Design of Ethernet Based Real-time Distributed Systems

Jiang Lu

A Thesis

in

The Department

of

Mechanical and Industrial Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science at

Concordia University

Montreal, Quebec, Canada

January 2004

Your file Votre référence
ISBN: 0-612-91073-3
Our file Notre référence
ISBN: 0-612-91073-3

ABSTRACT

Design of Ethernet Based Real-time Distributed Systems

Jiang Lu

Real-time distributed control and simulation systems have broad applications in automotive, aerospace and industrial automation industries. Ethernet is the most popular network standard in the world because it is simple, fast, and cost-effective with it's rapid evolution of new technology. However, it does not support real-time communication. This thesis investigates the possibility of adding a higher-level TDMA protocol to Ethernet to provide real-time capability so that it can provide a cost-effective solution for real-time distributed systems. The thesis describes the design and implementation of an Ethernet based real-time distributed system framework including real-time driver development, a general testing and verification method, an accurate clock synchronization method and a TDMA protocol on Ethernet. The new approach is verified with experiments to demonstrate the performance of Ethernet based real-time distributed systems.

ACKNOWLEGEMENTS

This thesis is one of the most important milestones in my life, and it would be mission impossible without the people who gave me their support and trust. I sincerely extend my gratitude to my research advisor Professor B.W. Gordon of the Mechanical Engineering Department at Concordia University. His trust, scientific excitement, and vision inspired me. He gave me his guidance, support, and patience while I completed this thesis. I also give my thanks to Tianhe Wen and Gilles Huard for their support on test equipment, and Bin Liu for his precious help in correcting this thesis. Also, I would like to thank Qin; I could not have accomplished writing this thesis without her support and understanding.

Jiang Lu

Montreal, QC

Table of contents

List of figures

List of tables

Nomenclature

Tx	Sending
Rx	Receiving
N_{test}	Number of tests
T_{dclock}	Time delay of clock reading.
T_{dpio}	Time delay of port I/O
T_{dtx_p}	Parallel port Tx time delay
T_{drx_p}	Parallel port Rx time delay
T_{d_p}	$T_{d\_p} = T_{dtx\_p} + T_{drx\_p}$, Parallel port total time delay
T_{dtx_NIC}	NIC Tx propagation time delay
T_{drx_NIC}	NIC Rx propagation time delay
T_r	System resting time
$T_{composing}$	Packet composing time
T_{theory}	Theoretical transmit time
T_{dround_p}	Parallel port time round trip delay
T_{dtest}	Total time delay of the test
T_{dhs}	Propagation time delay of hub or switch
P_{TDMA}	Period of a TDMA cycle
P_{syn_cycle}	Period of a synchronization cycle
$P_{\_syn}$	Period of a synchronization slot
P_{syn_server}	Period of a synchronization slot on server

P_{syn_client}	Period of a synchronization slot on client
$P_{communication}$	Period for TDMA communication slots
Drift	Clock drift time
T_{jitter}	Jitter time
T_{buffer}	Server synchronization buffer time
$T_{syn\_Tx\_min}$	Server sync packet sending time minimum
$T_{syn\_Tx\_max}$	Server sync packet sending time Maximum
T_{syn_start}	Synchronization packet start time
T_{syn_end}	Synchronization packet end time
T_{buffer_client}	Client synchronization buffer time
$T_{syn\_Rx\_min}$	Server sync packet receiving time minimum
$T_{syn\_Rx\_max}$	Server sync packet receiving time maximum
$T_{difference}$	Difference of time
T	Sampling time
$T_{control}$	Control step time
f_{TDMA}	TDMA communication frequency
k	Spring stiffness coefficient
m	Mass
v	Speed
ω_d	Desired angular speed
ω	Angular speed

θ	Angular position
b	Damping coefficient
J	Inertia
τ	Torque
K_p	PID controller proportional gain
K_i	PID controller integral gain
K_d	PID controller derivative gain

1 Introduction

1.1 Motivation

Networked control and simulation systems have become increasingly common over the last decade as network communication technology has become less expensive with higher levels of performance. Distributed systems have the potential for greater flexibility and higher performance over traditional centralized systems due to their inherent parallel and decentralized structure. In addition, distributed systems have the advantages of enhanced diagnostic and remote operation capability.

In industrial automation systems, point-to-point communication such as RS232 protocol, has been used for several decades. However, network communication has become more popular because of its flexibility, reliability, cost effectiveness, and higher data rate [1].

Networked automotive control systems often referred to as X-by-wire [2] systems, will be implemented by automotive manufactures over the next several years to provide the flexibility required for advanced automotive control systems. Figure 1-1 below shows a function block of an X-by-wire system including break-by-wire and steering-by-wire. All of the ECUs (Electrical Control Unit), sensors, and actuators are connected by a real-time data network [2].

In the simulation industry, high performance distributed computer clusters are often being used for complex large-scale simulation tasks instead of expensive centralized supercomputers [3]. In these applications, the time delay associated with the network communication is critical to simulation performance [4].

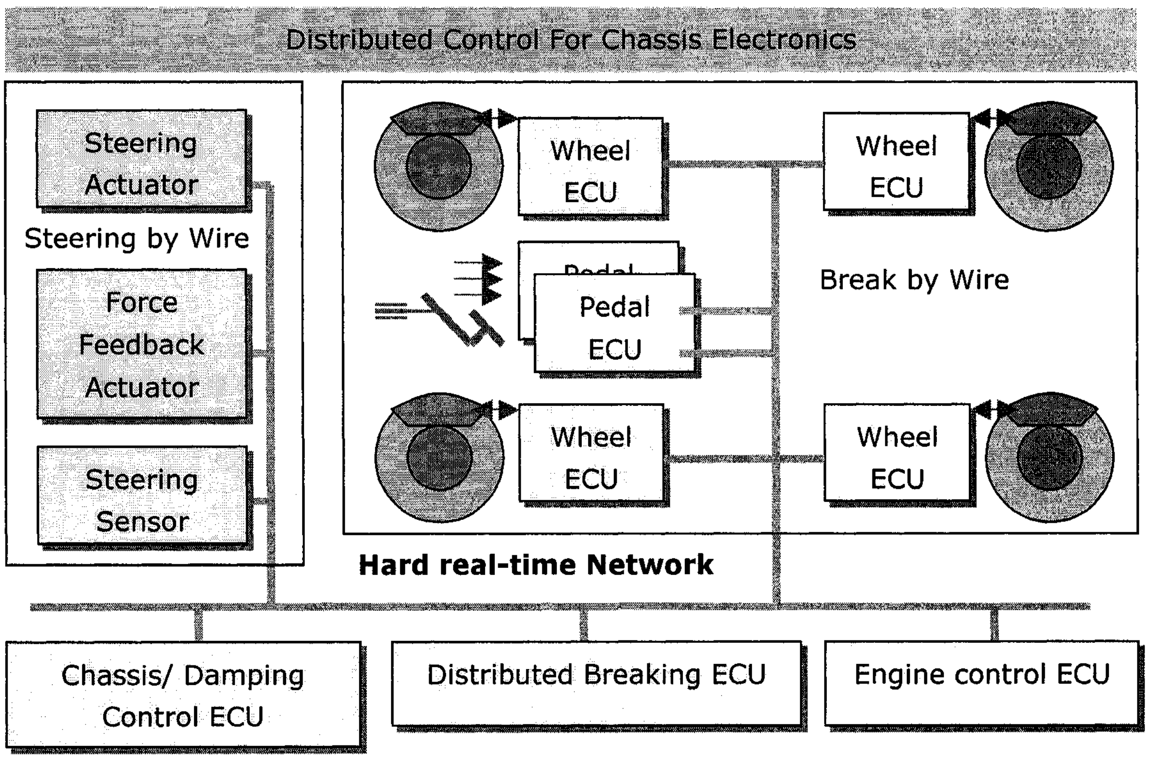


Figure 1-1 X-by-wire control system [2]

In recent years, Ethernet (IEEE803.2) [5] has become the most popular type of data network. Ethernet connects most of the networked computers in the world. The current Ethernet standard includes speeds of 10 Mbps (Ethernet), 100 Mbps (Fast Ethernet), 1000 Mbps (Gigabit Ethernet), and 10000 Mbps (10 Gigabit Ethernet).

Ethernet has many advantages over other network technologies. Compared with other control networks such as ControlNet (5Mbps) [6] and DeviceNet (0.5Mbps) [6], which operate at 5Mbps and 0.5Mbps respectively, Ethernet has 10/100/1000 Mbps data rates. Ethernet is also very inexpensive due to the economy of scale and production as a result of its widespread popularity. Fast Ethernet equipment is much less expensive, and Gigabit Ethernet, which has far superior performance, can be obtained for similar costs to other types of networks. Furthermore, Ethernet is easy to maintain and upgrade; adding and removing nodes from the network does not require an adjustment of the network. Equipment is also backward compatible so new equipment will work with older equipment at the older equipment data rate.

Distributed simulation and control systems also use some other types of networks such as CAN [1], Control Net [6], firewire [7], Myrinet [8], ATM [13]. However, none of these networks currently enjoy the same popularity, compatibility, and price/performance ratio as Ethernet (see Table 1-1). With all of these advantages, there is significant incentive for developing protocols that allow Ethernet networks to perform real-time communication for control and real-time simulation applications.

Table 1-1 Comparison of current network technology

Network type	**Cost per node**	**Data rate**
ControlNet	\$25	5 Mbps
Device Net	\$25	0.5 Mbps
ATM	\$500 ~ \$3,000	155 Mbps
Myrinet	\$2,000 ~ \$3,000	2,000 Mbps
Ethernet	\$20	10 Mbps
Fast Ethernet	\$20	100 Mbps
Gigabit Ethernet	\$120 ~ \$300	1,000 Mbps
10 Gigabit Ethernet	\$10,000 ~ \$20,000	10,000 Mbps

Real-time distributed simulation and control needs real-time communication. However, Ethernet is not a real-time network. At worst, Ethernet cannot guarantee the time delay. In other words, the time delay cannot be calculated theoretically in the worst case. Ethernet is not a deterministic system. For example, when the load is heavy on the network and one computer is sending a large amount of data to another computer, the network will be very active and most of the other computers will have less of an opportunity to access the network. Furthermore, collisions can occur and will increase delays and decrease data rate flow. For these reasons, Ethernet cannot be used for real-time communication without some type of modification. One of the main objectives of this thesis is to propose such a modification so that Ethernet can be used for distributed real-time simulation and control.

1.2 Problem Definition

A real-time network must guarantee the worst-case scenario time delay, so that it can be used in control. Control systems always need a deterministic time delay because time delay can change the system's dynamic characteristics. With a real-time network, the control system can be distributed in different locations. Sensors and actuators can be distant from the central controller; they use the network to send data and receive commands because the network is real-time, and data and commands will be received within a bounded time. Distributed real-time simulation is another important application of real-time networks. Stability and performance of real-time simulation also depends on having time delays that are bounded. This is especially important for hardware-in-the-loop (HIL) simulations since hardware can be damaged in an unstable simulation.

The main problem to address in this thesis is the development of a real-time higher-level protocol that works over the low-level Ethernet network interface card (NIC) to achieve real-time communication capability. This is accomplished by scheduling the communication so that only one node is transmitting to the network at a time. It will be shown experimentally in this thesis that deterministic response and bounded time delays can be achieved under this restriction. The main design criteria for the protocol will be to minimize network communication delays and protocol computation while maximizing data rate flow and reliability.

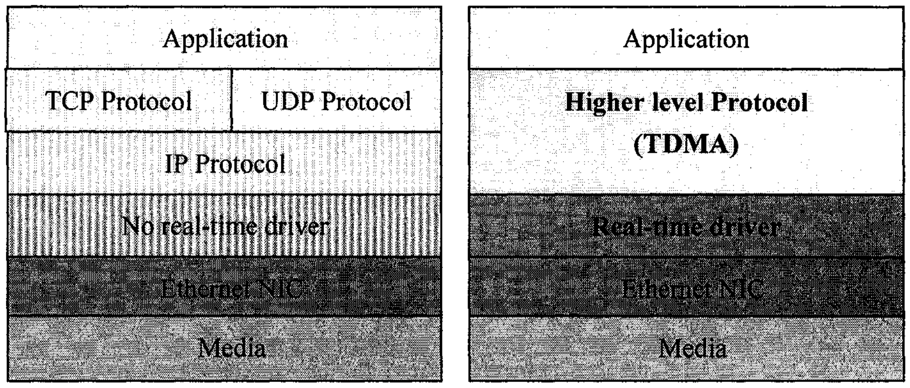


Figure 1-2 Comparison of TCP/IP and proposed real-time protocol

As shown in Figure 1-2, the higher-level protocol will use the Ethernet hardware (with its lower-level protocol) to send and receive information. If there is no request to send, the lower-level protocol will not send. By controlling the request sending time, higher-level protocol can control when the lower level will send and not send information. If the higher-level protocol is designed properly, the network will behave deterministically and have real-time capabilities.

There are many ways for a higher-level protocol to achieve this objective. TDMA (Time Division multiplexed Access) [9] based protocol is a good candidate because of its fairness, its simplicity, and because it is deterministic. It informs Ethernet when to send messages and when to receive messages. This TDMA based protocol cooperates with the other nodes to guarantee that when a message is going to be sent, there are no other nodes putting, or about to put messages on the network. Through an approved higher-level protocol, Ethernet will send a message on the network without danger of collision at anytime.

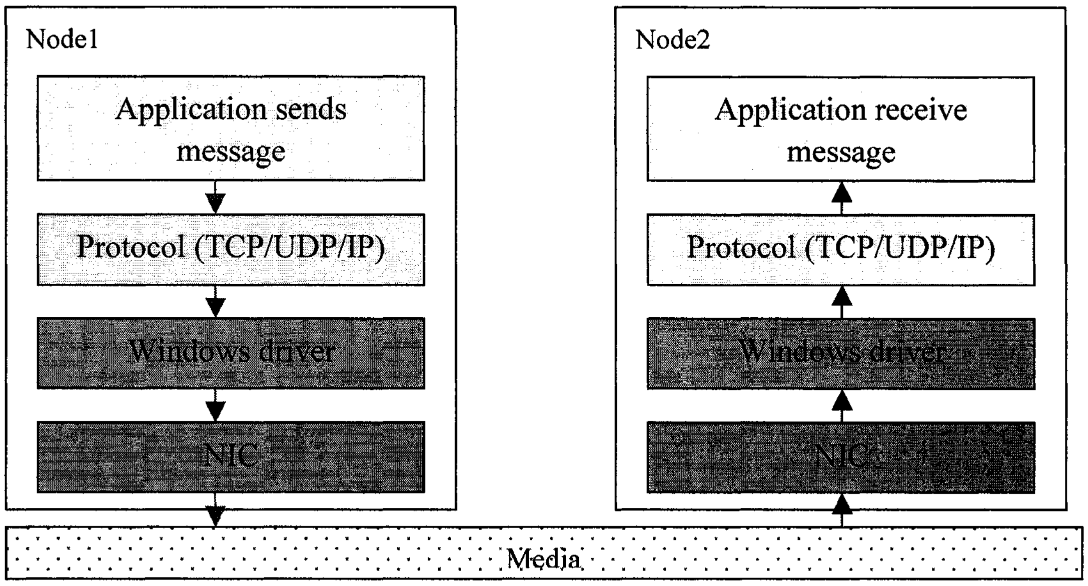


Figure 1-3 Information path from application in node 1 to application in node 2

Protocols such as TCP/IP protocol based on Ethernet have problems making a deterministic time delay because, first of all, there is a high possibility of collisions with Ethernet. If multiple nodes want to send a message at the same time, all of the packets are put on the network at once. The result is that all of the packets collide and none are sent out, and the same situation can occur again and again. In fact, this situation can easily happen when the network is heavily loaded. By this method, a deterministic time delay for a packet transport is impossible. Secondly, the general protocols access Ethernet through a non-real-time operating system such as Windows 2000, which uses the Ethernet driver to access the network as shown in Figure 1-3. During this process, any other process such as mouse movement can interrupt it and cause time delay. When the message arrives at the destination NIC, it needs to go through the whole process again, but in the opposite direction from when the destination application had received it. All of these layers are non deterministic and the time delay for a packet transfer inside the computer is also non-deterministic. The solution for these two problems will be discussed in this thesis.

1.3 Organization of Thesis

This thesis is organized into 8 chapters. Chapter 1 describes the motivation for the Ethernet based real-time distributed system and the problems that need to be solved. Chapter 2 introduces Ethernet's background and its basic concepts. Chapter 3 compares the existing solutions and briefly introduces the Ethernet based approach. Chapter 4 introduces the RTX real-time operating system, and focuses on real-time Ethernet NIC driver development process. Because of the difference in operating systems and application, this device driver is different from the Windows driver and Linux driver. Chapter 5 describes a method of time delay measurement and the results of testing of the parallel port, NICs, real-time drivers, hubs and switches. Chapter 6 deals with clock synchronization and TDMA protocol development. Chapter 7 presents a distributed simulation application and a distributed control application using the Ethernet based real-time distributed system. Chapter 8 concludes the thesis.

2 Ethernet Networks

Ethernet is a Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol. As illustrated in the Open System Interconnect (OSI) Reference Model [10], Ethernet works on the physical layer and data link layer, which are the base of the network communication.

Bus topology and star topology are the two most popular types of Ethernet connections. Ethernet hardware includes NICs, hubs and switches. A shared media uses a hub to connect the network, while a switched media uses a switch to connect the network. Hubs broadcast messages to all of the stations in a half-duplex mode, while switches create private connections between the two parts of the communication in a full-duplex mode. Switches deliver packets through a dedicated channel thereby creating less collisions but having more of a time delay than with a Hub.

The reason why Ethernet has gained such tremendous popularity is not only because of its simplicity, but also for its high data transfer rate and economy of scale. The drawback of Ethernet is that it is not designed for real-time application. Both shared and switched media have opportunities to collide, drop packets, and have non-deterministic delays.

2.1 Overview of Ethernet

Ethernet is a CSMA/CD protocol. The mechanism of the CSMA/CD is that before a node transmits a message to the network, it listens to the network. If the network is idle, the node transmits the message. Then it listens to the network for a collision signal. If there are no other nodes attempting to transmit at the same time, the sending node will continually transfer the packet to the network, and the network will be busy for a while. If this transmission is unlucky, there may be one or more nodes trying to transmit packets at the same time and a collision will occur. The sending nodes will hear a collision signal. After hearing a collision signal, the node uses a back off algorithm to wait for a while and then tries again. Dr. Norman Abramson and his colleagues at the University of Hawaii invented this concept during the early 1970's. The real development of Ethernet began at the Xerox Palo Alto Research Centre (PARC). Then DEC, Intel and Xerox worked together to publish Ethernet version 1. In 1982 they introduced version 2, which is the basis of the IEEE 802.3 CSMA/CD standard. For more detailed material on this subject, please refer to "Ethernet Networks: Design, Implementation, Operation and Management" written by Held, G. [24].

Ethernet can be described according to the Open System Interconnect (OSI) Reference Model [10]. In the OSI Reference Model, application software works on layers 5, 6, and 7. TCP/UDP works on layer 4, IP works on layer 3, and Ethernet works on layer 2 and layer 1. Each layer is relatively independent from the other layers so that developers and manufacturers can develop and manufacture different compatible hardware and software.

Figure 2-1 shows the OSI model.

