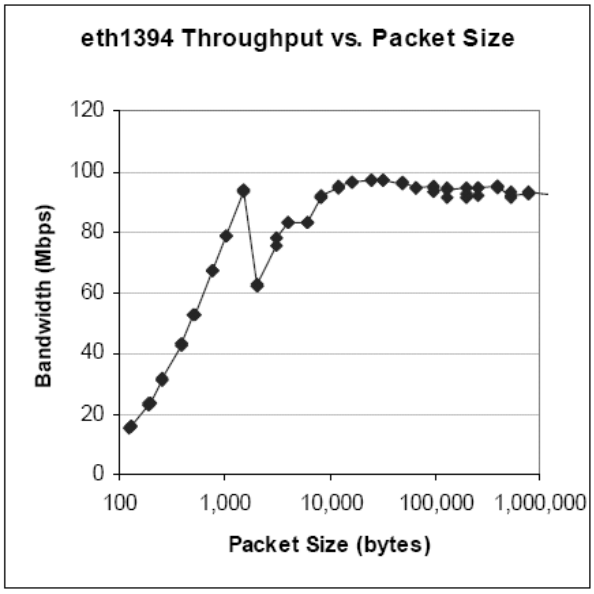


Figure 3-2: Performance of ip1394 and eth1394

When we consider using the Firewire as an interconnection interface for cluster computing, IP-over-Firewire may not be an ideal option. The TCP/IP protocol is designed to be a general purpose protocol, and to satisfy the demand for internet connections. There is a large overhead in the TCP/IP protocols to support internet addressing, options etc [82]. As an example, there is always a 20-byte IP header for each packet regardless of the size of the payload, and a 20-byte header for TCP or UDP mechanism (Figure 3-3). While these headers do not impose a real problem for internet data transfer as a data packet is often several kilobytes in length, however, it may become a major overhead if the payload is only consisted of tens of, or hundreds of, bytes, which is quite common in certain kind of cluster computing applications. In the TCP mechanism, it uses some form of generic flow control mechanism such as Sliding Window and Delay ACK [82] to enhance the performance for stream transport, but this also makes TCP especially not suitable for short message transportation, because it is quite

likely that there will not be enough data packets to trigger the Delayed ACK, and data that is waiting for transmission has to wait for time out which is often in the scale of hundreds of millisecond. Most of the applications for cluster computing cannot tolerate such long latencies. Although there is no such flow control scheme in UDP, the overhead is still a significant factor, so a protocol that is specifically designed for the interconnection of cluster systems is required.

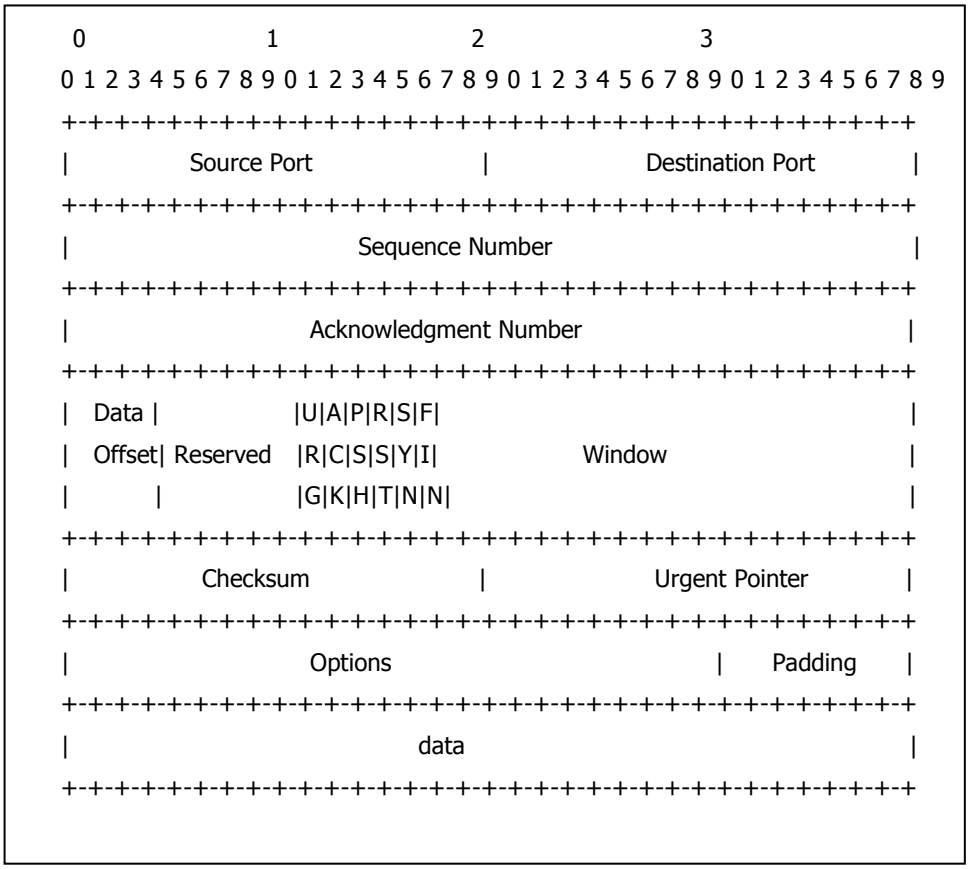


Figure 3-3: TCP Header Format

There are a few techniques to reduce the overhead, given in [108], [59], [32], most of them are based on the Linux1394 driver, and by modifying the protocol

stack to optimize the performance for specific demands. In [108], they proposed a priority queuing system based on the linux1394 driver to support guaranteed QoS. In [32], they develop a new link layer IC module to avoid the overhead of the IEEE 1394 protocol, so that the maximum capability of the IEEE 1394 physical layer can be used. Figure 3-4 and 3-5 show the IEEE 1394 round-trip time and throughput data given by [32], they used their own communication library called CheFcat (*CF*) on a real-time Linux operation system, working beyond the Linux 1394 driver. In this way the TCP/IP protocol is bypassed and related overhead is reduced. The feature of remote memory write that is supported by the OHCI interface [91] is used to improve the performance. As reported in [32], a minimum round-trip time of 18.8 microseconds, and a maximum throughput approaching 300 Mbps, can be achieved and this is very close to the maximum transfer rate, 400 Mbps, stipulated in the 1394 standard. Moreover, it also out-performs the results obtained from the FireNet, ip1394, or eth1394.

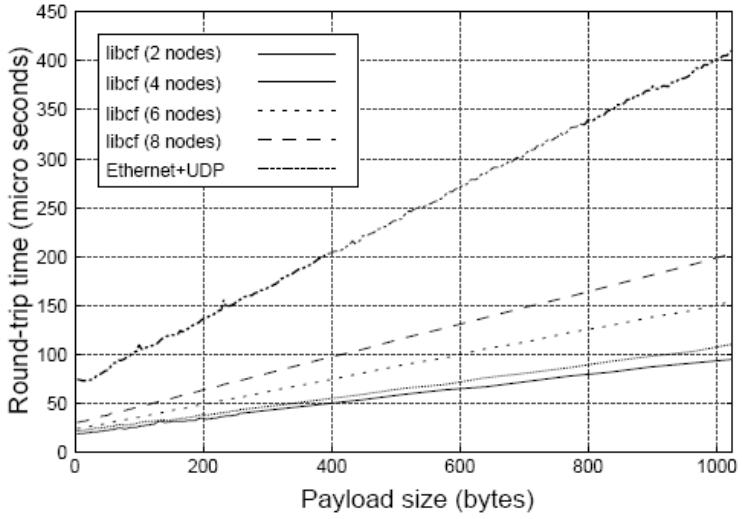


Figure 3-4: The round-trip time of CF on IEEE 1394

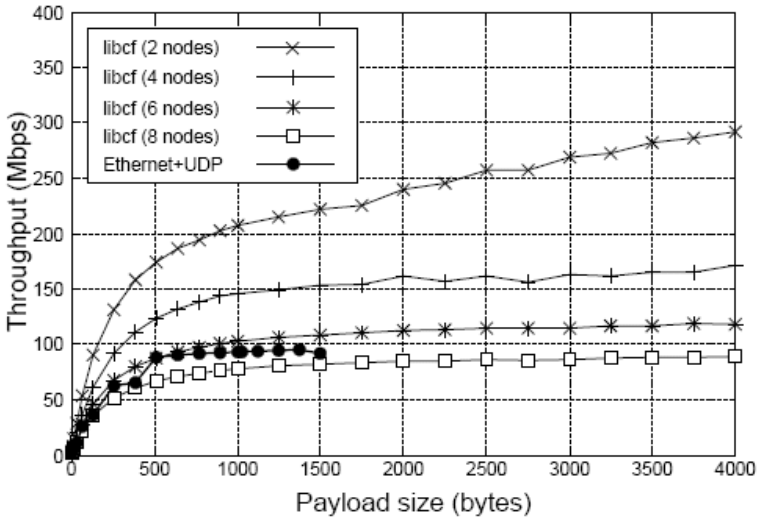


Figure 3-5: The throughput of CF on IEEE 1394

From the above results, we can observe that although IP-over-Firewire is readily available, it is not suitable for cluster computing, a new protocol which is designed specifically for the IEEE 1394 device and cluster computing environment is needed. As proved by [32], a transfer-rate of 300 Mbps can be achieved comparing to only 140 Mbps obtained in the standard approach. In the following section, the implementation details of a network transmission protocol which is developed for the Linux system is described, and the performance of the protocol is evaluated.

3.4 A new communication protocol

We adopted the Linux OS as the implementation platform, and the Linux1394 driver is employed as the underlying layer to implement our communication

protocol. It is because of the open-source policy of the Linux OS. So the library program development of the communication protocol can be carried-out more easily.

3.4.1 The Linux1394 driver

The Linux1394 device driver is now an official Linux driver project, which is included in Linux versions from 2.2 to 2.6. It supports Texas Instruments PCILynx/PCILynx2 and OHCI compliant chips (produced by various companies). The driver is now under active development in the Linux kernel of version 2.6.

Figure 3-6 shows the hierarchy of the Linux1394 driver. The core of the entire 1394 subsystem is the module ieee1394, as shown in Figure 3-6. It manages all high and low-level drivers in the subsystem, handles transactions, and provides a mechanism for event triggering. Underneath the ieee1394 module are the low-level (hardware) driver modules. There are two low-level drivers, which are hardware dependent. In our test environment, all the interface cards are OHCI1394 compatible so we use the ohci1394 module to interface with the 1394 interface card. It is possible to have all three low-level driver modules loaded and activated at the same time. All low-level drivers can control more than one card (the default maximum is four per low-level driver). Above the ieee1394 module are the high-level driver modules. One such high-level driver

is the “raw1394”, which provides an interface for user-space applications to access the raw 1394 bus. To access the raw 1394 bus from user-space, one can interact with the raw1394 modules through the device file */dev/raw1394*, or choose to link applications with a programming library called *libraw1394*, which handles the communication with the raw1394 high-level driver, to ease the manipulation of the device file. In our preliminary tests, we use the libraw1394 to control the IEEE 1394 bus.

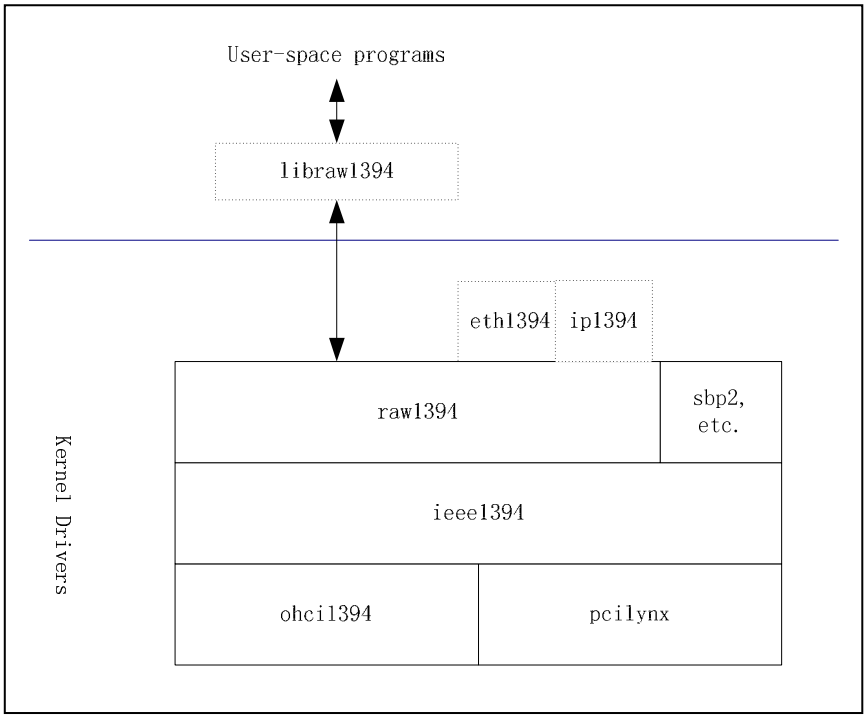


Figure 3-6: The Linux 1394 Driver Modules' Architecture

3.4.2 The Transmission feature of IEEE 1394

As discussed in Section 2.2, the IEEE 1394 device supports two basic data transfer services, namely the asynchronous data transfer and isochronous data trans-

fer.

The asynchronous data transfer service provides a packet delivery protocol for variable length packets to an explicit address and return of an acknowledgment. Transactions are multithreaded, in that more than one transaction can be started by a requester before the corresponding response is returned. These are called a split-response transaction. The isochronous data transfer service provides a broadcast packet delivery protocol for variable-length packets that are transferred at regular intervals. As shown in Figure 2-1, the asynchronous data transfer service uses the transaction layer, whereas isochronous data transfer service is application driven, thus the isochronous transfer has fewer overload than the asynchronous transfer.

As shown in Figure 2-2, the IEEE 1394 bus operates in a cycle mode, isochronous transmissions always occur before all asynchronous transmissions. The isochronous cycle begins after a *cycle start* signal is sent on the bus, and ends when a subaction gap is detected. During an isochronous cycle, only isochronous subactions can be active. An isochronous cycle occurs every 125μs, on average. For the asynchronous transmission, the Serial Bus architecture limits the maximum number of data bytes in a transaction to the largest power-of-two such that the whole asynchronous link-layer packet transmission takes less than 62μs (this is half the isochronous cycle time, and restricting asynchronous subaction to such value can help minimizing buffer requirements in the IEEE 1394 interface card). If we consider the bus operating at 100 Mbps

data rate, this means that the data payload of asynchronous packets is limited to 512 bytes. Longer packets can be sent at higher data rates and shorter packets at lower data rates. If the bus is operating at 400 Mbps data rate, the maximum payload size is 2048 bytes (Table 3-2).

Table 3-2: Maximum payload size for IEEE 1394 asynchronous packets

Data rate	Maximum payload size (bytes)
100 Mbps	512
200 Mbps	1024
400 Mbps	2048

The different definitions of the asynchronous mode and isochronous mode in the 1394 standard induce totally different transmission features for the two modes. The isochronous mode always takes higher priority than the asynchronous mode, therefore, if the two modes are competing for the bus, the isochronous mode is guaranteed to occupy most of the bandwidth, and only a few of the asynchronous mode transmission can be completed. The asynchronous mode's packet size is restricted so that a single asynchronous mode transmission cannot exceed half of the bus's bandwidth, even when there is no isochronous mode applications is competing with it.

In order to evaluate the basic performance of the IEEE 1394, a series of experiments was performed based on the Linux1394 driver with the raw1394 interface; we concentrated on the maximum throughput and latency of the asynchronous and isochronous modes. Our tests were conducted on a simple network

of 2 generic workstations, and the hardware configuration of the workstations included Pentium III 500MHz microprocessor, 128M RAM of memory, and each has an IEEE 1394a PCI interface installed, with a maximum data transfer rate of 400 Mbps, the OS is Linux 2.6 with the latest IEEE 1394 driver, configuration of the test bed is shown in Figure 3-7.

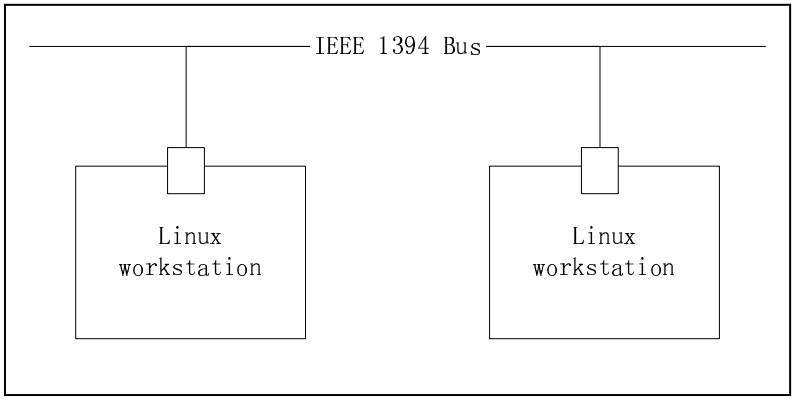


Figure 3-7: The test-bed for IEEE 1394 transmission feature

3.4.2.1 Asynchronous mode transmission feature

There are three different transaction types of the asynchronous mode transmission.

a) Read—data at a particular address within a responder is transferred to a requester.

b) Write—data is transferred from a requester to an address within one or more responders.

c) Lock—data is transferred from a requester to a responder, processed with data at a particular address within the responder, and then transferred back

to the requester.

Read and write transactions are mandatory in the IEEE 1394 standard, and they are for block data payload transmission. Lock transaction is optional in the standard and is designed for data manipulation on data of either 32 bits or 64 bits. As we are interested in transmitting block data through the IEEE 1394 bus, therefore, we concentrate on testing the performance of the read and write transactions.

The asynchronous write transaction's feature was evaluated first. We want to examine the relationships between throughput and latency with respect to the packet size. The test was carried out by running two programs on two workstations, one of the programs was responding to send data to the other program through the IEEE 1394 bus. Data received by the other program was verified, but no acknowledgment was return, as this will introduce extra dataflow on the bus. The packet size varied from 8 bytes to the most significant value defined, which is 2048 bytes for the 400 Mbps IEEE 1394 card. For each packet size, the write transaction was performed 1024 * 100 times back-to-back, the starting time and the ending time of the operation were recorded, and the resultant latency and throughput were the average of the 1024 * 100 transactions.

Figure 3-8 and Figure 3-9 show results obtained from the tests (referring to Appendix A, Table A-1 and Table A-2 for detailed results). Figure 3-8 depicts latencies of the write transaction corresponding to different sizes of data packet. The minimum latency obtained is 50 μs, which is obtained at the minimum

packet size (eight bytes). Based on our results, the latency is linearly increased with the packet size. When the packet size is 2048 bytes, the maximum value, the latency reaches a maximum value of 125 μs. This is because if one packet occupies all the allowable bus transmitting time with the maximum packet size in one cycle (20% of the whole cycle time [3]), the following packet has to wait for the next cycle time in order to initiate a transfer, which is exactly 125 μs from the cycle structure of the IEEE 1394 bus (Figure 2-2). Figure 3-9 depicts the throughputs of the write transaction with respect to the different data packet sizes. The throughput increases when the packet size increases, and the maximum throughput reaches 125 Mbps when packet size is also the maximum, i.e. 2048 bytes, and this is the limiting throughput that a single asynchronous mode transaction can obtain.

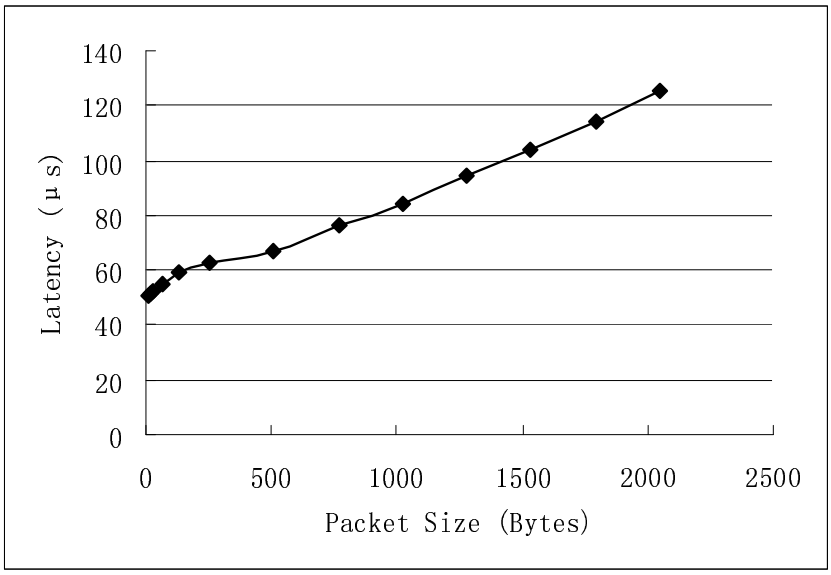


Figure 3-8: Latency of the asynchronous write transaction

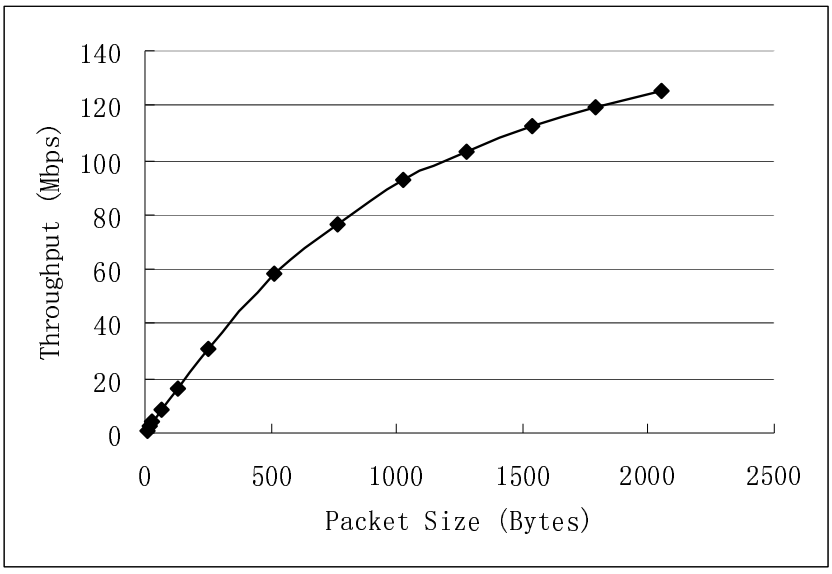


Figure 3-9: Throughput of the asynchronous write transaction

Tests for asynchronous read transaction were also conducted, and the results are listed in Table 3-3. Basically the asynchronous read transaction shows a similar characteristic as that of the asynchronous write transaction. The minimum latency time is around 50μs while the maximum is 125μs. The maximum throughput is also around 125 Mbps.

Table 3-3: Latency and throughput vs. packet size of the asynchronous read transaction

Packet Size	Latency(μs)	Throughput(Mbps)
8	50.776045	1.202046
16	52.636934	2.3191
32	50.950156	4.791754
64	56.195713	8.688941
128	57.586152	16.958287
256	62.513848	31.243078
512	62.965928	62.03752
768	75.679121	77.42393
1024	83.425527	93.646396

1280	95.567822	102.185283
1536	104.431104	112.315131
1792	113.312256	120.656631
2048	125.402773	124.59852

3.4.2.2 Isochronous mode transmission feature

Based on the same test-bed, the performance of the isochronous transfer mode was also examined. During the tests, there were also two programs running on the two workstations, one of the programs initiated an isochronous data flow to a specific isochronous channel (an isochronous channel is an identifier to a group of nodes of the IEEE 1394 bus, all nodes which listen on the same channel receive data that are sent to this channel); the bus speed was set to 400 Mbps. The other program which was running on the other workstation received data flowing on the same isochronous channel, only necessary data validation was made and no acknowledgment was returned. A total of 1024 * 30 packets were sent back-to-back, the latency and throughput value were once again the average value of the 1024 * 30 packets. The results are shown in Figure 3-10 (detailed results show in Appendix A, Table A-4), since the packet size limit for isochronous mode is 4096 bytes at 400 Mbps transfer rate, a single isochronous mode transfer data rate can reach a maximum throughput of 250 Mbps at 4096 bytes packet size. Figure 3-11 shows that the latency, around 125μs, of the isochronous transfer mode is almost unaffected by the changes in packet size. This is because as defined in the IEEE 1394 standard, when the isochronous mode is

transferring at the speed of 400 Mbps; the actual packet that is transferred in the physical layer should always be 4096 bytes long, which occupies all the allocated time of a cycle. The user packet which is sent from an application to the IEEE 1394 bus interface will always be padded to 4096 bytes, so the different packet length makes almost no difference in the test.

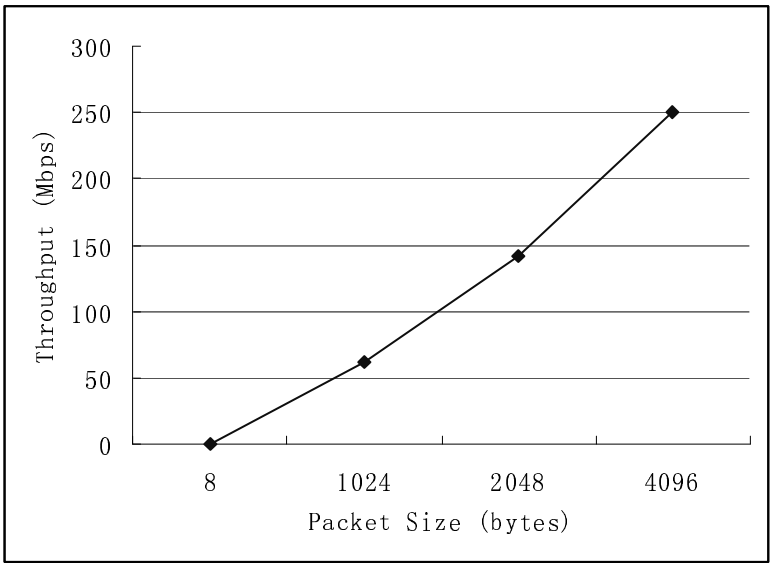


Figure 3-10: Throughput of the isochronous mode transmission vs. packet size

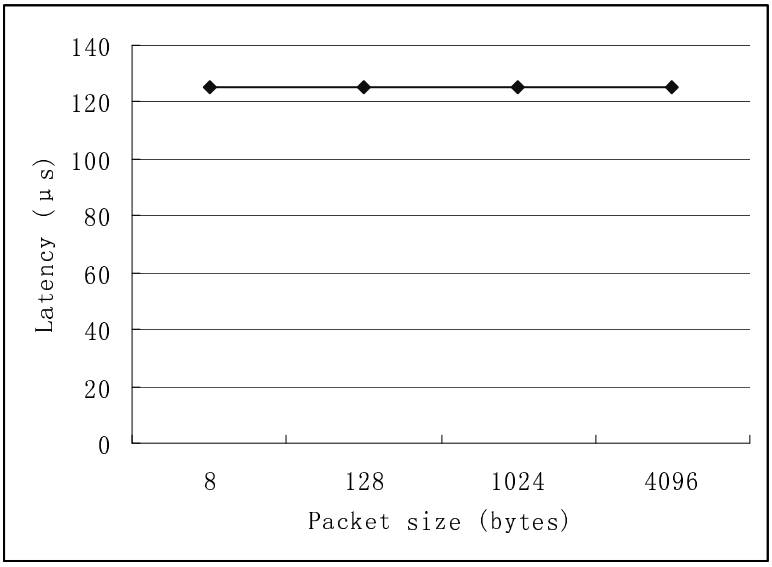


Figure 3-11: Latency of the isochronous mode transmission vs. packet size

According to our test results, the maximum throughput for the asynchronous mode is about 125 Mbps, which is similar to results presented by other studies, including [73], [32]. On the other hand, for isochronous mode, the maximum throughput is 250 Mbps, which is far better than the asynchronous transfer mode. However, the design of the isochronous mode transfer is not ideal for general purpose networking applications therefore, further studies are required. In addition, we would like to investigate techniques to fully utilize the device's maximum capacity by operating in both transfer modes.

3.4.2.3 Mixed mode transmission feature

In this section, the performance feature of a mixed mode transmission will be studied. We will examine how the asynchronous mode and isochronous mode transmissions influence each other when they are both active on the same IEEE 1394 bus. In order to test the mixed mode features, an asynchronous mode transmission and an isochronous mode transmission were initiated between two workstations at the same time. The bus rate was set to 400 Mbps, an isochronous mode data flow of 4096 bytes size was sustained between the two workstations. The reason that we used a constant packet size for the isochronous mode was based on the results discussed in Section 3.4.2.2, which shows the performance features of the isochronous mode are almost unaffected by the variation of the isochronous packet size. At the same time, an asynchronous write data flow between the two workstations was also active, the packet size of the write transac-

tion varied from 8 to 2048 bytes. Latency and throughput corresponding to each packet size were obtained by averaging the results of 1024 * 100 transactions obtained at each packet size.

Figure 3-12 and Figure 3-13 depict the throughput results of this mixed mode transmission, with respect to the variation of the packet size of the asynchronous mode transmission (the packet size of the isochronous mode transmission was fixed at 4096 bytes). As shown in Figure 3-12 and Figure 3-13, the isochronous mode transfer is hardly influenced by the concurrent asynchronous mode transmission at all, its latency is at a steady state at around 125μs, and the throughput is sustained at around 250 Mbps. On the contrary, the asynchronous mode transmission is highly affected by the isochronous mode transmission. The asynchronous mode latency goes beyond 300μs even at the 8 bytes packet size. Although a single asynchronous mode transmission shows much lower latency than that of the single isochronous mode, its latency is much higher when competing with an isochronous mode transmission, even higher than that of the isochronous mode, which is the bus cycle time, 125μs. This implies that the asynchronous mode transmission which can formerly be accomplished with one bus cycle, is now required three or more bus cycle time to accomplish. The maximum throughput of the asynchronous mode can only reach 28 Mbps at the mixed mode transmission. These phenomena are predictable since the isochronous mode is designed to be of higher priority than the asynchronous mode. Also from Figure 3-12, we can observe that the sum the two transfer mode gives

an overall throughput of 278 Mbps, which is quite close to the IEEE 1394 maximum nominal rate of 400 Mbps.

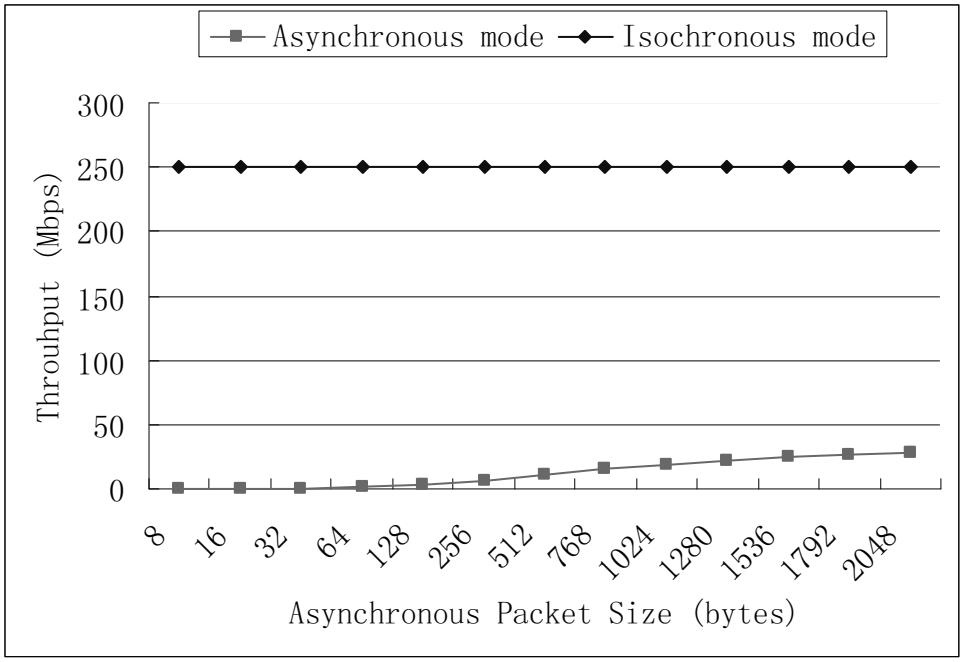


Figure 3-12: Throughput of mixed mode transmission vs. asynchronous mode packet size

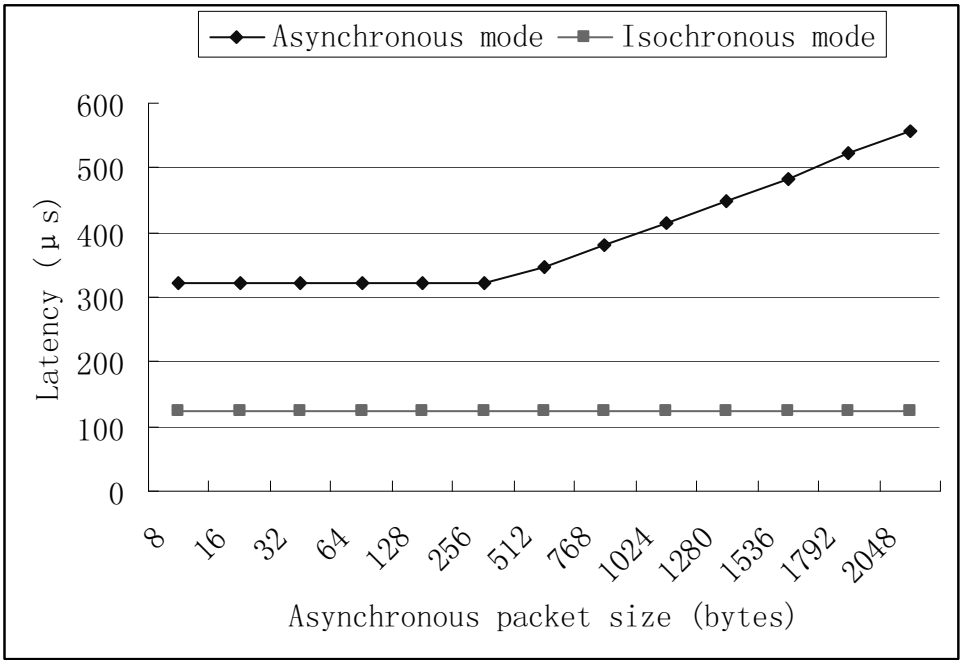


Figure 3-13: Latency of mixed mode transmission vs. asynchronous mode packet size

3.4.3 Services of the protocol

In the last section, we examined many aspects of the transmission features of the IEEE 1394 bus. The asynchronous mode supports guaranteed data delivery, it transmits data in low latency, but it can only sustain a maximum throughput of 125 Mbps. The isochronous mode can sustain much higher throughput around 250 Mbps, but it does not support guaranteed data delivery, and it operates only in broadcasting mode. Comparing to the performance of IP-over-Firewire, the isochronous mode certainly provides much higher throughput, but cannot provide guaranteed data delivery service. To fully utilize the IEEE 1394 device's bandwidth, we proposed a communication protocol which utilizes the throughput of the isochronous mode of the IEEE 1394 device, while at the same time supports the guaranteed data delivery service, and facilitates parallel computing of transportation systems on the IEEE 1394 cluster. In order to achieve this, we need two basic services, including send service and receive service.