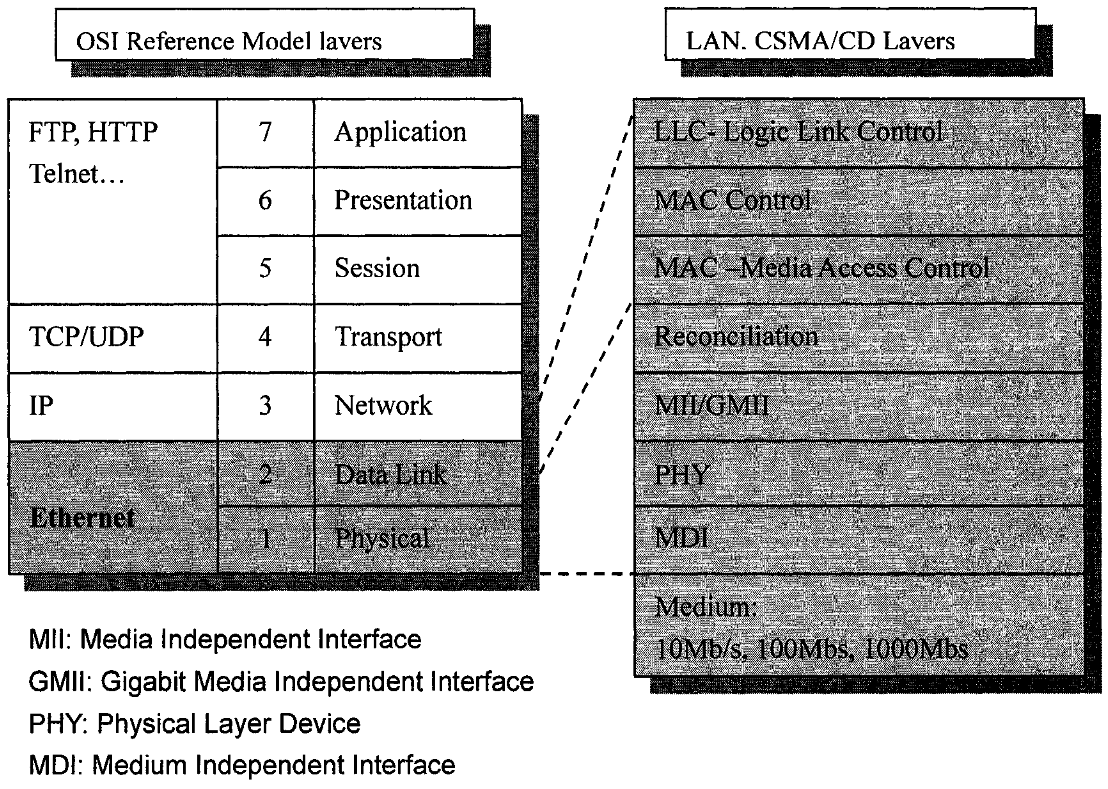

OSI Reference Model lavers
LAN. CSMA/CD Lavers
FTP, HTTP Telnet...
7 Application
6 Presentation
5 Session
TCP/UDP
4 Transport
IP
3 Network
Ethernet
2 Data Link
1 Physical
LLC- Logic Link Control
MAC Control
MAC –Media Access Control
Reconciliation
MII/GMII
PHY
MDI
Medium: 10Mb/s, 100Mbs, 1000Mbs
MII: Media Independent Interface
GMII: Gigabit Media Independent Interface
PHY: Physical Layer Device
MDI: Medium Independent Interface

Figure 2-1 OSI reference model and Ethernet layers

Ethernet works on the physical layer and data link layer, which is the base of the network communication system. The function of the data link layer is to ensure accurate communication between two nodes in a network. This layer contains formats of frames, error checking, and flow control. The data link layer is divided into a logic line control (LLC) layer and a media access control (MAC) layer. LLC is in charge of logic links between nodes in a network, while MAC controls access of transmission medium.

The most popular Ethernet connections are the bus topology and star topology. These two topologies are briefly introduced here. If more information is desired, please refer to "Ethernet Networks: Design, Implementation, Operation and Management" written by Held, G. [24]. Figure 2-2 shows a common bus topology where all of the devices communicate through the bus. A device senses the bus to check whether the bus is busy before it sends a signal. When two or more devices send a signal simultaneously, a collision occurs. [10]

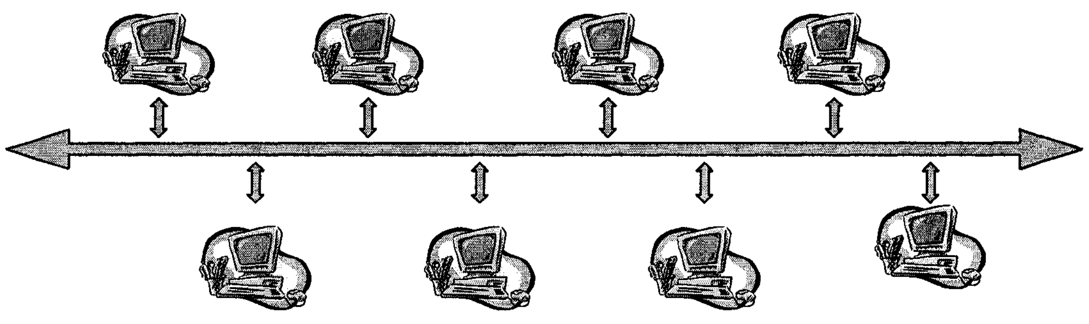


Figure 2-2 Bus topology

Star topology uses a central device that communicates to all other computers as a hub. Figure 2-3 shows the star topology. If a computer wants to send a message to other computers, it simply sends the message to the central device, and the central device will pass the message on to the receiver. In star topology, if the central device fails, the whole system goes down.

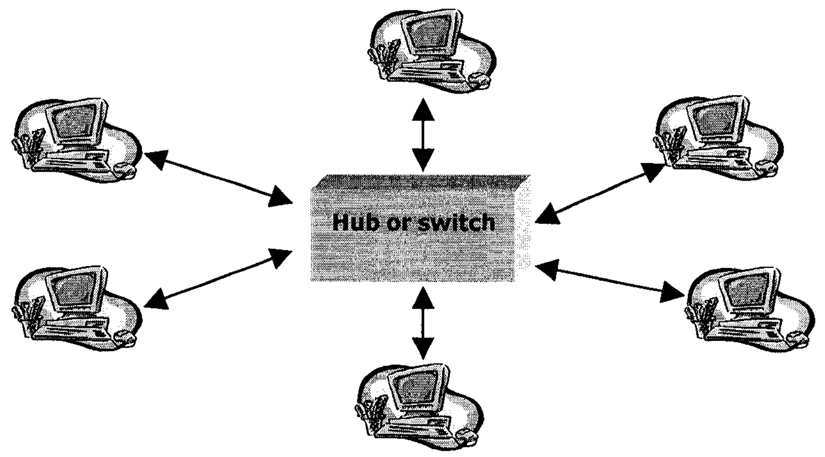


Figure 2-3 Star topology

Traditionally, Ethernet is connected as a bus topology by a coaxial cable. However, in recent years star topology has been used as the standard installation. In star topology, hubs broadcast packets, while switches transfer the packet only to the destination port.

Number of the octets

7	1	2 or 6	2 or 6	2	46~1500	4
Preamble	Start of frame delimiter	Destination address	Source address	Data length or Data type	Data/ Pad	Frame check sequence

Figure 2-4 Ethernet frame

Note: An octet is really a byte, but since the LLC is a bit-oriented protocol, we can't call eight bits a byte. Instead we call them an octet.

An Ethernet packet contains more information than the content transferred. It has a header containing source information and the destination address from each layer; like a complicated letter address. Figure 2-4 shows the Ethernet header and Table 2-1 describes each of them.

Table 2-1 Description of Ethernet frame parts [10]

Name	Octets	Description
Preamble:	7	Is a seven-octet pattern composed by alternating 0s and 1s. Its function is synchronization.
Start of frame delimiter	1	This is a special pattern 10101011 that indicates the start of a frame.
Destination address	2~6	If the first bit is 0, this field specifies a specific station. If it is 1, the destination address is a group specified by the address. Each station's interface knows its group address and responds when it sees this address. If all bits are 1, the frame is broadcast to all stations.
Source address	2~6	Specifies where the frame comes from.
Data Length Field	2	Specifies the number of octets in the combined data and pad fields in IEEE803.2 standard. In Ethernet it is used as a data type field to indicate what kind of data is in the data field. This is one of the differences between IEEE803.2 and Ethernet.
Data field	46~1500	The data that needs to be sent.
Pad field	0	The Data field must be at least 46 octets. If there is not enough data, extra octets are added to the data to make it at least 46 octets.
Frame check sequence:	4	Error checking using 32-bit CRC.

In the data field, we have an upper and lower limit (46~1500). The upper limit is used to prevent one transmission from monopolizing the medium for too long. The lower limit is to make sure the collision detection works properly. Considering all the factors, the 802.3 standard defines a minimum frame length as 512 bits (64 octets). If all of the fields (except the pad) have the smallest number of octets possible, the total comes to 18 octets. The 46 pad octets make up the difference.

2.2 Ethernet Hardware Components

Ethernet hardware typically includes nodes, NICs, buses, hubs and switches. Nodes are the joints on the network and can be all kinds of devices such as printers, sensors, and Web cams. However, most of the joints are computers. They send and receive packets to and from the network. The Network Interface Card (NIC) sends and receives packets. The bus is a media connected to the NIC with a bus topology, which transmits the data from node to node. It could be a coaxial copper cable, twisted wires, or an optical cable.

Hubs connect nodes and have several ports connecting each node's NIC via cables. Some hubs are called intelligent, or manageable, and these ports can be configured, monitored, enabled, and disabled by a network operator through a hub management program. Basically, it reconditions the signals and repeats them to all of the ports. Hubs can be linked together to extend the network. Figure 2-5 shows the OSI model and Ethernet hardware.

OSI Model	Hardware
Application (Layer 7)	Gateway
Presentation (Layer 6)	
Session (Layer 5)	
Transport (Layer 4)	Switch
Network (Layer 3)	Router, Switch
Data Link (Layer 2)	Bridge, Switch, NIC
Physical (Layer 1)	Hub (repeater), NIC

Figure 2-5 Hardware on the OSI model

Layer 2 switches (The Data-Link Layer switches) read the physical destination address and establish a direct connection between the destination and source without impacting other nodes. Several connections can be established at the same time between multiple devices. There are also some layer 3 and layer 4 devices called switches, but these are out of the scope of this paper. It is only necessary to discuss layer 2 switches in this paper.

There are some differences between Hubs and switches and these are illustrated in Table 2-2. Because hubs broadcast messages, every station is capable of receiving all transmissions from all stations, but only in a half-duplex mode. Switches allow for a private connection between the two parts of a communication. Nodes other than these two will not hear the packets. A switched network may operate in full-duplex mode. Full-duplex switching enables traffic to be sent and received simultaneously. Switches reduce the message collision opportunities dramatically. On the other hand, hubs have fewer operations and are faster than the switches; they just receive, condition and resend the signals. Switches have to store, read the packet and sort the list of MAC addresses, and then send the packet to the right port. Also, the time delay on the hub is less than with a switch. If the traffic load is too heavy, switches will drop packets.

Table 2-2 Comparison of hubs and switches

	Hubs	**Switch**
Way of transfer	Broadcast message	Switch message to destination or multicast
Operation mode	Half duplex	Full duplex
Collision Opportunities	Large	Small
Latency	Short	Long
Drop packets	No	Yes
Operation	Receive signal, Recondition, Resend signal.	Received packets, Store packets, Read the destination address, Check the address list, Send the packets to the destination port.

2.3 Advantages and Disadvantages of Ethernet

Ethernet is a simple protocol; it has a large base of installation and it has continual development. It also has a very high data transfer rate. The highest, 10Gbps, is available now, and 1Gbps has become popular while 100Mbps is the standard for new computers.

The disadvantage of Ethernet is that it is not designed for real-time applications. Both shared and switched media have opportunities for collision and for dropping packets.

A shared media uses a hub to connect the network. Hubs broadcast packets to every port on it. The media works like a bus where every node knows what is on the network and every node hears the packets for the other nodes. The time delay in the hub is relatively small compared with the switched media. The NIC or higher-level protocol should choose which packet should be accepted and which should be discarded. Figure 2-6 shows shared media and switched media.

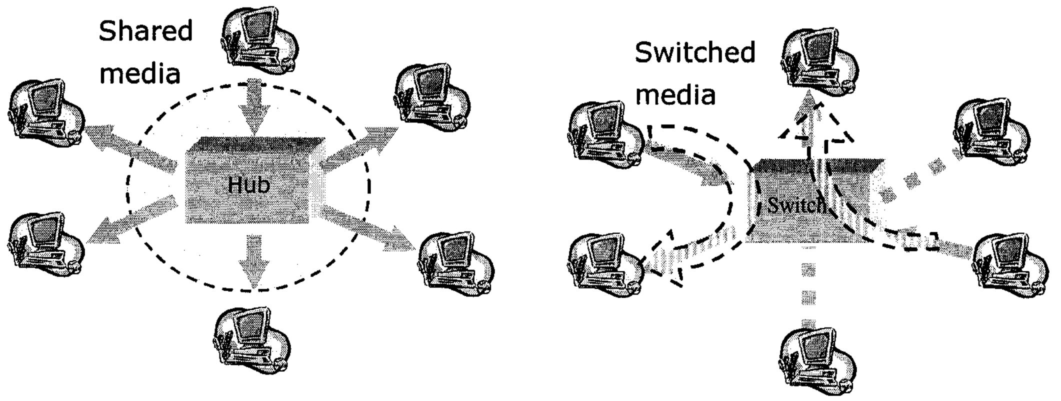


Figure 2-6 Shared and switched media

Switched media uses switches to connect the network. Switches read the destination address of the packets and send them to the corresponding ports. They give a private channel from the source node to the destination node. Other nodes cannot share this channel. However, other nodes can use other channels simultaneously.

Theoretically, a switch has more of a time delay than a hub. It delivers packets through a dedicated channel but there are still opportunities for collision. When the two nodes send packets to switches at the same time, and if the destination address is the same, a collision happens. Switches improve the general performance of Ethernet LAN; however, it cannot make Ethernet work as a real-time network. Switches must drop frames in cases of high loads. [11]

Hubs and switches both have to be implemented and tested in the project. For gigabit Ethernet, there is no gigabit Hub on the market. Compared with a hub, the switch's operation is more sophisticated and takes more time to process.

3 Real-time Ethernet Based Communication

Real-time Ethernet based communication is the goal of this thesis. In this chapter, the definition of real-time, alternative solutions, the advantage and disadvantage of existing solutions, and the new Ethernet based real-time communication approach is presented. It uses a higher-level protocol to control the time of sending and receiving on Ethernet and not only has the advantage of Ethernet's inexpensiveness and existing popularity, but also has deterministic performance.

3.1 Overview of Existing Approaches

The definition of real-time has two criteria. First, every command has to be executed successfully. Second, the result must arrive before the expected time limit.

Real-time in a network means that first, every packet must arrive at the intended destination, and second, every packet must arrive within a time limit. Generally, there are two kinds of real-time. The first is *hard real-time*, which means that if the real-time action fails, i.e. a message is missing or arrives late, it can cause a catastrophic or safety critical result. For example, an airplane control system is a hard real-time system. If the command sent by the control system goes missing or arrives late, the airplane could lose control and even crash. The other kind of real-time is *soft real-time*. When the processes are not completed, or are completed after the deadline, the system does not fail.

Even if the system does fail, there are no serious consequences. One example of *soft real-time* is a real-time game where even if the system fails, it does not cause critical safety concerns.

Network basically has two categories: one is a data network, and the other is a control network. The data network focuses on data transmission which normally needs a wide bandwidth, high speed, but not real-time. The control network focuses on control signal transmissions, which needs to be time deterministic, have high reliability, but generally have a narrow bandwidth and low transfer speed. Table 3-1 is a review of some of the popular networks and protocols [6] [12].

Except for the examples shown in the tables, some companies and organizations have started to work on proprietary software packages for distributed control and simulation. Venturcom developed the RT-TCP/IP which makes Ethernet and TCP/IP protocol real-time. However, even with the RT-TCP/IP, packet loss can still occur because of the Ethernet contention. Thus, it cannot be used in hard real-time applications. MPI-RT (Message Passing Interface / Real Time) is developed in MPI/RT forum [36] and provides a reliable real-time large data point-to-point channel. Since it is a point-to-point channel, it does not fit in our real-time distributed control and simulation system which generally needs more channels and more flexibility.

Table 3-1 Protocol comparison [12].

Protocol categories	Protocol Example	Advantage	Disadvantage
CSMA/CD	IEEE 802.3 (Ethernet), Bacnet, CAB	Small latency in low traffic load. No network initialization or configuration. Node can enter or leave the network without any interruption. Supports many nodes. Probabilistic global prioritization is possible. Extensive installed base and support.	It is designed for periodic traffic. Unbounded individual message latency. Poor efficiency under heavy loads. Collision detection is an analog process, which is not always practical.
Polling	MIL-STD-155 3B, 1773, Profibus, Bacnet, AN192	Bounded latency for real-time application. Simple protocol to implement. Very popular years ago.	Single point of failure from centralized master could make the entire system fail. Polling consumes bandwidth. Network size fixed during installation (not robust). Prioritization is local to each node.
Token Bus	IEEE802.4, PROFIBUS, Arcnet, AN192, MAP	Bounded latency for real-time control applications. High throughput during heavy traffic. On-the-fly reconfiguration.	Token passing latencies under light traffic conditions. Prioritization is local to each node. Reconfiguration process is lengthy. Token initialization, loss, and duplication recovery overhead is expensive. Collisions may occur during initialization and reconfiguration.

			Complex protocol especially at MAC sub layer.
Token Ring	IEEE802.5, TRON, FDDI	Bounded latency for real-time control applications. It has high throughput during heavy traffic. Global and local priority mechanisms are available. On-the-fly reconfiguration with node bypass hardware.	Moderate latency for light traffic (token passing overhead is relatively large. However it is better than the token bus). It needs a centralized monitor. Token initialization, loss and duplication recovery are all part of overhead. Propagation delay is based on the number of nodes. A cut in one wire disables the entire network.
CSMA/CA (Implicit Token)	Echelon	Small latency for light traffic. Good throughput under heavy traffic. Global prioritization is implemented through fixed slots. Bounded latency through rotating slots.	Restating time slots from an idle bus can be difficult (sending dummy messages to avoid idle state). Collisions can occur. Node complexity in mapping Sth slot to Nth node.
Binary Count Down (Bit Dominance)	CAN, SAE J1850	High throughput under light loads. Local and global prioritization are possible. Arbitration is part of the message – low overhead.	Propagation delay limits bus length. Unfair access – node with a high priority can "hog" the network. Poor latency for low priority nodes.
TDMA	Satellite Networks, DATAC, TTP, FlexRay	A simple protocol to implement. Deterministic response time. No wasted time for master polling messages.	Single point of failure from the bus master causes the whole network to fail. Wasted bandwidth when some nodes are idle. Stable clocks or clock synchronization. Network size is fixed during

			installation.

As we can see from Table 3-1, Ethernet is a CMSA/CD protocol. The drawback of Ethernet is the probability for collision. This collision could happen when two or more nodes send the packets to the media simultaneously, especially when the workload is heavy on the network and the collision probability increases dramatically [6]. In a switched network, when the workload is heavy, the switch may drop packets. For these reasons, Ethernet is a non-deterministic network.

However, if the collision can be avoided through upper level protocol, Ethernet can send packets very fast and deterministically. The idea is to organize the packet sending time on the network. When any node sends packets onto the network, all of the other nodes will not attempt to send their own packets until there is an available sending time. TDMA protocols have very deterministic response times and no collision. Use of Ethernet hardware to emulate a TDMA protocol will avoid the collision on Ethernet and achieve a deterministic response time. Because Ethernet works on the physical layer and data link layer, the new TDMA protocol can work on a network layer instead of a TCP/IP protocol.

The TDMA real-time Ethernet protocol has several advantages.

- A deterministic response time.

- High data rate.
- Various hardware available for use.
- A fair protocol; no nodes can monopolize the network.
- Low cost because the hardware is cheap.
- Great potential to upgrade to new Ethernet hardware i.e. 10gigabit Ethernet.

3.2 Proposed Approach

The TDMA real-time Ethernet is a collision free protocol based on fast Ethernet or gigabit Ethernet hardware. To achieve the deterministic performance, all of the components are real-time, including a real-time operating system, real-time protocol, and a real-time NIC driver. Also, the performance of the hardware has to be tested for its deterministic characteristics. Figure 3-1 shows the proposed structure. The operating system is an RTX system, which is a real-time extension of the Windows (NT, 2000, XP) system. Chapter 4 will introduce the operating system in detail.

The two main software components are a real-time Ethernet driver and real-time TDMA high-level protocol. Real-time TDMA high-level protocol runs over the real-time Ethernet driver and controls when the driver sends packets out on the media. Real-time TDMA protocol guarantees that when the driver attempts to put packets on the media, there are no other nodes sending packets and the media is idle. Because the TDMA protocol and driver are real-time, the time delay is deterministic.

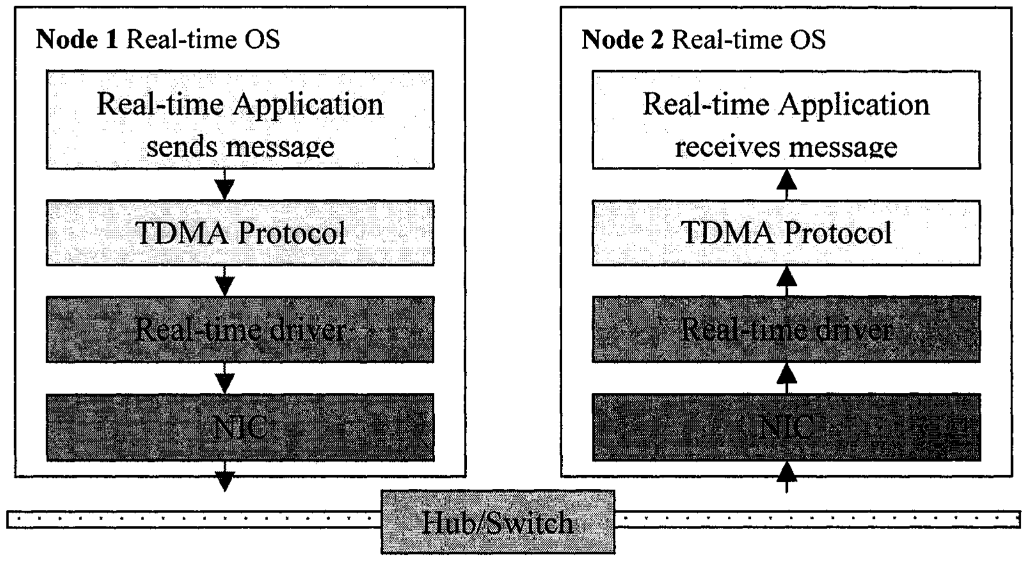


Figure 3-1 Information path from node 1 application to node 2 application

The hardware components include Ethernet NICs, Ethernet Hubs or Switches, Ethernet Category 5e cables and parallel port cables.

The goal of this project is to develop a deterministic Ethernet network. To achieve this goal, the following criteria must be met.

I) The NIC driver is deterministic.

II) TDMA protocol is deterministic.

III) As part of the proposal, using off-the-shelf NIC hardware without special modifications.

IV) Reliability:

A, time/number of packets before failure

B, fault tolerance

a) Ethernet collision

b) Missing node

c) Transmission out of turn/ overlap

d) Missing clock sync signal

e) Missing packet

f) Overloaded network, both overload of NIC and switch

Real-time Ethernet Driver

Real-time NIC drivers communicate between an application program and the NIC hardware. The driver receives the packets from the application and sends them to the network. It also receives the packets from the network and sends them to the application. All operations have to be real-time. Because most real-time operating systems do not have such drivers available, they have to be developed in this project.

Latency and Performance Testing

During driver and protocol development, all of the hardware and software components have to be tested for time delay and jitter, which can be used to choose the right hardware components and optimize software performance.

The test equipment has to be real-time and must have a higher accuracy, shorter response time, and more stable characteristics than the target system. For this reason, a parallel port is a good candidate. The hardware of a parallel port is simple and reliable and requires no extra data buffering. The time delay in a parallel port is stable and deterministic. The Tekronix TDS 200 real-time oscilloscope is the other hardware broadly used in the tests and has a 1GHz sampling-rate.

TDMA Communication Protocol

TDMA protocol is a protocol based on the time slot. Because clocks on different computers have drifts, all of the nodes have to be synchronized to one global clock. There are many ways to synchronize clocks. Two approaches have proved effective and function well in this thesis: first, use a square wave on the parallel port as a synchronized signal, and second, use a synchronization packet on the Ethernet Network as a synchronized signal to which every client is synchronized with. The following is the description of a clock synchronization algorithm.

A synchronization interval is the time from the raising edge of the synchronization signal to the next on a parallel port, or from receiving a synchronization packet to the next receiving on the Ethernet. At the beginning, all of the computers are waiting for a synchronization signal. Then every computer corrects its clock according this signal. Computers then send and receive by the clock time. When the slot time has arrived, the computer sends the packet. Just several microseconds before finishing this synchronization cycle, the computer starts to wait for the synchronization signal again. During the waiting time, computer is bind. Figure 3-2 shows TDMA protocol synchronized by a parallel port. Figure 3-3 shows TDMA protocol synchronized by packets. Figure 3-4 shows the timeline of TDMA protocol.

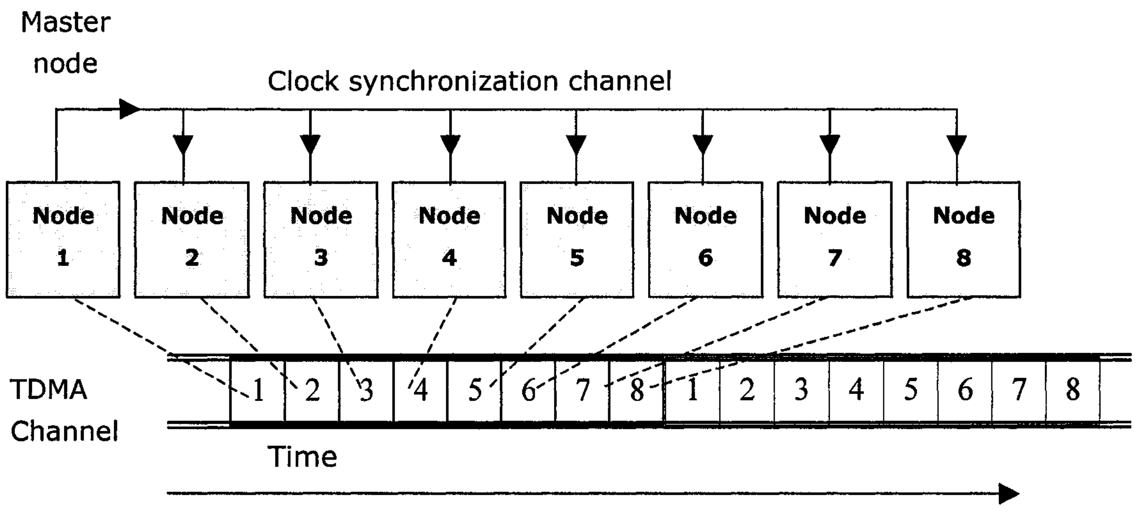


Figure 3-2 TDMA protocol synchronized by parallel port

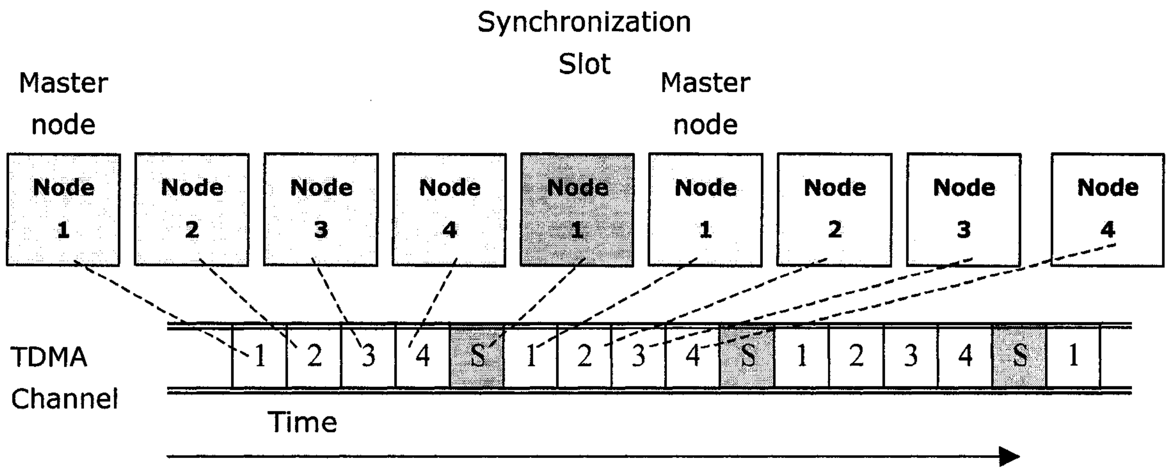


Figure 3-3 TDMA protocol synchronized by packets

Implementation and Testing

There are several design proposals for different applications.

A) Simple star topology with parallel port synchronization.

B) Simple star topology packet synchronization.

Design A and B are suitable for use in most simulation applications when there are no serious safety concerns if the network goes down. Design A, Figure 3-5, uses a separate synchronization channel to synchronize the clock. The network bandwidth is fully used for information transmitting. Design B, Figure 3-6, is very simple and has the same connection with the normal Ethernet network. However, it uses part of the network bandwidth to synchronize the clock.

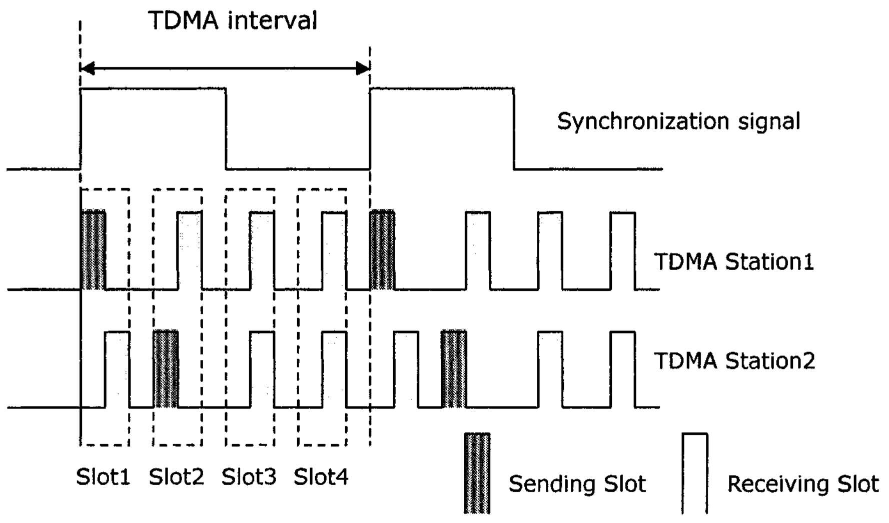


Figure 3-4 Clock synchronization signal and TDMA

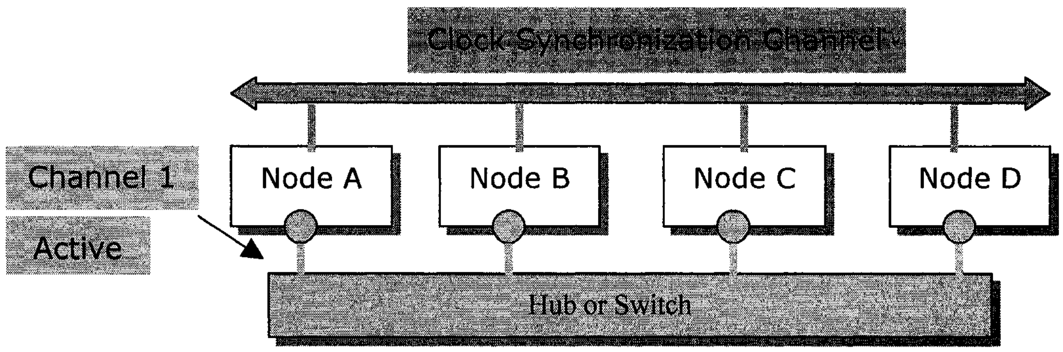


Figure 3-5 Design A - simple star topology synchronized by separate channel

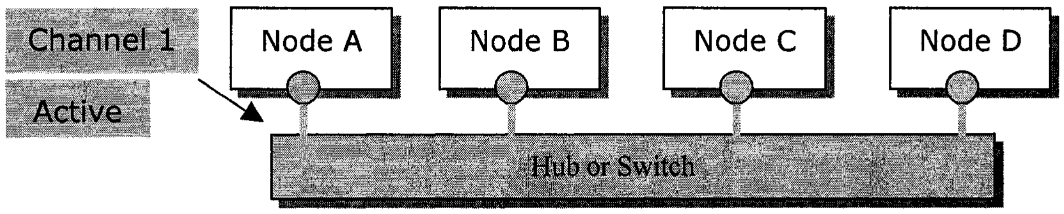


Figure 3-6 Design B - simple star topology synchronized by packet

4. Real-time Ethernet Driver Development

An NIC Driver is a program which communicates between the application (or operating system) and the NIC hardware. A real time driver works under a real-time operating system. In this chapter, the RTX operating system is introduced. The process and main functions of a real-time NIC driver are described. Simple functioning tests and debugging techniques are also illustrated. The results of the functioning tests demonstrate that the real-time NIC driver shows very good reliability and determinism. The intensive real-time testing will be taken up in Chapter 5.

4.1. Venturcom RTX Real-time Operating System

Venturcom developed the RTX (real-time Extension) operating system (OS). As its name implies, RTX is an extension of the Windows (NT, 2000, XP) operating system. RTX adds real-time capabilities to Windows NT, Windows 2000, and Windows XP and provides us with the use of a single, low-cost platform to satisfy a full range of real-time and embedded application requirements. [14]

RTX adds a real-time subsystem, named RTSS, to windows (NT, 2000, XP). RTSS supports its own environment and API, as with other Windows NT subsystems (such as

Win32, POSIX, WOW and DOS). However, instead of using Windows (NT, 2000, XP) scheduler, RTSS has real-time thread scheduling. In a uniprocessor environment, all RTSS thread scheduling occurs before any of the Windows scheduling, including Windows-managed interrupts and Deferred procedure Calls (DPCs). RTSS also supports Inter-Process Communication (IPC) that enables simple and standard communication and synchronization between real-time and non–real-time programs. In multiprocessor systems, RTX runs in one of the CPUs while others continue to run windows [14]. Figure 4-1 shows the structure of the RTX OS system.

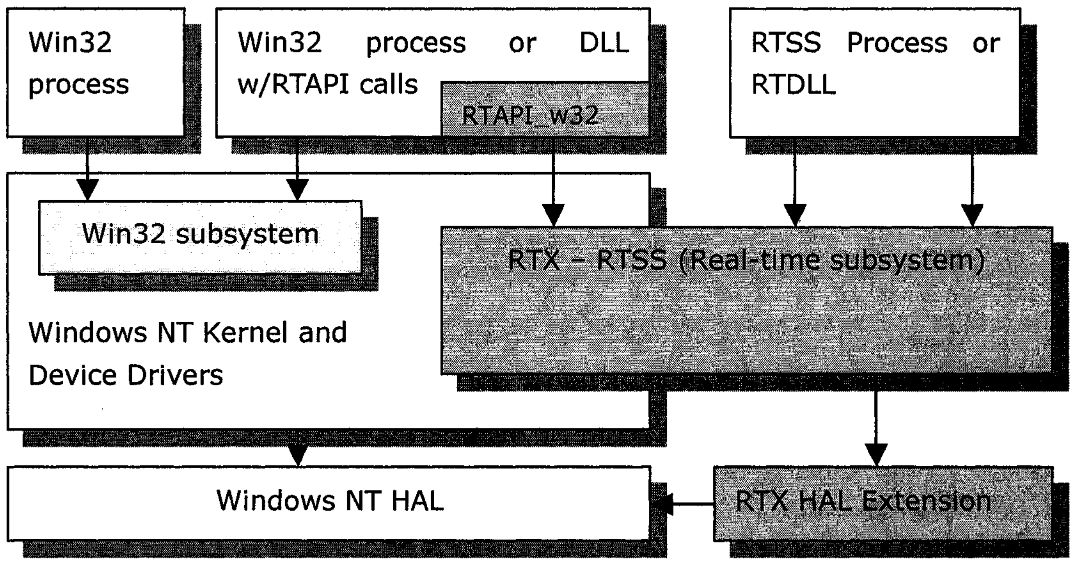

Win32 process
Win32 process or DLL w/RTAPI calls
RTAPI_w32
RTSS Process or RTDLL
Win32 subsystem
RTX - RTSS (Real-time subsystem)
Windows NT Kernel and Device Drivers
Windows NT HAL
RTX HAL Extension

Figure 4-1 RTX and Window OS structure [14]

RTX includes a real-time enabled Hardware Abstraction Layer extension. This extension maintains interrupt isolation between RTSS and Windows (NT, 2000, XP).

Windows interrupts are masked during RTSS processing. This means that Windows interrupt cannot interrupt the RTSS thread. [14]

Selecting an operating system is directly related to the application system design. In distributed real-time simulation and control systems, RTX exhibits great advantages because it has many popular application softwares and API runs in the Windows platform as Matlab [15], DirectX [16]...etc. However, Windows is originally designed for an office environment, not for real-time application. For example, mouse movement is designated with a very high priority designed to let people check whether or not the computer is working. On the other hand, there are a lot of real-time operating systems (RTOS) available on the market, such as VxWorks [17], QNX [18], ...etc. Linux [19] also has some versions of Real-time extensions. However, these other systems do not have the popularity or the number of applications that Windows has.

One of the advantages of RTX is that the user can use Windows (NT, 2000, XP) and RTX on one computer simultaneously. This gives users and designers the potential to use applications and development tools on Windows; meanwhile, there is still real-time communication on the network. The user can have a real-time communication by just installing a small piece of software. The developer can develop real-time applications with the real-time communication running with Windows. Table 4-1 shows the advantages and disadvantages of the RTX.

RTLinux has similar mechanisms as the RTX and Windows (NT, 2000, XP) combination. Linux has an advantage in that the developers have the source code of the OS. The difficulty is that until now, Linux has a small amount of installation compared with Windows (NT, 2000, XP). However, it still has great potential. We can develop the same real-time network communication program under real-time Linux in the future.

Table 4-1 Advantages and disadvantages of the RTX operating system

Advantages of RTX	Disadvantages of RTX
• Deterministic. • RTX can run with windows in one computer. • Easy to exchange data with windows. • Easy to develop. • It is able to directly control the hardware.	• The accuracy of the timer. • Booting computers still need windows.

4.2 NIC Device Driver Development

This section discusses the development of the real-time NIC driver. First of all, choosing a right NIC is critical for development. Performance, cost, and documentation are three main factors to consider when making this choice. In fast Ethernet sections, SMC 1211 series 10/100 mbps NICs have a stable performance and high throughput. They are based on the Realtek RTL8139 chip series produced by REALTEK

Semiconductor Corp [20]. Other big chip suppliers such as Intel [21], 3Com [22] also supply good chips and NICs. Cost is almost as important as the performance.

There are some difficult issues for a new NIC driver developer to consider. First is the hardware interface. Figure 4-2 shows the basic structure of RTL8139 NIC. This interface communicates between software and hardware. Device driver developers must know the functions of the NIC, related registers, and which resources will be used. The second difficulty is implementing the functions of the NIC in a specific operating system and includes how to manage the memory, interrupt and other resources. The third difficulty is testing the driver. The driver should be tested in all situations to make sure it works and then the bugs should be fixed and performance improved. Testing and repairing are often time-consuming tasks.

From my experience on this project, the NIC driver development has the following phases:

1 Initiation phases: detect the NIC and read the information, then initiate the NIC and buffers. This is the first phase; finding the proper NIC and its resources.

2 Sending phase: making the NIC begin working. Sending out the right information to the right destination. The buffer management is relatively simpler than the receiving.

3 Receive phase: receive packets from other nodes in the right sequence and find out

what is in the packet.

4 Testing phase: design experiment to test the reliability of the driver and measure the performance of the driver and hardware. After testing, bugs and problems are found, fixed, and then tested again.