Sending and receiving messages by processes are the basic communication mechanism of the protocol. The basic point-to-point communication operations are **fw_send** and **fw_receive**. Their uses are illustrated in the example, as shown in Figure 3-14.

In this example, process zero (myprocessID = 0) sends a message to process one using the send operation **fw_send**. The operation specifies a send buffer in the sender memory from which the message data is taken. In the example above, the send buffer consists of the storage containing the variable message in the

memory of process zero. The location and size of the send buffer are specified by the two parameters of the send operation. The message sent will contain the 13 characters of this variable. In addition, the send operation associates an envelope with the message. This envelope specifies the message destination and contains distinguishing information that can be used by the receive operation for selecting a particular message. The last two parameters of the send operation specify the envelope for the message sent.

```
main( argc, argv )
int argc;
char **argv;
{
char message[20];
int myprocessID;
int mygroupID=10;
int status;
fw_Init( &argc, &argv );
fw_setGroupID(myhostID);
fw_getProcessID(&myprocessID);
if (myprocessID == 0) /* code for process zero */
{
strcpy(message,"Hello, there");
fw_send(message, strlen(message), 1, mygroupID);
}
else /* code for process one */
{
fw_receive(message, 20, 0, mygroupID);
printf("received :%s:\n", message);
}
fw_finalize();
}
```

Figure 3-14: Sample code of the send and receive services

Process one (myprocessID = 1) receives this message with the receive operation fw_receive. The message to be received is selected according to the value of its envelope, and the message data is stored in the receive buffer. In the above ex-

ample, the receive buffer consists of the storage containing the string message in the memory of process one. The first two parameters of the receive operation specify the location and size of the receive buffer. The last two parameters are used for selecting the incoming message.

The protocol provides the user-space program with reliable message transmission. A message sent is always received correctly, and the user does not require to check for transmission errors, time-outs, or other error conditions. In another word, the protocol does not provide mechanisms for dealing with failures in the communication system. Since the isochronous mode transmission of IEEE 1394 is an unreliable data transmission mechanism, Acknowledgment mechanism should be implemented in the protocol to insulate users from this unreliable feature.

The send and receive procedures devised can operate in the following two modes:

Non-blocking

The procedure may return before an operation completes, and therefore the user is allowed to re-use resources (such as buffers) specified in the non-blocking call.

Blocking

Return from the procedure indicates that a user is allowed to re-use resources specified in the call.

Although implementation of the blocking mode is much easier than that of the

non-blocking mode, the non-blocking mode has to be implemented to improve the performance in some cases. Another subtle issue arises because of the nature of asynchronous communications of the non-blocking mode: the protocol called may initiate operations that continue asynchronously after the call returns. Thus, the operation may return with a code indicating successful completion, yet later causes an error exception to be raised. If there is a subsequent call that relates to the same operation (e.g., a call that verifies that an asynchronous operation has completed) then the error argument associated with this call will be used to indicate the nature of the error. In a few cases, the error may occur after all calls that relate to the operation have completed, so that no error value can be used to indicate the nature of the error. Such an error must be treated as fatal, since information cannot be returned for the user to recover from it. In the previous paragraphs, we described the send and receive services. In the following, we continue with the description of the syntax of the functions based on services devised above.

The syntax of the fw_send operation is given below.

int fw_send (buf, count, dest, group)

buf	initial address of send buffer
count	number of bytes in send buffer (nonnegative integer)
dest	destination process ID(integer)
group	destination group ID(integer)

The fw_send call is blocking. It does not return until the message data and en-

velope have been safely stored away so that the sender is free to access and overwrite the send buffer. The message might be copied directly into the matching receive buffer, or it might be copied into a temporary system buffer.

Message buffering decouples the send and receive operations. A blocking send can complete as soon as the message is buffered, even if no matching receive has been executed by the receiver yet. On the other hand, message buffering can be expensive, as it entails additional memory-to-memory copying, and it requires the allocation of memory for buffering. Our protocol offers the choice of several communication modes that allow one to control the choice of the communication protocol.

The send call does not define the message buffering behavior. It is up to the protocol to decide whether outgoing messages will be buffered. The protocol may buffer outgoing messages. In such a case, the send call may complete before a matching receive is invoked. On the other hand, buffer space may be unavailable, or the protocol may choose not to buffer outgoing messages, for performance reasons. In this case, the send call will not complete until a matching receive has been posted, and the data has been moved to the receiver.

Thus, a send can be started whether or not a matching receive has been posted. It may complete before a matching receive is posted. The standard mode send is non-local, successful completion of the send operation may depend on the occurrence of a matching receive.

int fw_receive (buf, count, source, group)

buf	initial address of receive buffer
count	number of bytes in receive buffer (nonnegative integer)
source	source process ID(integer)
group	source group ID(integer)

The receive buffer consists of the storage containing a counter of consecutive bytes of data, starting at address *buf*. The length of the received message must be less than or equal to the length of the receive buffer. An overflow error occurs if all incoming data does not fit, without truncation, into the receive buffer. If a message that is shorter than the receive buffer arrives, then only those locations corresponding to the (shorter) message are modified.

The receive operation is blocking; it returns only after the receive buffer contains the newly received message. A receive can complete before the matching send has finished (of course, it can finished only after the corresponding send has started).

3.4.4 Implementation of the protocol

In the above sections, we have described the major features of the send/receive functions. In order to support the reliabilities of the services as described above, an acknowledgement mechanism over the IEEE 1394 isochronous transmission mode is developed so that the success of data delivery can be guaranteed while at the same time higher throughput can be achieved. A packet multiplexing

mechanism over the isochronous mode will also be introduced so that the isochronous mode can be used to carry data between random nodes instead of broadcasting data.

3.4.4.1 Implementing acknowledgement over the isochronous mode

We use the asynchronous mode to implement the Acknowledgement. The data payload is fitted into a packet, together with a packet header. The packet header includes a Source field (Src), that indicates which node in the bus is the sender, and a Sequence Number field (SN), that shows the sequence number of the packet, and the SN field is increased by one after each operation. Figure 3-15 depicts the acknowledgement mechanism.

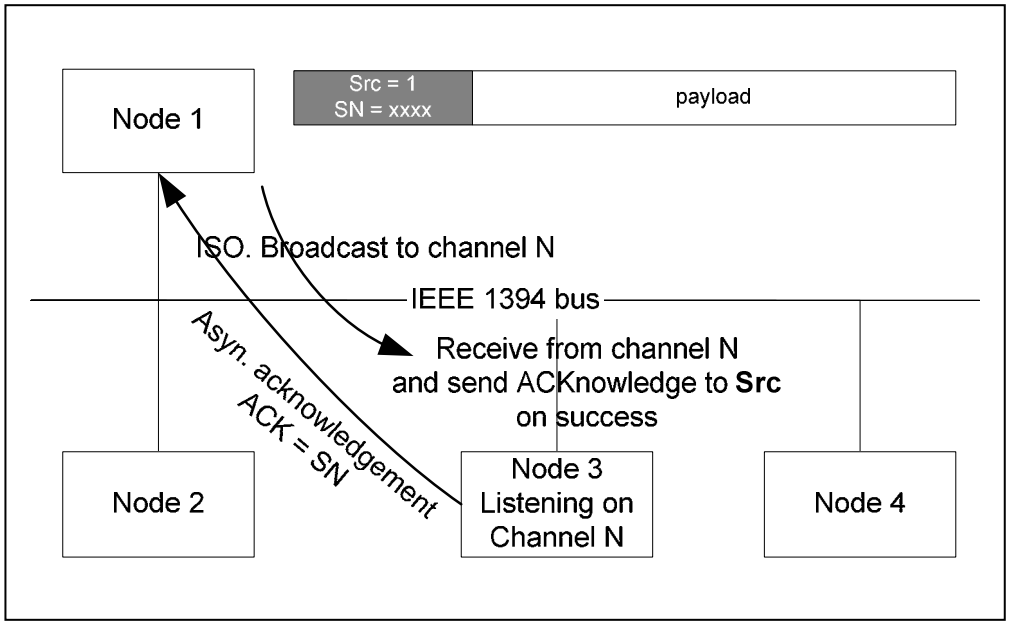


Figure 3-15: The acknowledgement mechanism

When two nodes are communicating, data packets are sent through the isochro-

nous transfer, on a channel number that is previously negotiated. The receiver receives the data packet at the same channel number, with each success delivery of the packet, an acknowledgement to that packet, with the SN number that is obtained from the SN field, will be returned through the asynchronous mode to the sender, which is obtained from the Src field. Since an acknowledgement is normally a small packet, this should incur little interference to the isochronous transfer.

A simple sliding window mechanism like in the TCP protocol [82] is also employed in our implementation of the acknowledgement, so that the data sending processes can overlap, i.e., an isochronous packet can always be sent without waiting for the acknowledgement of the packet which is sent before it. Since when packets are sending out, the sequence numbers are in sequence, the acknowledgements that are received by the sender should also be in sequence. So an out of sequence acknowledgement means that a packet lost or damage of packet has occurred, and all unacknowledged packets are then resent, which is an automatic process already included in our communication protocol.

Figure 3-16 illustrates the overlapped data sending mechanism. We highlighted that the isochronous data transmission time of a packet is 125μs, as we obtained in previous test. The data transmission time for an asynchronous mode acknowledgement packet is 50μs, which is the time of sending an asynchronous mode packet with only 8 bytes payload. From this figure, we can see that the accomplishment of an isochronous data packet sending time with the acknowl-

edgement, or latency, can be calculated as 125μs + 50μs + processing time on both sides, which is around 180μs in our empirical results. Although the acknowledgement mechanism increases the latency, the sliding windows mechanism enables overlapped data sending and the protocol can initiate next packet data sending as soon as the bus can be used, thus the throughput is almost not affected by the acknowledgement mechanism.

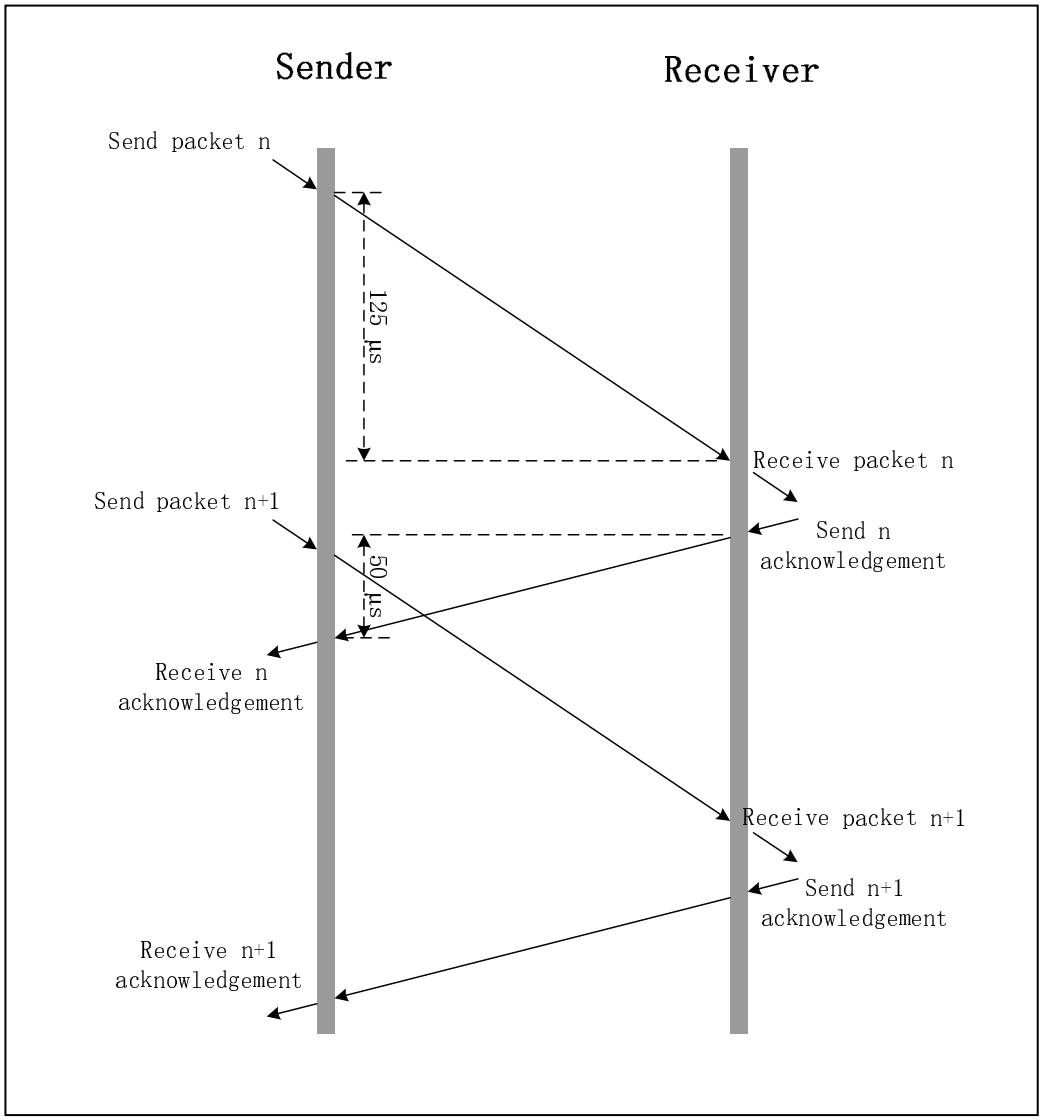


Figure 3-16: The illustration of the overlapped data transmission

Upon utilizing such acknowledgement mechanism in our communication library, we can now use the isochronous transfer to obtain higher throughput, and the success of data delivery can be guaranteed at the same time. We conduct a simple test of the isochronous mode throughput with the acknowledgement mechanism using our library. The test is carried out on a simple network with two nodes linked by an IEEE 1394a, and nodes' configurations are the same, including a Pentium 4 2.4GHz CPU with 256M memory.

In this test, we send back-to-back isochronous data packets from one node to the other, the data packet sizes are set to be 1024, 2048, 2072, 4096 bytes respectively, and the result is shown in Figure 3-17.

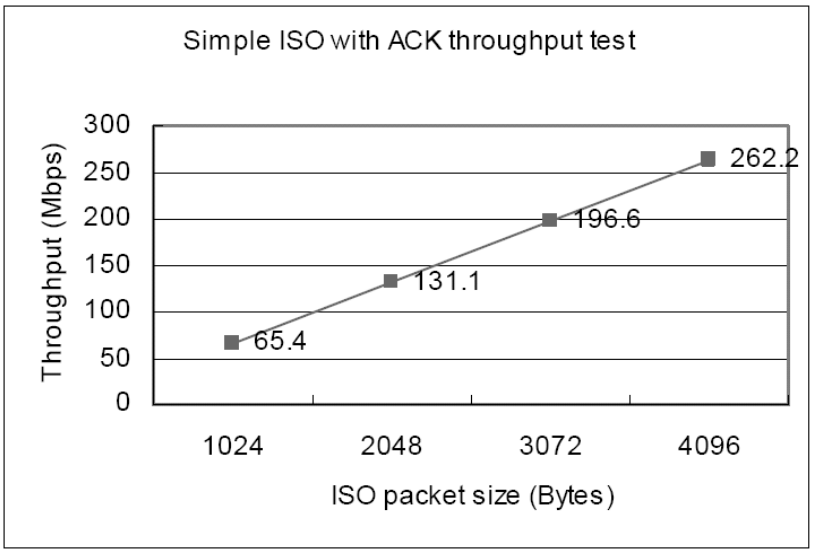


Figure 3-17: The throughput of isochronous mode with acknowledgement mechanism

From the results, we can observe that the throughput is almost linearly related to

the isochronous packet size, with the maximum throughput of 262.3 Mbps which is achieved at the packet size of 4096 bytes. Such a result is very close to the throughput result of the bare isochronous mode without the acknowledgement given in Figure 3-10.

Another possible method that can be used to improve the network throughput is to use multiple asynchronous mode data flows. Although as shown in Section 3.4.2.1, a single asynchronous mode transfer can only achieve 125 Mbps throughput at the maximum allowed asynchronous packet size of 2048 bytes, we can use multiple asynchronous transfers at the same time between the communicating nodes, so that the overall throughput can be improved.

Such a test was carried out on the same network. In a single data flow, the asynchronous packet size is set to 2048 bytes, the throughput of 1 to 4 data flows are recorded. The results are shown in Figure 3-18. We can see that as we expected, using multiple asynchronous transfer improved the overall throughput, but the maximum throughput is achieved when using 3 asynchronous transfers at the same time, using more concurrent transfers cannot increase the overall throughput. Most importantly, the maximum throughput that can be achieved in such a mode is only 180 Mbps which is still less than the maximum throughput, 262 Mbps that can be achieved in the isochronous mode with acknowledgement.

The reason for such a result could be due to the arbitration process of the asynchronous mode, since arbitration for the bus must be performed before each asynchronous packet transmission, and no data can be transferred during the ar-

bitration process, so the time is consumed during the arbitration. The multiple asynchronous transfers incur more unused bus transmission time, and when the concurrent asynchronous transfers are too frequent, the competition among these transfers will increase the overhead and therefore the overall throughput will be degraded.

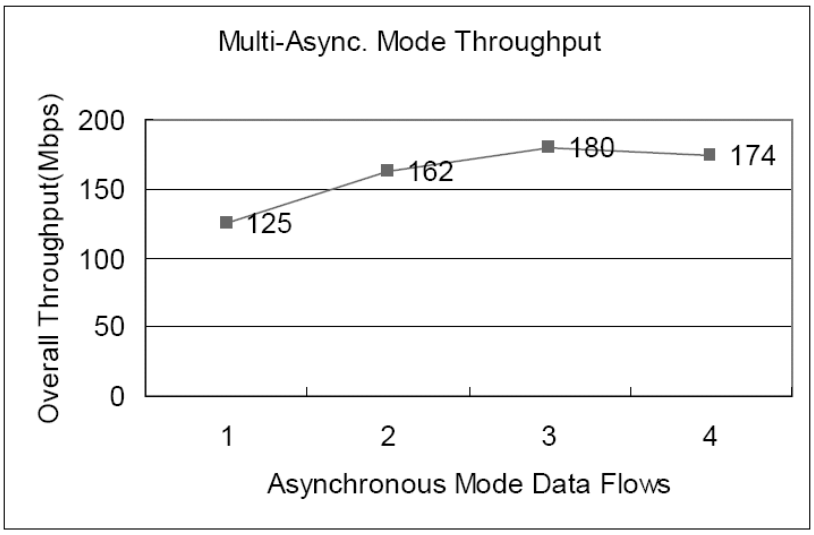


Figure 3-18: The throughput of multiple asynchronous transfers

3.4.4.2 Packet multiplexing for isochronous mode

With the acknowledgement mechanism imposed on the isochronous transfer, our communication protocol can offer better throughput with guaranteed data delivery. But there is a drawback of the isochronous transfer, when there are several isochronous transfers in the IEEE 1394 bus at the same time, the sum of the packet size of all the transfers cannot exceed 4096 bytes. If one wants to accommodate 8 isochronous transfers on the bus at the same time, each transferred

packet size can only be 512 bytes. Such packet size is too small for applications in many cases, and that will cause too much protocol overhead when transferring random size application data. In order to minimize such shortcoming, we introduce a multiplexing mechanism in our communication library, by taking advantage of the broadcasting nature of the isochronous mode. As shown in Figure 3-19, node 1 is broadcasting isochronous data through a specific channel, and the other three nodes all listen to this channel, so that all the data can be received at every node. Node 1 initiates all data sending request to the three nodes, and multiplex those request packets into one single IEEE 1394 physical packet if possible. Since all other nodes can see and examine the whole physical packet, while extracting the upper layer packet that is bound for them.

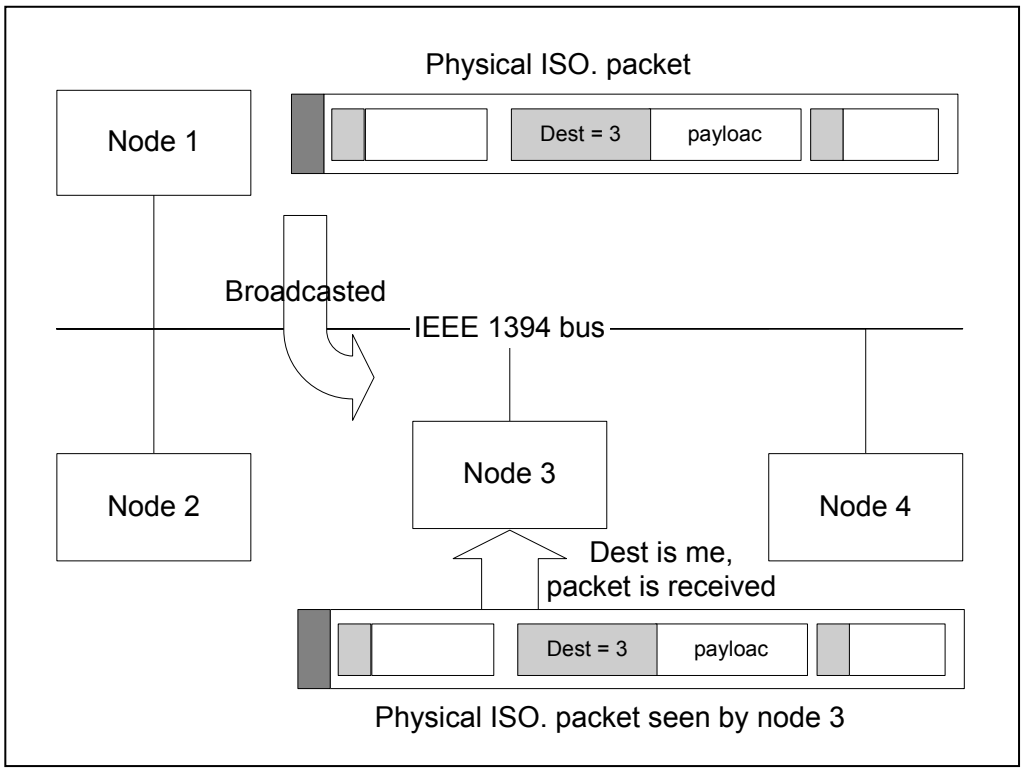


Figure 3-19: The multiplexing of the isochronous mode

With such multiplexing mechanism, one node needs only one isochronous mode transfer to send data to all other nodes. This makes carrying data packet in different sizes more flexible, large size protocol packet can occupy the whole physical packet, while small size protocol packet can be multiplexed into one physical packet to reduce the waiting time, and larger physical packet size is achieved by reducing the overall Isochronous mode transfer number. Such multiplexing mechanism is especially suitable for the Railway Simulation cluster as in Figure 2-4, where a single isochronous transfer with multiplexing as well as acknowledgement can be used to send animation data from the central simulator to multiple virtual reality display terminals, details are discussed in Chapter 5.

The throughput of such multiplexing mechanism is tested on an IEEE 1394 network with 4 nodes. We tested the throughput with fixed packet size, from 1 sender to 2 receivers, packet size is 2048 bytes, and 1 sender to 3 receivers, packet size is two 1024 bytes and one 2048 bytes, and 2 senders each sending data to 2 receivers, packet size of each is 1024 bytes respectively. The results are depicted in Figure 3-20.

We also tested the throughput with random data packet size on the network, we sent packets from one node to the other three, using the isochronous mode, with the acknowledgement and multiplexing mechanism. The data flow characteristic in the test is rather simple, the physical packet size is always 4096 bytes, at each time the sending node needs to send 3 protocol packets to the 3 nodes respectively, and the packet size is sampled from a uniform random distribution.

In our tests, 4 distributions were tested, in test No.1, the packet size of each dataflow was uniformly distributed between 128 - 2048. In test No.2, the packet size of each dataflow was uniformly distributed between 1024- 2048. In test No.3, the packet size of each dataflow was uniformly distributed from 2000 to 2048. In test No.4, the packet size of each dataflow was fixed at 2048. The test results are given in Figure 3-21.

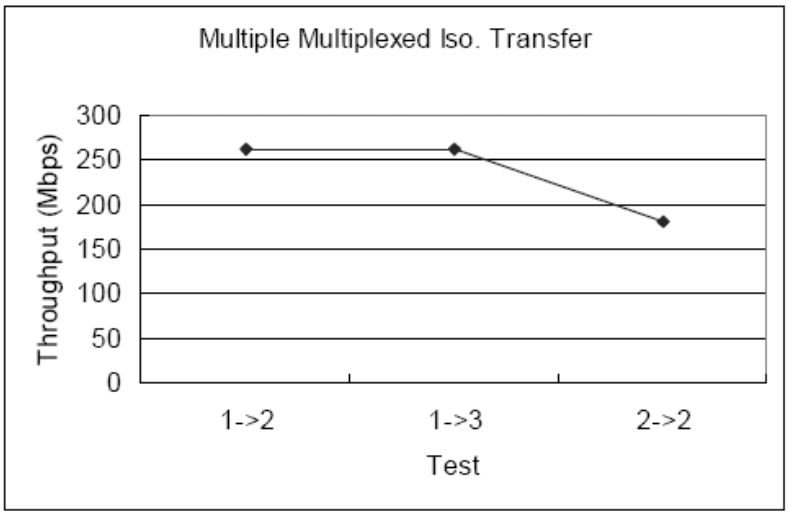


Figure 3-20: The multiplexing throughput of the isochronous mode with fixed packet size

In Figure 3-20 and 3-21, we can see that with the fixed packet size, maximum throughput is around 260 Mbps, that is almost the same as using bare Isochronous mode, but when data flows that transfer simultaneously is 4, the performance drops, this shows that such a mechanism cannot accommodate too many data transfers, but considering the IEEE 1394 is being used in a point-to-point structure as shown in Figure 2-5, it should not be a major problem. With the random packet size, the throughput is around 160 – 200 Mbps, it is still better than using the asynchronous mode. In test No.2 we get the minimum throughput,

that maybe due to the inefficient utilization of the physical packet size introduced by the multiplexing at that point.

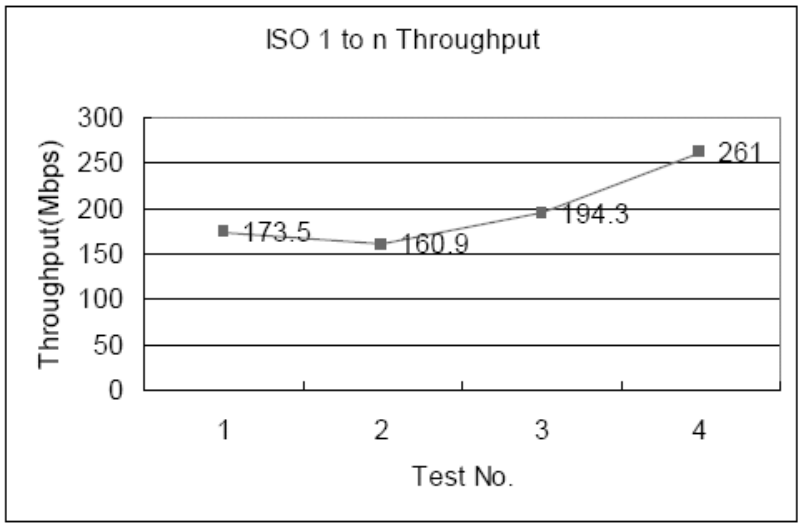


Figure 3-21: The multiplexing throughput of the isochronous mode with random packet size

3.5 Concluding remarks

In this Chapter we discussed the implementation details of a new communication protocol over the IEEE 1394 device. Message passing mechanism is offered in the communication protocol, so that the protocol can be used to build the interconnections of cluster using the IEEE 1394 bus. This enables parallel applications which employ the message passing mechanism to run on an IEEE 1394 cluster. By identifying the IEEE 1394 data transmission features of different transmission modes, we chose to implement an acknowledgement mechanism over the isochronous mode of IEEE 1394, to enable the isochronous mode

transmission to support reliable data transmission. The maximum throughput which we can achieve with our protocol is around 250 Mbps, and conforms to the study in [69]. This throughput is much higher than that of other implementations like the IP-over-Firewire under Windows or Linux. The minimum latency of our protocol is around 50μs, and is far lower than that of IP-over-Firewire. In [32] they achieved lower latency around 20μs and higher throughput around 280 Mbps, but their implementation is based on a customized real-time Linux operating system, and using a customized hardware layer beyond the standard IEEE 1394 interface card. This certainly will increase the cost for building such a system in addition; it also restricts the type of applications which can run on the system.

The performance of our protocol is much better than that of the Fast Ethernet, and is comparable with the Gigabit Ethernet, by comparing to Figure 3-1. When our communication protocol achieves better reliable delivery throughput using the isochronous mode, there are cases that using asynchronous mode can achieve better throughput, and different cluster applications may have different requirements for the interconnections, such as the throughput factor sometimes is not as important as the latency factor. In general, both transfer modes are needed to optimize the performance. In the next Chapter, we will introduce a control mechanism integrated into our communication protocol which employs the two transfer modes to fulfill the various requirements for cluster applications.

4 A control mechanism for dynamic performance tuning for cluster application

4.1 Introduction

In Chapter 3, the communication protocol that we devised for optimizing the communication performance of the IEEE 1394 was described in details. We have identified that the isochronous mode is more effective in terms of throughput; however, there are cases, where the asynchronous mode can also sustain a high throughput. In this Chapter, we try to identify the different data transmission characteristics of the IEEE 1394 asynchronous mode and isochronous mode, and propose a control mechanism for the communication protocol over IEEE 1394 to dynamically switch between the two transmission modes of the device, by doing so, we can maximize the overall performance. Fuzzy logic is employed in the mode switching process for identifying the proper switching point, so that the performance of IEEE 1394 device under different application specific data transmission features can be optimized.

Performance analysis and optimization of network system of computer clusters, or grids are well studied problems, and many works have addressed this issue. Queuing theory and stochastic process were the prevail methods in computer network modeling and analysis [87], [99], or the computer cluster modeling and analysis [101], [112], [77]. But the queuing theory is proved to be not suitable for the modeling of computer networks in some cases [57].

The fuzzy logic and fuzzy control have been used in the study and control of the computer network [7], [80], [5], [20], [31], and fuzzy logic analysis of the performance was seen recently [13], [51], [52]. In [98], [18], fuzzy logic is used in performance monitoring of parallel and distributed programs, and in [93], [94], the fuzzy logic is used to exploit in data analysis techniques of computer clusters performance classification etc.. While the fuzzy logic performance analysis in these works is often employed in analyzing performance related data after applications finish running on the cluster, we employ the fuzzy logic in real-time performance analyzing and performance tuning. The fuzzy logic control mechanism is proved to be an ideal approach for these purposes.