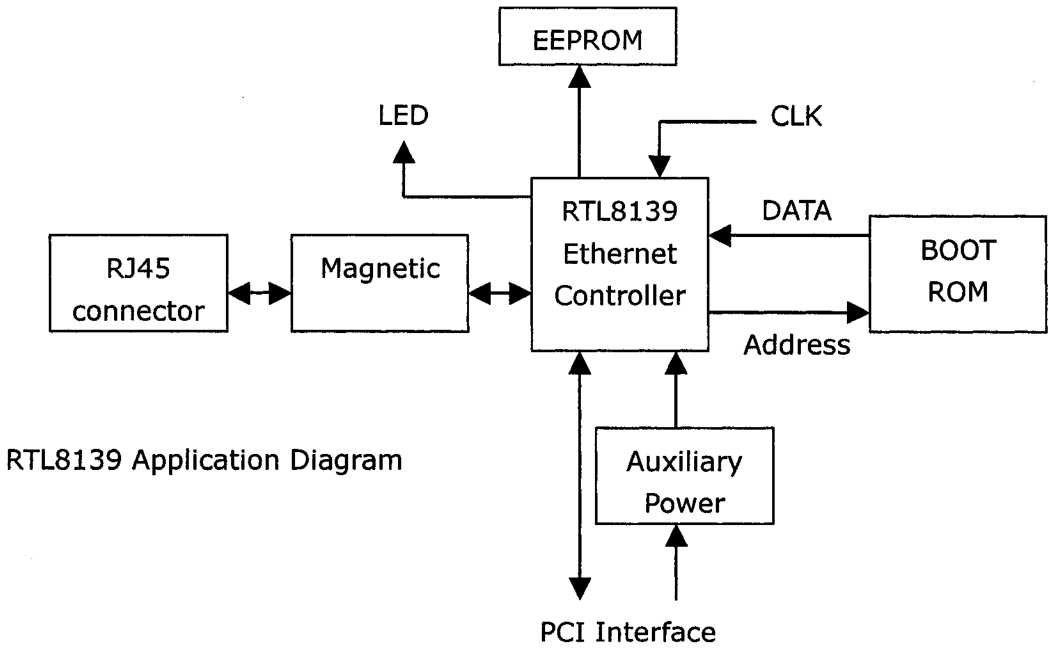


Figure 4-2 Basic structure of RTL8139 NIC [25]

4.3 Driver Organization

Simply put, the driver is a group of function modules, which allow application programs to communicate with and control the hardware. In this project, the NIC interface has four main functions: activate_NI, deactivate_NI, send2, and recv2. Some of the functions encapsulate other subroutines. The subroutine structure is shown in Table 4-2.

Function *activate_NI* detects NIC and initializes the NIC and buffers. Function *send2* composes and sends the packet. Function *recv2* checks whether the NIC receives the packet and delivers the packet. Function *deactivate_NI* closes the NIC and frees the buffer and other resources.

Table 4-2 device driver function modules

Function name	Function	Functions called
activate_NI(...)	Find NIC, Initialize NIC,	Pci(...),probe1(...),rtl_hw_start(...), read_eeprom(...),mdio_write(...), mdio_read(...),mdio_sync(...)
send2(...)	Sends packets	Composepacket(...)
recv2(...)	Receives packets	
deactive_NI(...)	Closes NIC	

<u>Module information transfer</u>

Information transfer between the driver's modules is mainly implemented by a data structure called net_device. This net_device data structure has basic information about the driver such as base address, IRQ, bus number, sending buffer address, receiving buffer address...etc.

```
struct net_device

{    char name[IFNAMSIZ];         // net device name

     unsigned long base_addr;          // device I/O address

     unsigned int irq;            // device IRQ number

     unsigned int bus;            // device pcibus number

     PVOID RxvAddress;                 // Rx virtual memory address
```

```
    PVOID TxvAddress;                  // Tx virtual memory address

    LARGE_INTEGER RxpAddress;   // Rx physical memory address

    LARGE_INTEGER TxpAddress;   // Tx physical memory address

    void *priv;                        // pointer to private data

};
```

Memory and bus layout (DMA)

Direct memory access is implemented in the Ethernet controller chips. Ethernet controllers access the sending and receiving buffers (memory) when some conditions are satisfied. For example, the RTL8139 chip accesses the receiving buffer's memory when the number of bytes in its embedded FIFO meets the early receive threshold [26]. It copies the bytes it receives to the receiving buffer in the main memory. Figure 4-3 shows the DMA map of RTL8139 Ethernet controller.

There are two memory operation concepts we will use throughout the driver development; virtual memory and physical memory. Physical memory is the memory hardware installed in a computer and physical memory address is the address of the circuit that connects to the memory chips. The physical memory address is not used directly by the application program. Virtual memory address is the address we get through a pointer. Usually virtual memory is larger than physical memory. The virtual memory address is combined with a page number and an offset within the page. A memory management unit maps the virtual memory page number to the physical page number, and the offset is unchanged. The paging method allows extra data to be stored on a hard disk and copied to physical memory when necessary [27]. For the hardware direct memory access (DMA), the physical address is needed because the hardware needs to read and write data as directed by the circuit. In the Windows operating system, a virtual memory address is easy to obtain. However, the correspondent physical address is difficult to get because there is no public information about how Windows converts the physical address to a virtual address.

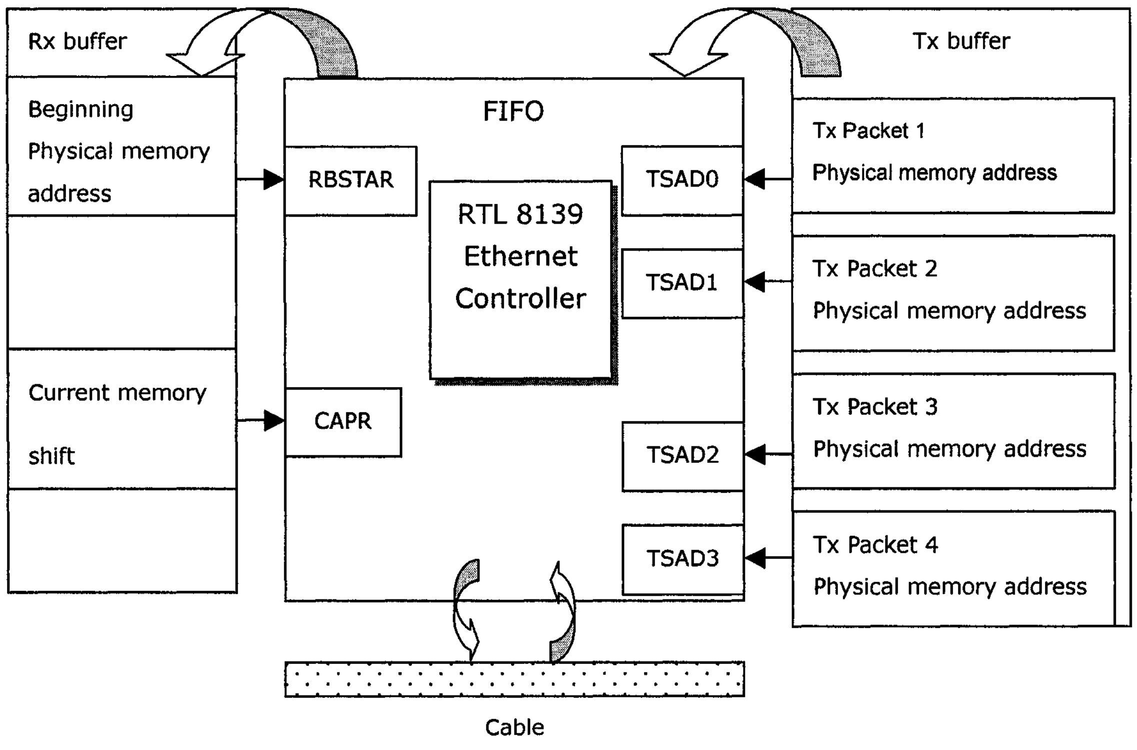

Rx buffer
Beginning Physical memory address
Current memory shift
FIFO
RBSTAR
CAPR
RTL 8139 Ethernet Controller
TSAD0
TSAD1
TSAD2
TSAD3
Tx buffer
Tx Packet 1 Physical memory address
Tx Packet 2 Physical memory address
Tx Packet 3 Physical memory address
Tx Packet 4 Physical memory address
Cable

Figure 4-3 DMA map of RTL 8139

The RTX operating system has its own special features even though it is attached to a

Windows operating system. RTX has functions to create a chunk of contiguous physical memory and convert the virtual address to the physical address.

4.4 Driver Components and Subroutines

This section describes real-time NIC driver components and subroutines in detail.

4.4.1 Activate NIC

Two steps are necessary to activate an NIC. The first step is to detect the NIC in the computer; NIC is plugging in one of the PCI slots. After the computer starts up, the NIC will have resources such as a memory address, interrupt number...etc. RTX gives developers a bus IO function to detect the card plugged into the PIC slots, and reads the information from the card. The function is *RtGerBusDataByOffse*t [28] and obtains information such as bus number, interrupt level and, most importantly, the basic memory address. This basic memory address is the beginning address of all of the registers. All of the register's addresses are the basic memory address plus the register's shift.

The second step is to initiate the NIC. The initiation includes resetting the NIC, reading the MAC address, sensing the connection, locating the sending and receiving buffer's physical address, enabling the sending and receiving process, initiating the interrupts, and writing all of the information to the corresponding registers. Every NIC has its unique

MAC address. This MAC address is stored in the EPROM of the NICs. In every initiation process, reading this MAC address is the basic step. Fast Ethernet NIC has 10M/100M and half/full duplex modes while Gigabit NIC has 10M/100M/1000M and half/full duplex modes. The NIC can sense the connection mode automatically, or the mode can be set manually. Sending and receiving buffers need to be created and initiated, and their physical addresses have to be written to the registers so that the NIC knows where to put received packets and where to pick up the sending packets.

4.4.2 Sending

Sending has two steps. The first is to compose packets. Data and destination addresses are given by the higher-level program. The driver will fill the information into the right positions in the packet frames. The driver does not need to fill the preamble, the start of frame delimiter, and the frame check sequence. Instead, they will be filled by the NIC automatically. The driver's filling begins with the destination MAC address, then the source MAC address. These addresses are followed by the data length or data type, which are two bytes long. The last part is the data. All of the data from the higher layer will be filled in this field. The maximum amount is 1500 bytes and the minimum is 46 bytes. If the data is less than 46 bytes, the driver adds (pads) some extra octants to make up the difference. Figure 4-4 shows the parts composed by a driver.

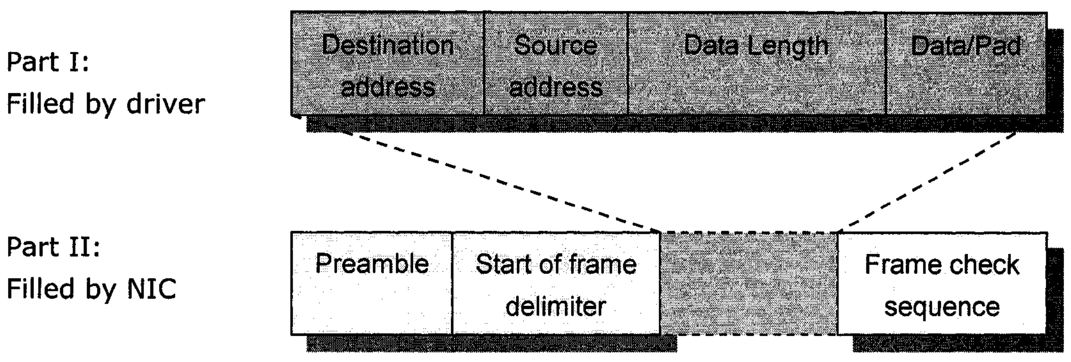


Figure 4-4 Composing a packet

The next step is sending. In the last step, packets are copied to the sending buffer. The RTL8139 has a fixed set of four sending descriptors in registers. The driver writes the physical address of the sending buffer to the corresponding descriptors and sets the sending signal for the NIC. The NIC will then send the packets when the network is free. The following are sample codes of sending.

Part of code of send2 ()

```
memcpy(dev->TxvAddress, pkthd, size);          // copy the packet to the sending buffer

add =(ULONG) dev->TxpAddress.QuadPart;  // got the buffer physical address

outl( add ,iobase+TxAddr0+ entry*4);           //outport the buffer physical address of packet

outw(size,iobase+TxStatus0+ entry*4);          //outport the packet length, now the NIC start to send
```

4.4.3 Receiving

The NIC receives the packet automatically without interference from the computer's CPU because the NIC has its own microprocessor. The NIC will store the packets in a buffer in the main memory. Different NIC chips have different buffer structures. The RTL8139 series chips use a single linear buffer called a ring structure. Incoming packets are stored sequentially in a ring buffer, and the driver copies them out to a piece of memory. In the initiation process, the length and address of the buffer is defined in the driver.

Ring structure is a buffer with its end connected to its beginning. We define the current read address as CRA, which is the address of the already read packet. The current write address is defined as CWA, which is the address of latest packet copied. CRA and CWA have correspondent registers in the Ethernet controller. After initiation, packets will be filled in from the start, at the very beginning of the ring structure; then, Ethernet controller chips will update the CWA. If new packets come, the NIC will automatically copy them after the CWA and update the CWA up to the end of the latest packet. The driver will update the CRA after the driver has read the packets in the ring structure. The packets before the CRA are already read. If CRA is equal to CWA, there is no new packet. If CRA is not equal to CWA, there must be some new packets and the driver should read on until CRA is equal to CWA. The ring structure is shown in Figure 4-5. Updating CRA needs some calculation and the formula is as below:

$$\textbf{New CRA} = \textbf{old CRA} + \textbf{packet length} + 4 + 3$$

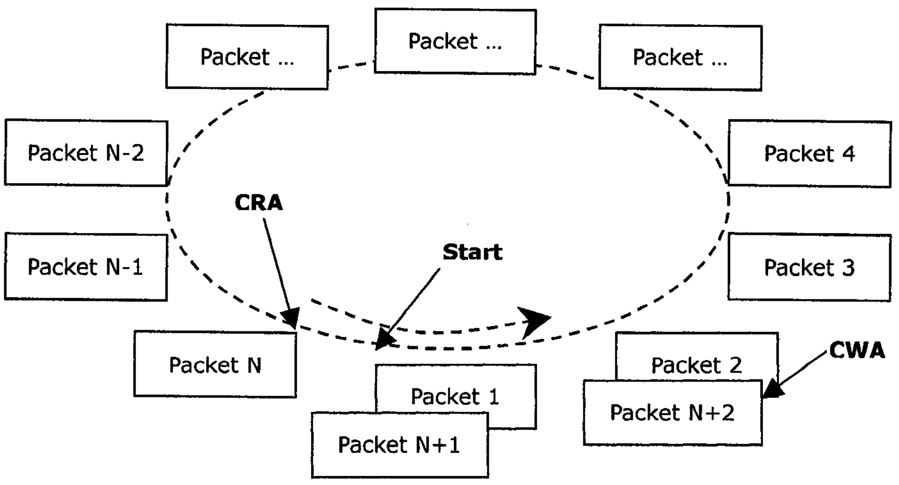


Figure 4-5 Receiving ring buffer

If the new CRA is larger than the buffer length, the buffer is full. The excess part of a new packet will be written to the beginning of the buffer and the next packet starts from the end of the previous packet [26].

Normal drivers in Windows and Linux use interrupt to communicate with the system. Ethernet NICs are designed to be this way as well. Nonetheless, in this real-time driver we use polling to check whether there are new packets. The reasoning behind this is that polling is simpler and faster than the interrupt, because interrupt takes more time to generate and more resources to implement. Another reason is that this driver will work under the TDMA protocol, which schedules every sending and receiving. There is no random sending and receiving. Polling is perfect for this kind of workload. Because the NIC chips are designed for interrupt, there is no specific interface for polling.

However, we can obtain the information by polling the interrupt register. This interrupt status register [25] shows the information when new packets come and when errors occur. By polling this register, the receiving function can be implemented very successfully.

Part of code of recv2( )

```
status = inw(iobase + IntrStatus);

outw(status, iobase + IntrStatus);

if (status & RxOK) {    // if there is Rx interrupt

    while( (inb(iobase + ChipCmd) & 1) == 0) {    //while there are new packets in the buffer

        now=inw(iobase + RxBufAddr);

        RxReadPtr = Rxring + RxReadPtrOffset;

        pPacketHeader = (PACKETHEADER *) RxReadPtr;

        pIncomePacket = (struct packethd *)(RxReadPtr + 4);

        PktLength = pPacketHeader->PacketLength;    //this length include CRC

        rx_status = *(u32*)(pPacketHeader);

        pkthdsize = sizeof(struct packethd);

        *len= PktLength-pkthdsize-4;

        //If Rx buffer reaches the end, copy the extra part from ring beginning to end

        if ( RxReadPtrOffset + PktLength +4 > Ringlen)

        memcpy( Rxring + Ringlen,    Rxring,    (RxReadPtrOffset + PktLength + 4 ) - Ringlen);

        //output the current packet
        memcpy( buf, (char *)(RxReadPtr+pkthdsize+4) , PktLength-pkthdsize-4);
        //fill destination MAC address
        destadd =(char *) address;
        k=0;
        for (j=5; j>=0; j--) {
        destadd[k] = pIncomePacket->mac.ethernet.h_source[j];
        k++; }
        //Update Read Pointer
        RxReadPtrOffset = (RxReadPtrOffset + PktLength +4+3)&RX_READ_POINTER_MASK;
        //4:for header length(PktLength include 4 bytes CRC) 3:for dword alignment
        tp->cur_rx = RxReadPtrOffset;
        outw( RxReadPtrOffset-0x10, iobase + RxBufPtr);  // -4:avoid overflow
} }
```

4.4.4 Deactivate NIC

To quit the driver, a deactivation function is needed. It frees the memory, releases resources and shuts down the NICs.

4.5 Testing

After developing the driver comes the test phase. In this phase, the driver needs to be tested in many different conditions to find bugs.

To test the driver and new protocols, a network monitor is a useful tool and instrument for driver development. The main function of a network monitor is to collect packets on the network and stamp the time on the packets. It can also analyze the packet by showing all of the bytes in the packets. The network monitor is installed on a computer, which has an Ethernet NIC using a Windows driver.

The sending test and receiving test are the two basic tests for an NIC driver. They use only two computers on the network to avoid interference. In the basic sending test, one computer runs the testing driver and sends the packet through the NIC, while the other runs the network monitor and receives the NIC. The monitor receives and analyses packets. This test checks the sending function and whether the packets are composed right. In the basic receiving test, a network monitor composes a packet and sends it out on the network. The receiving driver on the other computer should receive the packet and print it out to compare it with the one the network monitor composed. This test checks whether the packet can be received and whether the received one is right.

The basic tests can check the basic function of the NIC and the new real-time driver. However, it cannot test the time critical performance of the NIC and real-time driver. The performance tests should be tested under a real-time environment, and this topic will be discussed in the later chapters.

4.6 Debugging

Debugging takes up most of the programming time. In this section, some techniques and skills are discussed. Before debugging the program, the original program is backed up. Keeping each version of the program and taking good notes makes programming and debugging efficient.

Always begin with the most obvious mistakes. Most of the bugs are the simplest mistakes. For example, the use of "=" instead of "= =" and misplacing the "{" and "}" are good examples of common errors.

Debugging needs more information than general operating. A lot of information is usually hidden from the user and developer. Printing the information on a screen or a file is a common and useful technique. However, use this technique with caution because sometimes the print statement can change the behaviour of the program. In this driver-developing project, the most important information is in the registers. Because registers are in the NIC hardware, they record some of the status of the driver and hardware. And because those registers are tested in hardware development, they have a high level of reliability. Some of the variables in the driver are important as well.

The second technique is to isolate bugs. Dividing the program into several smaller isolated components is a good way to isolate bugs. Test those components separately if one of the components shows a malfunction or strange behaviour; it is possible that the bug is in this component. Add a new component to a tested and good component, and test them together. If the bug appears, that means the bug is connected to the relationship to the new component, although it may not be in the new component. Some of the bugs did not occur when the components were tested separately. In this situation we could use a virtual component instead of the original one. This virtual component simulates the results of the real one, however, it does not affect the real components or hardware.

5 Latency and Performance Testing

In the NIC driver development phase, some basic sending and receiving tests are processed. Nevertheless, these tests cannot illustrate the NIC and driver's behaviours in real-time environment and applications. To improve performance and prevent mistakes, latency tests need to process step by step to figure out where the bottleneck is. The latency tests include a parallel port characteristic test, an NIC sending latency test, a one-way send and receive test, a two-way round trip test, and a data rate test. In terms of hardware, a RTL8139 fast Ethernet NIC, a D_link DGE-550T Gigabit NIC, and several hubs and switches are tested. Prof. B. W. Gordon provided the D_link DGE-550T Gigabit NIC driver.