4.2 Characteristics of throughput and latency for the IEEE 1394

In section 3.4.2, we identified that the asynchronous mode has different transmission features when comparing to the isochronous mode. The asynchronous mode transmits data in low latency, but it can only sustain a maximum throughput of 125 Mbps. The isochronous mode can sustain much higher throughput around 250 Mbps, but in higher latency. This is due to the different arbitration methods applied when the two modes are transmitting on the bus.

Since the IEEE 1394 is a bus system, before a node can transmit data over the bus, it has to arbitrate for the resource, so that it can gain control of the bus.

As shown in Figure 4-1, for an asynchronous subactions, the source node sends a data prefix signal (including a speed code, if needed) to the bus, addresses of the source and destination nodes, a transaction code, a transaction label, a retry code, data, one or two cyclic redundancy checks (CRCs) and a packet termination (either another data prefix or a data end signal). Isochronous subactions include a short channel identifier rather than source or destination addresses and do not have the transaction label or retry code.

Acknowledgment must be returned for an asynchronous subaction which is destined to a unique destination. Acknowledgments are also preceded by a data prefix and terminated by another data prefix or a data end. All asynchronous subactions are normally separated by idle bus cycle period called “subaction gaps”. A gap opens up on the bus between the packet transmission and acknowledgment reception. This “ack gap” is of varying lengths depending on the location of the receiver in the bus relative to the senders of the link request and acknowledgment (ack). However, the maximum length of the ack gap is sufficiently shorter than a subaction gap to ensure that other nodes on the bus will not begin arbitration before the acknowledgment has been received. Table 4-1 gives a detailed Arbitration gap times of the IEEE 1394.

Similarly, isochronous subactions are separated by periods of idle bus called “isoch gaps,” as shown in Figure 4-2. The subaction gap is much longer than the isoch gap and this makes asynchronous mode consumes more bus time than the isochronous mode. The Serial Bus architecture limits the maximum number of

data bytes in a transaction to the largest power-of-two such that the whole asynchronous link-layer packet transmission takes less than 62μs. This means that the data payload of asynchronous packets is limited to 512 bytes at the cable base rate of 98.304 Mbps, longer packets can be sent at higher data rates and shorter packets at lower data rates, for the 400 Mbps rate, the asynchronous packet is limited to be 2048 bytes.

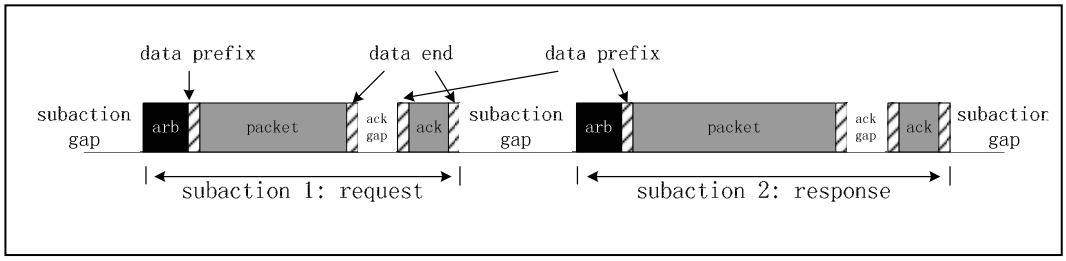


Figure 4-1: Asynchronous subactions

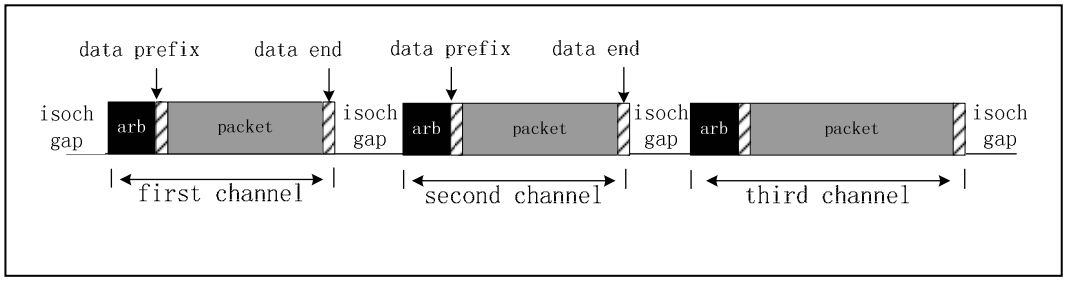


Figure 4-2: Isochronous subactions

As shown in Table 4-1, the arbitration gap time of the asynchronous mode, i.e. the subaction gap, is around 10μs, which is far greater than that of the isochronous gap, which is only 0.05μs. Most importantly, the maximum packet size of the asynchronous mode is restricted to half of the packet size of the isochronous mode. For the 400Mbit/s 1394 device, the maximum packet size for asynchronous mode is 2048 bytes, while for isochronous mode, it is 4096 bytes. The ex-

tra arbitration time and the restriction of the maximum packet size make the asynchronous mode transmission throughput incomparable with that of the isochronous mode.

Table 4-1: The IEEE 1394 Arbitration gap times

Gap type	Detection time		Comment
	Minimum	Maximum	
Acknowledge gap Isochronous gap	0.04μs	0.05μs	~4/BASE_RATE
Subaction gap	(27 + gap_count * 16) / BASE_RATE	(29 + gap_count * 16) / BASE_RATE	After two resets, gap_count is 63, so subaction gaps are ~10μs
Arb reset gap	(51 + gap_count * 32) / BASE_RATE	(53 + gap_count * 32) / BASE_RATE	After two resets, ~20μs

4.2.1 Multiple asynchronous mode transmission performance

Although the Serial Bus architecture limits the maximum number of data bytes in a transaction to the largest power-of-two such that the whole asynchronous link-layer packet transmission takes less than 62μs, it does allow multiple transactions to be initiated in one bus cycle time, so that we can improve the overall performance of the asynchronous mode by initiating multiple transactions at the same instance. Examination of the multiple asynchronous mode transmission performance with fixed packet size was described in Section 3.4.4. In this section, we carry out experiments of multiple asynchronous mode transmission with various packet sizes to examine the performance features in detail.

Four experiments were conducted on the same test bed, to study the overall

throughput from two to five concurrent asynchronous mode transmissions at the same time. Experiments show that many transaction failures occur if more than five concurrent asynchronous mode transmissions were initiated between two nodes, which made the IEEE 1394 bus unusable, so those results are not included. Only asynchronous write transaction was used in all the tests, since as discussed in section 3.4.2, the performance features of the read transaction and write transaction are similar. We will compare results in this section with those of the single write transaction's which we have already shown in section 3.4.2.

The experiments were carried out by running two programs on two workstations, as shown in Figure 4-3. Each program initiated threads to handle the data sending and receiving. For the experiment of two concurrent asynchronous transmissions, the sending program initiated two threads (send_th), and each of them kept sending data to the other program through the IEEE 1394 bus, the other program which ran on the other workstation also initiated two threads (rec_th), received data which was sent from the two corresponding threads (send_th), and verified the correctness of the received data. The size of the packet being sent varied from 8 bytes to the most significant value defined (2048 bytes for 400 Mbps IEEE 1394 device). For each packet size, the write transaction was performed around 10^5 times back-to-back, and data transmissions in all the threads were started and stopped at the same time so that the two concurrent data transmission were active through out the same testing period. The throughput and latency results were calculated based on these conditions.

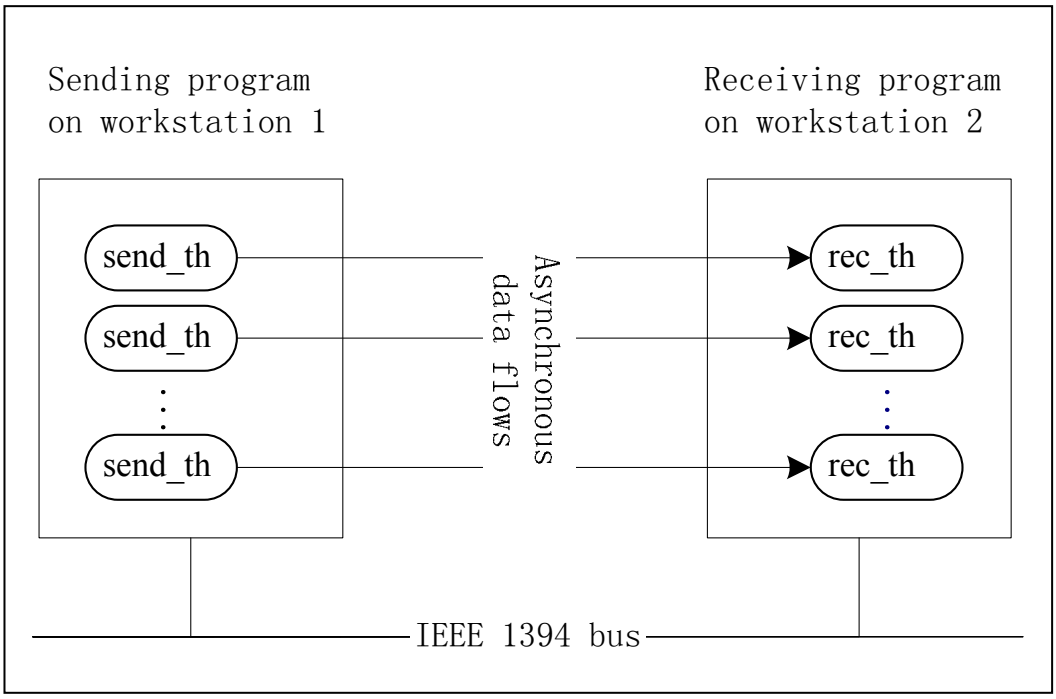


Figure 4-3: Test bed of multiple asynchronous mode transmissions

For the experiment of three, four and five concurrent asynchronous transmissions, three to five threads pairs are initiated in the two programs separately, the throughput and latency results are shown in Figure 4-4 and Figure 4-5 respectively.

Figure 4-4 shows the throughput results of the experiments, the throughput results in the figure are the sum of all the concurrent transmissions executed in the experiment. Since every single asynchronous transmission of the concurrent transmissions almost shares the same performance features, their details are not shown in the figure. Generally speaking, throughputs in these experiments are improved by initiating multiple asynchronous transmissions on the IEEE 1394 bus and the maximum throughput occurs at the packet size of 2048 bytes and

with three concurrent transmissions. The overall throughput of two asynchronous transmissions is significantly improved when comparing with that of a single asynchronous transmission. Therefore, adding more concurrent transmissions can improve the throughput further, but with four concurrent transmissions, the performance gain is reduced and is below that of the three concurrent transmissions case when the packet size is around 1024 bytes. For the five concurrent transmissions case, one can observe that the performance is only improved with very small packet size, less than 500 bytes, it becomes lower than that of the single asynchronous transmission's throughput when the packet size is greater than 512 bytes. With even larger packet sizes, the excessive concurrent data transmission will overload the bus and cause the bus to perform a reset action, thus rendering the bus unusable for further data transmission.

By further examining the throughput results of Figure 4-4, we find that the throughput increases rapidly when the packet size is small, say around 512 bytes. But the increase in throughput becomes slower when the packet size is larger than 1000 bytes. The throughput almost situates when more concurrent transmissions are active, and even drops with five concurrent transmissions.

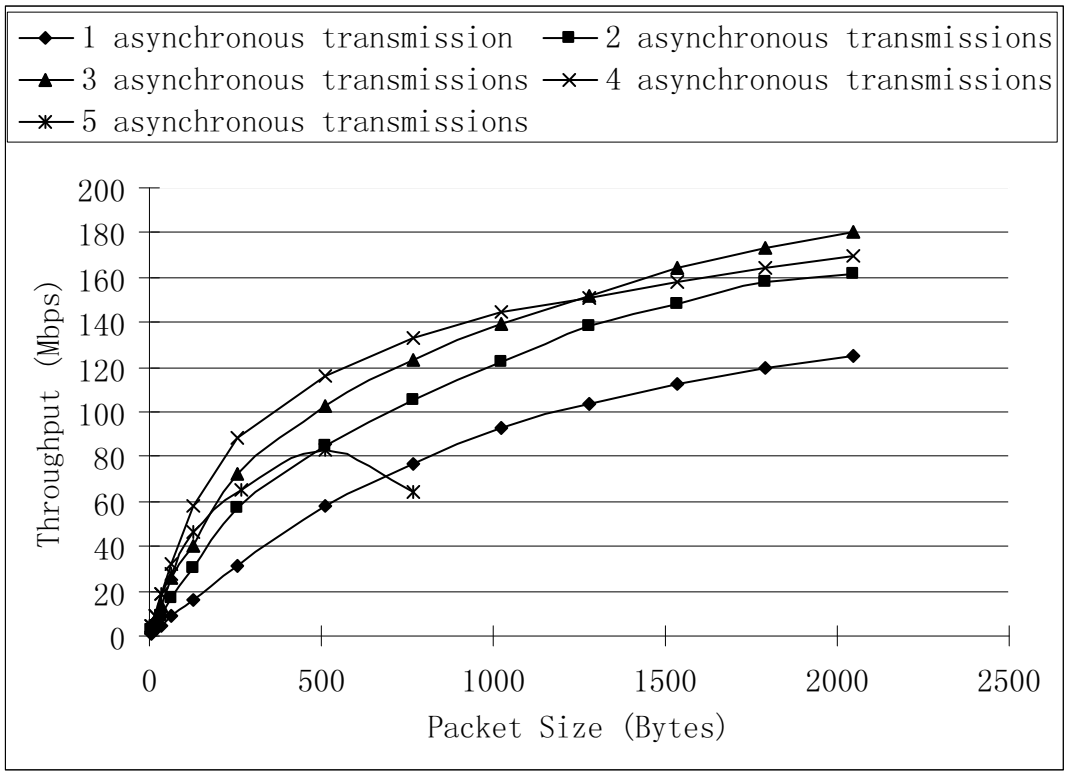


Figure 4-4: Throughput of multiple asynchronous mode transmission at the same time, comparing with the throughput of single asynchronous mode transmission.

Figure 4-5 depicts the latencies corresponding to various data packet sizes of these experiments. We can observe that the latencies are always around 50 μs when the packet size is very small (smaller than 128 bytes), even with five concurrent asynchronous transmission. When the packet size is greater than 128 bytes, the latencies increase almost linearly. The maximum latency is close to 400 μs with five concurrent asynchronous transmissions when the packet size is 2048 bytes, except in the case of five concurrent asynchronous transmissions, where latencies increase very rapidly with an increase in the packet size with a small packet size (768 bytes), the latency is already very close to 400μs.

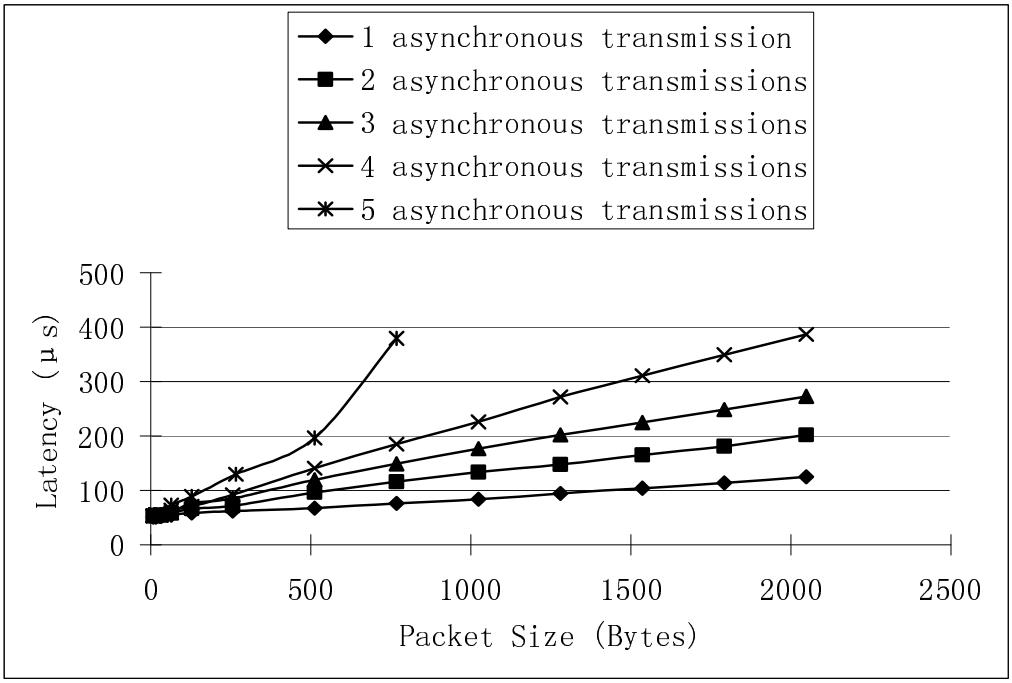


Figure 4-5: Latency of multiple asynchronous mode transmission at the same time, comparing with the latency of single asynchronous mode transmission

4.2.2 Comparison of the asynchronous and isochronous mode performance

As discussed in Section 3.4.4, the performance of the isochronous mode transmission with acknowledgement included in our communication library resembles that of the bare isochronous mode transmission. The latency is around 180 μs and does not affect much by the packet size. The throughput feature is almost the same as that of the bare isochronous mode transmission, because a sliding window system is used in the data transmission, so that data transmission can be

overlapped, thus the extra latency which is introduced by the acknowledgement packet causes very little overhead, and the throughput is almost the same as that of the bare isochronous mode transmission. The comparison of the performance features of the isochronous and asynchronous modes transmissions is shown is Figure 4-6 and Figure 4-7.

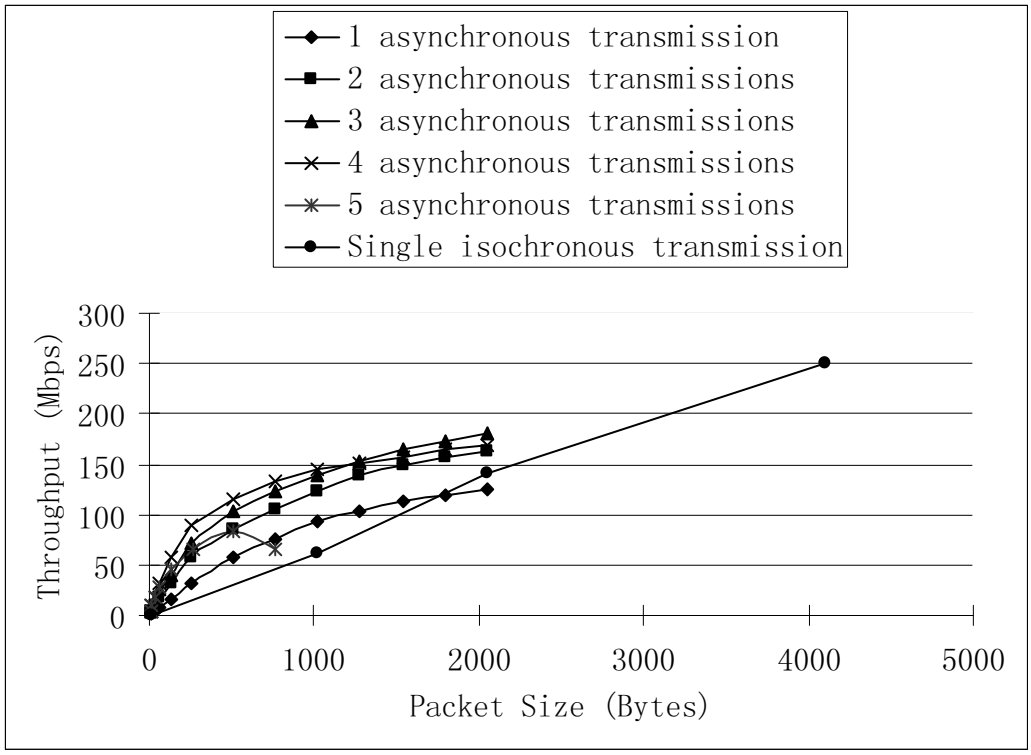


Figure 4-6: The throughput comparison of the asynchronous and isochronous mode transmissions

Referring to Figure 4-6, we can see that the throughput of the isochronous mode increases linearly with the packet size. The maximum packet size of the isochronous mode is 4096 bytes, and the maximum throughput is obtained at this point,

which is close to 250 Mbps. Considering the maximum throughput, the asynchronous mode transmission is not comparable with the isochronous mode transmission. Even though employing multiple concurrent asynchronous transmissions can improve the overall throughput, the maximum throughput that can obtain is only 180 Mbps, which is about 30% lower than the maximum throughput of the isochronous mode transmission. However, if we consider the throughput corresponding to packet size that is smaller than 2048 (see Figure 4-6), it is clear that in most cases, the throughput of the single asynchronous mode transmission is better than that of the isochronous mode transmission.

Figure 4-7 represents the latency induced in the different transmission modes and it is obvious that the latency of the isochronous mode transmission is in a steady-state for all packet sizes, which is around 180μs. Latencies of the asynchronous mode transmission vary with respect to the packet size. The single asynchronous mode transmission latency is always below the isochronous mode latency, while for other multiple concurrent asynchronous modes, the latencies increase steadily and almost proportional to the packet size. For example, in the case of a 3 asynchronous transmissions, when the packet size is less than 1000 bytes then the asynchronous mode is superior, however, when the packet size gets larger (over 1000 bytes) then the latency becomes higher than the isochronous case.

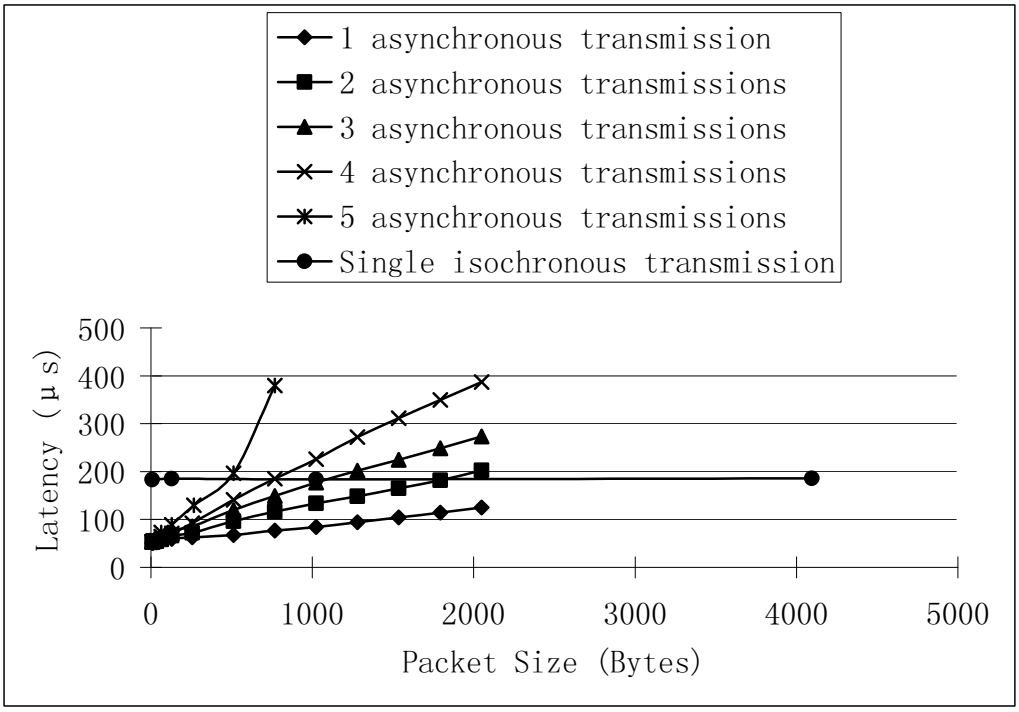


Figure 4-7: The latency comparison of the asynchronous and isochronous mode transmissions

From the above discussion, we found that both the asynchronous mode transmission and isochronous mode transmission have advantages and disadvantages. The asynchronous mode transmission has a lower latency, but its throughput is restricted, although the throughput can be improved by using multiple concurrent transmissions, it still can not compete with the throughput of the isochronous mode transmission when the packet size is large. While the isochronous mode transmission is better in throughput, its latency is higher than that of the asynchronous mode transmission in most cases. It is obvious that if we can use both transmission modes of IEEE 1394 device, and use the suitable transmission mode under different network conditions, or traffic, then the utilization of the

bus can be greatly improved. In this Chapter, we proposed to include a control mechanism in our communication protocol to dynamically monitor the data transmission situation of the IEEE 1394 bus. The control mechanism will switch the bus to use appropriate transmission modes in order to carry the data to better serve the application that runs on the network.

4.3 The fuzzy transmission mode switching controller

In this section, we will first introduce the design requirements, the trade off between lower latency and higher throughput. We will discuss details of fuzzy control for mode switching.

4.3.1 Analysis and performance metrics of computer cluster system

4.3.1.1 Speedup factor

"How much faster can the cluster solve the problem under consideration?" This question is perhaps the first point of interest when developing software solutions on a cluster system. In doing this comparison, one would use the best solution on the single computer, that is, the best sequential algorithm on the single computer system to compare against the parallel algorithm under investigation on the cluster. If we identify the number of computers or processors as p, the speedup factor [101], [85], $S(p)$, is a measure of relative performance, which is

defined as:

$$S(p) = \frac{t_s}{t_p} \tag{4.1}$$

Where t_s is the execution time of the best sequential algorithm running on a single processor and t_p is the execution time for solving the same problem on a multiprocessor. $S(p)$ gives the increase in speed in using the multiprocessor. Note that the underlying algorithm for the parallel implementation might not be the same as the algorithm on the single-processor system, and in most cases, it is different.

In a theoretical analysis, the speedup factor can also be presented in terms of computational steps:

$$S(p) = \frac{\text{Number of computational steps using one processor}}{\text{Number of parallel computational steps with } p \text{ processors}} \tag{4.2}$$

It is easy to understand that the maximum speedup possible is usually p with p processors (linear speedup). The speedup of p would be achieved when the computation can be divided into equal-duration processes, with one process mapped onto one processor and no additional overhead in the parallel solution.

$$S(p) \leq \frac{t_s}{t_p / p} = p \tag{4.3}$$

Super-linear speedup, where $S(p) > p$, may be seen on occasion, but usually this is due to using a suboptimal algorithm. One common reason for super-linear speedup is extra memory in the multiprocessor system. For example, suppose the main memory associated with each processor in the multiprocessor system is the same as that associated with the processor in a single-processor system.

Then, the total main memory in the multiprocessor system is larger than that in the single-processor system, and can hold more of the problem data at any instant, which leads to less disk memory traffic.

4.3.1.2 Message-passing computations

Several factors will appear as overhead in the parallel computing and limit the speedup, notably

a) Periods when not all the processors can be performing useful work and are simply idle.
b) Extra computations in the parallel version not appearing in the sequential version; for example, to recomputed constants locally.
c) Communication time between processes.

Among these factors, a) and c) are highly related to message-passing of a cluster parallel computing system, and they can cause significant overhead in the computation. In message-passing computation, messages are sent between processes to pass data and for synchronization purposes. Thus,

$$t_p = t_{comm} + t_{comp} \tag{4.4}$$

Where t_{comm} is the communication time, and t_{comp} is the computation time. As we divided the problem into parallel parts, the computation time of the parallel parts generally decreases because the parts become smaller, and the communication time between the parts generally increases (as there are more parts communicating). At some point, the communication time will dominate the overall exe-

cution time and the parallel execution time will actually increase. It is essential to reduce the communication overhead because of the significant time taken by inter-processor communication.

From (4.1) and (4.4), we have:

$$S(p) = \frac{t_s}{t_{comm} + t_{comp}} \tag{4.5}$$

In this thesis, our main concern is the communication time t_{comm} of the IEEE 1394 interconnection network.

4.3.1.3 Latency and throughput

The communication time will depend on the number of messages being transmitted, the size of each message, the underlying interconnection structure, and the mode of transfer. The communication time of each message will depend on many factors, including network structure and network contention. For a first approximation, we will use the following equation for the communication time of a message:

$$t_{comm} = t_{startup} + wt_{data} \tag{4.6}$$

Where $t_{startup}$ is the startup time, sometimes called the *message latency*. The startup time is essentially the time needed to send a message with no data. It includes the time to pack the message at the source and unpack the message at the destination. The term *latency* is also used to describe a complete communication delay, i.e. t_{comm} can also be called *latency*. The startup time is assumed to be constant. The term t_{data} is the transmission time to send one data word, also

assumed to be constant, and there are *w* data words. The transmission rate, or *throughput*, is usually measured in bits/second and would be b / t_{data} bits/second when there are *b* bits in the data word. The equation shows a linear relationship between communication time t_{comm} and data bits *w*. The latency of the asynchronous mode transmission of IEEE 1394 vs. packet size given in Figure 3-8 resembles the result from equation (4.6), but not a perfect linear relationship. It is because in a real system, many factors can affect the communication time, like the arbitration for using the IEEE 1394 bus.

In (4.6), t_{data}, which is the transmission time to send one data word, can also be noted as:

$$t_{data} = \frac{1}{r_{data}} \tag{4.7}$$

Where r_{data} is the data rate of the interconnection network, or throughput.

Thus (4.6) becomes:

$$t_{comm} = t_{startup} + \frac{w}{r_{data}} \tag{4.8}$$

Although in theory the latency t_{data}, and the throughput r_{data}, are inversely proportional, it may not be held in some cases. The latency and throughput are two key factors of a network system. A network can obtain low latency often imply that it can also sustain a high throughput, but this is not always true. As we have seen, the IEEE 1394 can achieve low latency when using the asynchronous mode transmission (from 50 μs to 125 μs with packet size varies from 8 bytes to 2048 bytes, as shown in Figure 3-8), but it can not achieve very high throughput

at the same time (125 Mbps maximum), because the packet size of the asynchronous mode is restricted to 2048 bytes (for 400 Mbps IEEE 1394 card). On the other hand, the IEEE 1394 can achieve high throughput when using the isochronous mode transmission (250 Mbps maximum as shown in Figure 3-10), but it cannot achieve low latency by using this transmission mode (125 μs fixed), because in order to simplify the hardware implementation, the IEEE 1394 device physical layer only sends fixed sized packet in isochronous mode [37], [59]. While considering the parallel applications which run on a cluster of computers, different applications may have different demands on the data transmission features of the network system. For example, a parallel sorting program [58] exchanges very few data after each computation step, but a parallel weather simulation system [25] often exchanges considerable amount of data after each computation step.

Interactive simulation systems for education, industrial training [106] and entertainment (multiplayer games [1], [2]) are one common type of application which runs on computer clusters. Such applications often are highly interactive real-life simulators, and support fine grain, close to instantaneous control of player actions and a high degree of interaction among players in a detailed, 3D virtual world. As we know, when a movie is playing at 24 frames per second, it is smooth to observers' eyes. Likely for 3D animations, to avoid the frame changing of a 3D scene to be observed by players, the application needs to render at least 24 to 30 frames per second. This frequency defines the time step for

each iteration, resulting in 30 to 40 ms time step for each iteration. So the demand of these applications, in terms of interconnection network features, is often low application to application latencies, but not necessary high throughput [1]. Because the data exchange of each time step between client and server is often a few Kbytes/s, so increasing number of clients often makes the system reach the bottleneck of the server's CPU processing cycles rather than the bottle neck of the network system. Another key factor of the network system which has to be taken into account is the variation of the latencies (or jitter), because big jitters will cause appreciable unsmooth refresh of the 3D scenes.

Unlike interactive simulation systems, other applications which run on the cluster system may require different latency and throughput features of the interconnection system. The requirement for the interconnection system may even change during the system run time. These requirements are all related to the factors of the communication time t_{comm} and the computation time t_{comp} as in (4.5), and the ratio

$$\text{Computation/communication ratio} = \frac{\text{Computation time}}{\text{Communication time}} = \frac{t_{comp}}{t_{comm}} \qquad (4.9)$$

can be used as another metric to evaluate the cluster performance.

Figure 4-8 illustrates hypothetical samples of three applications of different computation/communication ratios. In each application two cases are considered, and the optimized communication time in one case is half of the other.

For sample application 1, t_{comp} is much higher than t_{comm}, it is obvious that reducing the t_{comm} to half of its original value does not have much influence on

the overall processing time. Scientific computation applications often show such computational features where sophisticated mathematical computation consumes much of the processing time while parameter exchanges between computing nodes are usually small. This may also imply that the computing time can be reduced by using more computational nodes in the cluster. As shown in equation (4.6), if the data word w is small enough so that wt_{data} is near to or less than $t_{startup}$, then the latency will be an important proportion in t_{comm}, and thus greatly influence the overall performance.

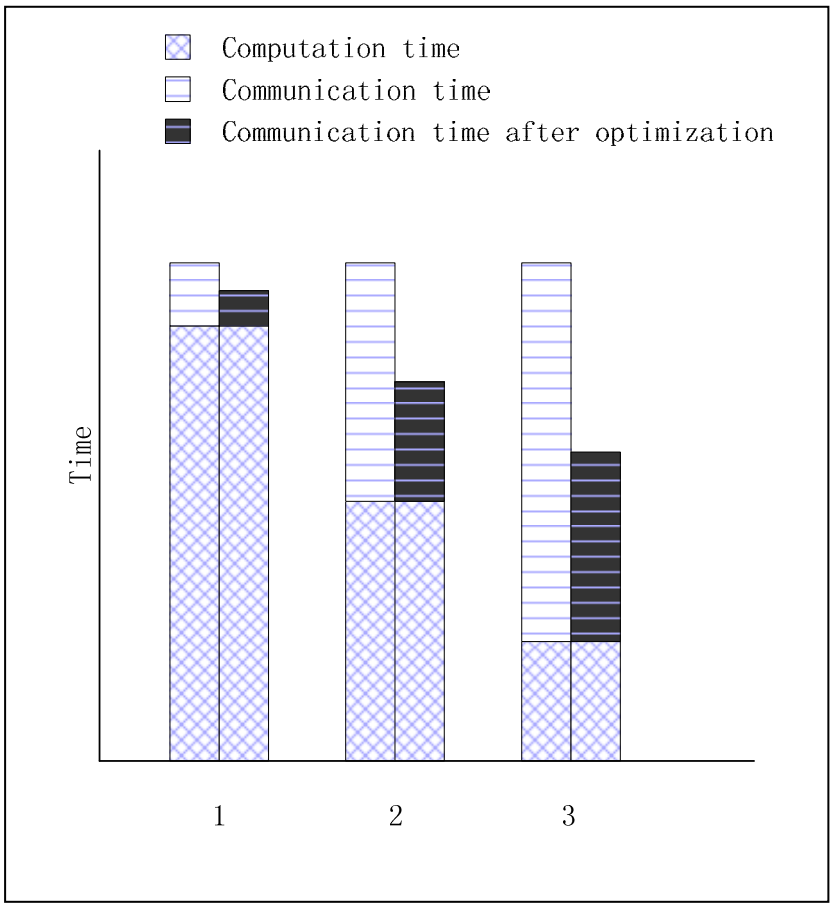


Figure 4-8: The computation/communication ratio of three different applications and comparison after the communication time is optimized to half of the original

For sample applications 2 and 3, communication time t_{comm} is close to or even higher than the computation time t_{comp}, so the interconnection network plays an important role in the total processing time. Under these circumstances, reducing the communication time t_{comm} will reduce the overall processing time significantly. In application 3, the communication time t_{comm} is much longer than the computation time t_{comp}, and dominates the total processing time. This may imply that the number of computing nodes in the cluster is unnecessarily high and introduce too much interconnection overload. At this time, much higher throughput of the interconnection network system is demanded to improve the overall cluster performance.

In the above, we have analyzed issues related to the two components, communication time and computing time, in affecting the effectiveness of cluster computing applications. Different applications which run on a computer cluster often incur different network transmission features, and have different demands in terms of system performance. To meet the requirements demanded by these applications, a control mechanism is needed to optimize the performance of the IEEE 1394 network. In the following sections, we will discuss two sample applications and their requirements for the proposed control mechanism.

4.3.2 Cluster design requirements of transportation simulation systems

The railway simulation system and the urban traffic simulation system are the

two target systems which will run on our cluster. The original objective of railway simulation system is to provide a simulation platform for the training of railway technical process operations [106], so the 2D and 3D graphical interface and human-machine interaction are of great importance in the system, but the system is often running in the same time scale as in the real-world, which means speedup is not a major design concern. In such a system, low data transmission latency, as well as low jitter in latency is crucial for the interconnection network of the cluster, so that frequent interaction of such a system can be guaranteed. Although high throughput is not required for the railway simulation system under such consideration, the system is under fast evolution, and it may be used to carry out prediction or verification of large scale transportation planning in the future. Speedup will become an issue when the system is used to perform those tasks, thus the requirements of the interconnection network need to be adjusted in the future.

The urban traffic simulation system is often for the purpose of transportation planning or forecasting [14], [92], [16]. Urban traffic problems often involve many human activities, and are hard to be modeled precisely, thus computer simulation is often employed. The most preferable approach for traffic simulation is a “microscopic” approach [67], in which individual traveler and transportation vehicle are modeled separately. Considering a big city with a population of over ten millions, simulating the traffic of such a city will be a challenging problem for most contemporary computers, so parallel computing for the

simulation of traffic system prevails. To understand the urban traffic situation of a single day, it may be necessary to simulate activities of that day hundreds of times. In order to predict the traffic situation for the next ten minutes of an urban area, it is reasonable to complete the prediction in about three minutes. This means that the speedup factor is a crucial factor of such kind of urban traffic simulation systems.

To facilitate the implementation of these applications on the IEEE 1394 cluster, our transmission protocol must meet the demands of various application features, which may vary during the runtime of an application, and the requirements of the control mechanism in the protocol are defined below:

a) Identify the transmission features of the application.

b) Monitor the performance of the application (in terms of *speedup*, *jitter* etc.)

c) Switching between the IEEE 1394 transmission modes to improve the network performance.

The requirement a) is to investigate the data transmissions of the application which is running on the cluster, and find out the type of application, that is, whether the application needs low latency and low jitter rather than high throughput, or it needs high speedup factor and high throughput.

The requirement b) is to monitor the transmission factors of the application, namely *latency*, *jitter*, *throughput*, and *speedup*.

The requirement c) is to tune the IEEE 1394 network performance to meet

the demand of the preconceived application.

The control mechanism is encapsulated in the protocol, so that it is transparent to applications. The application needs no knowledge about the underlying IEEE 1394 network system and the optimization is achieved by the control mechanism automatically. The control mechanism runs in a loop and keeps monitoring the network performance so that tuning of the network parameters can be performed. Figure 4-9 illustrates the logic of the control mechanism.

The control mechanism runs in parallel with the data transmission of the protocol, but in a lower priority so that the performance of the data transmission is not influenced by the controller.

Figure 4-9 depicts how the control mechanism can achieve the objective of maximizing the overall throughput of an application. If throughput is the most preferred transmission feature, this control mechanism tries to offer the maximum throughput under all condition, by initiating various numbers of concurrent asynchronous mode transmissions, or by switching between the asynchronous mode and isochronous mode. The control reaction is supposed to be only related to the packet size, the bold line, as shown in Figure 4-10, gives the optimized maximum throughput.

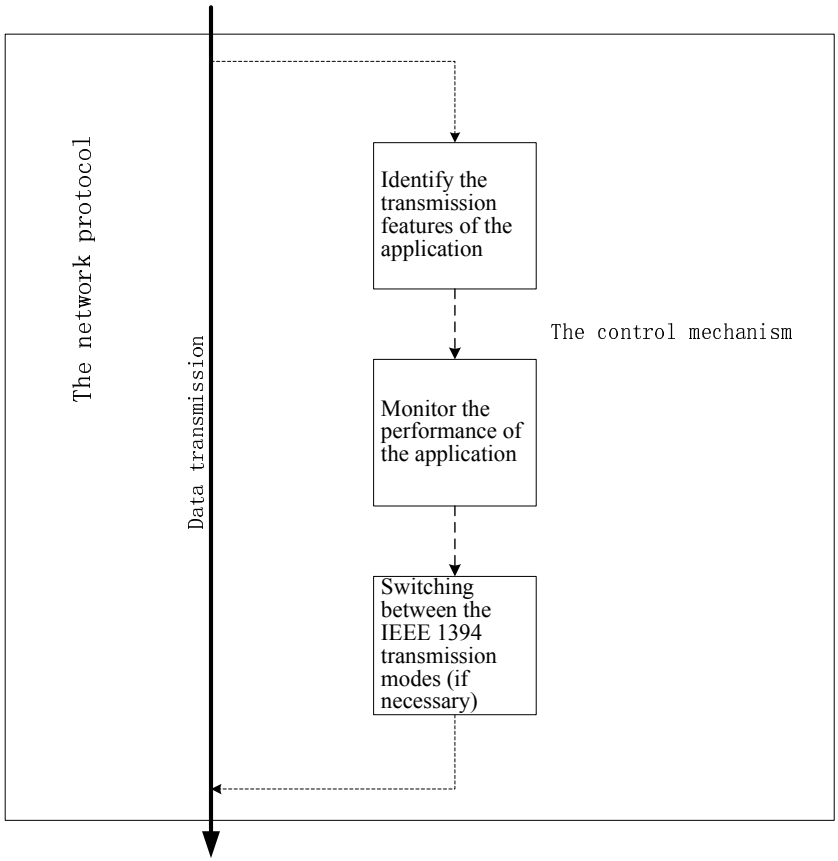


Figure 4-9: The control mechanism in the protocol

In practice, the maximum throughput that can be achieved at run time may vary from the result that we obtain from an experiment as shown in Figure 4-10, since network condition, the CPU usage and other factors may vary from time to time, thus the maximum throughput may not be equal to the value that we measured. This uncertainty of the transmission features makes it difficult for a crisp control mechanism to be implemented and to adapt to changes of the cluster system, so the fuzzy control is employed in our control mechanism. Another reason for using the fuzzy control mechanism is that to perform statistical analysis on the network transmission data and to make a control decision based on that will

be time consuming, and unnecessarily occupy the CPU cycles and memory, both of which are sacred resources in any parallel computing systems.

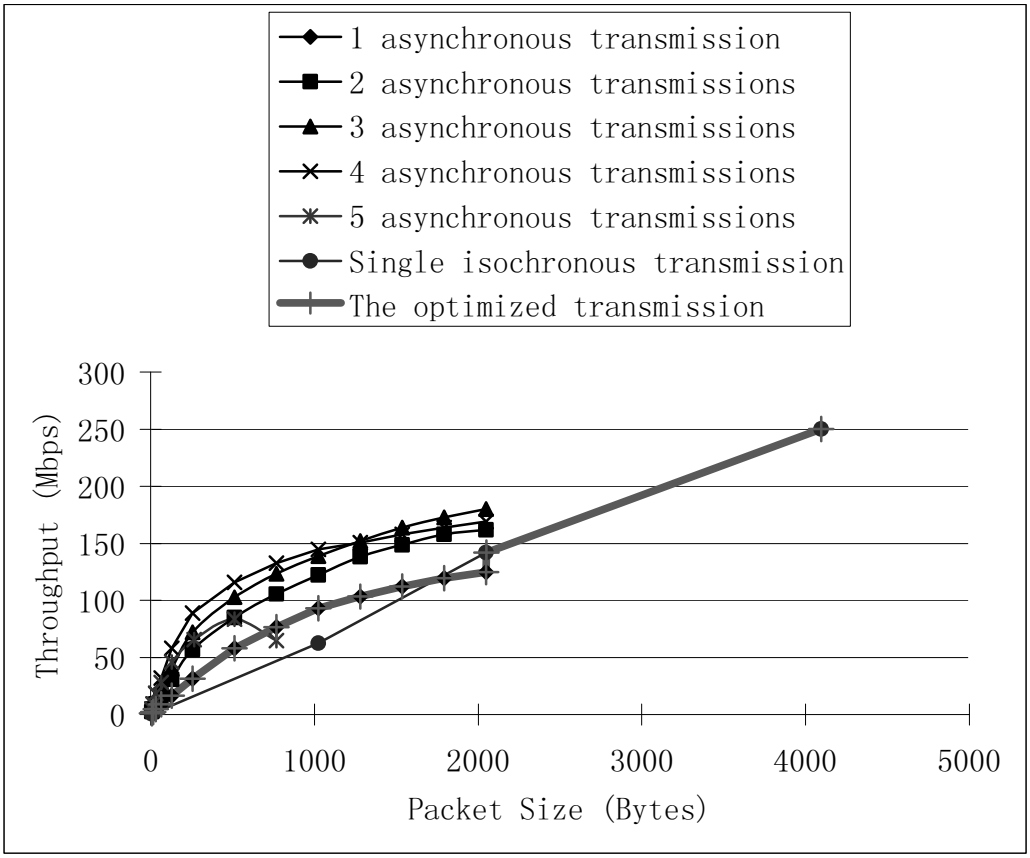


Figure 4-10: A sample objective of the control mechanism for throughput optimization

4.3.3 Principles of fuzzy control

Fuzzy control is a combination of control theory and fuzzy logic [109], [110]. When traditional mathematics was unable to solve complex practical problems, fuzzy logic is filling the gap and with tremendous success in areas including home appliances, aircraft control, production systems, medical applications and

much more.

Fuzzy logic assumes that there are propositions with an infinite number of truth values in infinitely varying degrees. Any logic then is just a subset of fuzzy logic. There are two extreme values, 1 (totally true) and 0 (totally false), and a continuum in between that justifies the term “fuzzy”.

Fuzzy logic, like probability theory, deals with uncertainty, but this uncertainty is masked in somatic and subjective ambiguity. Unlike probability theory, fuzzy logic deals with degrees of occurrence, whereas probability theory deals only with occurrence. It deals with degrees of truth that are provided in the context of fuzzy sets by what is called membership functions. To be able to perform logical, albeit fuzzy, reasoning, fuzzy operator such as OR, AND, IF, and THEN ought to be defined.

Fuzzy control systems are rule-based systems in which a set of rules, called fuzzy rules, defines a control mechanism to adjust the system. Figure 4-11 shows the block diagram of a fuzzy logic controller that comprises of four principal components: a fuzzification interface, a rule base, and decision making engine, and a defuzzification interface.

As illustrated in Figure 4-11, a fuzzy rule-based system receives a number of crisp inputs, which describe the system that is controlled through a set of parameters, and suggests an action as a response to these inputs. In order to accomplish this, the system first “fuzzifies” the crisp numerical values that it receives from the input, according to a set of selected rules that are derived from

experience and the desired response of the system. Each rule's consequence then contributes to the final results to the degree that its descendent is true. A defuzzification process translates the fuzzy value back to a crisp, numerical value (or set of values for systems with more than one output), to provide the final result. So in practice the controller uses the crisp value to perform the control task. The fuzzification is performed by membership functions which map crisp sets into fuzzy sets. Triangular or trapezoidal membership functions and Gaussian membership functions are frequently used. There are several defuzzification methods, e.g. *center of gravity, height method* etc. [29]. Height method is convenient and advantageous when using triangular or trapezoidal membership functions, center of gravity is recommended when using Gaussian membership functions.

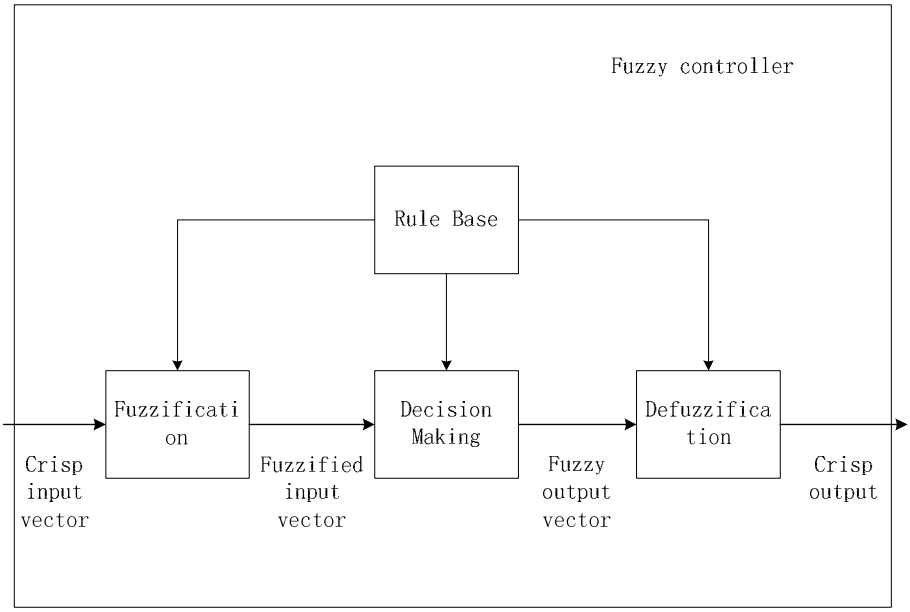


Figure 4-11: Block diagram of a fuzzy controller

4.3.4 Classification of transmission features of data flow

The first task of the control mechanism is the classification of data flow. As we mentioned before, different applications have different transmission requirements, therefore, we need to identify the transmission features of the application and make control decision accordingly. Moreover, this task is important because the requirements of transmission parameters for a given application may change during the application's run-time. As in a parallel transportation simulation application, the number of vehicles in the simulated area often varies on a large scale during the simulation, and the data transmitting between the cluster nodes will change accordingly.

Fuzzy logic is employed in classifying the transmission patterns of data flow in our control mechanism, in that it supports fast reasoning and decision making than other analytical, statistical or stochastic queuing theories.

As discussed in Section 4.3.2, there are three distinct types of data flows which need to be classified, namely:

a) Multimedia data flow of 3D animation.

b) Data flow that requires low latency but not high throughput.

c) Data flow that requires high throughput but not necessarily low latency.

The classification of these data flows all involves the following factors: packet size of an individual data packet, variation of packet size, interval of back-to-back data packet, variation (jitter) of the interval, throughput of the data

flow, variation of the throughput.

In the next section, we will present the classification criteria and membership functions for these three distinct data flows, which are derived by examining the performance features discussed in the Chapter 2, 3 and Section 4.2.

4.3.4.1 Multimedia data flow

Multimedia data flow of 3D animation often exhibits a rather "smooth" characteristic. The packet size of individual packet is usually not very large. In the case of the railway simulation system, it is in the order of tens of kilo-bytes. In addition, there are packet size variations, sometimes the variations can be very significant (imagine a train passing through the simulated station at very high speed), but this kind of burst variation is not frequent. The most significant characteristic of the multimedia data flow is that the variation of packet interval is often very small, and packet interval is often short (<30ms), although the actual packet interval may be longer due to the congestion of the network, we can measure the interval between which the application initiates the two data sending procedure of the protocol instead. Since the throughput is not an important factor for such an application, the throughput factor is not taken into account to these criteria. If we note Δt to be the measured time period, a fuzzy set of the input vector is defined as

$$\hat{X} = \left\{ p, |\Delta p|, \frac{|\Delta p|}{\Delta t}, i, |\Delta i| \right\}$$

where $X_1 = p$ is the packet size, $X_2 = |\Delta p|$ is the variation of packet size during

Δt in absolute value, $X_3 = \frac{|\Delta p|}{\Delta t}$ is the frequency of burst packet size variation in absolute value, $X_4 = i$ is the interval between packets, $X_5 = |\Delta i|$ is the variation of interval, in absolute value.

Let $L(X_i)$ be a set of linguistic values (words) characterizing any measurement over X_i . We define these as

$$L(X_1) = \{Small(S), Big(B), Very\ Big(VB)\}$$
$$L(X_2) = \{Small(S), Big(B)\}$$
$$L(X_3) = \{Nonfrequent(NF), Frequent(F)\}$$
$$L(X_4) = \{Short(S), Long(L)\}$$
$$L(X_5) = \{Small(S), Big(B)\}$$

From above criteria we have the following rules:

Rule 1: *if* X_1 *is S and* X_2 *is S and* X_3 *is NF and* X_4 *is Short and* X_5 *is S, then the result is YES*

Rule 2: *if* X_1 *is B and* X_2 *is S and* X_3 *is NF and* X_4 *is Short and* X_5 *is S, then the result is LIKELY YES*

Rule 3: *if* X_1 *is B and* X_2 *is B and* X_3 *is NF and* X_4 *is Short and* X_5 *is S, then the result is LIKELY NO*

These three rules are given as examples. The full rule set can be derived similarly, see Appendix C for details.

The membership functions of the input variables are chosen to be triangular. The functional forms of the membership functions of these values depend on the corresponding input and will be determined according to the criteria and rules derived above (Figure 4-12).

The physical domain for the packet size ($X_1 = p$) is [0, +∞). The differentiation between *Small*, *Big*, *Very Big* is 15, 25, 30 Kbytes respectively. These val-

ues are defined based on a preliminary test on the transmission data of the railway simulation system, where the packet sizes are mainly distributed between 1K – 25Kbytes.

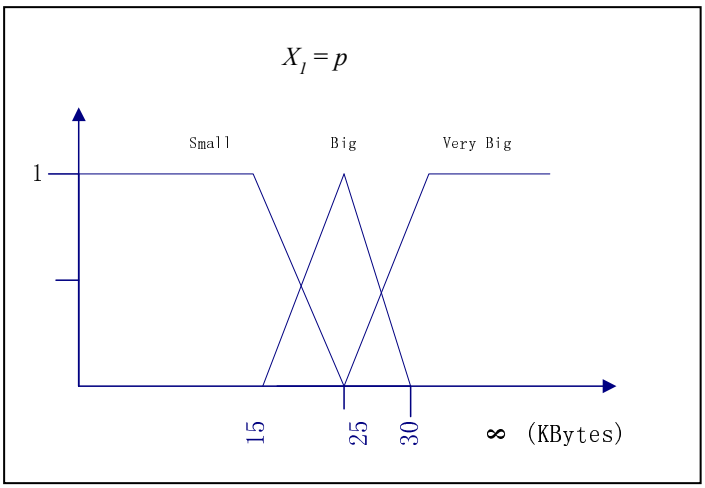


Figure 4-12: Membership function for packet size (X_1)

The membership function of variation of packet size $X_2 = |\Delta p|$ is depicted as in Figure 4-13, the differentiation of term small and big is at 1 Kbytes and 20 Kbytes, which are defined based on the same test, in which the variation of packet size is often around several kilo-bytes, while with a burst of around 20 kilo-bytes.

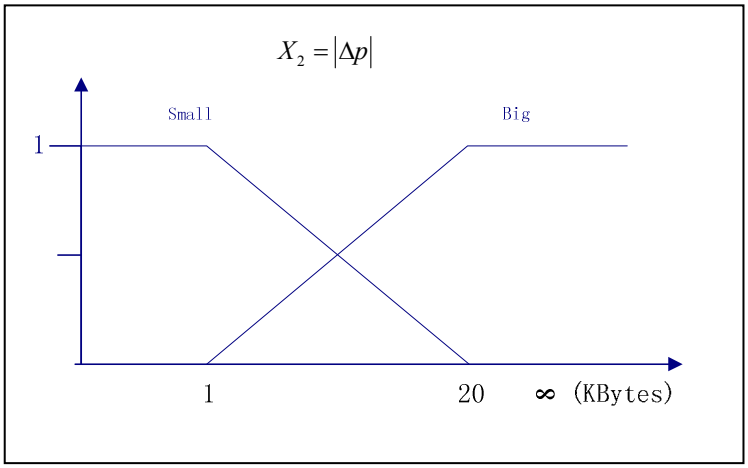


Figure 4-13: Membership function for variation of packet size (X_2)

The membership function of $X_3 = \frac{|\Delta p|}{\Delta t}$, the frequency of burst packet size variation, is shown in Figure 4-14, the differentiation of 0.005 and 0.1 Hz are defined based on the observation that the burst variation of packet often happens within 1 to 3 minutes intervals.

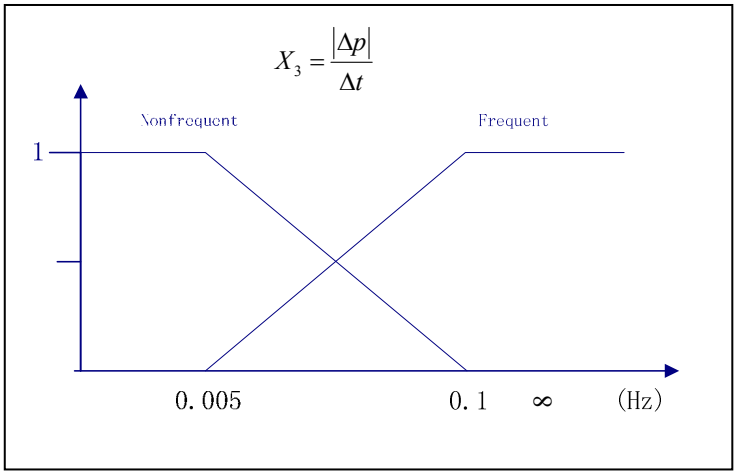


Figure 4-14: Membership function for frequency of burst packet size variation (X_3)

The membership function of interval between packets $X_4 = i$ is shown in Figure 4-15, Since the interval between packets is corresponding to the rendering rate, if we assume that the rending rate between [10, 100] fps is a reasonable value for the 3D application, the interval boundary then should be around [10, 100] ms, with 30ms interval in the middle, which is the preferred time interval.

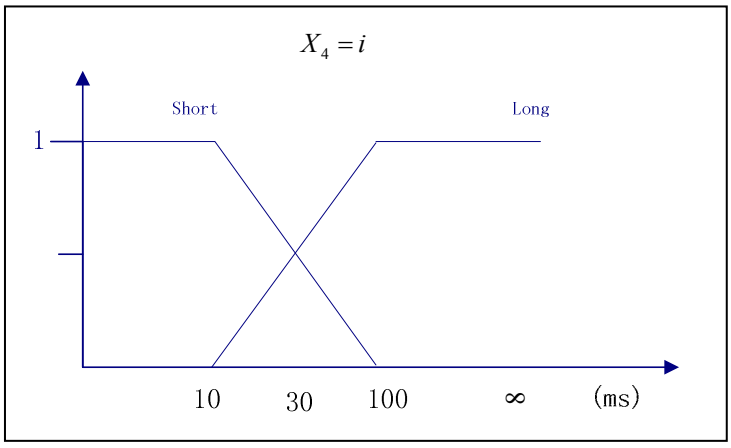


Figure 4-15: Membership function for interval between packets (X_4)

The membership function of variation of interval between packets $X_5 = |\Delta i|$ is depicted in Figure 4-16. The differentiation for big is 20ms, which will delay a rendering rate from 30fps to 20fps, which we regarded as a significant changes to the rendering rate.

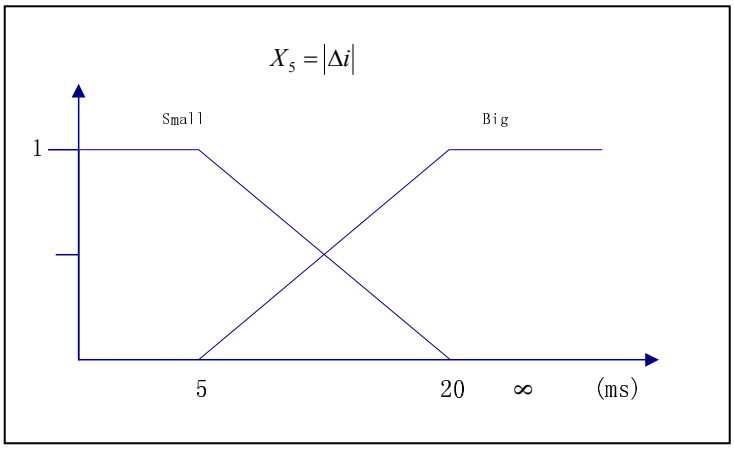


Figure 4-16: Membership function for variation of interval (X_5)

We have described the membership functions for the five parameters of multi-media data flow. In the next section, we will discuss the membership functions

for additional parameters that is required for identifying latency sensitive data flow.

4.3.4.2 Latency sensitive data flow

Some applications require prompt response to specific messages, such as messages which convey control information. Take the railway simulation system [106] as an example, if a button is pressed on the control terminal, the corresponding message is sent to a central simulator and a response is expected to be returned promptly. These data flows are latency sensitive. The classification of the latency sensitive data flow is based on the classification in the former section, if the data flow does not fall into the category of multimedia data flow, we need to further identify whether it is a latency sensitive data flow.

The control mechanism tries to identify whether there is a prompt response to a given data packet at the receiver side to decide whether the data flow requires low latency. We define $X_6 = r$, where r is the response interval between the receiving of a data packet and the sending of a response packet. Thus the rule for classifying a data flow as latency sensitive one is:

$$if\ \ r\ is\ short, then\ the\ data\ \ flow\ is\ latency\ sensitive\ .$$

The membership function of r is given in Figure 4-17.

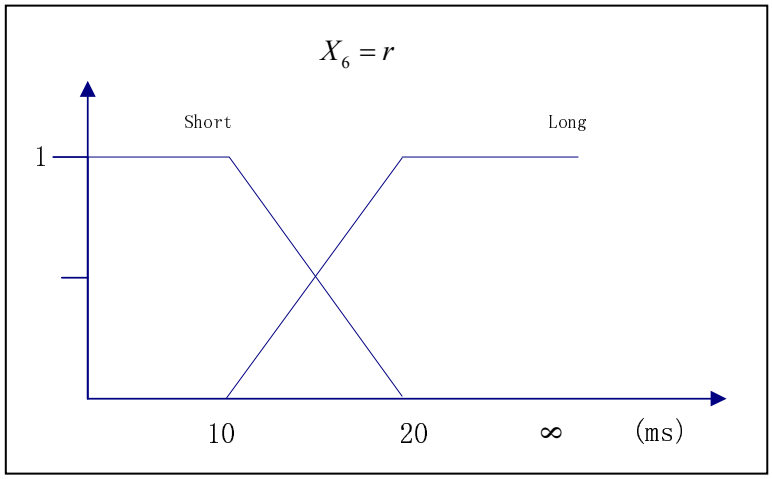


Figure 4-17: Membership function for r (X_6)

4.3.4.3 Throughput sensitive data flow

A throughput sensitive data flow is identified by monitoring the internal buffer queue length of our protocol. If a packet sending operation is initiated and the buffer is not empty, the data flow is considered to be throughput sensitive. The control mechanism is required to arrange more throughput capacity to the data-flow.

Since in a realistic cluster computing condition, the arriving of data packet is often in groups, once the queue is not empty, it is likely that the data will keep coming-in and the queue length will keep growing. Considering an M/M/1 queuing system, a solution given in [84] shows that the possibility of queue length P_n is related to the utilization ρ of the queuing system. The possibility of P(n>3) is less than 0.13 for ρ less than 60%, and it increases to 0.66 when ρ is 90%. This underlies that if the queue length is found greater than three, it is

probably that the utilization is between 60-90%. According to this, we set the queue length of three as the key point for differentiating the queue length as long. The queue length is defined as $X_7 = q$, then the rule for classifying a throughput sensitive data flow is

$$if\ \ q\ is\ long, then\ the\ data\ \ flow\ is\ throughput\ sensitive$$

The membership function of $X_7 = q$ is given by Figure 4-18.

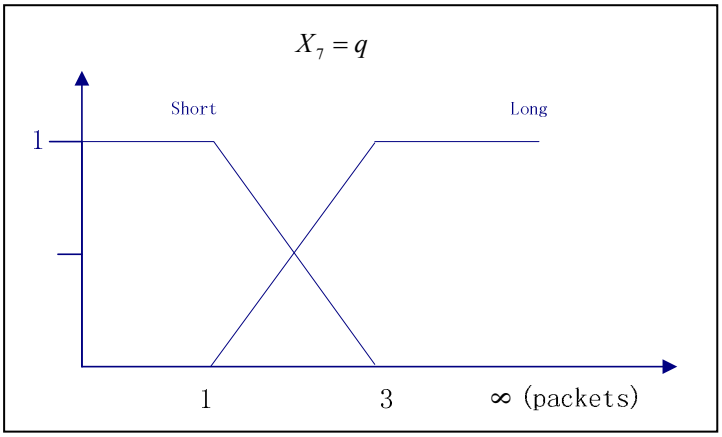


Figure 4-18: Membership function for q (X_7)

4.3.5 Performance prediction and decision making

Once the data flow is classified into the above categories, the fuzzy controller makes decisions of mode switching based on the data flow category and its current transmission performance. The fuzzy controller monitors the performance features such as packet size and packet size variation, and derives a prediction of the future performance from the current performance. Although it is impossible to make an exact network status forecast of cluster running various applications,

we still can make the prediction based on some criteria like minimum variance, which is referred to as minimum variance prediction [100]. The mode switching decision is made by the fuzzy controller based on the predicted future performance and the category of a specified data flow. Since there are considerable uncertainties in the network data transmission, the fuzzy controller is expected to yield better performance over the same cluster network than a traditional controller which is based on a mathematical model and makes mode switching on some threshold. In the next section, we describe the performance prediction and decisions making of data flows in three categories, results are presented to illustrate how the fuzzy controller can improve the performance of the cluster system.

4.3.5.1 Multimedia data flow

As we have described in section 3.4.2, the isochronous mode of IEEE 1394 bus supports guaranteed delivery time, and is designed to carry multimedia data. When a data flow is classified as a multimedia data flow, the controller should switch the IEEE 1394 bus to use isochronous mode to carry this data flow, provide that there is available isochronous resources. Figure 4-19 illustrates the topology of the controller for multimedia data flow.