From the test results, the parallel port shows stable, fast and small jitter characteristics. It is a capable tool to test the other equipment. The performance of the real-time driver is better than the Winsock in terms of latency and jitter. The gigabit Ethernet NIC is much better than the fast Ethernet NIC in terms of latency and jitter. From the results of the NIC latency tests, the less hardware in the host computer, the better the performance the NIC has. Both NICs are suitable for real time application. Compared with hubs, switches have a large latency and jitter as expected. Hubs have different performance; because they use different algorithms, some of them have very good performance while others have latency and jitter rates approximately as large as switches.

5.1 Latency Quantification and Measurement

Before introducing latency, it is helpful to look at the definition of response time. Response time is defined as the elapsed time between the end of an inquiry or demand on a computer system and the beginning of a response; for example, the length of the time between an indication of the end of an inquiry and the display of the first character of the response at a user terminal [29]. Latency is that in a computer system that increases response time beyond the time desired for an operation.

In the networking field, latency is the same as delay. It is the time from when a packet is sent out until its arrival at the designated node. In this project, latency is defined as the time from when a request is sent out until the process is finished.

The minimum latency is the shortest time delay in a test. The maximum latency is the largest one. A jitter is the difference between the minimum and the maximum latency. Jitters show how stable a system's performance is.

The results of the latency tests are presented with a diagram and a histogram in Figure 5-1. The diagrams show the time delay of tests on the Y-axis in μs, ms or seconds.

The X-axis is the test number or the time. Tests generally run several thousands times. Between two tests, there are several milliseconds of rest. Histograms show the distribution of the results.

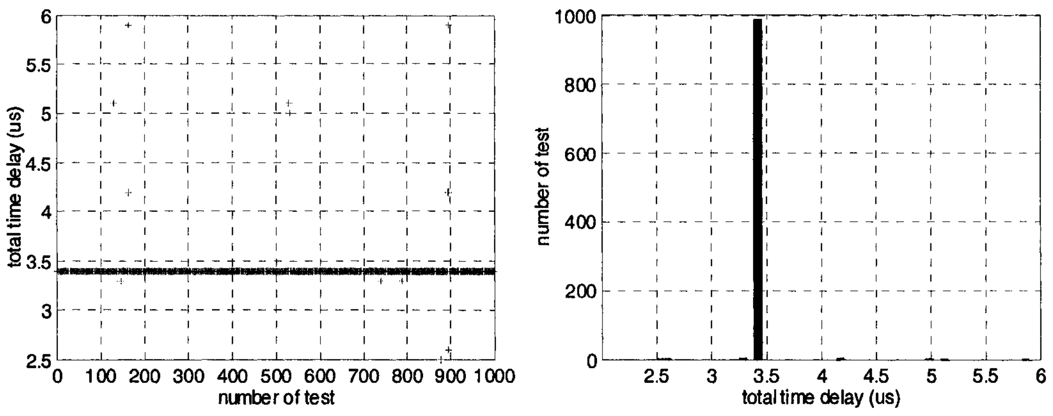


Figure 5-1 Example of test result data and histogram

There are maximum latency, minimum latency and average latency. The average is a mathematical average. It is possible that most latency does not happen in the average time. If the test is using the polling technique, the result is discrete according to the polling period.

5.2 Operating System Latency

In latency tests, the first test is to test the operating system. Windows and RTX Operating systems have different latencies.

5.2.1. Compare RTX to Win32

Windows and RTX operating systems have different latencies, although they are working on one computer. Clock reading tests show how fast and stable an operating system can perform. The following tests show the delay of the reading clock statement on different operating systems. All tests take place on the same computer. For the hardware configuration of computers, please refer to Appendix 2.

Test 1 group is designed to test the operating system clock reading time delay. It contains two tests. Test 1-a is under the RTX OS. Test 1-b is under the Windows 2000 OS. The two tests have the same operation steps.

Test 1: operating system clock reading delay test.

Operation:

Step 1, read the clock, get time T_1.

Step 2, read the clock again, get time T_2.

The interval between each round of tests is 1 ms.

Measure: $T_{dclock:} = T_2 - T_1$, as shown in Figure 5-2.

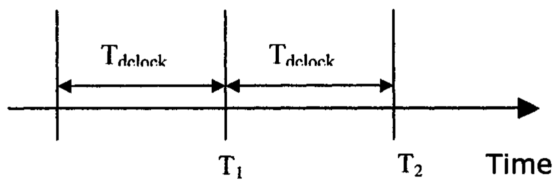


Figure 5-2 Test 1 illustration

Test1-a: RTX performance:

Test 1000 cycles, takes 999996.60 μs. Figure 5-3 shows the result.

T_{dclock}: Max =5.90 μs, Min = 2.50 μs, Average = 3.40680 μs, Jitter = 3.40 μs.

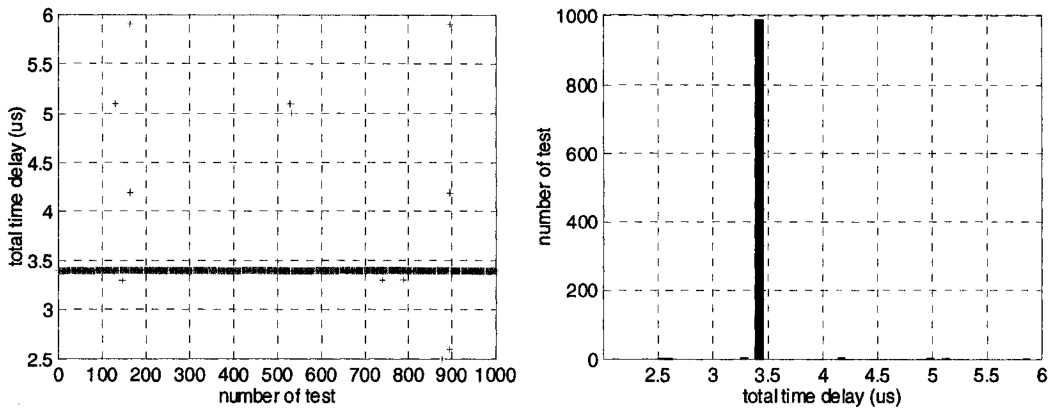


Figure 5-3 Test 1-a result data and histogram of RTS system

Test 1-b: Win32 performance:

Test 1000 cycles, takes time 10054472.30μs. Figure 5-4 shows the result.

T_{dclock} : Max = 1.0740e3 μs, Min = 1.170e1 μs, Average = 1.45018e1 μs, Jitter = 1063 μs

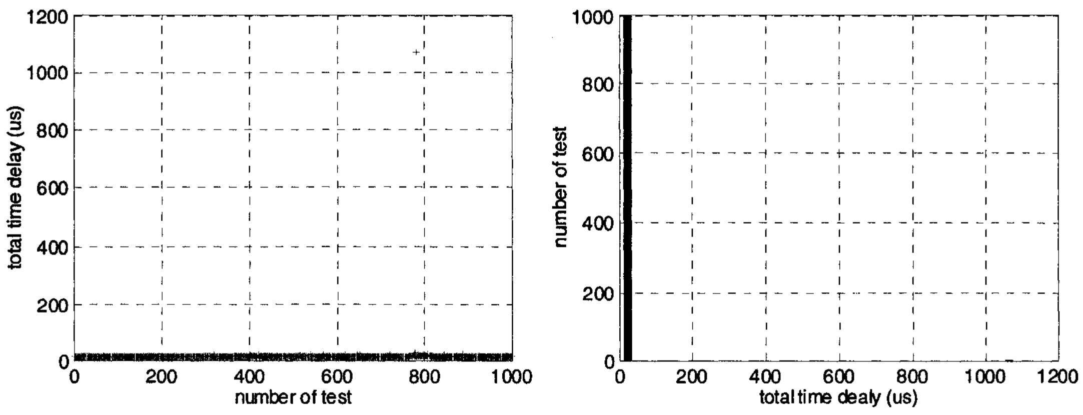


Figure 5-4 Test 1-b result diagram and histogram

From the results of the win32 test in Figure 5-4, the general performance is acceptable for office applications. Its average time delay is 14.5 μs, 4.26 times of RTX test results. However, occasionally win32 will get a very large time delay, which goes up to 1074 μs. The jitter of the win32 system is 1063 μs. This jitter is not acceptable in a real-time system, especially in hard real-time applications.

5.2 Parallel Port Communication Latency

In this section, parallel port time delay is tested because the parallel port will be working as test equipment. Different from the previous section which tests the port I/O operation, the new tests focus on the communication delay.

The parallel port is a bi-directional port. It has 12 output pins and 5 input pins. Crossover parallel cables can cross the 5 input pins to the 5 output pins in order to construct 5 channels (Appendix 1). Parallel port I/O communication latency tests run as an rtss process in the RTX operating system. Figure 5-5 shows the test hardware connection. To guarantee the accuracy of the next tests, some features of the parallel port should be done first. Group Test 3 is designed to test the settling time and bandwidth of the parallel port. Test 3-a is the settling time test and Test 3-b is the bandwidth test.

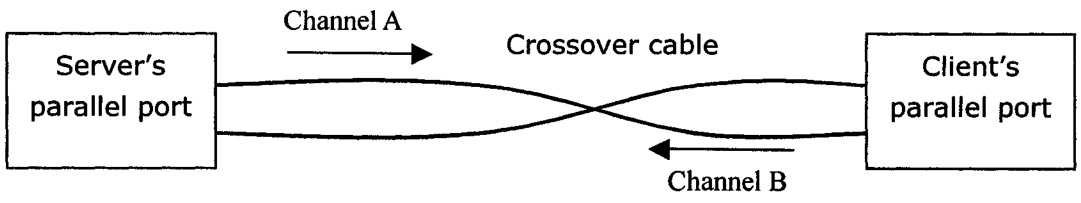


Figure 5-5 Cross over parallel port channel illustration

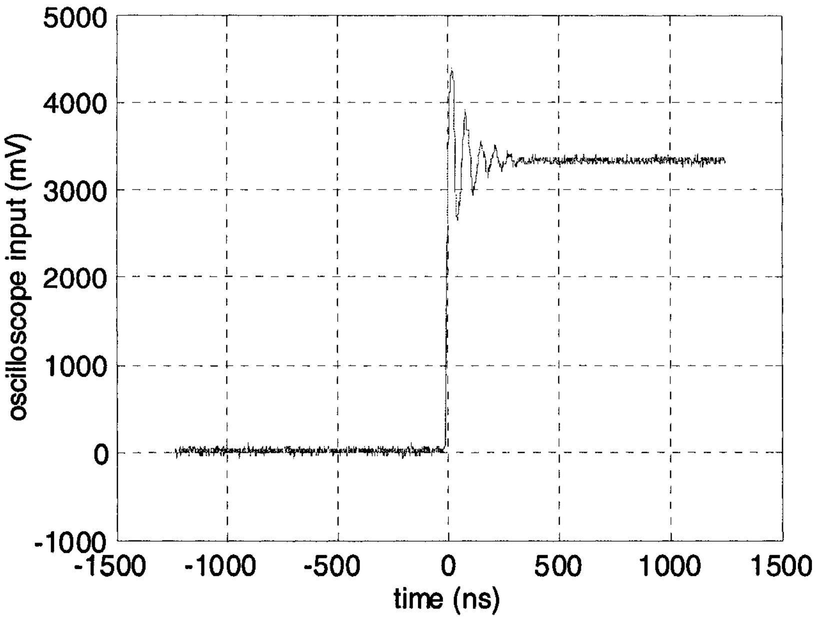


Figure 5-6 Test 3-a settling time of a rising edge sent by the parallel port, oscilloscope screenshot

Test 3, parallel port basic test

Test 3-a, parallel port settling time test

Operation:

Step1, parallel port sends out a square wave.

Step2, use an oscilloscope to observe the rising edge, and find out the settling time.

Result: $T_{settle} = 250$ ns, for the computer configuration see Appendix 2. Figure 5-6 shows the result. From this test, we discover that the settling time is 250 ns, which can be used in other test results' compensation.

Test 3-b, parallel port bandwidth test:

Test 3-b is testing the highest working frequency and bandwidth of one channel of the parallel port, and also can verify the sending and reading time delay. The test will bind the computer for 10,000 milliseconds, and be stopped by the RTX watchdog. Figure 5-7 illustrates the test.

Operation:

Step1, Send a high voltage to the parallel port,

Step2, Send a low voltage to the parallel port,

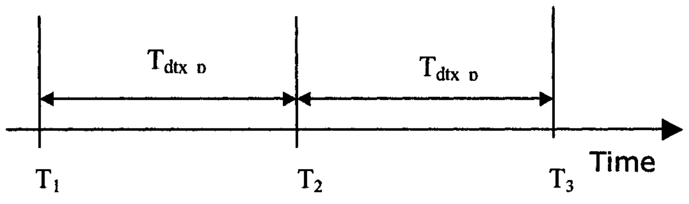


Figure 5-7 Test 3-b parallel port bandwidth illustration

Result:

Observe the wave from an oscilloscope in Figure 5-8. From the oscilloscope:

$T_1 = 0.0\ \mu s$, $T_2 = 3.60\ \mu s$. $T_{interval} = 3.6\ \mu s$. $T_{dtx\_p} = (T_2 - T_1) / 2.0 = 1.80\ \mu s$.

Where: T_1 is the time that the high voltage is sent out.

T_2 is the time that the low voltage is sent out.

$T_{interval}$ is the interval of the square wave.

Frequency of one channel of the parallel port is $1/T_{interval} = 277.7$kHz

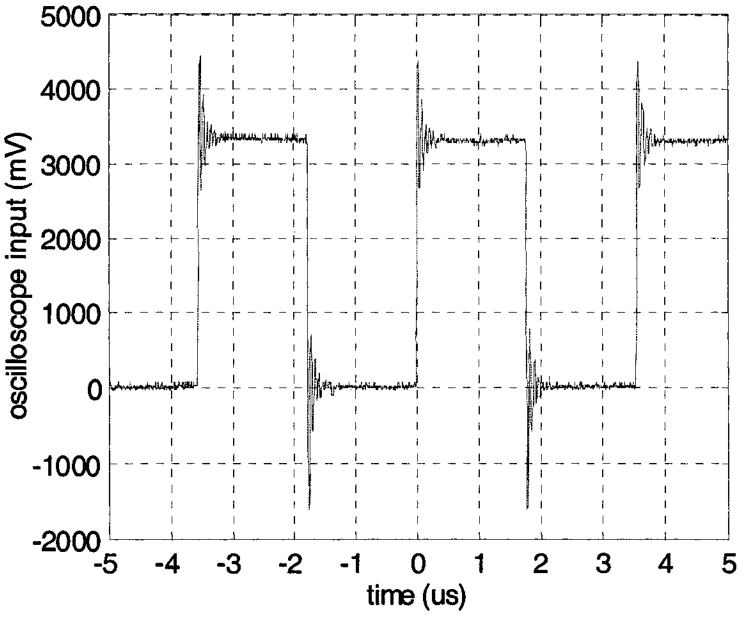


Figure 5-8 Result of one sending to parallel port test observed by an oscilloscope.

Bandwidth is the difference between the limiting frequencies. Here the bandwidth is 277.7kHz. From the oscilloscope, only one sample of the wave is recorded, and a jitter on the wave is also observed.

Test 4 group, parallel port time delay test

The parallel port will be used to measure other hardware and software. Its time delays have to be measured first. Test 4 groups are designed to test the parallel port communication time delay. Test 4 has two groups of tests to be processed. Test 4-a is a receiving (Rx) test while Test 3-b is a sending (Tx) test.

Test 4-a, parallel port Rx test:

Operation:

Step1, Read clock, get time T_1.

Step2, Read parallel port with inw (in16).

Step3, Read clock again, get time T_2, as shown in Figure 5-9.

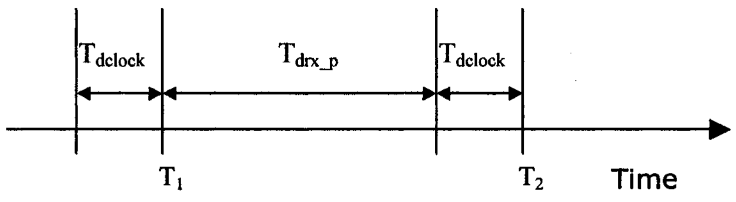


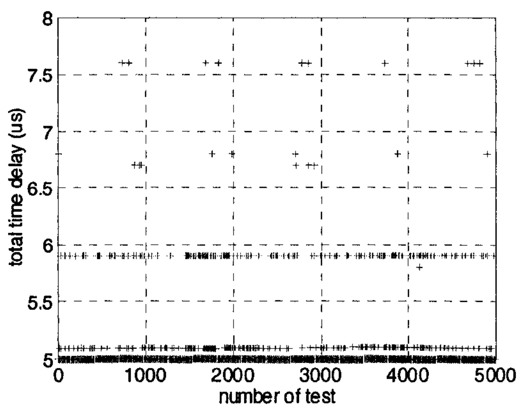
Figure 5-9 Test 4-a parallel port Rx illustration

Test 4-A test 5000 cycles, takes the time of 24999995.80 µs, as shown in Figure 5-10.

Result:

$T_2 - T_1$: Max = 7.60 µs, Min = 5.0 µs, Average = 5.048520 µs, Jitter = 2.60 µs.

Average $T_{drx\_p} = T_2 - T_1 - T_{dclock} = 5.048520 - 3.40 = 1.64852$ µs



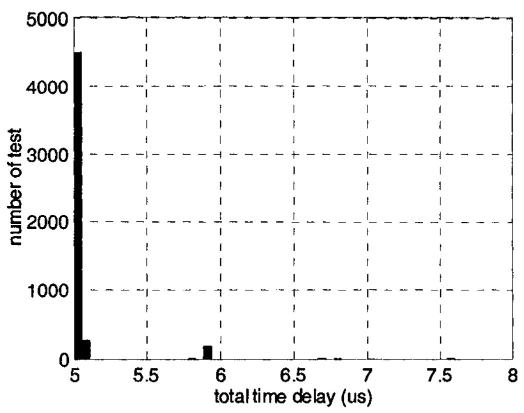


Figure 5-10 Test 4-a parallel port Rx results

Test 4-b, parallel port Tx test operation:

Operation:

Step1, Read clock, get time T_1.

Step2, Send a high voltage to parallel port by outw (out16).

Step3, Read clock again, get time T_2. $T_{dtx\_p} = T_2 - T_1 - T_{dclock}$, as shown in Figure 5-11.

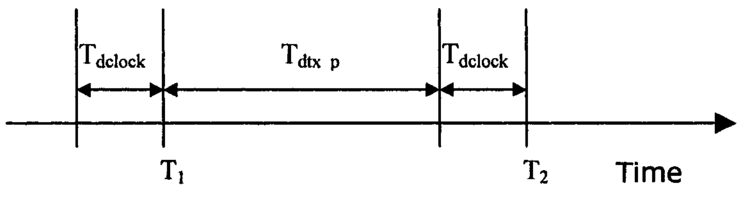


Figure 5-11 Test 4-b parallel port Tx test illustration

Test 4-b test 5000 cycles, takes time 24999996.60 μs.

Results:

$T_2 - T_1$: Max = 7.600000e+000μs, min=5.000000e+000μs, average=5.072280e+000μs , Jitter=2.6μs, as shown in Figure 5-12.

Average T_{drx_p} = $T_2 - T_1 - T_{dclock}$ = 5.072280 - 3.40 = 1.672280 μs

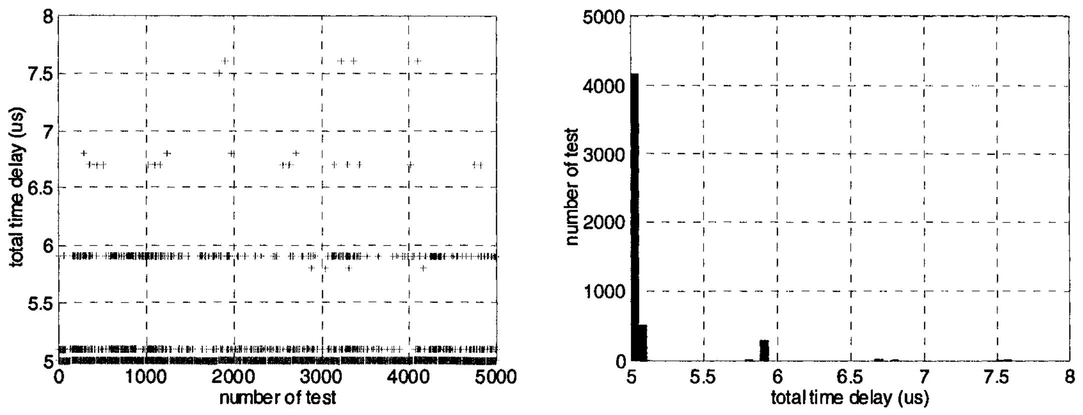


Figure 5-12 Test 4-b parallel port Tx test result

Test 4-c, parallel port round trip latency test:

Group Test 4-c is designed to test the round trip latency in two parallel port channels. The round trip latency test has a server and a client. The server sends out signals through a parallel port, while the client echoes the signal. The time T_1 and T_2 are recorded on the server side. The client does not record any data. The sever sends out two signals: high voltage (logic 1) is the testing signal, by which the time was measured and low voltage (logic 0) means that this round of tests is finished. The client sends back the same signal as an acknowledgement. The acknowledgement method makes sure that every high voltage signal on the line is sent out in the current round and not from the previous round.