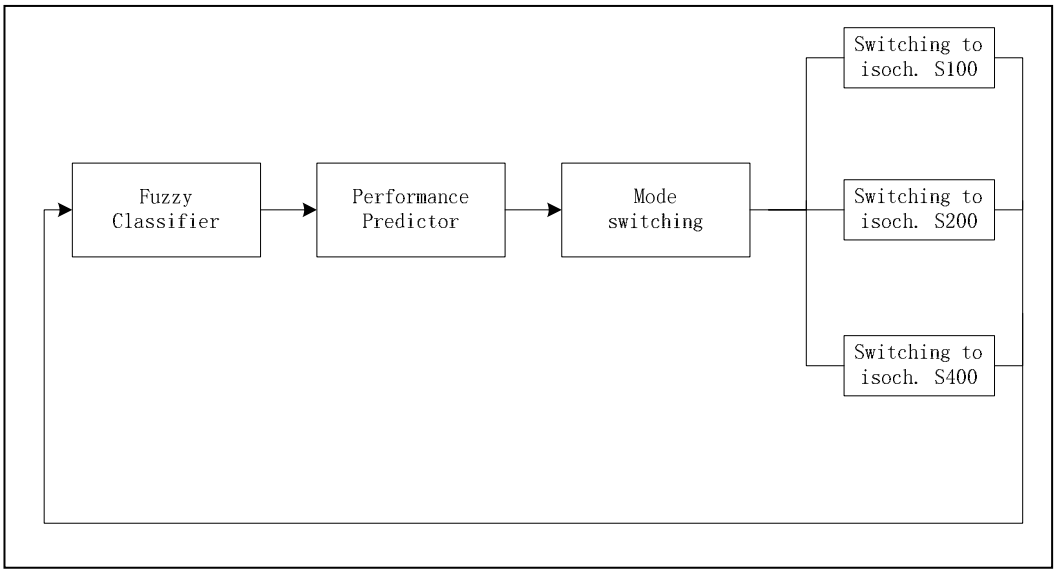


Figure 4-19: Controller topology for multimedia data flow

As shown in Figure 4-19, after the fuzzy classifier identifies a data flow as multimedia data flow, the performance predictor keeps monitoring the data flow and provides performance prediction. The mode switching is carried out with reference to the performance prediction. For the multimedia data flow, the IEEE 1394 bus can operate in three modes: isochronous mode S100 (with 62.5 Mbps throughput), isochronous mode S200 (with 125 Mbps throughput) and isochronous mode S400 (with 250 Mbps throughput).

To predict the future performance for a multimedia data flow, we define a fuzzy variable $X_8 = A$ as the arrival data rate of the multimedia data flow, and a fuzzy variable $X_9 = dA/dt$ as the rate of change of A. Figure 4-20 and Figure 4-21 show the membership functions of the two variables.

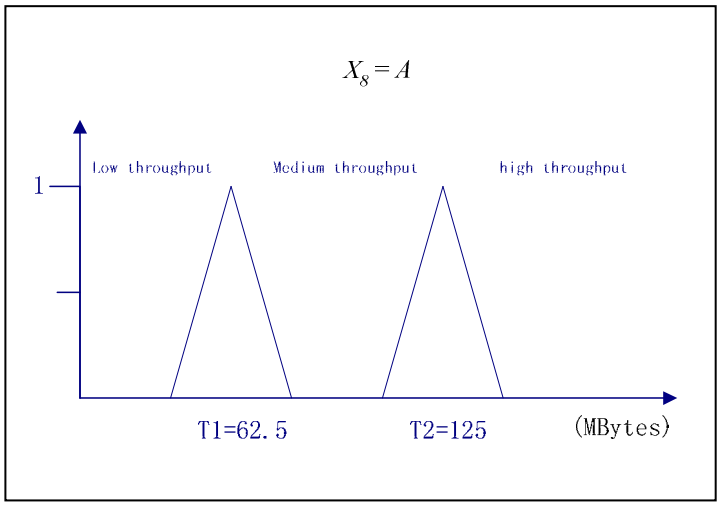


Figure 4-20: Membership function for arrival rate (X_8)

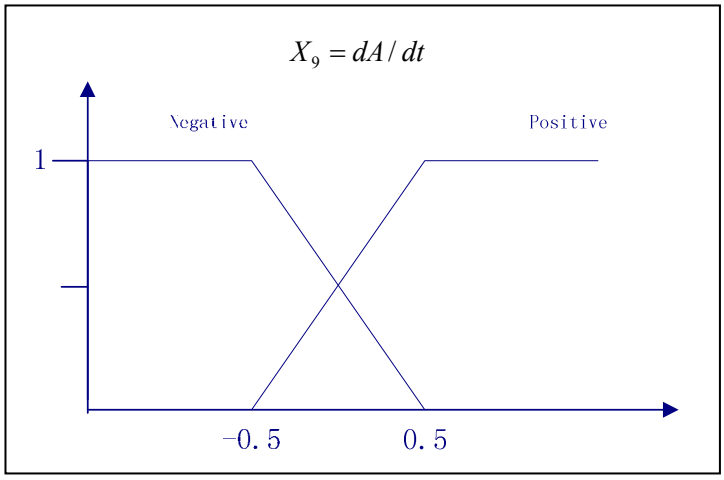


Figure 4-21: Membership function for rate of change of A (X_9)

As shown in Figure 4-20, two thresholds 62.5 Mbps and 125 Mbps categorize the variable $X_8 = A$ into three areas. In Figure 4-21, the rate of change of A is differentiated into *negative* and *positive* to represent the changing trend of the arrival rate. Four rules can be derived from these variables and their values, one example rule is:

*if A is T*1, *and dA*/*dt is positive*, *the predicted performance is medium*

The mode switching decision is thus to be switching the IEEE 1394 bus to isochronous mode S200.

We carry out an experiment to verify the improvement of the latency performance. As shown in Figure 4-22, we initiate one multimedia dataflow and a control data flow between two workstations at the same time. One frame of the multimedia data flow is about 150 Kbytes, and there are 50 frames of data per second (to offer better 3D animation performance, the frame rate is double of the ordinary 25 fps), so the workload of the multimedia data flow is around 7500 Kbyte/s. The throughput of the control data varies from 10 Kbytes/s, 100 Kbyte/s to 1 Mbyte/s. Each of the Data flow lasts for 2000 seconds.

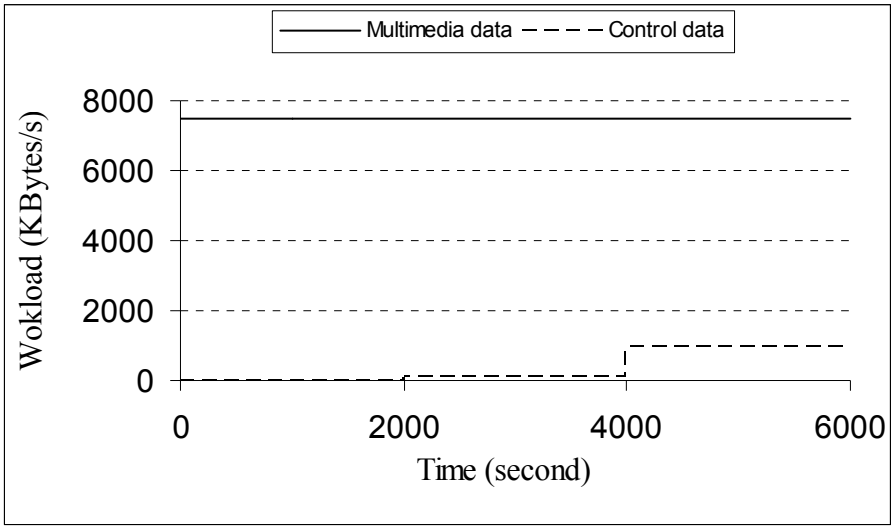


Figure 4-22: Workload of multimedia and control dataflow

Figure 4-23 and Figure 4-24 depict the comparison latency of the multimedia data flow without and with the fuzzy controller. In Figure 4-23, with no fuzzy controller, the multimedia data flow is transmitted in asynchronous mode. The

result shows that the multimedia data flow is influenced significantly by the control data. The variation in latency is quite clear even when the control data is only 10 Kbyte/s. Both latency and variation of latency increase when the control data throughput increases. When the throughput of control data is around 1 Mbyte/s, the latency of the multimedia data is sometimes over 35ms, and introduces perceivable jitter in the animation for multimedia data. In Figure 4-24, with the fuzzy controller, the multimedia data flow is identified and the transmission mode is switched from asynchronous mode to isochronous mode, since the throughput of the multimedia data flow is 7500 Kbyte/s, the fuzzy controller uses the S100 isochronous mode to carry the data flow, this makes the latency increase from 10ms to 18ms, but the variation of latency drops dramatically when comparing to Figure 4-23, and the variation is still very low when the throughput of control data flow increases to 1 Mbyte/s. This result shows that the multimedia data flow benefits a lot from the fuzzy controller.

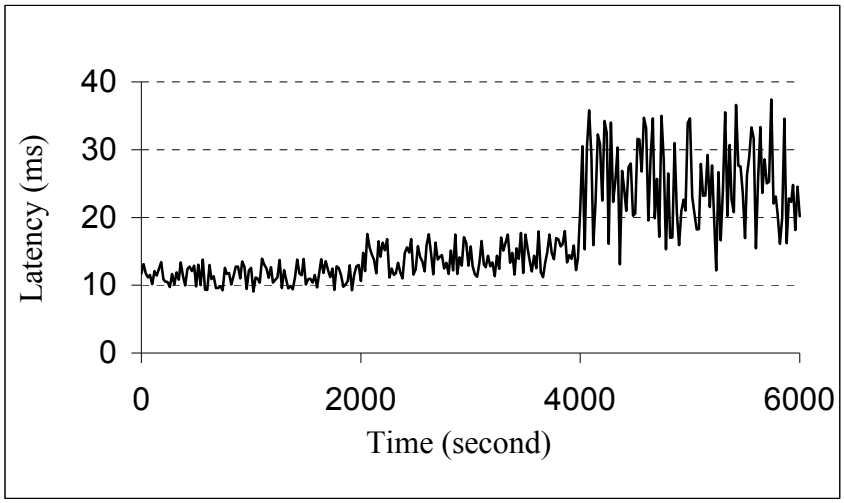


Figure 4-23: Latency of the multimedia data flow without fuzzy controller

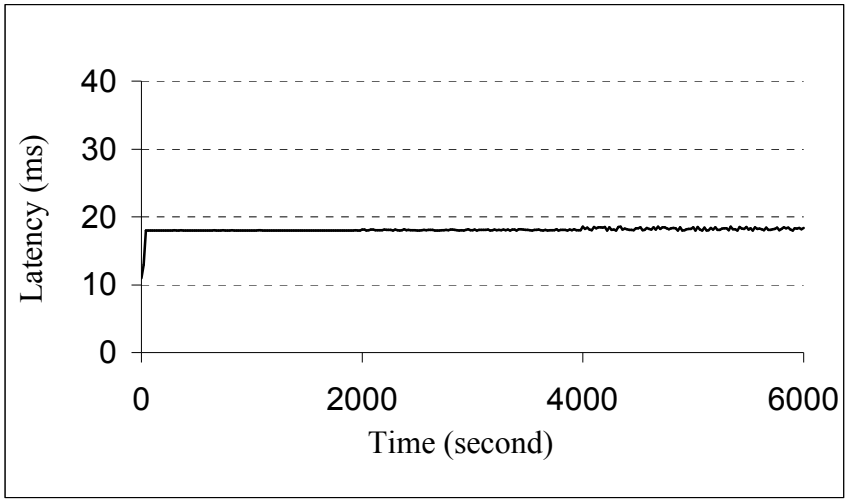


Figure 4-24: Latency of the multimedia data flow with fuzzy controller

4.3.5.2 Latency sensitive data flow

When the data flow is latency sensitive, the fuzzy controller chooses to use asynchronous mode to carry the data flow, and the single-way latency of this data flow is monitored, so that when the latency is increased beyond a given point, 180 μs as we have measured in Section 3.4.2, the bus will be switched to isochronous mode to carry the data flow, because the asynchronous mode cannot offer better latency than the isochronous mode.

Since the asynchronous mode performance is often influenced greatly by the isochronous mode, the mode switching for latency sensitive data flow depends greatly on the network conditions. Figure 4-25 shows the results for the empirical latency performance of latency sensitive data flow against packet size. The packet size of the data flow varies from 8 bytes to 4096 bytes. Three different conditions are studied, i.e. latency sensitive data flow with no isochronous data

flow, S100 isochronous data flow, and S200 isochronous data flow respectively. As shown in Figure 4-25, when there is no isochronous mode transmission, the latency sensitive data flow can benefit from asynchronous mode when the packet size is smaller than 2048 bytes, and only have to switch to isochronous mode when the packet size is greater. When there is a S100 isochronous mode data transfer accompanied with the latency sensitive data flow, the latency of the data flow carrying with asynchronous mode increases greatly, and when the packet size is around 1500 bytes, the fuzzy controller switches to isochronous mode to transfer the data flow, the latency is around 180 μs when packet size is smaller than 3072 bytes, but increases sharply after that, which is because the packet cannot be carried in one isochronous cycle, and has to use the next cycle. When there is a S200 isochronous mode data transfer accompanies with the latency sensitive data flow, the latency of the data flow increases greatly from small packet size, and the fuzzy controller switches to use isochronous mode when packet size is around 500 bytes. The latency is around 180μs when the packet size is smaller than 2048 bytes, and there is a sharp increase when the packet size is larger, it is because when there is a S200 isochronous mode, the other isochronous mode carrying the latency sensitive data flow can only use 2048 bytes in an isochronous cycle, and have to use the next cycle to carry the packet. Table 4-2 summarizes the comparison of latency performance of the data flow with various network conditions.

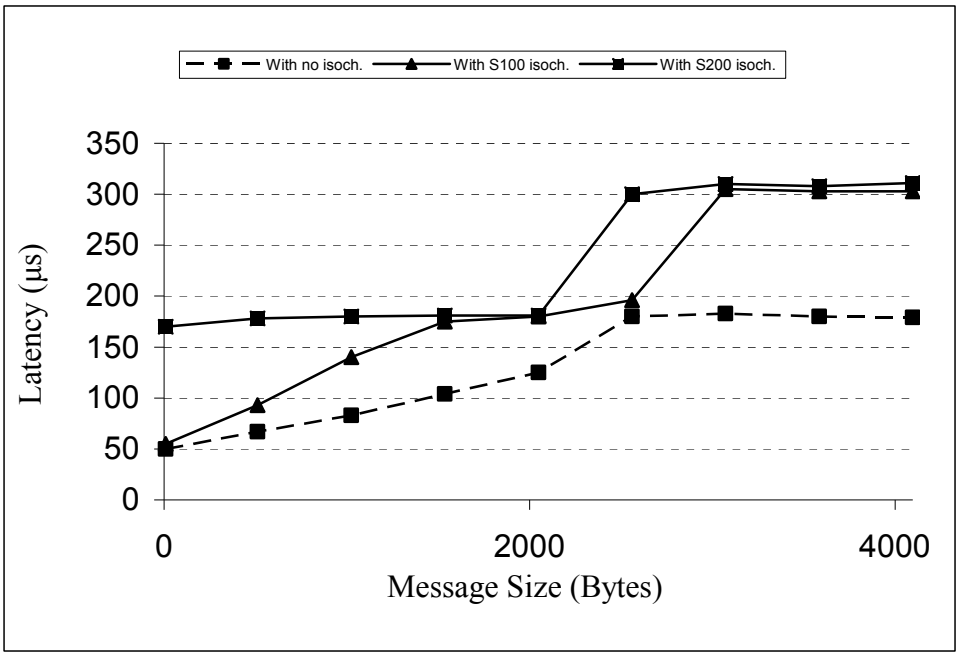


Figure 4-25: Latency of the latency-sensitive data flow with various network conditions

Table 4-2: Comparison of latency with various network conditions

Network condition	Mode used (packet size)	Latency Performance
Light loaded	Asynchronous (<2000) Isochronous (>2000)	Low Fairly low
Medium loaded	Asynchronous (<1500) Isochronous (>1500)	Low Acceptable
Heavy loaded	Asynchronous (<500) Isochronous (>500)	Fairly low Acceptable

By applying the fuzzy controller in the communication protocol, we can offer latency sensitive data flow a best-effort service, when the packet size is small, the protocol can achieve very low latency by utilizing the asynchronous mode, when the packet size increases, the protocol can still achieve acceptable latency by utilizing isochronous mode.

4.3.5.3 Throughput sensitive data flow

If the data flow is throughput sensitive, the fuzzy controller switches between modes to offer maximized throughput for the data flow. As we have shown in Figure 4-6, the throughput of asynchronous mode is higher when packet size is smaller than 2048 bytes, and the throughput of isochronous mode is presumably better when packet size is greater than 2048 bytes. But with the communication protocol which supports reliable delivery by acknowledgment, the latency of isochronous mode is increased to 180 μs. So using multithread asynchronous mode can sometimes offer better performance when packet size is greater than 2048 bytes. Figure 4-26 shows that when the packet size is between 2048 and 3000 bytes, the multithread asynchronous mode throughput is higher than the isochronous mode throughput. When the packet size is greater than 3000 bytes, the isochronous mode throughput is better. The fuzzy controller needs to monitor the packet size of a throughput sensitive data flow, and to choose the suitable mode according to the packet size.

Real time ratio (RTR) is an important factor for describing the performance of parallel simulation on clusters, it describes how much faster the simulation is than the reality. An RTR of 100 means that 100 minutes of reality events are simulated in 1 minute of computing time. Throughput of the interconnection in a computer cluster is the dominant factor for the real time ratio of parallel discrete

event simulation system. Urban traffic simulation is one of such systems.

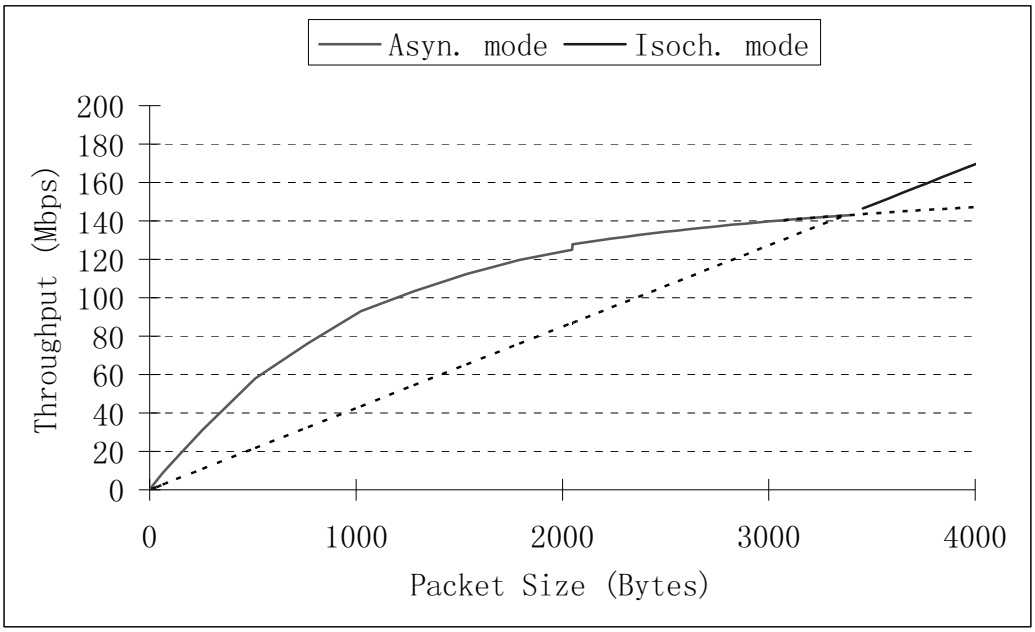


Figure 4-26: Maximized throughput by employing the fuzzy controller

Consider a very simple cluster which consists of two nodes and one IEEE 1394 link and Fast Ethernet link respectively, we can derive the real time ratio of a parallel simulation application regarding to interchanging packet size between nodes directly from the throughput of the interconnection device. To observe the real time ratio of the two nodes cluster using Gigabit Ethernet, we use the Gigabit Ethernet performance data from the study of [11] and [28] about data transmitting through MPI over TCP. The time for computation on each of the computation node (t_{comp}) is 1ms. Figure 4-27 shows the result.

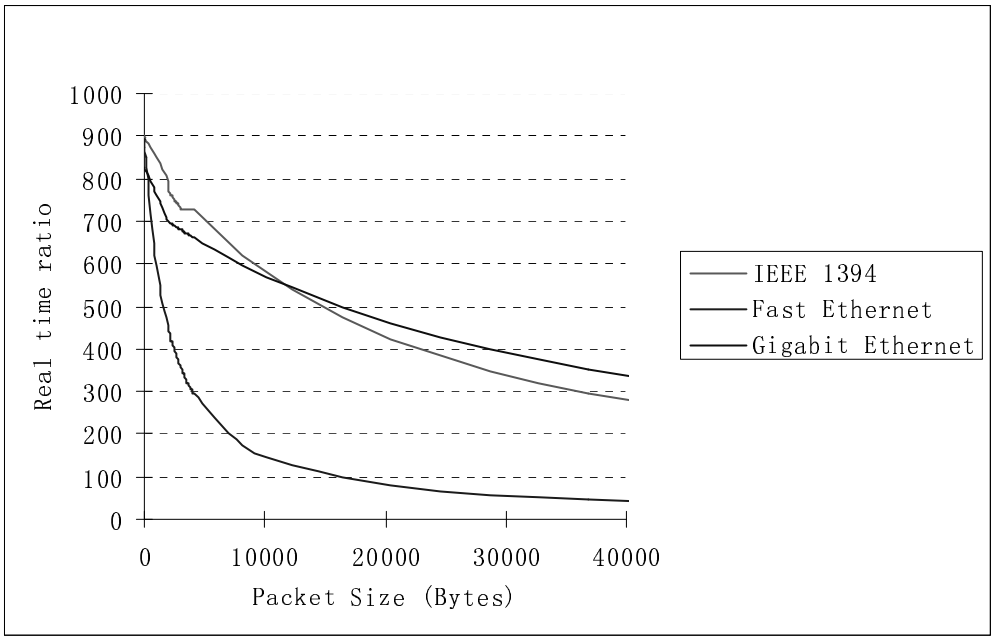


Figure 4-27: Real time ratio of two nodes cluster interconnected by IEEE 1394, Fast Ethernet, Gigabit Ethernet

As shown in Figure 4-27, the real time ratio of IEEE 1394 is better than that of the Gigabit Ethernet when the packet size is smaller than 10 Kbytes, and it drops lower than the real time ratio of the Gigabit Ethernet when the packet size is greater than 10 Kbytes. Both the real time ratio of IEEE 1394 and Gigabit Ethernet are better than that of the Fast Ethernet. This result shows that by employing our light weight communication protocol and the fuzzy controller, the IEEE 1394 can achieve high throughput when the packet size is relatively small, and can outperform the Gigabit Ethernet. But when packet size becomes larger, the performance of IEEE 1394 drops and cannot compete with the Gigabit Ethernet. Under all condition, the IEEE 1394 performance is much better than the Fast Ethernet. Software overhead is the main reason for the performance of

the Gigabit Ethernet is not as good as the nominal value – 1 Gbps. This overhead introduces extra latencies and causes the performance significantly slow when the packet size is small.

4.4 Concluding remarks

In this Chapter, we discussed the fuzzy logic data flow analysis and classification, and the fuzzy control mechanism to optimize the IEEE 1394 network for cluster computing. Although the fuzzy logic performance analysis of the cluster or grid system was address by several works, dynamic performance monitoring and tuning mechanism of the computer cluster system is unseen. We have implemented a control mechanism which intelligently controls the underlying IEEE 1394 network and tunes the network to provide optimized cluster performance.

Look-up table is another approach that can achieve the control task. The look-up table approach is usually faster and incurs shorter latency than the fuzzy decision approach for small number of combination. But in our case, there are many possible parameters, and the number of combinations is enormous, so the look-up table approach may be not faster than the fuzzy decision approach. Above all, the fuzzy decision is easy to adapt to real world communications, it is a more appropriate approach for decision making in the communication protocol than the look-up table approach.

Three types of data flow on the cluster can be classified by the fuzzy control mechanism, namely the multimedia data flow, the latency sensitive data flow and the throughput sensitive data flow. Dynamic mode switching is carried out by the control mechanism to optimize the performance for a specific data flow type. Results show that the jitter in latency of multimedia data flow is alleviated by switching to isochronous mode. Transmission latency for latency sensitive data flow is optimized intelligently with various packet sizes and different network conditions, latency lower than 125 μs can always be achieved when the packet size is less than 4 Kbytes under different network conditions. Throughput with various packet sizes is improved for the throughput sensitive data flow. By employing the control mechanism, the real time ratio of the IEEE 1394 cluster is comparable to that of the Gigabit Ethernet cluster when the packet size is less than 20 Kbytes. This control mechanism is proved to be valuable in utilizing the IEEE 1394 cluster for parallel computing.

5 Case studies of transportation simulation applications

5.1 Introduction

In this Chapter, we present two case studies of transportation simulation applications on the IEEE 1394 cluster system. Comparisons are made between Ethernet cluster and IEEE 1394 cluster, running the same applications. The results are presented to illustrate the advantages of adopting the IEEE 1394 cluster over the Ethernet Cluster.

Transportation research is concerned with the analysis of phenomena occurring in real world traffic. As the amount of people traveling, as well as transport of goods, is increasing continuously, existing transportation resources such as roads and vehicles get more and more overloaded, resulting in congestion or breakdown of resources. The situation is poor in an urban traffic system, where a limited amount of road capacity must accommodate an increasing amount of traffic. Transportation research can help to understand the characteristics of traffic and propose solutions for existing problems. To achieve these goals, computer simulation of traffic is an important tool. Many works have been carried out in the past to develop simulation models appropriately representing the reality of the transportation system [66], [10], [21], [23], [24], [50], [78], [79], [97].

There are two fundamentally different types of models in traffic simulation namely macroscopic model and microscopic model. Macroscopic models try to

characterize traffic flow with fluid-dynamic approaches [25], giving an aggregated view of the problem. Microscopic models, in contrast, represent all entities (e.g. persons, vehicles, traffic lights, intersections) of the simulated system as individual objects. Besides more accurate simulation results, this has the advantage that individual choices of travelers can be considered. A straightforward approach is to describe the movement of vehicles by equations, directly or indirectly derived from physical laws. In order to reduce the computation time, there are other approaches which model traffic as cellular automata [103], where the dynamic behavior of a vehicle is discretely modeled in time and space. Such an approach is recognized as simple and computationally efficient, yet exhibiting realistic behavior.

Due to the nature of Microscopic Transportation Simulation systems, a huge number of individual entities are needed to be simulated at the same time, the computational demands of such a large-scale traffic simulation can be very high, leading to the need for parallelization. High performance supercomputers are one choice to fulfill the requirement for microscopic transportation simulation systems [62], [63], but they are often very expensive. Recently, computer clusters, which are cheaper alternatives for supercomputers, become popular for solving parallel problems. But high-end clusters are still expensive and this makes it difficult to be widely deployed in a microscopic transportation simulation systems.

The networking system is a major element in the construction of a comput-

ing cluster, cost and performance of the cluster are greatly affected by different networking mechanisms. The IEEE 1394 bus, or the FireWire, which has a transfer rate of 400 Mbps but costs much less than the Gigabit Ethernet, is a suitable candidate for building a cost-effective computer cluster. In previous chapters, we have already devised different techniques to further improve the overall performance of the device, and in this Chapter, we will deploy the device in two microscopic transportation simulation problems. The two different types of microscopic transportation simulation problems are the railway simulation system and the urban traffic system. These two systems show different characteristics and thus have different demands for the computer cluster. In the following sections, we first discuss the railway simulation system, and followed by the urban traffic simulation system. Comparison of implementation results obtained from an Ethernet cluster and the IEEE 1394 cluster are presented for each system.

5.2 Railway simulation system

The railway simulation system is a distributed interactive simulation system for the demonstration and training of railway operations. Most major railway operation procedures are included in the simulation system, like operation of the general railway stations and the dispatching process of railway marshalling station [105], [106], [107], [45].

In our case study, we consider only a marshalling station simulation system.

Marshalling stations are key stations in railway network. Located in the joint-point of several lines of the network, marshalling stations perform the sorting work of freight trains. Wagons that are bounded to different destinations are decoupled from a fleet and those are bounded to the same direction aggregate together and form a new train. To offer a better platform for the training purpose, the system employs 3D animations in the marshalling yard simulation. Figure 5-1 shows a screen-shot of the 3D animation for the marshalling yard of the marshalling station simulation system. As shown in the figure, a 3D animation system displays the marshalling yard situation in real-time, trainees looking at the screen can make control decision such as to brake the wagons according to the wagons' sliding speed from the 3D animation. Figure 5-2 depicts the system structure. There are usually two symmetric marshalling systems in a large marshalling station for the up and down train running directions, only the down direction system is shown in Figure 5-2. The marshalling yard simulator is responsible for the computation for all states including moving objects like wagons, signal lights as well as rail switches. The statuses of these objects are displayed concurrently on many virtual reality displays on different locations on the network. Obviously, the moving objects – the wagons' data must be transmitted in real-time over the network, so between the central simulator and the graphical workstations that run the virtual reality displays, there are huge volume of data communications (heavy communications). Such data communications are in constant time interval and almost in a steady throughput. They also

have the broadcasting nature, or can benefit from broadcasting. Other than the marshalling yard, there are other yards like the receiving yard and the departure yard. Technical process on such yards is limited so no virtual reality display is needed. There are only small amount of random data communications (often train handing over between stations and control panel status changes, so it is regarded as light communications) between the connected receiving yard, departure yard and marshalling yard.

Figure 5-1: The running railway simulation system with 3D display and a control panel

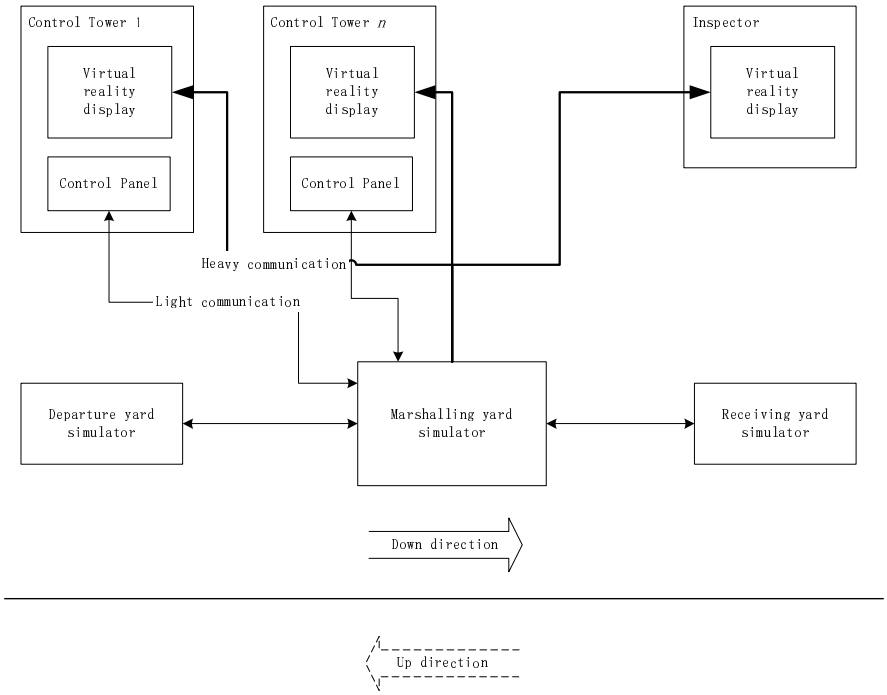


Figure 5-2: The railway simulation system structure

The railway simulation system is originally implemented on a Fast Ethernet cluster. We sampled the data communication requirements for the railway simulation system. Using these data, we simulated the performance of such a system on both Fast Ethernet network and the IEEE 1394 network. The strong data communications consist of data to render every 3D scene. These data include the 3D x-y-z axial locations and stance angles for wagons and engines, light status of signals and opening directions of switches etc. For a given marshalling yard, the number of signal and switches are fixed, so the data size for the description of those objects is also fixed, in our example, the data size for these objects is around 1 Kbytes. Number of wagons and engines in the yard, on the other hand, vary all the time. When a fleet of wagons is being pushed into the yard the number increases, and when wagons are led out of the yard the number decreases. Since these processes often are carried out at the same time, the number of wag-

ons and engines in the yard is actually random. Different number of wagons and engines incur different amount of 3D scene data, and thus influences the performance of the system. In order to observe the performance under various conditions, the data sample we used in the test (Figure 5-3) is excerpted from an empirical simulation. The data was collected from the marshalling station representing situations when the station is empty to when it is almost saturated (about 500 wagons in the yard in this sample), when fleets arrive faster than fleets depart. Given that the data for the description of a wagon is around 100 bytes, the resulting data frame of each 3D scene, as in Figure 5-3, increases gradually from 1 Kbytes to 50 Kbytes. The light communications are sampled using a random distribution, with an average of 0.5 Kbytes/s, simulating the control signal information that is exchanged between yards; in addition, short burst of 20 Kbytes/s, which simulates the trains' information exchange when a train is passing through stations is applied. With such data, we simulated the railway simulation system running on a Fast Ethernet cluster, and on the IEEE 1394 cluster with our communication protocol. Results of the network transfer latencies are shown in Figure 5-4.

There are distinctions between the heavy communications and light communications. The heavy communications carry data for 3D animation, and require low jitter in latency; the light communications carry data for control messages, and require low latency. Since there is no QoS service in Fast Ethernet to facilitate different types of communication, for the Fast Ethernet cluster, we use the

typical Ethernet topology, and all data are transferred directly on the cluster using the UDP protocol. The IEEE 1394 cluster is also using a simple structure. All nodes are connected to the same bus. But different from the Fast Ethernet cluster, in the IEEE 1394 cluster, the strong communications are transferred using the isochronous mode, by employing our communication protocol, the communication pattern of the data flow is recognized by the fuzzy controller and the mode is then switched to isochronous mode. Broadcasting is also used so that a single sending function will distribute data to all the destinations. While the light communications, also identified by the protocol, are transferred using the asynchronous mode.

The performance of such a system on an Ethernet cluster is rather poor, the rendering rate (equals to the strong communication data frame delivery speed) of the 3D animations averaged about 20 frames per second (fps), even on a 100M Fast Ethernet cluster, although its throughput is theoretically sufficient for the data sample as shown in Figure 5-3, but when there is more than 3 virtual reality displays connected, the average rendering rate will be under 20 fps, because many copies of the same data have to be transferred repeatedly on the network and the burst transfer also interferes the network traffic. At the same time, the 3D animation display sometimes is not very smooth, because of the unsteady delivery time of the data packets. While the IEEE 1394 result is very satisfactory comparing to the Fast Ethernet result. The rendering rate of the virtual reality displays is almost steadily at around 30fps, and the latencies of the

IEEE 1394 cluster are ranging from 1 ms to around 2.5 ms, as shown in Figure 5-4 (see Appendix A, Table A-8 for details), this is almost unperceivable by human eyes.

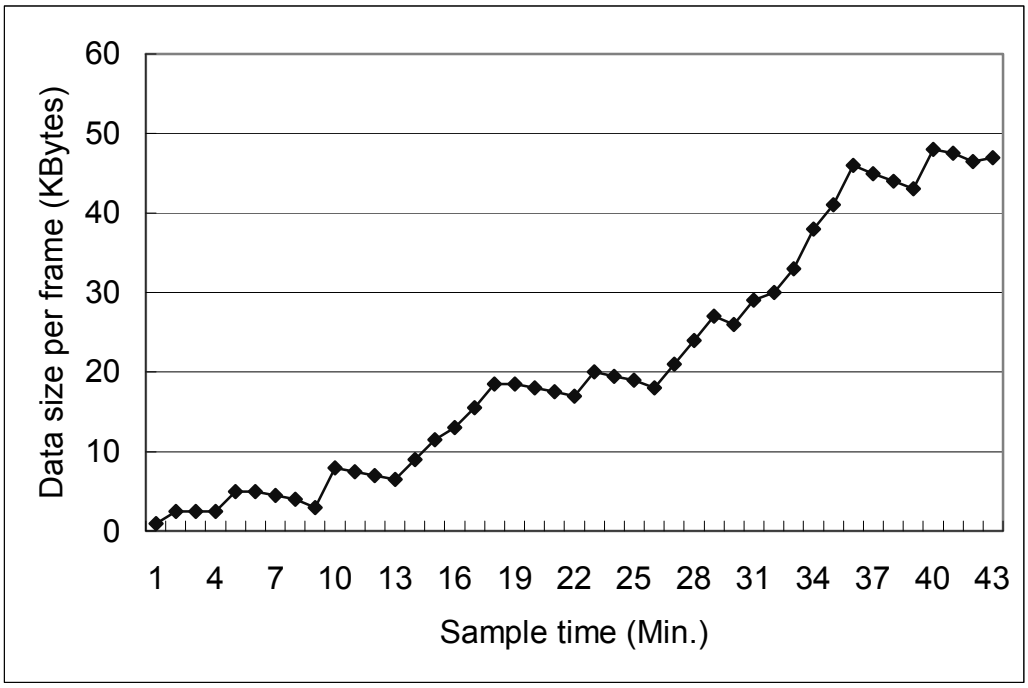


Figure 5-3: The data sample for the railway simulation test

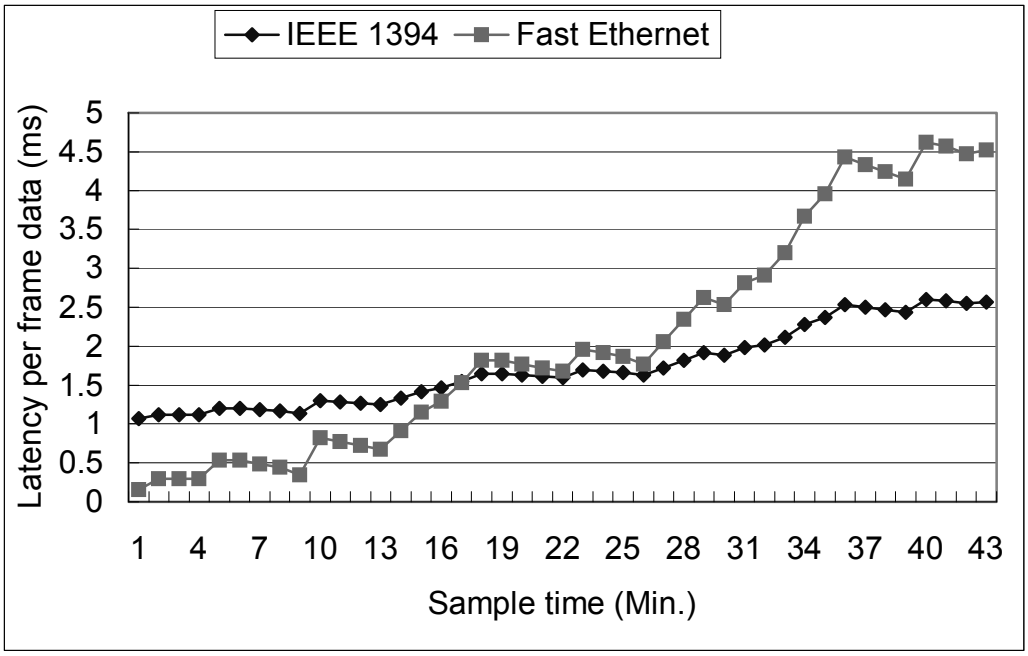


Figure 5-4: The latency results of the railway simulation test

This case study shows that although the Fast Ethernet is presumably to be capable of simulating the railway system, it is not ideal to carry the 3D animation data. The IEEE 1394 cluster shows its excellence in carrying multimedia data by using the isochronous mode. Although when the data packet size is small, the latency of IEEE 1394 is greater than that of the Fast Ethernet, it is still tolerable. During the testing period, the IEEE 1394 cluster retains very low jitter in delivering time for each data packet, that is crucial for carrying multimedia data.

5.3 Urban traffic simulation system

Many researchers addressed issues regarding parallel simulation of urban traffic system on computer clusters [68], [83]. There are two important factors describing the speed of a parallel system, i.e. real time ratio and speed-up. Speed-up, as in equation (4.1), describes how the running time is shortened when a program runs in parallel comparing to the program runs in serial. Real time ratio is a major concern of simulation systems; it is described as the real world running time of a system divided by the simulation time of the system on a computer cluster. In [14], [15], they reported Beowulf clusters (clusters of Pentium computers with Linux operation system) can be used successfully for large scale problems. The latency of Ethernet communication sets a hard limit on computing speed to about 150 simulation time steps per second (if one time step simulates one second of real world time, the real time ratio is 150), no matter what the problem

size is. They further showed that the use of Myrinet communications technology overcomes this problem. A real-time ratio of 180 was reached on 64 CPUs Myrinet clusters. In [27], they describe an implementation of transport simulation based on a graphical parallel programming environment called P-GRADE. The transport simulator called MadCity, simulates a specific road network of a city and shows cars moving on the roads. They achieved a speed-up around 6 with a 16 nodes cluster, which means the simulation time is 6 times shorter running on the cluster than running on a single computer.

The urban traffic simulation system is a microscopic simulation system that is used for transportation forecasting and planning. The simulation is not carried out by abstracting vehicles into traffic flows, but by simulating each vehicle as an individual object. Each vehicle is represented by a corresponding data structure, as shown in Figure 5-5. The vehicle is identified by the traveler's unique ID. There are data fields which describe the vehicle's movement status (velocity, acceleration, etc.), and data fields which describe the vehicle's location on the road (road ID, lane ID, etc.). There are also other data fields related to the origination and destination of the traveler, travel plan and strategy of the traveler, etc. The data structure for describing a vehicle under different simulation models can be different. In our simulation, we use a data structure that is around 100 Bytes.

When vehicle travels from one location to another, the corresponding data is sent through the network from one node to another. Since there will be typically millions of cars running in a metropolitan traffic network, a vast amount of

memory is needed for such a system, also very high network throughput is required. Such system is a typical application for a cluster system, and they often use a domain decomposition method to divide the problem as in [15], [16]. The metropolitan road map is divided into several domains, and each domain is handled by one node in the computing cluster, as is illustrated in Figure 5-6. The traffic zone consists of intersections and roads. Symbols of the road intersections are in different shapes, intersections with the same shape and the roads connected to them are partitioned to the same computation node. Thus the traffic zone in Figure 5-6 is partitioned to four computation nodes.

The sample cluster structure of our urban traffic simulation system is shown in Figure 5-7. We tested both the Fast Ethernet cluster and IEEE 1394 cluster. In our test, each cluster consists of four computation nodes.

For the Fast Ethernet cluster, the structure is the simple bus type, for the IEEE 1394 cluster, we use one IEEE 1394 bus for each link. In the urban traffic system, the whole system status is updated step by step, with one step represents 1-second time in the simulated world. The time that is consumed in updating the system status for one step follows a uniform distribution, with limits between 0.5 ms to 1.5 ms and with a mean of 1 ms. The data that is exchanged between nodes (vehicle travels across computing node's border) at the end of each step are ranging from 8 Bytes to 50 Kbytes.

```
int ID;                 //Traveler ID
double V;               //Vehicle velocity
double A;               //Vehicle acceleration
int locID;              //The road ID the Vehicle is on
int locX;               //Which Lane is the Vehicle on
float locY;             //Location start from road end
int OrgID;              //The traveler's origination ID
int DestID;             // The traveler's destination ID
List travelPlan;        //The traveler's plan of roads to go to reach the destination
List travelStrategy;    //The strategies the traveler use to make decisions upon "cirCondition"
List cirCondition;      //The pointers to circumstances factors that may influence the traveling plan of the traveler (weather; road condition from traveler's observation, other public media or ITS system; random instance etc.)
.
// other internal data
.
```

Figure 5-5: Vehicle data object

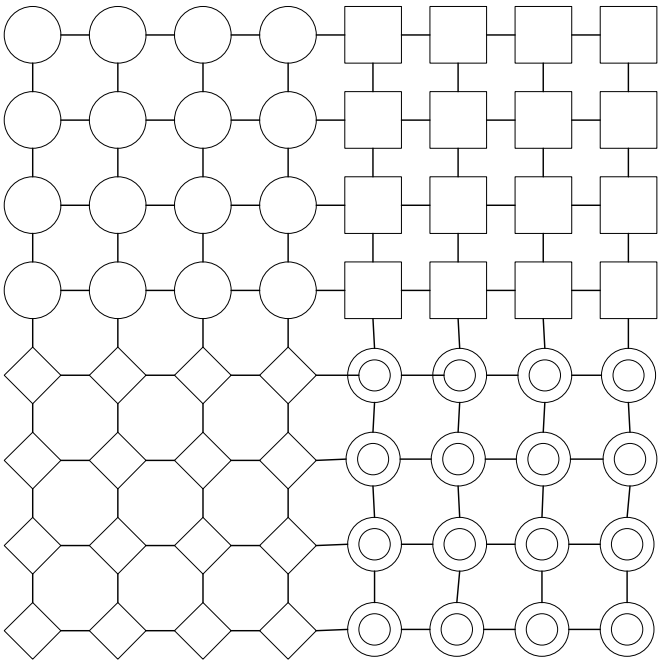


Figure 5-6: The road network and domain decomposition

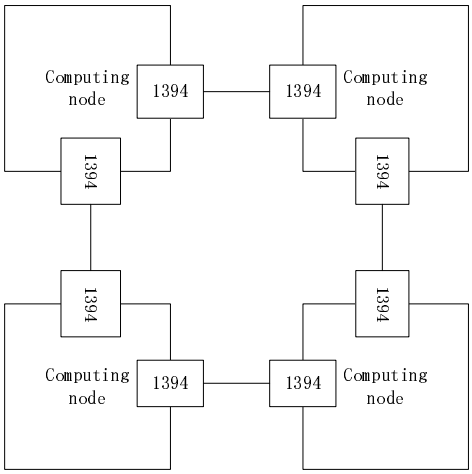


Figure 5-7: The cluster structure

We first tested the performance of the Ethernet cluster and the IEEE 1394 cluster based on the original Linux driver. The results are shown in Figure 5-8 and 5-9. In Figure 5-8, the Ethernet performance is compared with the performance of the IEEE 1394 cluster using isochronous mode only.

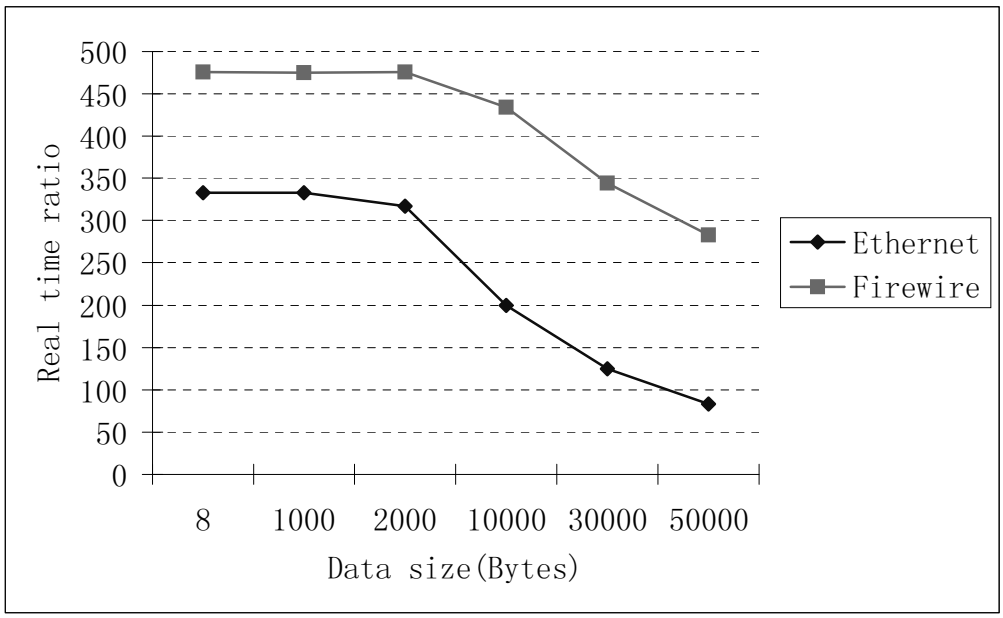


Figure 5-8: Real time ratio of the Fast Ethernet cluster and IEEE 1394 cluster using isochronous mode

The real time ratio (RTR) is determined by the number of steps that can be up-

dated in 1-second, so a higher RTR value implies a better performance. For the Fast Ethernet cluster, the maximum RTR is 330, and drops below 100 when data size becomes 50 Kbytes. For the IEEE 1394 cluster, the maximum speed up is 480, and also drops when data size increases, but a RTR around 300 can be achieved when data size is 50 Kbytes. The RTR of IEEE 1394 is much higher than the Fast Ethernet cluster, and even more than double when the packet size is larger than 10 Kbytes.

The RTR is not as high as expected for the IEEE 1394 when the data size is small; it is because for the isochronous mode latency cannot be less than 125 μs even when the data size is smaller than 4 Kbytes. In Figure 5-9, anther test using asynchronous mode is carried out, the result shows the RTR improves greatly when the packet size is small, with the maximum value near 800. While the RTR drops very fast comparing to the isochronous mode value, when the data size is around 20 Kbytes, it drops below the isochronous value.

Since data size ranging from 10 Kbytes to 50 Kbytes is the typical value in the real simulation system, it is necessary to switch between the transmission modes so that we can sustain maximum RTR in all conditions.

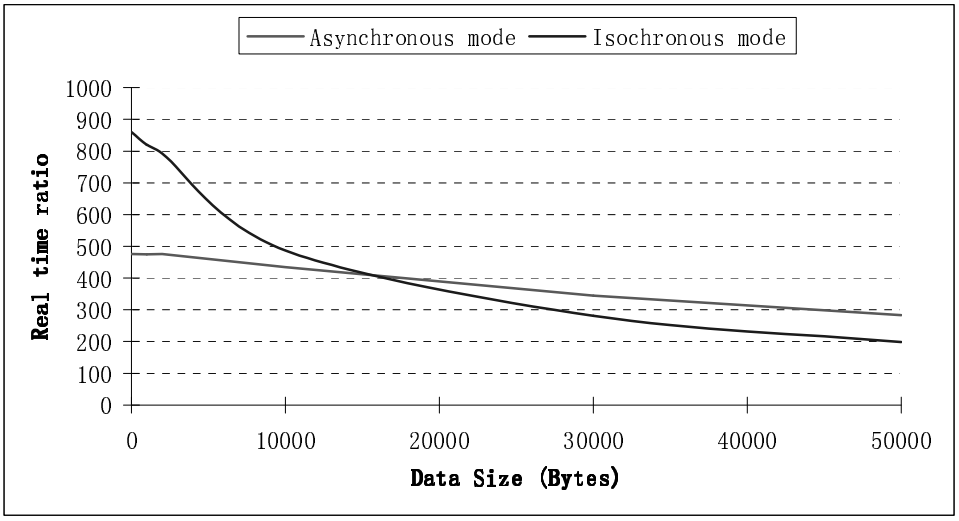


Figure 5-9: RTR of the IEEE 1394 cluster using asynchronous mode comparing to using isochronous mode

Figure 5-10 shows the result when running the urban traffic simulation system on the IEEE 1394 cluster with our new communication protocol. The real time ratio decreases when the data size increases. The maximum RTR is around 800, with the smallest data size at 8 bytes. When the data size is between 2000 bytes to 10000 bytes, the performance with the new protocol is slightly better than using asynchronous mode, this is because the fuzzy controller uses two concurrent asynchronous mode transmissions to carry the data payload. The figure shows that using our protocol the overall performance of the urban traffic simulation is optimized, very high real time ratio can be achieved by utilizing the asynchronous mode. When the asynchronous mode cannot improve the performance with packet size greater than 10000 bytes, isochronous mode is used to improve the performance. So advantages of both asynchronous and isochronous mode contribute to the overall performance.

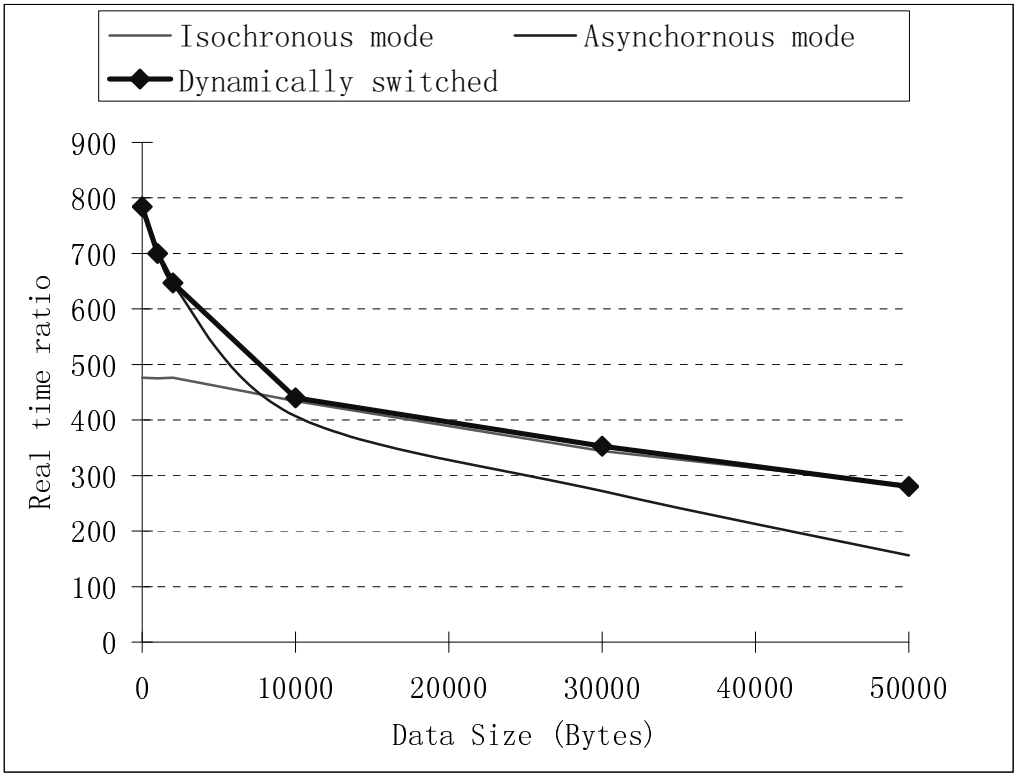


Figure 5-10: Speedup of the IEEE 1394 cluster using the control mechanism in the protocol, with comparing to the isochronous mode and asynchronous mode

5.4 Concluding remarks

In the case study of the railway simulation system, the IEEE 1394 cluster shows its excellence in carrying multimedia data by using the isochronous mode. The performance of 3D animation which is previously suffered from the big jitter of the Ethernet network is greatly reduced. Other type of data, such as data carrying control messages also can benefit from the high throughput nature of the IEEE 1394 bus.

In the case study of the urban traffic simulation system, we achieved much better real time ratio when using the IEEE 1394 cluster than using the Fast

Ethernet cluster. We can achieve a maximum real time ratio of around 960 with our new protocol, which is even better than the real time ratio, 800, which is reported in [15] by using a 64 computing nodes Myrinet cluster. Although it is hard to compare the two results directly because we used a different prototype of the simulation system and employed a different scale cluster. This still shows that the IEEE 1394 cluster has great potential in the high-speed networking area.

The IEEE 1394 cluster was employed in solving transportation problems in these case studies. The same cluster can be applied to other parallel computation problems and only a few minor alterations of the data transmission code of those problems are needed as is shown in Figure 5-11. With the API similar to MPI or socket programming, other problems can easily adopt this communication protocol.

There may be cases that the data transmission features of other types of problem are different from the categories we have discussed, therefore, our communication protocol cannot produce optimized results. In such cases, tuning of the fuzzy control mechanism is needed and the communication protocol can adapt to these parallel computation problems after tuning. While the IEEE 1394 cluster is suitable for urban traffic simulation problem and problems which can benefit from the domain decomposition approach like atmosphere simulation etc., it may not be suitable for other problems like parallel sorting, in which non-adjacent nodes may exchange data frequently, and sophisticated routing problems arise. A switched network would be more suitable for those problems.

Problem which uses socket API	**Problem which uses the Protocol over 1394**
func()	func()
{	{
.	.
.	.
.	.
initsock();	fw_Init(0, 0);
.	.
.	.
sendto(socket, message, len,0, &to, len);	fw_send(message, len, did, gid);
recvfrom(socket, message,len,0,&from,&len);	fw_receive(message, len, sid, gid);
.	.
.	.
closesocket(socket);	fw_finalize();
.	.
.	.
}	}

Figure 5-11: Data transmission codes to be altered in a parallel computation problem.

6 Performance prediction of larger scale IEEE 1394 cluster performance

6.1 Introduction

In Chapter 3 and Chapter 5, we presented results obtained from an IEEE 1394 cluster for various conditions, including in the case study of the emulation of macroscopic traffic simulation. Our results are based on a test-bed that consists of only four workstations. Certainly, such a small scale system is only suitable for experimental purposes; most likely system in a much larger scale is required for solving real life problems, for example system consists of 64 nodes is used to implement a traffic simulation system [14]. However, due to the limitation in terms of resources, it is not possible to implement a full-scale cluster with 64 nodes to carry out our test, so it is necessary to provide a performance prediction for larger scale clusters.

Stochastic process and queuing theory models are widely used in the performance analysis of computer systems. Such as Erlang's formulas [22] were heavily used to study resources requirement for a telephone network, and Jackson's theory for networks of queues [42], [43] was used to study the performance of the ARPANET, a precursor of the Internet. These models provide simple formulas that can be used directly to evaluate the performance of single-server systems and some of the simplest models of multi-server systems such as those

studied by Erlang and Jackson. These explicit formulas not only can be evaluated numerically but also can give us good intuition on how the system parameters affect the performance. But stochastic process and queuing theory are not always ideal solutions for performance modeling. On one hand, these models are often based on strict assumptions, such as a Markov process must has the Markov property, i.e. in such a process, the past is irrelevant for predicting the future given knowledge of the present, while this assumption may be true for a telephone network, it is often not the case for a traffic network in which adjacent persons' activity apparently influenced by each other. Furthermore, these models may not be held in some cases even the assumptions are met [57]. On the other hand, when employing such stochastic theory for more complex models, the formulas obtained become more complex. This is particularly true for modeling the performance of multi-node computer clusters which are often sophisticate multi-server systems. Multidimensional Markov chains may be used to predict the performance of computer clusters, but these modules result in very complicated expressions that are difficult to evaluate numerically [71], [49], [47], [56], and often have to resort to simulation method to reduce numerical results [102]. Though it is difficult to model the performance of a computer cluster with stochastic process and queuing theory, it is still possible to systematically predict the performance with simplified stochastic model [40], [44], [72], and if the network transmission features of a specific application are taken into account, the performance prediction for that specific application can produce ideal result.

In this Chapter, we will present a model to predict the performance of the IEEE 1394 cluster under different configurations, mainly with different numbers of nodes. Performance prediction of different kinds of applications on clusters is a sophisticated problem, and is related to specific application features. On the other hand, performance of larger scale cluster is not a major concern for applications that require short latency and low throughput like the railway simulation system. In this Chapter, we will concentrate on the performance prediction of parallel micro-simulation, taking the urban traffic simulation system as a prototype. With the help of such a model, users can gauge the performance of the IEEE 1394 and determine a suitable scale for the cluster for the corresponding application.

6.2 Performance modeling

In order to systematically predict the performance of a parallel simulation system, several assumptions about the computer architecture need to be made. In the following, we demonstrate how to derive predictive equations for coupled workstations and for parallel computers based on the study of [66].

The method for this is to systematically calculate the time for one simulation time step of the micro-simulation. Assume that the time for one time step includes components in computation, T_{comp}, and communication, T_{comm}. If these do not overlap, as it is reasonable to assume for coupled workstations, we have

$$T(p) = T_{comp}(p) + T_{comm}(p) \tag{6.1}$$

Where p is the number of computation nodes.

Time for computation is assumed to the following:

$$T_{comp}(p) = \frac{T_1}{p}(1 + f_{ovr}(p) + f_{dmn}(p)) \tag{6.2}$$

Where T_1 is the time of the same code on one computation node (assuming a problem size that fits on available computer memory), p is the number of commutation nodes, f_{ovr} includes overhead effects (e.g., split links need to be administered by both computation nodes), f_{dmn} includes the effect of unequal domain sizes.

Time for communication typically has two factors: latency and bandwidth. Latency is the time required to initiate the communication, as a result it is independent of the message size. Bandwidth describes the number of bytes that can be communicated per second. So the time for one message is

$$T_{msg} = T_{lt} + \frac{S_{msg}}{b} \tag{6.3}$$

Where T_{lt} is the latency, S_{msg} is the message size, and b is the bandwidth.

However, for many of today's computer architectures, bandwidth is given by at least two contributions: node bandwidth, and network bandwidth. Node bandwidth is the bandwidth of the connection between the computation node and the network. If two computers communicate with each other through the network, this is the maximum bandwidth that they can reach. Because of this, it is sometimes also called the "point-to-point" bandwidth.

The network bandwidth is governed by the technology and topology of the network. Typical technologies are bus topologies (Fast Ethernet, IEEE 1394 etc.), switched topologies, two-dimensional topologies (e.g. grid/torus), hyper-cube topologies, etc. A traditional local area network uses Fast Ethernet, and it has a shared bus topology. In a shared bus topology, all communication goes through the same medium; i.e., if several pairs of computers communicate with each other, they have to share the bandwidth.

For example in a 100 Mbps Fast Ethernet network, the node bandwidth was found to be about b_{nd} = 40 Mbit [6]. It implies that if two computers communicate point-to-point at the node bandwidth, the Fast Ethernet network can accommodate four computers to communicate in this pattern at the same time, i.e. using 80% of the 100 Mbit/s, while more computers were limited by the network bandwidth. For example, ten computers could maximally get 100/5=20 Mbit/s each.

A switched topology is similar to a bus topology, except that the network bandwidth is determined by the backplane of the switch. Very often, the backplane bandwidth is high enough to have all nodes communicate with each other at full node bandwidth, and for practical purposes one can thus neglect the network bandwidth effect for switched networks.

If computers become massively parallel, switches with enough backplane bandwidth become too expensive, an 8-port small Myrinet CLOS type switch (support direct n-to-n parallel full speed transmission) costs more than 4000

USD [64]. As a compromise, such clusters usually use a communication topology where communication to "nearby" nodes can be done at full node bandwidth, whereas global communication suffers some performance degradation. Since we partition our traffic simulations in a way that communication is local, we can assume that we do communication with full node bandwidth on a cluster. That is, on a parallel cluster, we can neglect constraints of the network bandwidth. This assumes, however, that the allocation of street network partition to computational nodes is done in some intelligent way which maintains locality.

As a result of the above discussion, we can assume that the communication time per time step is

$$T_{comm}(p) = n_{nb}(p)T_{lt} + \frac{N_{spl}(p)}{p}\frac{S_{bnd}}{b_{nd}} + N_{spl}(p)\frac{S_{bnd}}{b_{net}} \qquad (6.4)$$

The term n_{nb} is the number of neighbor domains that each computation node should talk to, all information which goes to the same computation node is presumed to be collected and sent as a single message, thus incurring the latency only once per neighbor domain. For $p = 1$, n_{nb} is zero since there is no other domain being connected. Since we use a 2D mesh for the cluster topology (Figure 6-1), n_{nb} can be 2 for nodes in the corner, 3 for nodes on the border and 4 for nodes in the middle.

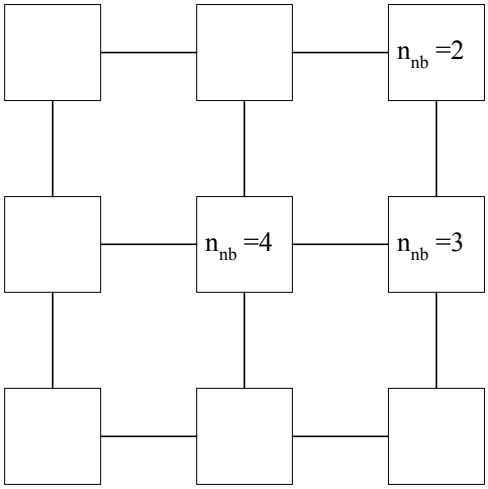


Figure 6-1: A 2D mesh cluster

T_{lt} is the latency (or start-up time) of each message. T_{lt} between 50 and 200 μs are typical values for our communication protocol over the IEEE 1394 cluster.

$N_{spl}(p)$ is the number of split links (split links is roads which are split and located in adjacent computation nodes, as in Figure 6-2) in the whole urban traffic simulation system, this will depend on the simulated road network and the domain decomposition. Accordingly, $N_{spl}(p)/p$ is the number of split links per computational node. S_{bnd} is the size of the message per split link. b_{nd} and b_{net} are the node and network bandwidths, as discussed above.