Operation:

Step1, Server: reads clock, gets time T_1, and sends high voltage to the parallel port.

Client: keeps polling the parallel port.

Step2, Server: keeps polling parallel port.

Client: receives the high voltage signal and sends a high voltage signal back.

Step3, Server: receives the signal and reads the clock again, gets time T_2, sends a low voltage.

Client: keeps polling the parallel port.

Step4, Server: keeps polling parallel port.

Client: receives the low voltage signal and sends a low voltage signal back.

Step5, Server: receives the low voltage, waits for a while, repeats steps 1 to 5.

Client: keeps polling the parallel port.

Interval between rounds is about 1ms.

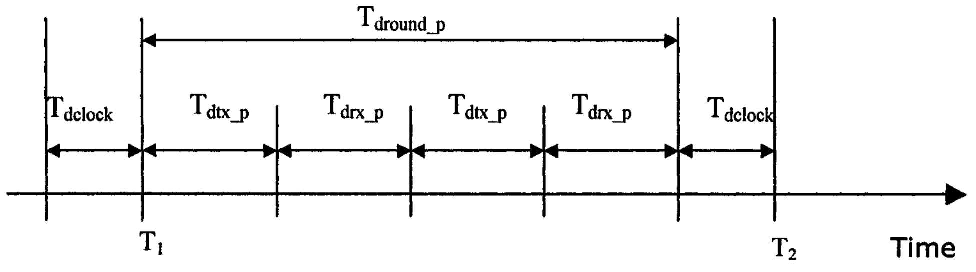


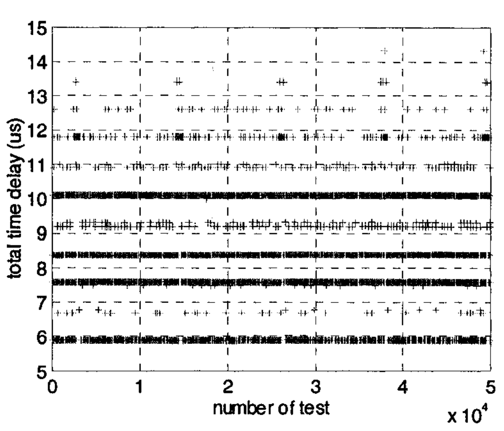
Figure 5-13 Test 4-c, parallel port round trip latency test illustration

Result:

Average of $T_{dclock} = 3.40\mu s$, from previous tests.

$T_{dround\_p} = T_2 - T_1 - T_{dclock}$

$T_{dtx\_p} + T_{drx\_p} = T_{dround\_p} / 2 = (T_2 - T_1 - T_{dclock}) / 2$, shown in Figure 5-13.



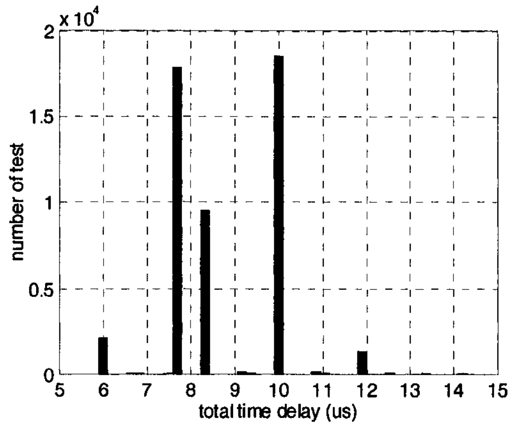


Figure 5-14 Test 4-c parallel port round trip latency test results

Test 50000 cycles, total time 49998891 μs.

$T_2 - T_1$: Max 14.30 μs, Min 5.90 μs, Average 8.742274 μs, Jitter: 8.40 μs. The data is shown in Figure 5-14.

Average:

$T_{dround\_p} = T_2 - T_1 - T_{dclock} = (8.742274 - 3.40) = 5.342774$ μs

$T_{dtx\_p} + T_{drx\_p} = 5.342774 / 2 = 2.671137$ μs

Note that:

From Figure 5-14, we find that there are long time delays happening about every 1.2s.

1mS x 1.2E4 = 1.2 S

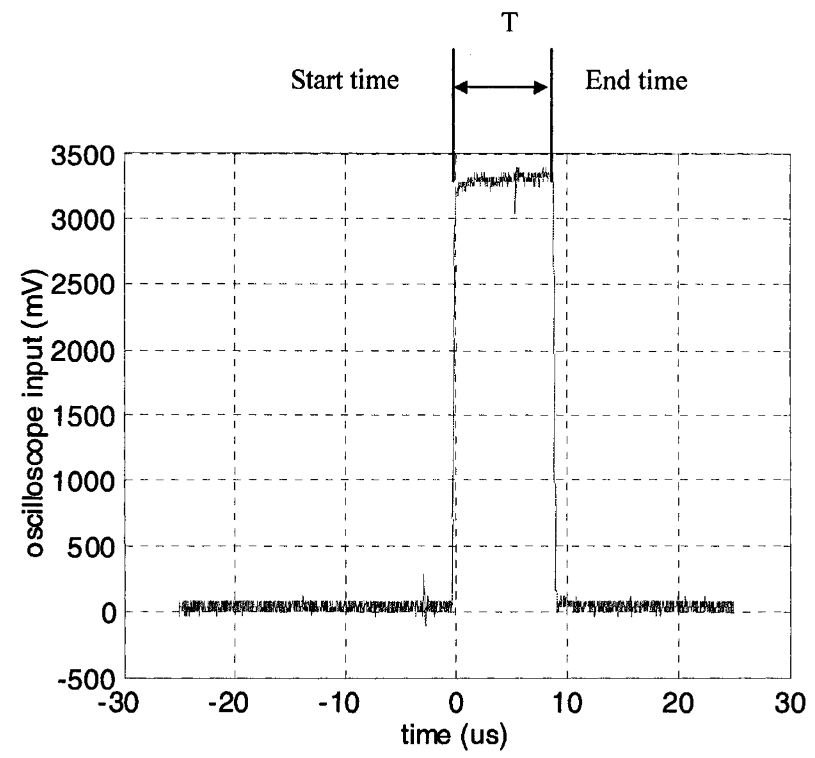


Figure 5-15 Test 4-c oscilloscope screen shot

From the tests we know that the parallel port has a very stable performance in latency and jitter. The settling time is 250 ns. Average Tx and Rx time is about 2.7 μs and the maximum is 5 μs. The initiation of the port and operation is simple. All factors make the parallel port ideal for testing other hardware.

The parallel port can be used as a feedback in the latency and jitter test for other hardware or software. This test can test one-way latency of the test objects, because the latency and jitter of the parallel port are known.

The second use of the parallel port is that we can observe the process of a program by sending a signal to the parallel port, and observing it with an oscilloscope. Before the operation, we send the parallel port a high voltage. After the operation, we send the parallel port a low voltage. From the oscilloscope, a pulse is observed. The raising edge is when the operation starts and the down edge is when the operation ends.

The third use of the parallel port is to synchronize the clocks of network computers. In this test, a crossover parallel cable is used. The server sends out a synchronization pulse periodically and other computers receive this signal. They adjust their clocks (maybe just a virtual globe clock) according to this signal.

5.3 Fast Ethernet NIC Latency

After the parallel port is tested, we can use the parallel port to measure the latency of the Ethernet NICs. Test5, Figure 5-15, is designed to measure the fast Ethernet NIC latency, which includes one transmitting and one receiving latency, and uses the parallel port as feedback. Test6, Figure 5-18, is designed to measure the two transmitting and two receiving latencies, which still use network NIC as feedback. Test7, Figure 5-20, is designed to measure the transmitting latency by polling a specific register. To eliminate other interference on the network, all of the tests are implemented on a two-computer network connected by a crossover cable.

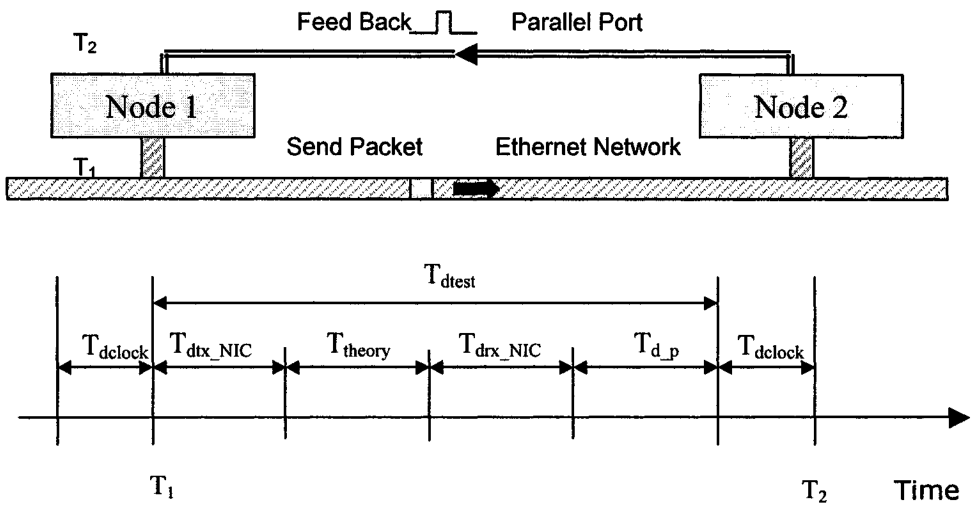


Figure 5-16 Test 5 NIC one-way latency test with parallel port feed back illustration

Test 5 NIC one-way latency test with parallel port feed back

Operation:

Step1, Server reads clock, gets time T_1, sends a packet to the NIC.

Client is polling the NIC register and waiting for the packet.

Step2, Server is polling parallel port.

Client receives the packet and sends a high voltage signal to the parallel port.

Step3, Server receives the signal and reads the clock again, gets time T_2.

Because both the length of the network cable and parallel port cable are less than 3m, the time delay on media is ignored. This experiment is repeated with different packet sizes. Figure 5-16 shows the test.

$T_{dtest} = T_2 - T_1 - T_{dclock}$; here T_{dclock} is 3.4 µs on average.

$T_{dtx\_NIC} + T_{drx\_NIC} = Tdtest - T_{d\_p}$; here T_{d_p} is 2.671137 μs

The hardware includes the following:

NIC: SMC1244 100Mbps, Media: Crossover cable, Cable: C5e Network cable

Result: Figure 5-17 shows the overall performance. Table 5-1 gives some data in μs.

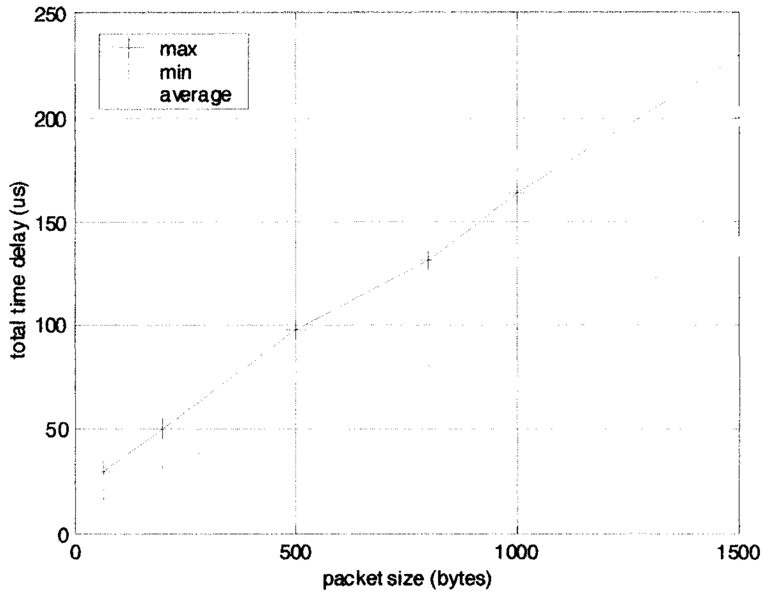


Figure 5-17 Overall of test 5 T_{dtest} in crossover cable with parallel port feed back test

Table 5-1 Result of test 5 crossover cable with parallel port feed back test

Packet size (bytes)	**64**	**200**	**500**	**800**	**1000**	**1500**
$T_2 - T_1$ (μs)	23.6	39.4	76.7	111.2	132.3	200.7
T_{dtest} (μs)	20.2	36.0	73.3	107.8	128.9	197.3
$T_{dtx\_NIC} + T_{drx\_NIC} + T_{theory}$	17.53	33.33	70.63	105.13	126.23	194.63
T_{theory} = bits/100Mbps (μs)	5.12	16	40	64	80	120
$T_{dtx\_NIC} + T_{drx\_NIC}$ (μs)	12.41	17.33	30.63	41.13	46.23	74.63

Test 6, fast Ethernet NIC round trip latency test

Figure 5-18 shows the test configuration and time line.

Operation:

Step1, Server reads clock, gets time T_1, sends a packet to the NIC.

Client is polling the NIC register, and waiting for the packet.

Step2, Server is polling the NIC register.

Client receives the packet and sends the packet back to server.

Step3, Server receives the packet and reads clock again, gets time T_2.

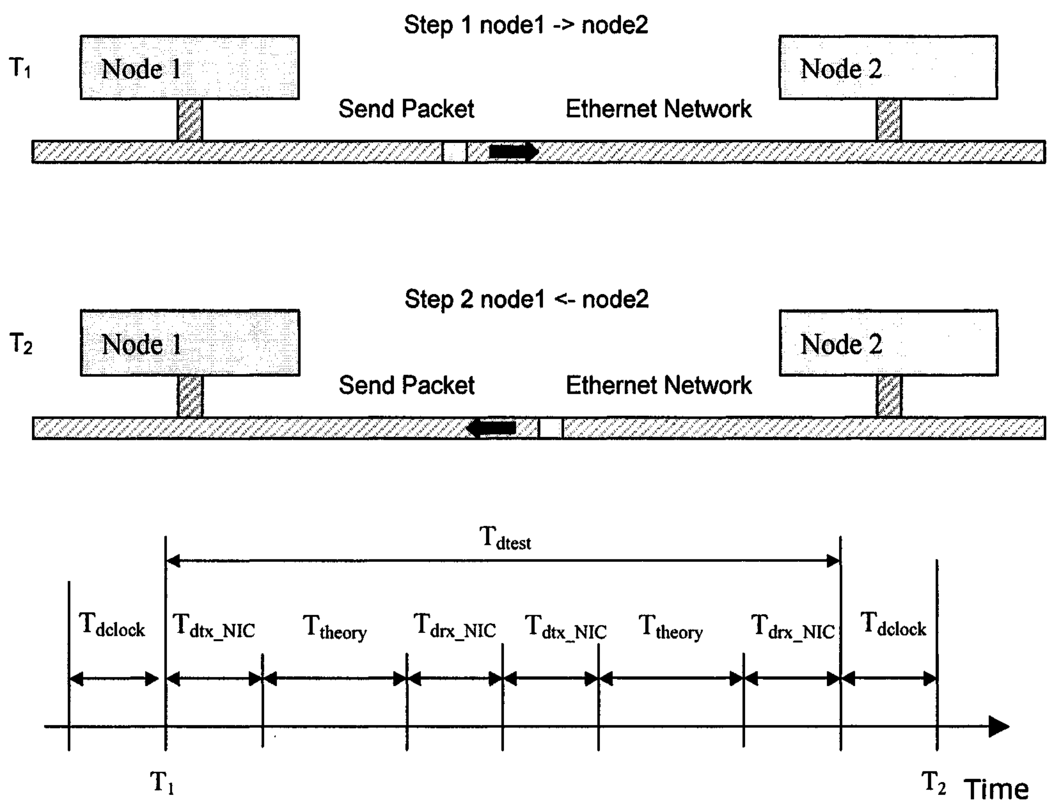


Figure 5-18 Test 6 NIC round trip latency test illustration

Because both the length of the network cable and parallel port cable both are less than 3m, the time on media is ignored.

Result: Figure 5-19 shows the result, and Table 5-2 shows the data.

$T_{dtx\_NIC} + T_{drx\_NIC} = (T_2 - T_1 - T_{dclock} - 2 * T_{theory})/2$, T_{dclock} is 3.4 μs in average.

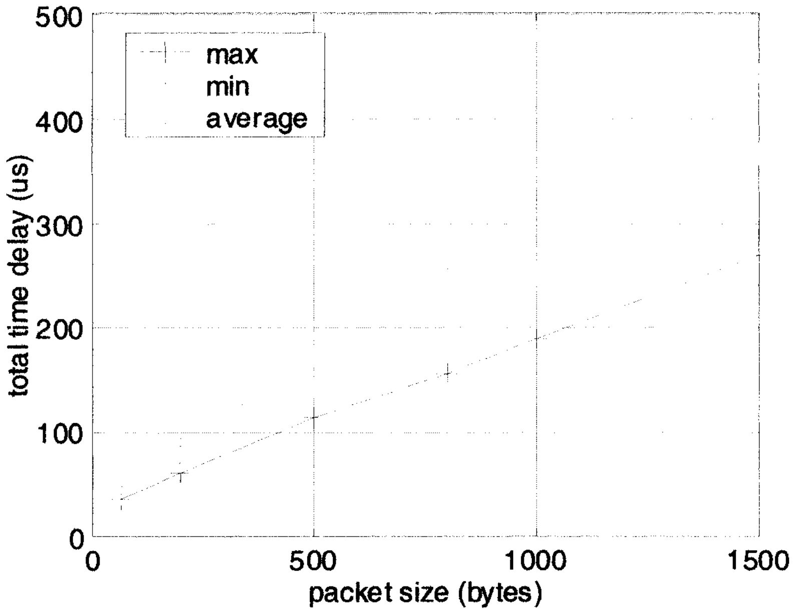


Figure 5-19 Fast Ethernet round trip test result

Table 5-2 Result of test 6 fast Ethernet round trip test

Packet size (bytes)	64	200	500	800	1000	1500
$T_1 + T_2$ (μs) Average	40.3	70.1	143.4	202.4	250	366.9
T_{dtest} (μs)	36.9	66.7	140.0	199.0	246.6	363.5
T_{theory} = bits/100Mbps*2 (μs)	10.24	32	80	128	160	240
$T_{dtx\_NIC} + T_{drx\_NIC}$	13.33	17.35	30.0	35.5	43.3	61.75

Test 7 fast Ethernet NIC transmit latency test

In Test 7 we test the packet sending time delay by using the transmit register. The

RTL8139 NIC has the register show if the packet is totally sent out. If we keep polling this register, we can find out when the transmit has finished

Operation:

Step1, send out 1 to parallel port, and record the time T_1;

Step2, send out packet to the NIC and check the status register until the packet is totally sent out. This check operation blocks the computer until it finishes.

Step3, send out 0 to parallel port, and record time T_2.

Step4, wait for a specified amount of time. (300μs).

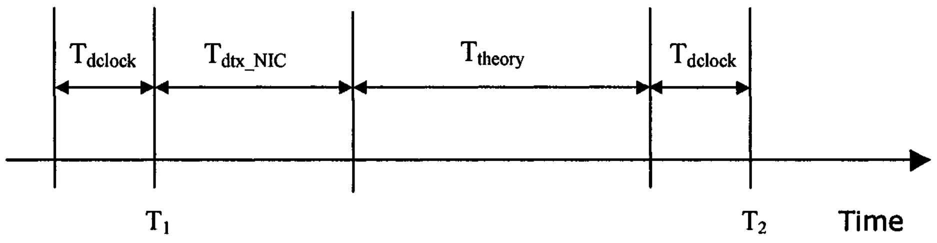


Figure 5-20 Test 7 NIC transmit latency test illustration

Figure 5-20 shows the time line of the test. Figure 5-21 shows the result. Table 5-3 gives the data.

Results:

Table 5-3 Result of test7 average

Packet size (bytes)	**64**	**200**	**500**	**800**	**1000**	**1500**
T_{dtest} (μs) average	15.9	24.9	52.0	85.3	99.4	142.1
T_{theory} = bits/100Mbps	5.12	16	40	64	80	120

(μs)						
T_{dtx_NIC} (μs)	10.78	8.9	12	21.3	19.4	22.1
$T_{dtx\_NIC} + T_{drx\_NIC}$ (μs) (Table 5-3)	12.41	17.33	30.63	41.13	46.23	74.63
T_{drx_NIC} (μs)	1.63	8.43	18.63	19.83	26.83	52.53

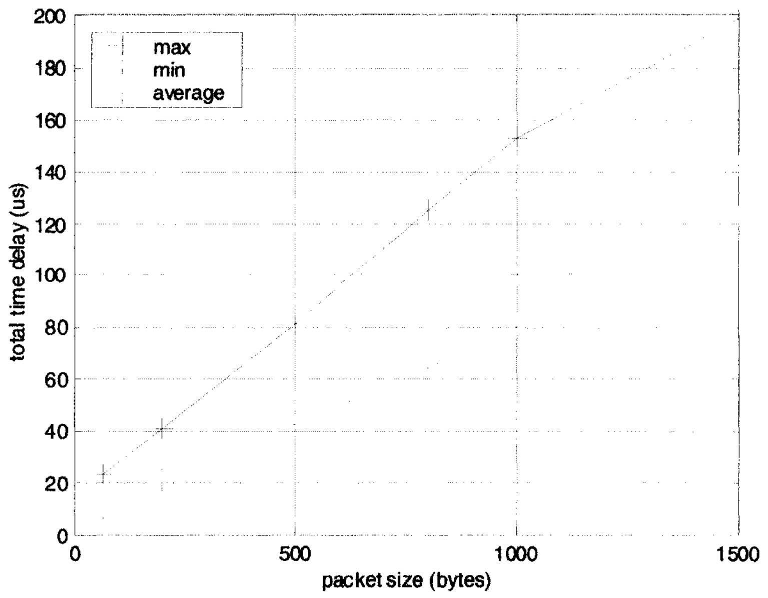


Figure 5-21 Overall result of fast Ethernet NIC Tx time delay test

This test can also generate some pulses observable by an oscilloscope or other computers. Figure 5-22 is the Oscilloscope waveform on packet size 1500 byte test. The high level period is the packet sending time. In this figure, the Tx sending time is about 125 μs.

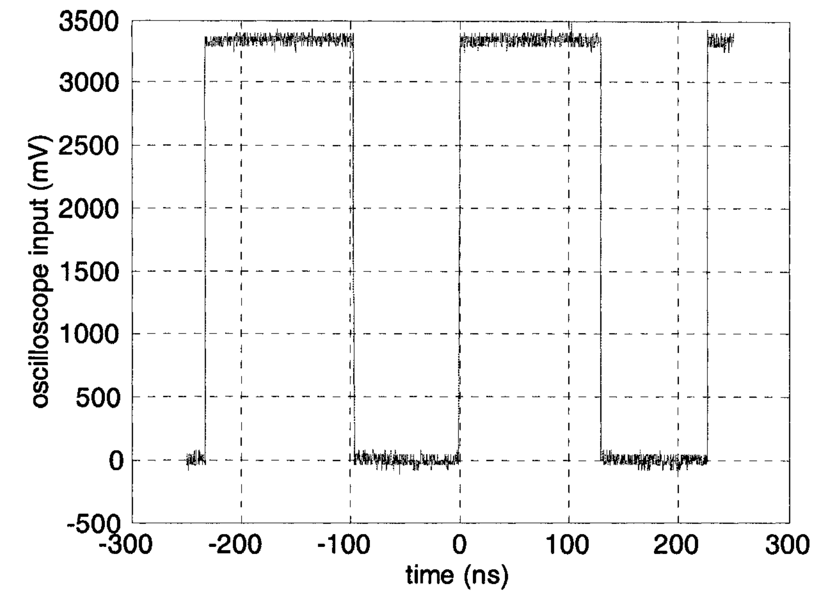


Figure 5-22 Test 7 oscilloscope screen shot

5.4 Fast Ethernet NIC Data Rate

In the driver of the Ethernet NIC, packets are put into a buffer and then wait for the NIC to send them out. If the application puts too many packets on the buffer that exceed the length of the buffer, some of the packets will be lost. Theoretically, the minimum time interval of continually putting packets on the driver buffer must be equal to the time of the NIC putting the packets on the network. However, for a specific NIC, we have no information about whether this theory is right or not. In Test 9 we determine how close packets can be spaced together and the maximum data rate for RTL8139 NIC and D-Link Gigabit NIC. Test 9 is implemented on a two-computer network connected by a crossover cable.

Test 8 fast Ethernet NIC data rate test

Operation:

Server: Hardware RTL8139 series chip fast Ethernet NIC

Step1, Server sends a packet.

Step2, wait a time T_r,

Client: Hardware: NIC Intel 10/100 integrated NIC

Step1, Client keeps receiving status:

Use Windows system to receive.

Use RTX driver to receive.

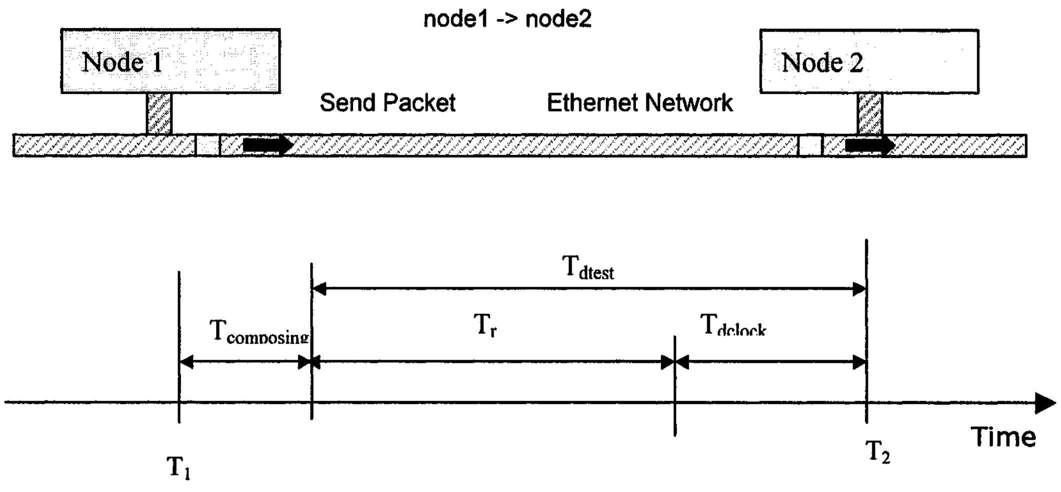


Figure 5-23 Test 8 NIC data rate test block illustration

Figure 5-23 shows the time delay of the test. Table 5-4 presents the result.

Result: compare the number of packets sent and received. If they match, T_r is good. If they do not match, the time T_r is too short. Increase T_r and test again. Test 9 is also repeated for different packet sizes.

Table 5-4 Minimum time space between two packets

Packet size (bytes)	Theoretical minimum time space (μs)	T_r (μs)
1500	120	128
1000	80	78
64	5.12	1

We find that the test result for a packet size of 1000 bytes is 2 μs less than the theoretical result and a 64-byte result is about 4 μs less than the theoretical. However, when we consider the clock reading time is around 3 ~ 6 μs, the test results approximately match the theoretical packet sending time.

Test 9 fast Ethernet NIC data rate test with sending block

Test 9 is designed to verify Test 8.

Operation:

Server: Hardware RTL8139 series chip fast Ethernet NIC

Step1, Server sends a packet.

Step2, Check the NIC status register until it shows the packet is sent.

(Block function)

Step3, Wait a time T_r,

Client: Hardware: NIC Intel 10/100 integrated NIC

Step1, Client keeps receiving status.

Use windows system to receive.

Use RTX driver to receive.

Result: compare the number of packets sent and received. If they match, T_r is good.

T_r can be 0 μs.

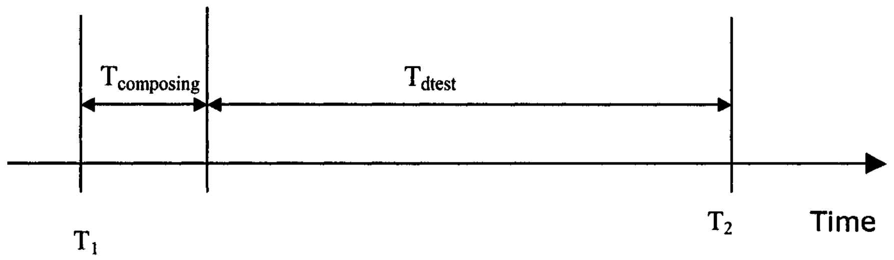


Figure 5-24 Test 9 illustration

Theoretically, in 100Mbps Ethernet, the maximum data rate is 12.5 byte/μs. Using a Send2 function with status check and block function, the packets can be sent one by one with 0 μs rest time in between. Figure 5-24 shows the time delay of the test.

5.5 Latency of Hub and Switch Components

The previous tests are based on a crossover Category 5e cable in fast Ethernet NIC; the following tests will connect the fast Ethernet network with hubs and switches.

Test 10 Latency measurement of hubs and switches

In this group of tests, four different brand name hubs, one faster Ethernet switch and two gigabit switches are tested. The physical connection topology used is star topology for both the hub and switch tests.

The test operation is the same as with Test 5. Because the only difference between these tests and Test 5 is that we use a hub or switch instead of a crossover cable, the difference in the results of time delay will come from the hub or switch. Figure 5-25 shows the calculations.

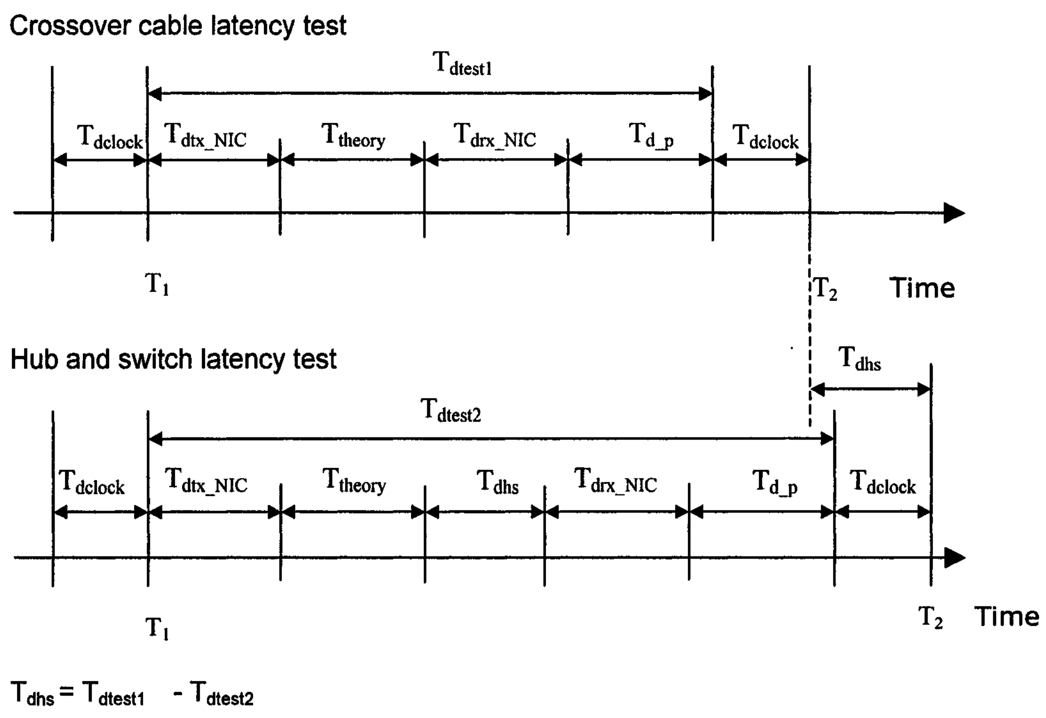


Figure 5-25 Hub and switch latency test illustration

Test 10-a, 3Com OfficeConnect 10/100 dual Speed hub 8. Figure 5-26 shows the result and Table 5-5 shows the data of the test.

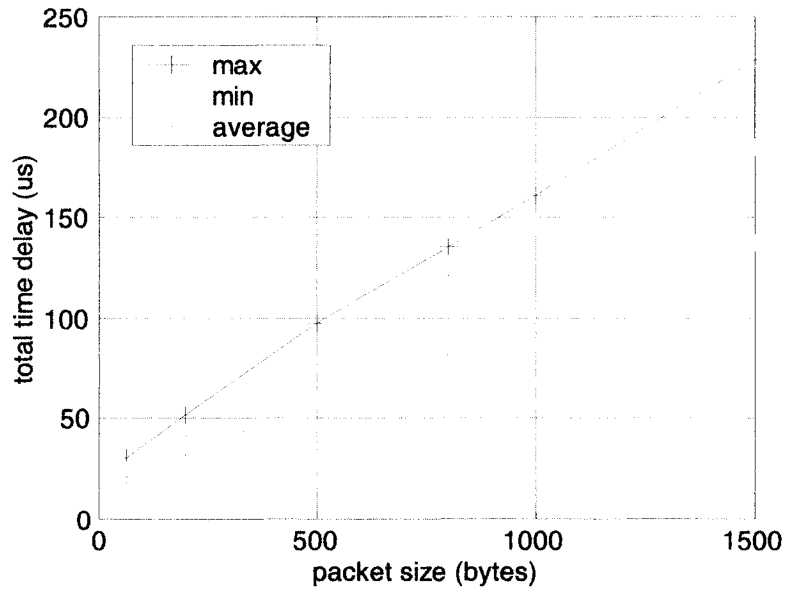


Figure 5-26 Result of test 10-a 3Com OfficeConnect 10/100 hub

Table 5-5 Result of test 10-a 3Com OfficeConnect 10/100 hub

Packet size (bytes)	**64**	**200**	**500**	**800**	**1000**	**1500**
T_{dtest} of crossover cable average (μs)	23.6	39.4	76.7	111.2	132.3	200.7
T_{dtest} of 3Com OfficeConnect hub average (μs)	24.8	41.6	76.7	121.3	146.7	185.1
Hub time delay Average (μs)	6.32	18.2	40	74.1	94.4	105.6

Test 10-b, NETGEAR DS108 10/100 hub. Figure 5-27 shows the results and Table 5-6 shows the data.

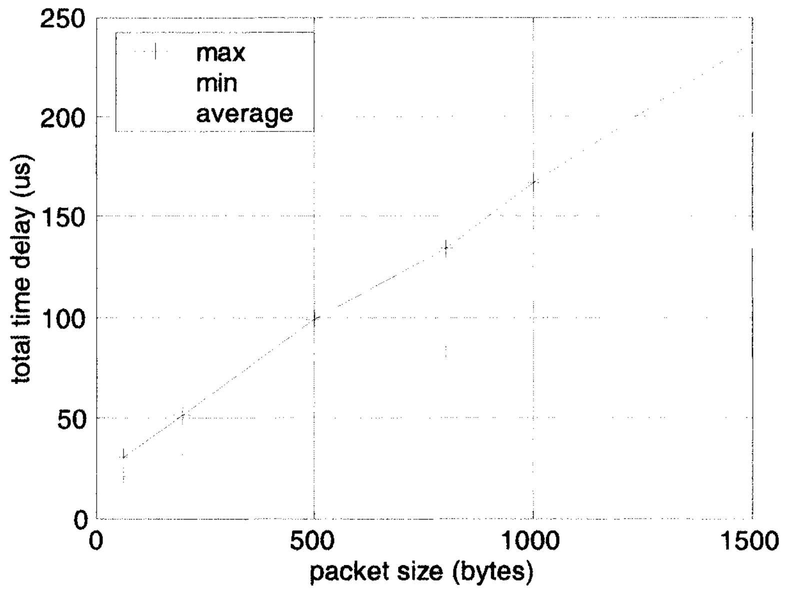


Figure 5-27 Test 10-b results of NETGEAR DS108 10/100 hub

Table 5-6 Result of test 10-b, NETGEAR DS108 10/100 hub

Packet size (bytes)	64	200	500	800	1000	1500
Tdtest of crossover cable average (µs)	23.6	39.4	76.7	111.2	132.3	200.7
T_{dtest} of NETGEAR DS108 hub average (µs)	24.9	42.9	81.9	115.8	138.9	197.0
Hub time delay average (µs)	6.42	19.5	45.2	68.6	86.6	117.7

Test 10-c, SMC-EZ5808DS 10/100 hub. Figure 5-28 shows the result and Table 5-7 gives the data.

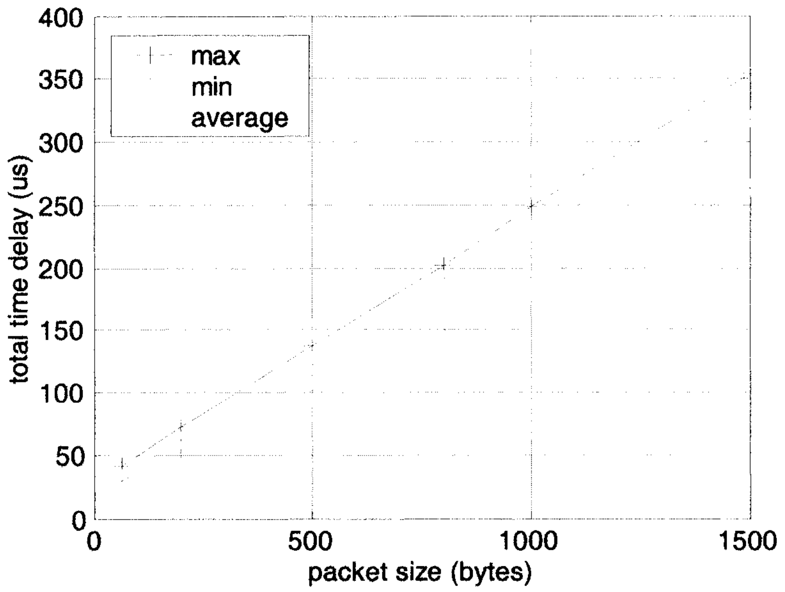


Figure 5-28 Test results of test 10-c SMC-EZ5808DS 10/100 hub

Table 5-7 Result of test 10-c SMC-EZ5808DS 10/100 hub

Packet size (bytes)	64	200	500	800	1000	1500
T_{dtest} of crossover cable average (μs)	23.6	39.4	76.7	111.2	132.3	200.7
T_{dtest} of SMC EZ5808DS hub average (μs)	32.4	61.1	122.2	191.6	231.2	317.6
Hub time delay average (μs)	13.92	37.7	85.5	144.4	178.9	236.9

Test 10-d, Linksys NH1005 V2 hub

LINKSYS NH1005 HUB summary, Test 10000 cycles, takes time 30000000.0 μs.

Figure 5-29 shows the result and Table 5-8 gives the data.