From above, the combined time for one time step is

$$T(p) = \frac{T_1}{p}(1 + f_{ovr}(p) + f_{dmn}(p)) + n_{nb}(p)T_{lt} + \frac{N_{spl}(p)}{p}\frac{S_{bnd}}{b_{nd}} + N_{spl}(p)\frac{S_{bnd}}{b_{net}} \quad (6.5)$$

For the 2D mesh topology, for $p \to \infty$ the number of neighbor scales as $n_{nb} \sim 4$ and the number of split links in the simulation scales as $N_{spl} \sim \sqrt{p}$. If f_{ovr} and f_{dmn} are small enough, we have:

- For a share or bus topology, b_{net} is relatively small and constant, and thus

$$T(p) \sim \frac{1}{p} + 1 + \frac{1}{\sqrt{p}} + \sqrt{p} \to \sqrt{p} \tag{6.6}$$

- For a switched or a parallel cluster topology, we assume $b_{net} = \infty$ and obtain

$$T(p) \sim \frac{1}{p} + 1 + \frac{1}{\sqrt{p}} \to 1 \tag{6.7}$$

Thus, in a shared topology, adding computation nodes will eventually increase the simulation time and make the simulation slower. In a non-shared topology, adding computation nodes will not make the simulation any faster, but at least will not be detrimental to the computational speed. The dominant term in a shared topology for $p \to \infty$ is the network bandwidth, while the dominant term in a non-shared topology is the latency.

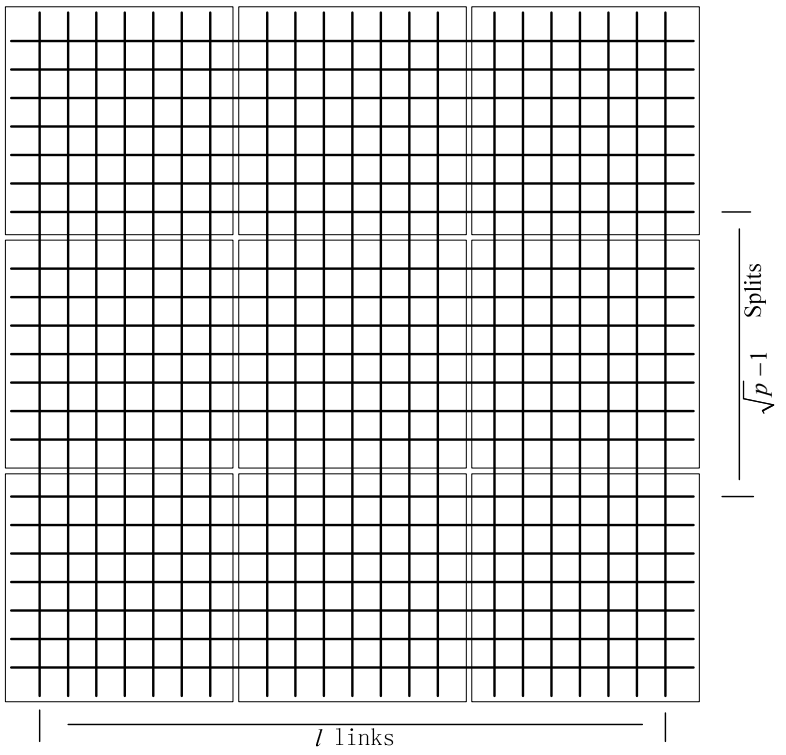


Figure 6-2: A prototype urban traffic simulation on p nodes cluster

6.2.1 Speed-up of the urban traffic simulation

If we consider a simple prototype urban traffic simulation system, in which all the road are straight lines and evenly distributed in the urban area, the $N_{spl}(p)$ can be calculated as

$$N_{spl}(p) = (\sqrt{p} - 1)2l \tag{6.8}$$

Where l is the number of roads along one of the edges of the square urban area.

For the given application, we assume that in a single node environment, time used for one simulation step is $T(1)$ = 0.1s, and the speed-up factor can be obtained as $speedup(p) = \frac{T(1)}{T(p)}$. For the IEEE 1394 cluster, T_{lt} is 50μs, b_{nd} is 31 Mbytes/s. For the Fast Ethernet cluster, T_{lt} is 0.8 ms, b_{nd} is 12 Mbytes/s. then we have the performance prediction as in Figure 6-3.

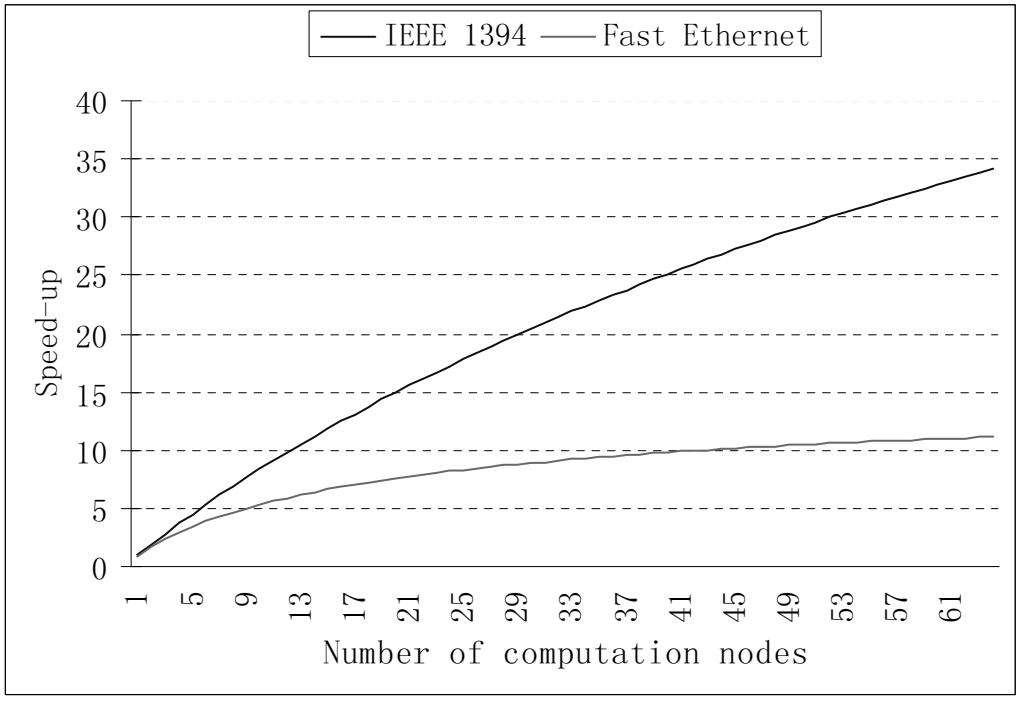


Figure 6-3: The speed-up factor of IEEE 1394 and Fast Ethernet cluster

The speed-up for the IEEE 1394 cluster of 4 computation nodes is 3.69, which is proved by our empirical study in the previous Chapter. The result shows that the speed-up of the IEEE 1394 cluster is much better than that of the Fast Ethernet cluster. Considering that the cost of building a IEEE 1394 cluster is almost the same as that of building the Fast Ethernet cluster when the computation node is less than 64, the IEEE 1394 cluster is definitely a better option to carry out such a large scale urban traffic simulation system.

6.2.2 Real time ratio of the urban traffic simulation

Speed-up describes how much faster the parallel application runs on a parallel computer than the application uses the same algorithm runs on a single computer. For a discrete-event simulation system similar to the urban traffic simulation, real time ratio is often another important factor which describes how much faster the parallel simulation runs on a parallel computer than the simulated system. Consider an urban traffic simulation system which performs the traffic prediction of an urban area, it is reasonable to expect a simulation for 1 hour traffic in the future should be finished with 5 minutes, in which the real time ratio is 60min/5min = 12.

In the last section, the time for one time step simulation $T(p)$ is derived based on the time of running the same system on a single computer, and the data size which is exchanged between nodes. While these factors for the computation

of $T(p)$ is excerpt from an existing urban traffic simulation system [66], simulation using other algorithms or even simulation using the same algorithm while simulating different urban areas may behave quite differently. It is difficult to predict performance of other simulations from the existing one. In this section, probability theory is employed to describe the performance of discrete-event simulation, so that performance prediction can be carried out when detailed information about the system is not available.

Synchronous iterative algorithm [61], [75], [104] is used in discrete-event simulations. In the simulation, each processor performs a portion of the required computation of each iteration, and barrier synchronization separates every iteration from previous and subsequent iterations. If $T_i(p)$ i=1,2,…,p is the iteration time for the processor i, then $\max(T_i(p) \mid 1 \le i \le p)$ governs the iteration time for the whole system.

If we set the assumption that the iteration times for each node are independent and identically distributed (i.i.d.), from order statistics [19], if i.i.d. random variables $T_1(p)$, $T_2(p)$, ..., $T_p(p)$ have a mean (μ) and a variance (σ) then

$$E[\max T_i(p) \mid 1 \le i \le p] \le \mu + \frac{p-1}{\sqrt{(2p-1)}}\sigma \qquad (6.9)$$

From this equation, an upper bound for the iteration time can be obtained. This equation requires that only the mean and variance of the iteration time are known. If information is available about the nature of the iteration time distribution, the upper bound can be tightened. If $T_1(p)$, $T_2(p)$, ..., $T_p(p)$ are i.i.d. random variables distributed exponentially with parameter λ, then from extreme value

theory[4],

$$E[\max T_i(p) \mid 1 \le i \le p] \approx \frac{\ln p + \gamma}{\lambda} \tag{6.10}$$

where γ is Euler's constant (0.5772). If the variable are uniformly distributed between a and b then

$$E[\max T_i(p) \mid 1 \le i \le p] \approx b - \frac{b-a}{p}. \tag{6.11}$$

If the variable are normally distributed with parameters μ and σ then

$$E[\max T_i(p) \mid 1 \le i \le p] \approx \mu + \sigma[\sqrt{2\ln p} - \frac{\ln(\ln p) + \ln 4\pi}{2\sqrt{2\ln p}} + \frac{\gamma}{\sqrt{2\ln p}}]. \tag{6.12}$$

We use these functions to model the communication time $T_{comm}(p)$ of a cluster system. If the $T_{comm}(p)$ is an i.i.d. random variable, where for p=1, μ=600 μs and σ=30 for the IEEE 1394 cluster, and μ=2000 μs and σ=100 for the Fast Ethernet cluster. Figure 6-5 depicts the prediction based on equation (6.12). The Ethernet result is also given in the figure.

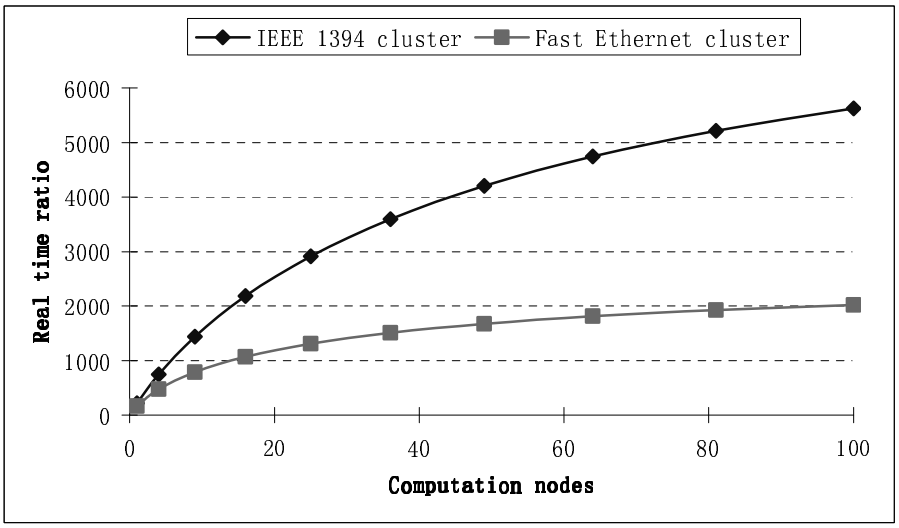


Figure 6-4: The real time ratio factor of IEEE 1394 and Fast Ethernet cluster

In Figure 6-4, we can observe that both of the IEEE 1394 cluster and the Fast Ethernet cluster can benefit from the increase of computation nodes. The real time ratio increases when the number of computation node increases. The increase is faster when the number of computation nodes is less than 20, and becomes slower after that. Due to the slower transmission speed and larger variance of the Fast Ethernet interface, the drop in the increasing rate is much larger than that of the IEEE 1394 interface. The real time ratio of the IEEE 1394 cluster is over 100% higher than that of the Fast Ethernet cluster for all numbers of computation node.

To observe the effect of the variance of the communication time, Figure 6-5 illustrates the IEEE 1394 cluster real time ratio with μ=600 μs and σ changed from 30, 60, 60, to 300. This result shows that the variance of the communication time has limited effects on the real time ratio when the number of computation node is under 9, but it influences the real time ratio greatly after that. Comparing the real time ratio when computation node number is 100, the result of σ=30, almost double the result of σ=90. With a very large σ=300, the increase of real time ratio is almost flat after computation node number greater than 20.

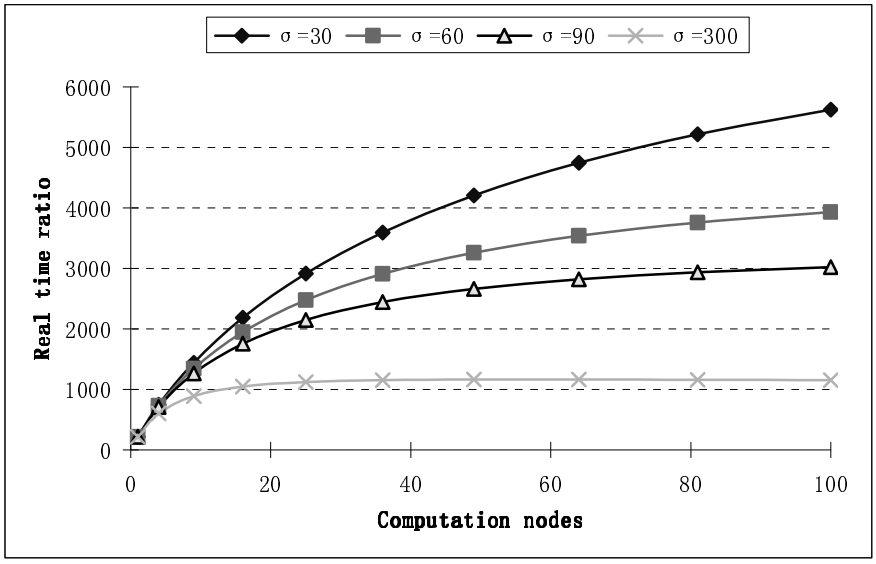


Figure 6-5: The real time ratio factor of IEEE 1394 cluster with different variance of communication time

6.3 Concluding remarks

In this Chapter, we have presented techniques to predict of the performance of the IEEE 1394 cluster with more computation nodes. Comparison of the IEEE 1394 cluster with the Fast Ethernet cluster is also given. The performance prediction is aimed at a specific type of parallel applications, i.e. urban traffic simulation applications. Features of these applications are used in the performance prediction, this makes the prediction agrees better to such applications, but also makes it hard to generalize it to predict the performance for other applications. Order statistics is used to derive the upper bound for the time of simulation iteration; this is based on the assumptions that the time is i.i.d random variables, and follows a known distribution, the prediction will be inaccurate if either of the two assumptions is violated.

Results show that the speed-up factor of IEEE 1394 cluster is much higher than that of the Fast Ethernet (more than 200% higher in the case of using 64 computation nodes). The real time ratio of the IEEE 1394 cluster is also better than that of the Fast Ethernet cluster; it is often 100% higher for the IEEE 1394 cluster than the Fast Ethernet cluster. The data transmission pattern of a parallel simulation is shown as an important factor to the cluster performance. If the message size variance of each iteration of a parallel simulation is very large, at some point, increasing the number of computation node eventually cannot improve the overall system performance.

7 Conclusion and future work

7.1 Conclusion

Computer clusters attract many attentions for implementing software simulation for transportation problems. Usually Fast Ethernet cluster in the low-end and Myrinet cluster in the high-end are employed in such problems [12], [6]. While the performance of the Fast Ethernet cluster is limited, and Myrinet cluster is quite expensive, we propose to employ the IEEE 1394 device as a new method to build a cost-effective computation cluster.

In this thesis, we compared the cost-effectiveness of implementing a computer cluster based on the IEEE 1394 device, as well as using the Fast Ethernet, Gigabit Ethernet and the Myrinet. Initially, we only tested the device based on the available software drivers, such as IP-over-Firewire, and our results showed that the communication bandwidth is only about 120 Mbps, which is much lower than the nominal value of 400 Mbps. After further investigations, we found that the communication modes (isochronous mode and asynchronous mode) supported by the IEEE 1394 have different characteristics and the total bandwidth can be improved by proper utilization of the communication modes. Basically, the isochronous mode can deliver higher bandwidth; however, it does not include features such as acknowledgement, or re-send, which are very important for implementing a computer cluster.

A new communication protocol is implemented in order to further exploit the capacity of the 1394 device. Message passing mechanism is supported in our devised protocol, so that it can be used to deploy a generic computer cluster using the IEEE 1394 bus. This enables parallel applications which employ the message passing mechanism to run on an IEEE 1394 cluster. We implemented an acknowledgement mechanism over the isochronous mode of the IEEE 1394 to enable the isochronous mode transmission to support reliable data transmission. The maximum throughput based on our new protocol is around 250 Mbps, which is much higher than that of other implementations like the IP-over-Firewire under Windows and Linux. The minimum latency of our protocol is around 50μs, and is far lower than that of IP-over-Firewire. The performance of our protocol is much better than that of the Fast Ethernet, and is comparable to the Gigabit Ethernet, as described in Chapter 4. The real time ratio of the IEEE 1394 cluster is even better than that of the Gigabit Ethernet cluster when considering the implementation of the traffic simulator when the amount of data being exchanged is small. While the performance is not comparable with that of the Myrinet, but the cost of IEEE 1394 is much lower, and therefore, IEEE 1394 is still a cost-effectiveness solution for high-speed networking. The cost of alternative high-speed networking devices such as Myrinet has been dropping and this makes these devices less expensive in the future, but the situation for the IEEE 1394 device is similar. Since high-speed networking devices like Myrinet are in nature expensive, for example, an 8-port Myrinet fi-

ber switch costs about 4000 USD, we can predict that the cost of the IEEE 1394 device will still be relatively low when comparing to these high-speed networking devices, therefore, the IEEE 1394 device will still be a cost-effective solution for high-speed networking in the future.

Although the bandwidth of the isochronous mode is higher than that of the asynchronous mode, there are cases where, the asynchronous mode can give a better performance. Therefore, techniques to switch between the two communication modes were investigated. A fuzzy control mechanism to optimize the IEEE 1394 network for cluster computing was proposed. This mechanism enables the dynamic performance monitoring and tuning of the computer cluster system. We developed a control mechanism which can intelligently control the underlying IEEE 1394 network and can tune the network to provide optimized performance. This control mechanism was proved to be valuable in supporting the IEEE 1394 cluster for parallel computing as described in our case studies.

In order to evaluate the actual performance of a computer cluster based on the 1394 devices, embodying the new communication protocol as well as the control mechanism, two types of existing transportation simulation systems were migrated to our IEEE 1394 cluster and their performance were examined. In the case of the railway simulation system, the IEEE 1394 cluster shows its advantages in carrying multimedia data by using the isochronous mode. The performance of 3D animation which was formerly suffered from jitters due to communication latency of the Ethernet network was greatly improved. Other

types of data can also benefit from the IEEE 1394 cluster. In the case study of the urban traffic simulation system, we achieved much better real time ratio using the IEEE 1394 cluster than using the Fast Ethernet cluster. We achieved a maximum real time ratio of around 960 steps with our new protocol, which is even better than the real time ratio about 800 steps, that is reported in [15] by using a 64 computing nodes Myrinet cluster. Although it is difficult to compare the two results directly since we use a different prototype of the simulation system and employing cluster system with different scales, it still reflects that the IEEE 1394 cluster is an alternative option to the Myrinet cluster.

We further investigated methods to predict the performance of the IEEE 1394 clusters when more computation nodes are required. The results show that the speed-up and real time ratio of the IEEE 1394 cluster is much better than that of the Fast Ethernet cluster. Considering that the cost of building a IEEE 1394 cluster is no higher than building the Fast Ethernet cluster when the computation node is less than 64, the IEEE 1394 cluster is definitely a better option to carry out such large scale urban traffic simulation system.

The main contributions of this thesis are given below:

- A new communication protocol over the IEEE 1394 device for interconnecting PCs into a computer cluster.
- A novel control mechanism in the communication protocol to fully utilize the capacity of the IEEE 1394 device.
- A study of two different structured IEEE 1394 cluster applied in two

types of transportation simulation systems.

- A performance prediction of urban traffic simulation system on a larger scale IEEE 1394 cluster.

In conclusion, our investigations proved that the IEEE 1394 device can be applied in implementing a cost-effective high-speed network. In order to fully utilize the device's capacity, a new protocol and a control mechanism have been devised. Our case studies further substantiated our findings.

7.2 Future directions

A full featured MPI interface over the IEEE 1394 cluster

The MPI message passing library [90] which is designed specifically for parallel computing on parallel computers, defines a full-set of functions to support message passing. The communication protocol that was implemented in our work, which contains basic send, receive, and initialization functions, is a subset of the MPI standard. The two studied transportation simulation systems have to be modified in the message passing part so that they can utilize our protocol to exchange messages. If we want to deploy other application which is designed specifically for the MPI library, it has to be modified as well. But to modify an existing computer program is often a time consuming task, and sometimes it is not even possible if the one who wants to deploy the application does not possess the source code of the software. Although there are IP-over-Firewire implemen-

tations which enable TCP/IP on the IEEE 1394 interface, and the MPI can utilize the IEEE 1394 interface via IP-over-Firewire, results show that the MPI over TCP/IP suffers a lot from the protocol overhead and delivers poor performance. This has made it difficult to let the IEEE 1394 cluster to accommodate more applications. Some MPI implementations offer an open architecture, therefore, it is possible to adopt our protocol to the open architecture and enable MPI call on the IEEE 1394 cluster.

The future IEEE 1394b cluster

The IEEE 1394b device that can communicate in a higher speed (800 Mbps) has emerged in the commodity market. The new IEEE 1394b device is different from the IEEE 1394-1995 device not only in the throughput, but also in the bus arbitration mechanism. This new mechanism lifts the restriction that asynchronous mode data transmission must not occupy more than half of the bus cycle time. It enables the asynchronous mode transmission to fully utilize the bus capability, so that it can offer both reliable data transmission and high throughput. This improvement of the IEEE 1394b device changes the communication features of the asynchronous and isochronous mode, so new measures need to be taken in our protocol which is previously designed for IEEE 1394a device to better utilize the advantages of both modes of the IEEE 1394b device, so that the IEEE 1394b capacity can be explored to support higher speed parallel computing.

A preliminary experiment is conducted to study the basic performance of the IEEE 1394b device. The test bed consists of two workstations which are directly connected by an IEEE 1394b link, and the *netio* toolkit is employed to perform data transmission. The Unibrain IP-over-Firewire driver and the Windows XP build-in 1394 driver are employed to measure the data transmission throughput against packet size. Figure 7-1 illustrates the results.

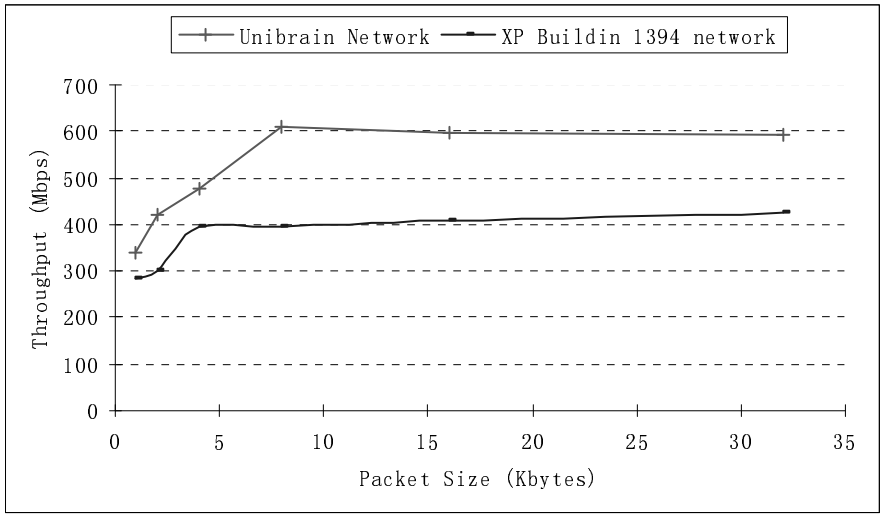


Figure 7-1: TCP performance over IEEE 1394

As documented in [81], only the asynchronous mode is use in the data transmission for IP-over-Firewire implementation. Figure 7-1 shows that using the asynchronous mode, the Windows XP build-in driver can sustain a throughput of 400Mbps, and the Unibrain IP-over-Firewire can sustain a throughput of up to 600 Mbps, which is a very high utilization rate of the IEEE 1394b bus (800 Mbps nominal bit rate).

While the IEEE 1394b device is excellent in performance, it is not expensive

in price. The IEEE 1394b PCI interface cards costs less than 100 USD, and per Mbps cost of it is as low as 0.12 USD/Mbps, which is far lower than other networking interfaces (Table 2-1). As the asynchronous mode can fully utilize the IEEE 1394b bandwidth, and support reliable data delivery, the IEEE 1394b device can be deployed as a generic network interface with fewer customizations than the IEEE 1394a device as what has been done in our communication protocol. Since the IEEE 1394b device supports data transmission speed of up to 800 Mbps, performance of computer clusters interconnected with it are expected to be comparable to that of the Gigabit Ethernet clusters or even the high-end Myrinet clusters, but for a much lower price. This situation makes the IEEE 1394 bus series a competing device for the computer cluster interconnection with impressive cost-effectiveness, and will stimulate the employment of parallel systems in various applications such as the transportation research and applications.

Features to enhance the IEEE 1394 device

Since the IEEE 1394 is originally designed for the interconnection of computer peripherals, not for the interconnection of network, it does not include features for networking. Such as no switching device is defined in the IEEE 1394 specification, and only the bus structure can be used in the network. This makes the IEEE 1394 devices less scalable for the interconnection of a large scale network. In this thesis we have discussed directly connected mesh structure for parallel

computing, but the number of interface card is increased as the square of the number of computing node. If we can use IEEE 1394 switch for the interconnection, only one interface card for each computing node is needed. So an IEEE 1394 switch device can both simplify the structure of the network, and reduce the cost of the network. Another important aspect is, in the bus structure, the bandwidth is shared by all nodes, if there are many nodes on the same bus, the performance will drops significantly. If an IEEE 1394 switch device which can support parallel transmission is available then the performance can be greatly improved. However, the cost of such a switch device could be quite expensive.

Another suggested change in the IEEE 1394 is an enhanced asynchronous mode that can fully utilize the capacity of IEEE 1394. As we observed in the thesis, the asynchronous mode transmission cannot fully utilize the capacity of IEEE 1394 device. If the asynchronous mode can fully utilize the capacity, it will be easier to implement the communication protocol over the IEEE 1394. By applying these changes, the IEEE 1394 will be even more effective as a generic network interconnection device, and will provide higher performance in a computer cluster.

Appendix A: The IEEE 1394 basic performance features

Table A-1: Latency and throughput vs. packet size of the asynchronous read transaction

Packet Size	Latency(us)	Throughput(Mbps)
8	50.861602	1.200024
16	51.272139	2.380831
32	52.356807	4.663016
64	54.954912	8.885125
128	59.216045	16.491518
256	62.525	31.237505
512	67.32085	58.024372
768	76.716006	76.377477
1024	83.939365	93.073137
1280	94.556592	103.278098
1536	104.268408	112.390226
1792	114.17584	119.744028
2048	125.025439	124.974566

Table A-2: Latency and throughput vs. packet size of the asynchronous write transaction

Packet Size	Latency(μs)	Throughput(Mbps)
8	50.776045	1.202046
16	52.636934	2.3191
32	50.950156	4.791754

64	56.195713	8.688941
128	57.586152	16.958287
256	62.513848	31.243078
512	62.965928	62.03752
768	75.679121	77.42393
1024	83.425527	93.646396
1280	95.567822	102.185283
1536	104.431104	112.315131
1792	113.312256	120.656631
2048	125.402773	124.59852

Table A-3: Latency and throughput vs. packet size of the roundtrip (asynchronous read & write) transaction

Packet Size	Latency(us)	Throughput(Mbps)
8	102.135303	1.195182
16	103.357529	2.362098
32	105.53709	4.626632
64	112.757471	8.660734
128	111.772178	17.474161
256	125.053398	31.236656
512	131.993857	59.18836
768	156.775547	74.748583
1024	184.710977	84.591616
1280	193.22415	101.080791
1536	214.941045	109.041528
1792	236.277871	115.727088

2048	250.195723	124.902215

Table A-4: Latency and throughput vs. packet size of the isochronous mode

Packet Size	Latency(us)	Throughput(Mbps)
8	124.9914	0.488315
128	125.1209	7.804951
1024	124.9682	62.51589
4096	124.9475	250.1051

Table A-5: Latency and throughput vs. packet size of mix modes

Packet Size	Asynchronous Latency(us)	Asynchronous Throughput	Isochronous mode Latency	Isochronous mode throughput
8	321.123232	0.190068	124.9192	250.1617
16	321.095986	0.380168	124.9507	250.0985
32	320.866328	0.76088	124.9383	250.1234
64	321.238525	1.519996	124.9284	250.1432
128	320.813857	3.044016	124.9521	250.0957
256	320.680781	6.090558	124.9452	250.1097
512	346.648291	11.268626	124.9669	250.0662
768	380.604082	15.394935	124.9474	250.1053
1024	414.22542	18.860504	124.9283	250.1435
1280	448.681699	21.765151	124.9579	250.0841
1536	482.849766	24.269971	124.9378	250.1245
1792	521.85377	26.198671	124.9571	250.0859
2048	556.010752	28.101975	124.9466	250.1069

Table A-6: Throughput of multiple asynchronous mode transmission at the same time

Packet size	Throughput			
	1 asynchronous	2 asynchronous	3 asynchronous	4 asynchronous
8	1.200024	2.4	3.6	4.8
16	2.380831	4.7	7.1	9.2
32	4.663016	9.3	13.8	18.5
64	8.885125	17.4	26	32
128	16.491518	30.8	40	57.9
256	31.237505	57	72	88.8
512	58.024372	85	102.8	116
768	76.377477	105.6	123.4	132.7
1024	93.073137	122.4	139	144.9
1280	103.278098	138.3	152	150.6
1536	112.390226	148.6	164	157.9
1792	119.744028	157.9	172.9	164
2048	124.974566	162	180	169.2

Table A-7: Latency of multiple asynchronous mode transmission at the same time

Packet size	Latency			
	1 asynchronous	2 asynchronous	3 asynchronous	4 asynchronous
8	50.861602	53. 16083395	53.33457639	53.60323122
16	51.272139	54.46808511	54.08450704	55.65217391
32	52.356807	55.05376344	55.65217391	55.35135135

64	54.954912	58.85057471	59.07692308	64
128	59.216045	66.49350649	76.8	70.74265976
256	62.525	71.85964912	85.33333333	92.25225225
512	67.32085	96.37647059	119.5330739	141.2413793
768	76.716006	116.3636364	149.3679092	185.1996986
1024	83.939365	133.8562092	176.8057554	226.142167
1280	94.556592	148.0838756	202.1052632	271.9787517
1536	104.268408	165.3835801	224.7804878	311.2856238
1792	114.17584	181.5832806	248.7449393	349.6585366
2048	125.025439	202.2716049	272.5333333	387.3286052

Table A-8: The latency results of the railway simulation test

Data size	IEEE 1394 Latency	Ethernet Latency
1	1.0725	0.155
2.5	1.12125	0.2975
2.5	1.12125	0.2975
2.5	1.12125	0.2975
5	1.2025	0.535
5	1.2025	0.535
4.5	1.18625	0.4875
4	1.17	0.44
3	1.1375	0.345
8	1.3	0.82
7.5	1.28375	0.7725
7	1.2675	0.725
6.5	1.25125	0.6775
9	1.3325	0.915

11.5	1.41375	1.1525
13	1.4625	1.295
15.5	1.54375	1.5325
18.5	1.64125	1.8175
18.5	1.64125	1.8175
18	1.625	1.77
17.5	1.60875	1.7225
17	1.5925	1.675
20	1.69	1.96
19.5	1.67375	1.9125
19	1.6575	1.865
18	1.625	1.77
21	1.7225	2.055
24	1.82	2.34
27	1.9175	2.625
26	1.885	2.53
29	1.9825	2.815
30	2.015	2.91
33	2.1125	3.195
38	2.275	3.67
41	2.3725	3.955
46	2.535	4.43
45	2.5025	4.335
44	2.47	4.24
43	2.4375	4.145
48	2.6	4.62
47.5	2.58375	4.5725
46.5	2.55125	4.4775
47	2.5675	4.525

Appendix B: IEEE 1394 architecture

Communication model

The protocol layers are defined to simplify the implementation of hardware and software. Each layer has an associated set of services defined to support communications between the application and the 1394 protocol layers, and for configuration and bus management.

The protocol consists of the:

- Bus Management layer – supports bus configuration and management activities for each node.
- Transaction layer – supports the CSR architecture request-response protocol for read, write, and lock operations related to asynchronous transfers. Note that a transaction layer exists in both the requester and responder. Note also that the transaction layer does not provide any services for isochronous transfers. Instead, isochronous transfers are driven directly by the application.
- Link layer – provides the translation of a transaction layer request or response into a corresponding packet, or subaction, to be delivered over the serial bus. This layer also provides address and channel number decoding for incoming asynchronous or isochronous packets. CRC error checking is also performed here.
- Physical layer – provide the electrical and mechanical interface re-

quired for transmission and reception of data bits (packets) transferred across the serial bus. The physical layer also implements an arbitration process to ensure that only one node at a time transfers data across the bus.

Each of the layers is described in more detail in the following sections.

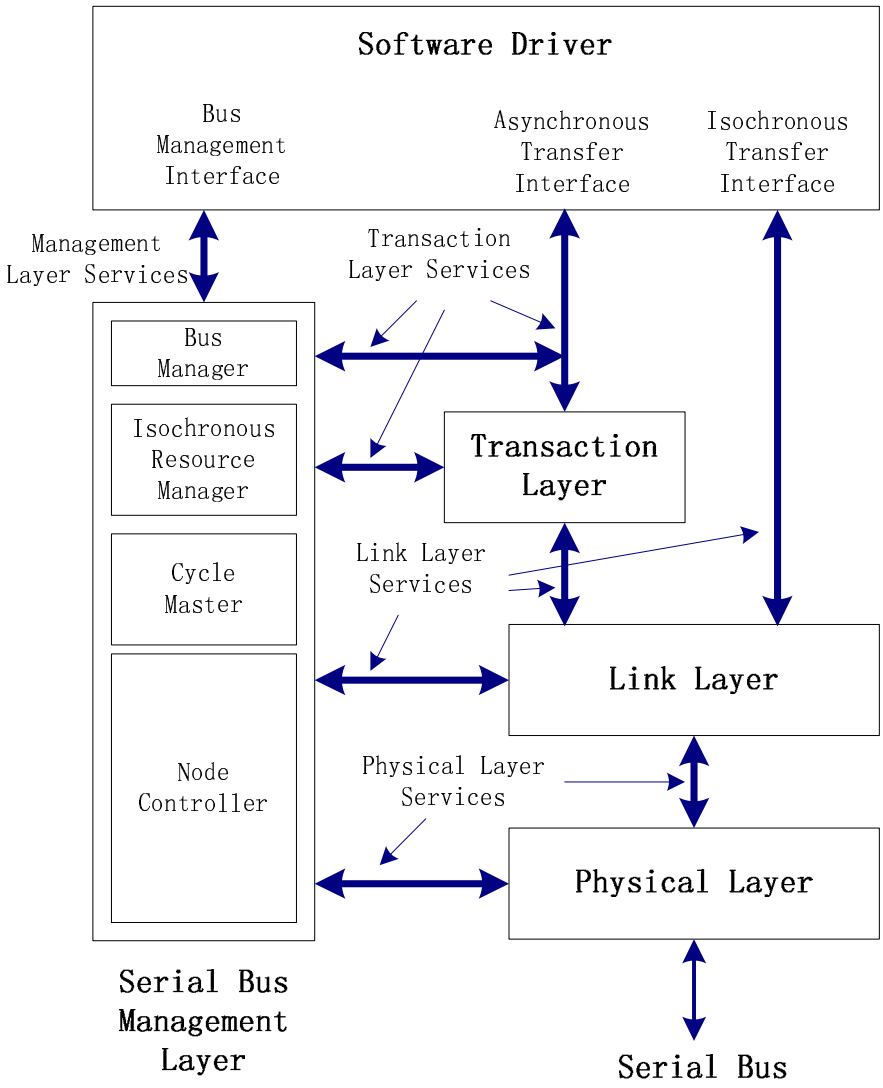


Figure B-1: The protocol layers

Bus management layer

Nodes implement the bus management layer to support a variety of functions including configuration and the application of power. The exact bus management support included depends on the capabilities of the node. All nodes must include support for automatic bus configuration, while other bus management functions are optional. For example, a given node may require power from the bus for its functional unit (e.g. video camera) and consequently will include bus management support for applying bus power.

Some nodes also participate in global bus management to ensure that the family of nodes residing on the bus live in harmony and can perform their functions efficiently. This global management consists of:

- Channel number and bus bandwidth allocation for isochronous transfers.
- Controlling the intervals at which isochronous transactions are performed.
- Verifying that all bus powered nodes have sufficient bus power.
- Tuning the bus to enhance performance (dependent on the bus topology).
- Providing services to other nodes (e.g. specifying the maximum speed at which two nodes can communicate with each other).

The 1394 specification identifies three global bus management roles that provide the support for a completely managed bus.

- Cycle Master
- Isochronous Resource Manager
- Bus Manager

Note that these roles may be performed by three separate nodes or one node may perform all three roles. Depending on the capabilities of the node residing on the bus, these roles may not be supported; thus, global bus management may be limited or may not occur at all.

Transaction layer

The transaction layer supports only asynchronous transfers. As discussed in the previous chapter, the 1394 bus supports three basic asynchronous transaction types:

- Read
- Write
- Lock

The asynchronous transaction model is based on communication between a requester node and response subaction, with the link and physical layers operation between the requester and responder transaction layer.

1394 applications typically have little knowledge of the intermediate layers within the 1394 communications model. Rather, they simply issue data transfer

requests the transaction layer. This software layer translates a transfer request into one or more transaction requests that are transaction type (read, write, or lock), and if the transaction consists of a write or lock the transaction layer also supplies data to be transferred during the request.

Note the transaction layer is not involved in isochronous transactions.

Transaction layer services

The transaction layer provides services related to transaction data flow. These service primitives are defined as:

- Request service – used by the requester to start a transaction (initiates the request subaction).
- Indication service – notifies the responder of the request (completes the request subaciton).
- Response service – used by the responder to return status or data to the requester (starts the response subaction).
- Confirmation service – notifies the requester that the response has been received (completes the response subaction).

Figure B-2 illustrates the transaction layer services, without regard to the intermediate layers. Note the transaction layer provides the interface between the application (function and the 1394 like layer).

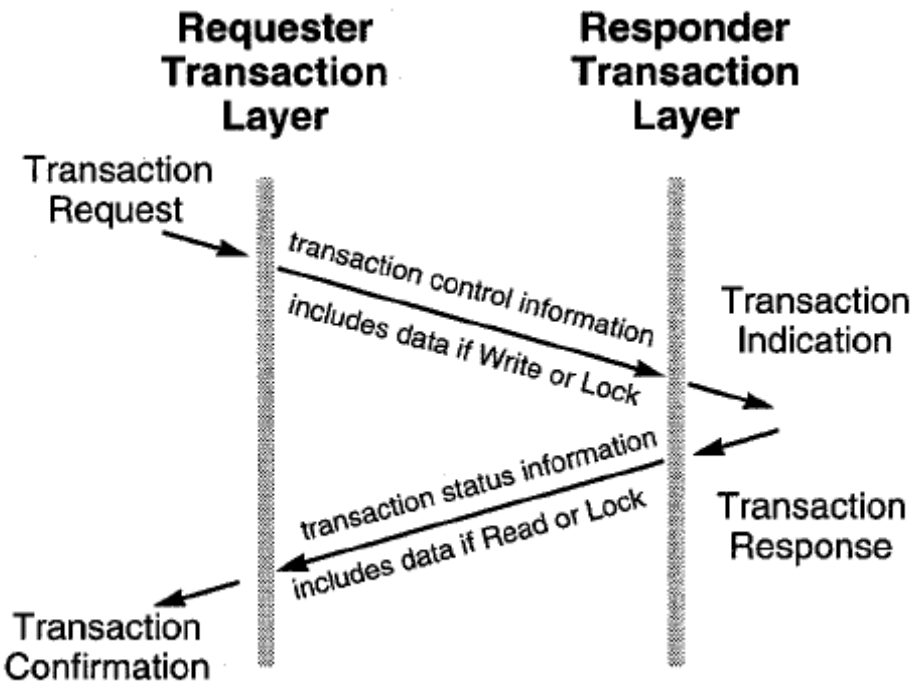


Figure B-2: Transaction layer Communication

The 1394 specification adds verification of packet delivery to the transaction protocol, which is not part of the CSR architecture model. The transaction layer supplies an acknowledgement for each packet transferred. That is, more packets transferred across the bus require that a 1-byte acknowledgment packet be returned to the sender to verify successful delivery of the packet. In the event of a failed transfer, retries can then be performed. The term "most" is used because two types of packets require no acknowledgement:

1. broadcast packet – the serial bus supports the broadcast of packets that may target more than one node on the bus. In this case, an acknowledge packet is not returned to avoid bus contention from multiple nodes simultaneously returning the acknowledgement.

2. Isochronous packets – delivery of isochronous packet requires a guaranteed transmission rate, thus any failed packet transmission cannot be retried because it might result in desynchronized data transfers. Therefore no acknowledgment needs to be sent because no corrective action can be taken in the event of failed packet transfer.

Link layer

For asynchronous transactions, the link layer provides the interface between the transaction layer and the physical layer and provides services based on the same request/response model used by the transaction layer. The requester's link layer translates transaction requests from the transaction layer into 1394 packets to be sent over the 1394 cable. When the packet is received by the responder, it is translated and forwarded on to its transaction layer.

For isochronous transactions the link layer provides the interface between the isochronous software driver and the physical layer. During transmission, the link layer creates the isochronous packets from the cable, decodes the packet's channel number and if the packet is destined for this node, the packet is forwarded to the software driver.

Appendix C: Fuzzy rules sets

Table C-1: Fuzzy rule set for identification of multimedia data flow

X_1	X_2	X_3	$X_{\setminus 4}$	X_5	result
S	S	NF	S	S	Yes
S	S	NF	S	B	Likely No
S	S	NF	L	S	Likely No
S	S	NF	L	B	No
S	S	F	S	S	Yes
S	S	F	S	B	Likely Yes
S	S	F	L	S	Likely No
S	S	F	L	B	No
S	B	NF	S	S	Likely No
S	B	NF	S	B	Likely No
S	B	NF	L	S	Likely No
S	B	NF	L	B	Likely No
S	B	F	S	S	No
S	B	F	S	B	No
S	B	F	L	S	No
S	B	F	L	B	No
B	S	NF	S	S	Yes
B	S	NF	S	B	Likely Yes
B	S	NF	L	S	Likely Yes
B	S	NF	L	B	Likely No
B	S	F	S	S	Likely Yes
B	S	F	S	B	Likely No
B	S	F	L	S	Likely Yes
B	S	F	L	B	No

B	B	NF	S	S	Likely No
B	B	NF	S	B	Likely No
B	B	NF	L	S	Likely No
B	B	NF	L	B	No
B	B	F	S	S	Likely Yes
B	B	F	S	B	Likely No
B	B	F	L	S	Likely No
B	B	F	L	B	No
VB	S	NF	S	S	Likely Yes
VB	S	NF	S	B	Likely No
VB	S	NF	L	S	No
VB	S	NF	L	B	No
VB	S	F	S	S	No
VB	S	F	S	B	No
VB	S	F	L	S	No
VB	S	F	L	B	No
VB	B	NF	S	S	No
VB	B	NF	S	B	No
VB	B	NF	L	S	No
VB	B	NF	L	B	No
VB	B	F	S	S	No
VB	B	F	S	B	No
VB	B	F	L	S	No
VB	B	F	L	B	No

References

[1] Abdelkhalek A., Bilas A. and Moshovos A., Behavior and Performance of Interactive Multi–player Game Servers, *Cluster Comp*uting, vol.6, no.4, pp.355-366, Oct 2003.

[2] Abdelkhalek A., Bilas A., Parallelization and Performance of Interactive Multiplayer Game Servers, *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, pp.72-82, Santa Fe, New Mexico, USA, 26-30 Apr, 2004.

[3] Anderson D., *FireWire System Architecture Second Edition IEEE 1394a*, MindShare, Inc. 1999.

[4] Ang A. H. S. and Tang W. H., *Probability Concepts in Engineering Planning and Designing Vol. II*, Rainbow Bridge, 1984.

[5] Ascia G., Catania V., Ficili G., Panno D., A fuzzy buffer management scheme for ATM and IP networks, *Proceedings of INFOCOM 2001*, pp.1539-1547, Anchorage, Alaska, 22-26 Apr, 2001.

[6] Bal H., Hofman R., Verstoep K., A Comparison of Three High Speed Networks for Parallel Cluster Computing, *Proceedings of the 1st International Workshop on Communication and Arch. Support for Network-Based Parallel Computing*, pp.184-197, Jun 1997.

[7] Bello L. L., Kaczyn´ski G. A., and Mirabella O., Improving the Real-Time Behavior of Ethernet Networks Using Traffic Smoothing, *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS*, vol.1, no.3, pp.151-161, Aug 2005.

[8] Boden N. J., Cohen D., Felderman R. E., Kulawik A. E., Seitz C. L., Seizovic J. N. and Su W. K., Myrinet – A Gigabit Per Second Local Area Network, *IEEE MICRO,* vol.15, no.1 pp.29-36, Feb 1995.

[9] Boszormenyi L., Holzl G. and Pirker E., Parallel Cluster Computing with IEEE 1394 – 1995, *Proceedings of the 4th International ACPC Conference Including Special Tracks on Parallel Numerics (ParNum'99) and Parallel Computing in Image Processing, Video Processing, and Multimedia*, pp.532-552, Salzburg, Austria, Feb 1999.

[10] Cameron G.D.B., Duncan C.I.D., PARAMICS – parallel microscopic simulation of road traffic, *Journal of Supercomputing*, vol.10, no.1, pp.25-53, Mar 1996.

[11] Caponetto R., Bello L.L., Mirabella O., Fuzzy Traffic Smoothing: Another Step towards Statistical Real-Time Communication over Ethernet Networks, *Proceedings of the 1st Intl Workshop on Real-Time LANs in the Internet Age*, pp.33-36, Vienna, Austria,18 Jun, 2002.

[12] Chen H. and Wyckoff P., Simulation studies of gigabit Ethernet versus Myrinet using real application cores, *Proceedings of CANPC'00, Workshop of High-Performance Computer Architecture*, pp.130-144, Toulouse, France, Jan 2000.

[13] Cheng R.G., Chang C. J., Design of a Fuzzy traffic controller for ATM networks, *IEEE/ACM Transactions on Networking (TON*), vol.4 no.3, pp.460-469, Jun 1999.

[14] Cetin N., Burri A., Nagel K., Parallel Queue Model Approach to Traffic Microsimulations, *Proceediengs of Swiss Transport Research Confernce*, Monte Verita, Switzerland, Mar 2002.

[15] Cetin N., Burri A., and Nagel K., A large-scale agent-based traffic microsimulation based on queue model, *Proceedings of the 3rd Swiss Transport Research Conference*, pp.42-52, Monte Verita, Switzerland, Mar 2003.

[16] Chrobok R., Wahle J., and Schreckenberg M., Traffic forecast using simulations of large scale networks, *Proceedings of the 4th International IEEE Conference an Intelligent Transportation Systems*, pp.434-439, 2001.

[17] Ciaccio G., Marco E. and Schnor B., Exploiting Gigabit Ethernet Capacity for Cluster Applications, *Proceedings of the 27th Annual Conference on Local Computer Networks (LCN' 02)*, 2002.

[18] Crovella M.E., *Performance prediction and tuning of parallel programs*, PhD thesis, University of Rochester, UK, 1994.

[19] David H. A., *Order Statistics, 2nd ed*. New York: Wiley, 1981.

[20] Douligeris C. and Develekos G., A fuzzy logic approach to congestion control in ATM networks, *Proceedings of the IEEE International Conference on Communications, ICC*, pp.1969-1973, 1995.

[21] DYNAMIT/MITSIM, Massachusetts Institute of Technology, http://its.mit.edu, Cambridge, MA, USA, 1999.

[22] Erlang A. K., Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges, *The Post Office Electrical Engineers' Journal* vol.10, pp.189-197, 1917.

[23] Federal Highway Administration, *Traffic Network Analysis with NETSIM – A User Guide*, Washington, DC, USA, 1980.

[24] Ferrer J., Barcelo J., AIMSUN2: advanced interactive microscopic simulator for urban and non-urban networks, Internal Report, Departemento de Estatdistica e Investigacion Operattiva, Faclutad de Informatica, Univeritat polittecnica de Catalynya, 1993.

[25] Freitas J. C., Crowley G., Space weather simulation on networks of workstations, *ASME FLUIDS ENG DIV PUBL FED*, vol. 250, pp.273-279, 1999.

[26] Gatner N. H. and Wilson N. H. M., editors. *Transportation and Traffic Theory*, Elsevier, New York, 1987.

[27] Gourgoulis A., Terstyansky G., Kacsuk P., Winter S., Creating Scalable Traffic Simulation on Clusters, *Proceedings of the 12th Euromicro Conference of Parallel, Distributed and Network-Based Processing (EUROMICRO-PDP)*, pp.60-65, 11-13 Feb 2004.

[28] Grove D.A., Coddington P.D., Communication Benchmarking and Performance Modeling of MPI Programs on Cluster Computers, *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, 26-30 April 2004, Santa Fe, New Mexico, USA. 2004.

[29] Hellendorn H., Thomas C., Defuzzification in Fuzzy Controllers, *Journal of Intelligent and Fuzzy System*, Vol. 1, pp.109-123, 1993.

[30] Hesheam E., Mostafa A., *Advanced Computer Architecture and Parallel Processing*, Hoboken, N.J., John Wiley, 2005.

[31] Hu R. Q., Petr D. W., A predictive Self-Tuning Fuzzy-logic feedback rate controller. *IEEE/ACM Transactions on Networking*, vol.8, no.6, pp.697-709, Dec 2000.

[32] Hyoudou K., Ozaki R. and Nakayama Y., A PC Cluster System Employing the IEEE 1394, *Proceedings of The 14th International Conference on Parallel and Distributed Computing and Systems*, pp.489-494, Cambridge, USA, 2002.

[33] IEEE 1394 for Linux, http://www.linux1394.org, 2003.

[34] IEEE Std. 802.3-2002, *LAN/MAN CSMA/CD Access Method*, 2002.

[35] IEEE Std. 1212-1994, *IEEE Standard for a control and Status Registers (CSR) Architecture for Microcomputer Buses*, 1994.

[36] IEEE Std. 1212-2001, *IEEE Standard for a control and Status Registers (CSR) Architecture for Microcomputer Buses (Revision of IEEE Std 1212-1994),* 2001.

[37] IEEE Std. 1394-1995, *IEEE Standard for a High Performance Serial Bus*, The Institute of Electrical and Electronics Engineers, 1996.

[38] IEEE Std. 1394a-2000, *IEEE Standard for a High Performance Serial Bus-Amendment 1*, 2000.

[39] IEEE Std. 1394b-2002, *1394b IEEE Standard for a High-Performance Serial Bus-Amendment 2*, 2002.

[40] Ipek E., Supionski B.R, Schulz M., and McKee S.A., An Approach to Performance Prediction for Parallel Applications, *Euro-Par 2005*, LNCS 3648, pp.196-205, 2005.

[41] ISO/IEC 7498-1:1994 *Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*, 1994.

[42] Jackson J. R., Networks of waiting lines, *Operations Research*, vol.5, no.4, pp.518-521, 1957.

[43] Jackson J. R., Jobshop-like queueing systems, *Management Science*, vol.10, no.1, pp.131-142, 1963.

[44] Jarvis S.A., Spooner D.P., Keung H.N.L.C., Cao J., Saini S., Nudd G.R., Performance prediction and its use in parallel and distributed computing systems, *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, 22-26 Apr, 2003.

[45] Jiang X., Yang Z.X., Du P., Miao J.R., Li H.Y., A research on an open architecture of integrated railway simulation system, *Proceedings of Traffic and Transportation Studies ICTTS 2002*, vol.2, pp.1355-1360, 2002.

[46] Jin H. and Yoo C., Latency Analysis of UDP and BPI on Myrinet, *Proceedings of the 18th IEEE International Performance, Computing, and Communication Conference (IPCCC'99)*, pp.185-191, Phoenix/Scottsdale, Arizona, USA, Feb 1999.

[47] Kang K., Kim C., Performance analysis of statistical multiplexing of heterogeneous discrete-time Markovian arrival process in an ATM network, *Computer Communications*, vol. 20, no.11, pp.970-978, 1997.

[48] Kevillem K.L., Tompkin R. IEEE 1394 and RFC 2734; a viable HSI for Hypercubes, *Proceedings of the 2001 IEEE International Conference on Cluster Computing (CLUSTER '01)*, pp.155-157, 2001.

[49] Knottenbelt W. J., *Parallel Performance analysis of large markov models*, PhD Thesis, Imperial College of Science, Technology and Medicine, University of London, UK, 1999.

[50] Kosonen I., *HUTSIM*, PhD Thesis, University of Helsinki, Finland, 1999.

[51] Kunz T. and Seuren M. F. H., Fast Detection of Communication Patterns in Distributed Executions, *Proceedings of the conference of the Centre for Advanced Studies on Collaborative research, IBM Centre for Advanced Studies Conference*, pp.9-269, 1997.

[52] Kurmann C., Rauch F., Stricker T.M. Cost/Performance Tradeoffs in Network Interconnects for Clusters of Commodity PCs, *Proceedings of Workshop on Communication Architecture for Clusters*, pp.196-198, 22Apr, 2003.

[53] Laamanen V., Lampinen T., Laurikkala M., Koivisto H., A Comparison of Fuzzy C-means Clustering and Rough Sets Based Classification in Network Data Analysis, *Proceedings of the 3rd WSEAS International Conference on Fuzzy Sets and Fuzzy Systems,* Interlaken, Switzerland, Feb 2002.

[54] Lee C., Jang J., Park E. K., Makki S., A simulation study of TCP performance over IEEE 1394 home networks, *Computer Communications*, vol.26, pp.670-678, 2003.

[55] Lee C., Jang J., Park E.K., Makki S., An Analysis of the Performance of TCP over IEEE 1394 Home Networks, *Proceedings of the Eight International Conference on Computer Communications and Networks*, pp.199-203, 1999.

[56] Lee Y. D., van de Liefvoort A., Wallace V.L., Modeling correlated traffic with a generalized IPP, *Performance Evaluation*, vol.40, no.1, pp.99-114, 2000.

[57] Li G. L., Li V. O. K., Networks of Queues Myth and Reality, *IEEE INFOCOM* 2003, pp.154-158, Dana Point, California, USA, 20-21 Oct, 2003.

[58] Li K.Q., Analyzing the Expected Execution Times of Parallel Programs, *Proceedings of the 1997 ACM symposium on Applied computing*, pp.488-495, San Jose, California, USA, 1997.

[59] Lim H., Park D., Kang S. and Oh B., Priority Queue-Based IEEE1394 Device Driver Supporting Real-time Characteristics, *IEEE TRANSACTIONS ON CONSUMER ELECTRONICS*, vol.46, no.3, pp.825-833, Aug 2000.

[60] Mache J., An Assessment of Gigabit Ethernet as Cluster Interconnect. *Proceedings of The 1st International Workshop on Cluster Computing (IWCC '99)*, pp.36-42, 2-3 Dec 1999.

[61] Madala S. and Sinclair J.B., Performance of Synchronous Parallel algorithms with Regular Structures, *IEEE Transaction on Parallel and Distributed Systems*, vol.2, no.1, pp.105-116, Jan 1991.

[62] Mahmassani H.S., Jayakrishnan R. and Herman R., Microscopic Simulation of Traffic in Networks: Supercomputer Experience, *American Society of Civil Engineers (ASCE) Journal of Computing in Civil Engineering*, vol.4, no.1, pp.1-19, 1990.

[63] Mouskos K. and Mahmassani H.S., Guidelines and Computational Results for Vector Processing of Network Assignment Codes on Supercomputers, *Transportation Research Record* 1251, pp.10-16, 1989.

[64] Myricom Inc., http://www.myrinet.com, 2005.

[65] Nagel K. and Rickert M., Parallel implementation of the TRANSIMS micro-simulation, *Parallel Computing*, vol.27, pp.1611-1639, 2001.

[66] Nagel K. and Schleicher A., Microscopic traffic modeling on parallel high performance computers, *Parallel Computing*, vol.20, pp.125-146, 1994.

[67] Nagel K., Rickert M., Rrye R., Stretz P., Simon P., Jacob R., Barrett C. L., Regional Transportation Simulations, *Proceedings of Advanced Simulation Technologies Conference*, Boston, MA, USA, 5-9 Apr 1998.

[68] Nakajo H., Ichikawa A., and Kaneda Y., A Distributed Shared-Memory System on a Workstation Cluster Using Fast Serial Links, *International Journal of Parallel Programming*, vol.28, no. 2, pp.179-194, Apr 2000.

[69] Norimatsu T., Takagi H., and Gail H.R., Performance Analysis of the IEEE 1394 Serial Bus, *Performance Evaluation*, vol.50, no.1, pp.1-26, Oct 2002.

[70] Norris R. C. and Miller D. M., Comparing the Performance of IP over Ethernet and IEEE 1394 on a Java Platform, *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp.481-484, Aug 2001.

[71] Osogami T. *Analysis of Multi-server Systems via Dimensionality Reduction of Markov Chains*, PhD Thesis, Carnegie Mellon University, Pittsburgh, USA, 2005.

[72] Ould-Khaoua M., Loucif S., Rabhi F.A., On the performance of multicomputer interconnection networks, *Journal of System Architecture*. vol.50, no.9, pp.563-574, Sep 2004.

[73] Park D. and Kang S., IEEE 1394 OHCI Device Driver Architecture for Guarantee Real-Time Requirement, *Proceedings of The 7th International Workshop on Real-Time Computing and Applications Symposium (RTCSA 2000),* pp.389-394, Cheju Island, South Korea, 12-14 Dec, 2000.

[74] Peeta S. and Zhang P.C., Architecture for Enabling Real-Time Traffic System Operations, *Computer-Aided Civil and Infrastructure Engineering*, vol.19, pp.306-323, 2004.

[75] Peterson G.D. and Chamberlain R.D., Beyond Execution Time: Expending the Use of Performance Models, *IEEE Concurrency*, vol.2, no.2, pp.37-49, 1994.

[76] Plaisant C., Tarnoff P., Keswani S., Saraf A. and Rose A., Understanding Transportation Management Systems Performance with a Simulation-Based Learning Environment, *Proceedings of Conference on Intelligent Transportation Systems' 99*, Washington D.C., USA, 1999.

[77] Qin X., Jiang H., Zhu Y., Swanson D. R., Towards Load Balancing Support for I/O-Intensive Parallel Jobs in a Cluster of Workstations, *Proceedings of the IEEE International Conference on Cluster Computing*, pp.100-107, Dec 2003.

[78] Rakha H.A., Aerde M.W., Comparison of simulation modules of TRANSYT and INTEGRATION modules, *Transportation Research Records*, no.1566, pp.1-7, 1996.

[79] Rathi A.K., Santiago A., The new NETSIM simulation program, *Traffic Engineering and Control*, pp.317-320, 1990.

[80] Rea S., Pesch D., Multi-Metric Routing Decisions for Ad Hoc Networks using Fuzzy Logic, *Proceedings of the 1st International Symposium on Wireless Communication Systems,* pp.403-407, 20-22 Sep 2004.

[81] RFC 2734 - *IPv4 over IEEE 1394*, 1999.

[82] RFC: 793 Transmission Control Protocol http://www.ietf.org/rfc/rfc0793.txt, 2002.

[83] Rickert M., *Traffic simulation on distributed memory computers*, PhD thesis, University of Cologne, Germany,1997.

[84] Robertazzi T.G., *Computer Networks and Systems, Queuing Theory and Performance Evaluation*, Second Edition, Springer-Verlag. 1994.

[85] Roosta S. H., *Parallel Processing and Parallel Algorithms*, Springer, 1999.

[86] Santamaria R., IEEE-1394: A Standard for the Next Millenium, *Proceedings of The 18th Digital Avionics Systems Conference*, vol.1, pp.1-7, St. Louis, MO, USA, 24-29 Oct, 1999.

[87] Schwartz M., *Broadband Integrated Networks*. Prentice Hall PTR, 1996.

[88] Stathopoulos A., Long-Term Travel Demand Forecasting in General Dynamic Transportation Networks, *Transportation Research Board Annual Meeting*, 2003.

[89] Steinberg D., Birk Y., An Empirical Analysis of the IEEE-1394 Serial Bus Protocol, *IEEE Micro*, vol.20, no.1, January, 2000.

[90] The Message Passing Interface (MPI) standard, http://www-unix.mcs.anl.gov/mpi/, 2005.

[91] The Promoters of the 1394 Open HCI, *1394 Open Host Controller Interface Specification, Release 1.1*. 2000.

[92] TRANSIMS – transportation analysis and simulation system, http://transims.tsasa.lanl.gov/, 2005.

[93] Truong H. L. and Fahringer T., Soft computing approach to performance analysis of parallel and distributed programs, *Proceedings of 11th International Euro-Par Conference (Euro-Par 2005)*, LNCS 3648, pp.50-60, Lisboa, Portugal, 30 Aug-2 Sep, 2005.

[94] Truong, H.L., *Novel techniques and methods for performance measurement, analysis and monitoring of cluster and gride application*, PhD Thesis, Vienna University of Technology, Austria, 2005.

[95] Unibrain, http://www.unibrain.com/, 2005.

[96] *Universal Serial Bus Specification, Revision 1.1*, Copyright © 1998, Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, NEC Corporation, 23 Sep, 1998.

[97] VISIM, Planung Transport uud Verkehr (PTV) GmbH, www.ptv.de, 2005.

[98] Vraalsen F., Aydt R.A., Mendes C.L., Reed D.A., Performance contracts: predicting and monitoring grid application behavior, *Proceedings of the 2nd International Workshop on Grid Computing*, GRID 2001, LNCS, vol.2242, pp.154-165. 2001.

[99] Walrand J., Varaiya P., *High-performance communication networks, 2nd Edition*, Morgan Kaufmann, 1996.

[100] Wellstead P.E. and Zarrop M.B., *Self-Tuning Systems: Control and Signal Processing*, New York: Wiley, 1991.

[101] Wilkinson B., Allen M., *Parallel programming Techniques and Applications Using Networked Workstations and Parallel computers, 2nd Edition*, Prentice Hall, 2005.

[102] Wilmarth T.L., Zheng G.B., Bohm E.J., Mehta Y., Choudhry N., Jagadishprasad P. and Kale L.V., Performance Prediction using Simulation of Large-scale Interconnection Networks in POSE, *Proceedings of Advanced and Distributed Simulation (PADS'05*), pp.109-118, 2005.

[103] Wolfram S., *Theory and Applications of Cellular Automata*, World Scientific, Singapore, 1986.

[104] Xu C.Z., Wang L.Y. and Fong N.T., Stochastic Prediction of Execution Time for Dynamic Bulk Synchronous Computations, *The Journal of Supercomputing*, vol.21, no.1, pp.91-103, 2002.

[105] Yang Z.X., Li H.Y., Jiang X., Yu L., Miao J.R. Du P., A study of railway transportation simulation system, *Proceedings of Traffic and Transportation Studies ICTTS 2002*, vol.2, pp.1385-1392, 2002.

[106] Yang Z. X., Jiang X., Yu Y., Tan L. G., Du P., Miao J. R., Simulation system of technological process at marshalling station, *Proceedings of the Conference on Traffic and Transportation Studies*, *ICTTS 2000*, pp.17-21, 2000.

[107] Yang X. Y., Yang W., Zeng M., and Shi Y., A Novel Network Traffic Analysis Method Based on Fuzzy Association Rules, *Proceedings of Modeling Decisions for Artificial Intelligence: First International Conference, MDAI 2004*, pp.81-91, Barcelona, Spain, 2-4 Aug, 2004.

[108] Yoshimoto H., Arita D. and Taniguchi R., Real-Time Image Processing on IEEE1394-based PC Cluster, *Proceedings of The 15th IEEE International Parallel and Distributed Processing Symposium*, pp.1177-1183, Apr 2001.

[109] Zadeh L.A., Fuzzy logic, neural networks, and soft computing, *Communications of the ACM*, vol.37, no.3, pp.77-84, 1994.

[110] Zadeh L.A., Fuzzy logic = Computing with Words. *IEEE Transactions on Fuzzy Systems* vol.4, no.2, pp.103-111, 1996.

[111] Zhu W., Lee D., Wang C., High Performance Communication Subsystem for Clustering Standard High-Volume Servers Using Gigabit Ethernet, *Proceedings of High Performance Computing in the Asia-Pacific Region*, vol.1, pp.184-190, Dec 2000.

[112] Zomaya Y. A., *Parallel & Distributed computing handbook*, McGraw-Hill, Inc. New York, NY, USA, 1996.