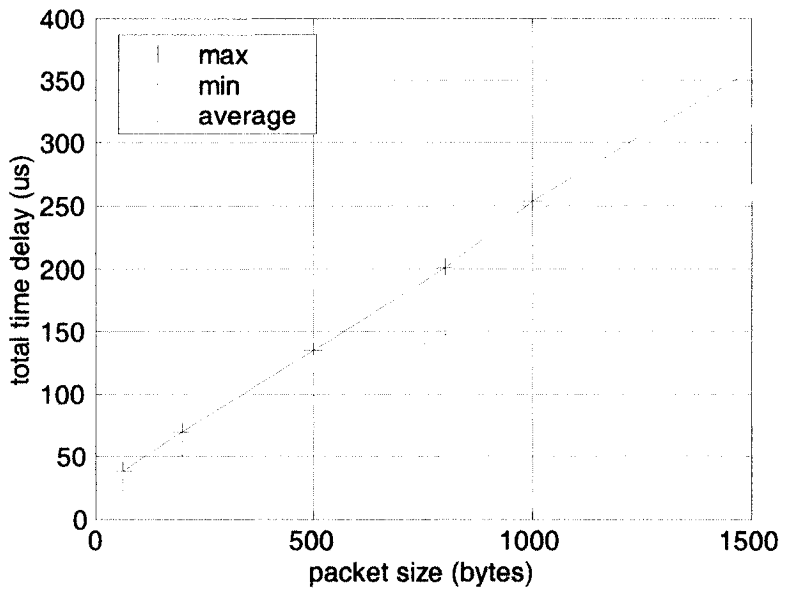


Figure 5-29 Test 10-d results of Linksys NH1005 V2 10/100 hub

Table 5-8 Result of test 10-d Linksys NH1005 V2 10/100 hub

Packet size (bytes)	64	200	500	800	1000	1500
T_{dtest} of crossover cable average (μs)	23.6	39.4	76.7	111.2	132.3	200.7
T_{dtest} of Linksys NH1005 V2 hub average (μs)	31.4	61.2	124.5	187.9	218.3	320.6
Hub time delay average (μs)	12.92	37.8	87.8	119.6	166	239.9

Test 10-e, 3Com 3C16477 Baseline gigabit switch

Figure 5-30 shows the overall of 3Com 3C16477 Baseline gigabit switch. Table 5-9 shows the data.

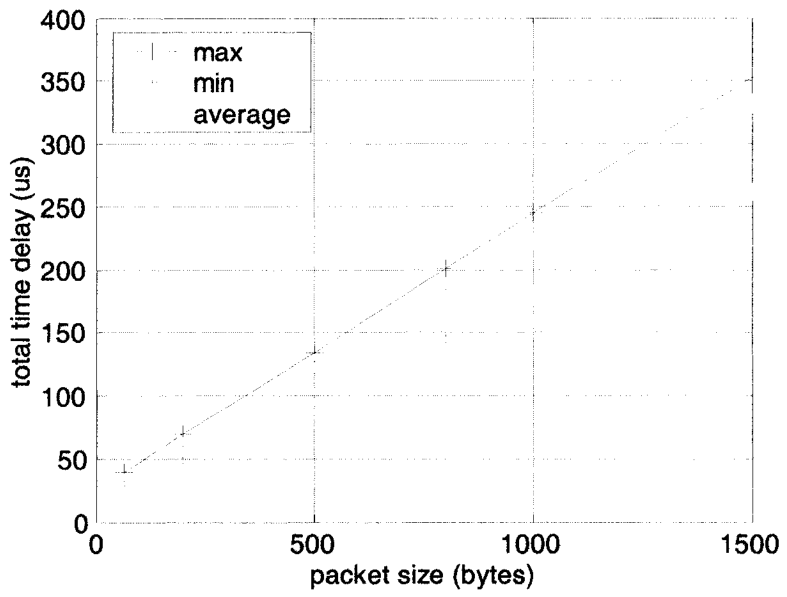


Figure 5-30 Test 10-e results of 3Com 3C16477 gigabit switch

Table 5-9 Result of test 10-e 3Com 3C16477 gigabit switch

Packet size (bytes)	64	200	500	800	1000	1500
T_{dtest} of crossover cable average (μs)	23.6	39.4	76.7	111.2	132.3	200.7
T_{dtest} of 3Com 3C16477 gigabit switch average (μs)	32.3	60.3	127.3	184.4	222.1	333.0
Switch time delay average (μs)	13.82	36.9	90.6	137.2	169.8	252.3

Test 10-f, NETGEAR FS108 fast Ethernet switch

The overall of NETGEAR FS108 fast Ethernet switch is shown on Figure 5-31. Table 5-10 provides the ODE and

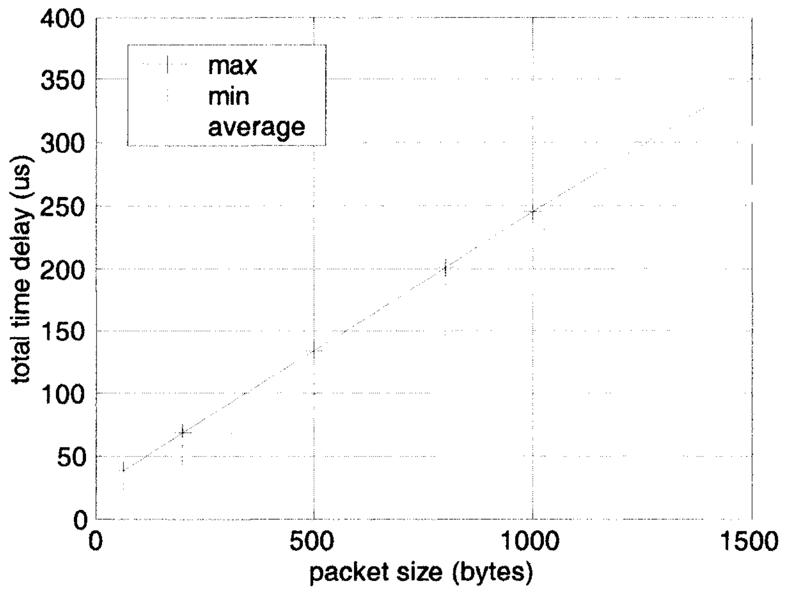


Figure 5-31 Test 10-f results of NETGEAR FS108 fast Ethernet switch

Table 5-10 Result of test 10-f NETGEAR FS108 fast Ethernet switch

Packet size (bytes)	64	200	500	800	1000	1500
T_{dtest} of crossover cable average (µs)	23.6	39.4	76.7	111.2	132.3	200.7
T_{dtest} of NETGEAR FS108 fast Ethernet switch average (µs)	30.5	57.6	114.8	186.6	225.9	306.4
Fast Ethernet switch time delay average (µs)	12.02	34.2	78.1	139.4	173.6	225.7

Summary of test 10:

Table 5-11 is the matrix of the different equipment. Figure 5-32, Figure 5-33, Figure 5-34 briefly show the results.

Table 5-11 shows the of equipment in test10

Equipment	E1	E2	E3	E4	E5	E6	E7	E8
Brand name	RealTek	N/A	3COM	NETGEAR	SMC	Linksys	3COM	NETGEAR
Equipment	NIC Tx	Cross over Cable	Hub, 100/10 Mbps	Hub, 100/10 Mbps	Hub, 100/10 Mbps	Hub, 100/10 Mbps	Switch, Gigabit	Switch, 100/10 Mbps

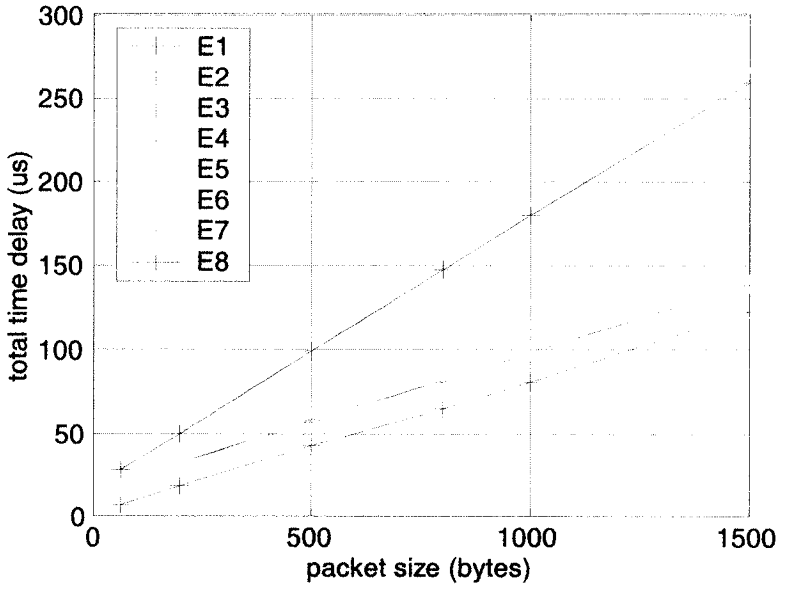


Figure 5-32 Test 10 summary - minimum time delay

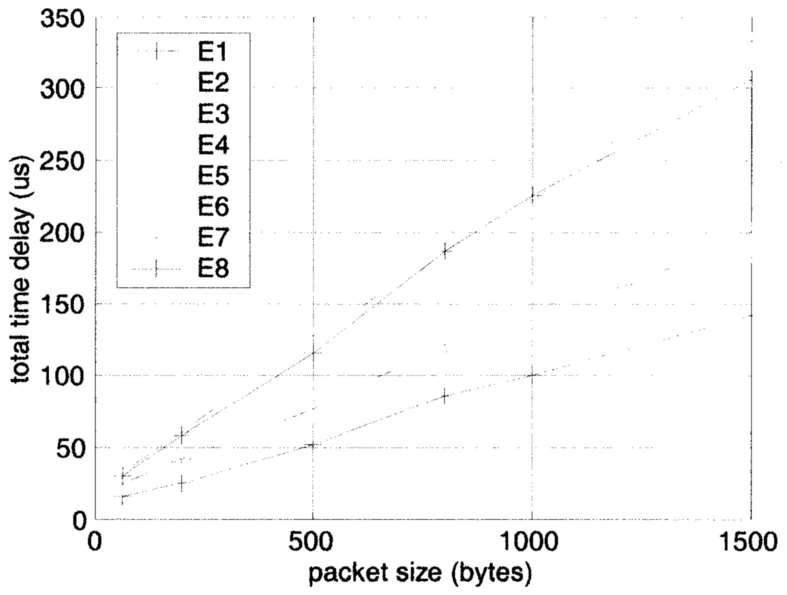


Figure 5-33 Test 10 summary - average time delay

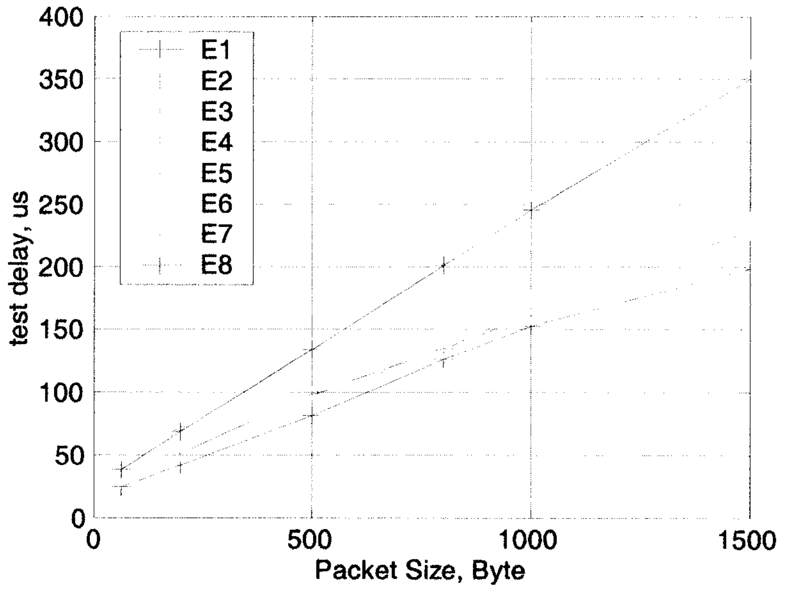


Figure 5-34 Test 10 summary - maximum time delay

5.6 Gigabit Ethernet NIC Latency

In this section, we use the same process to do the Gigabit Ethernet NIC latency tests as we did in Test 5 in the fast Ethernet section. Test 12 is the Winsock comparison test. Test 13 is a one-way latency test with parallel port feed back. By est 13 we can calculate the latency of the two Gigabit switches.

Test 11, Gigabit NIC on Winsock 2.0 one-way latency test with parallel port feedback

The operation steps are the same as with the Test 5 NIC one-way latency test with parallel port feed back. The media is a crossover cable. The only difference is that instead of using a real time driver, this test uses Winsock 2.0 to send out packets. Figure 5-35 shows the results of Test 11. It shows a very large jitter and a large latency.

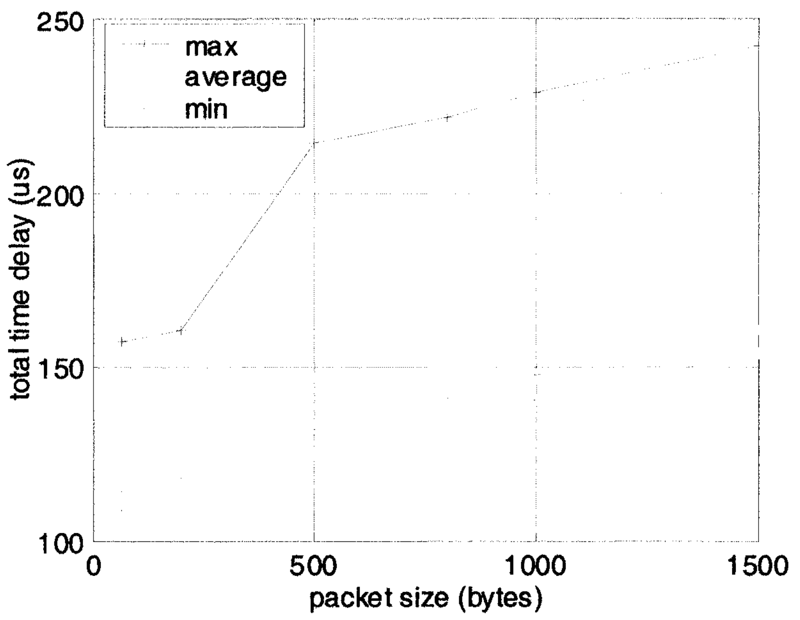


Figure 5-35 Test 11 results of Windows Winsock 2.0 UDP packet latency

Test 12, Gigabit NIC one-way latency test on real-time driver with parallel port feedback and switches latency test

All of the steps are the same as in Test 5. The NICs are D_link DGE-550T Gigabit NICs. The host computer is a Dell PowerEdge 600sc (for details please refer to appendix 2). Test 12-a is a crossover cable test, Test 12-b is a 3Com gigabit switch test, and Test 12-c is a Netgear gigabit switch test. By deducting the crossover cable average latency from the average latency of the switch test, we obtain the switch latency. Figure 5-25 illustrates the time delay calculation.

Test 12-a, Crossover cable test,

Figure 5-36 and Table 5-12 show the results.

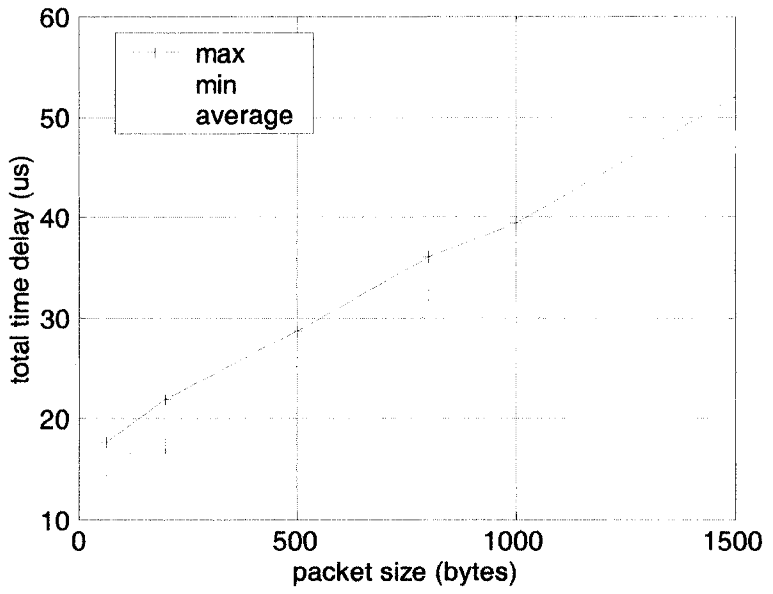


Figure 5-36 Test 12-a results of on D_link DGE-550T gigabit NIC with driver version 8 crossover cable feedback result

Table 5-12 Result of test 12-a average of crossover cable test

Packet size (bytes)	**64**	**200**	**500**	**800**	**1000**	**1500**
T_{dtest} of crossover cable average (μs)	15.6	18.0	26.0	32.7	37.5	48.1

Test 12-b, 3com switch test

Figure 5-37 and Table 5-13 show the results.

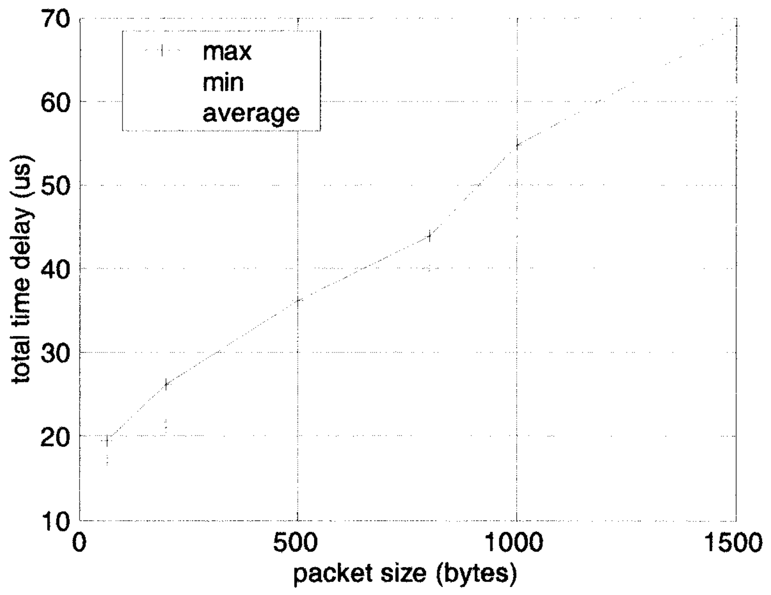


Figure 5-37 Test 12-b results of 3Com Gigabit switch

Table 5-13 Result of test 12-b, 3com Gigabit switch test

Packet size (bytes)	64	200	500	800	1000	1500
T_{dtest} of crossover cable average (μs)	15.6	18.0	26.0	32.7	37.5	48.1
T_{dtest} of 3com Gigabit switch average (μs)	17.3	21.6	31.7	40.3	45.1	61.3
Gigabit switch time delay average (μs)	2.212	5.2	9.7	14.0	15.6	25.2

Test 12-c, Netgear switch test

Figure 5-38 and Table 5-14 show the results.

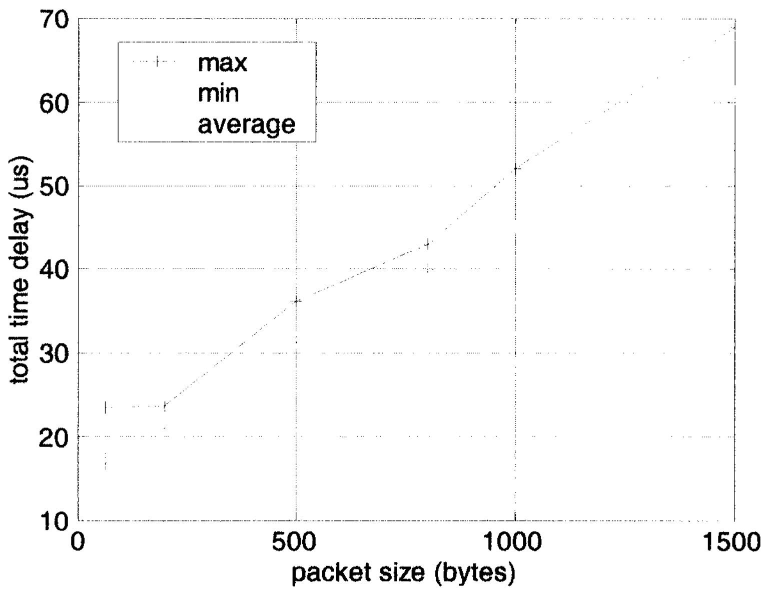


Figure 5-38 Test 12-c results of NETGEAR gigabit switch

Table 5-14 Result of Test 12-c NETGEAR gigabit switch test

Packet size (bytes)	**64**	**200**	**500**	**800**	**1000**	**1500**
T_{dtest} of crossover cable average (µs)	15.6	18.0	26.0	32.7	37.5	48.1
T_{dtest} of NETGEAR Gigabit Switch average (µs)	17.4	21.6	31.8	40.3	45.2	61.4
Gigabit Switch time delay average (µs)	2.312	5.2	9.8	14.2	15.7	25.3

Test 12-d, D_link DGE-550T gigabit NIC Tx time delay test.

Figure 5-39 and Table 5-15 show the results.

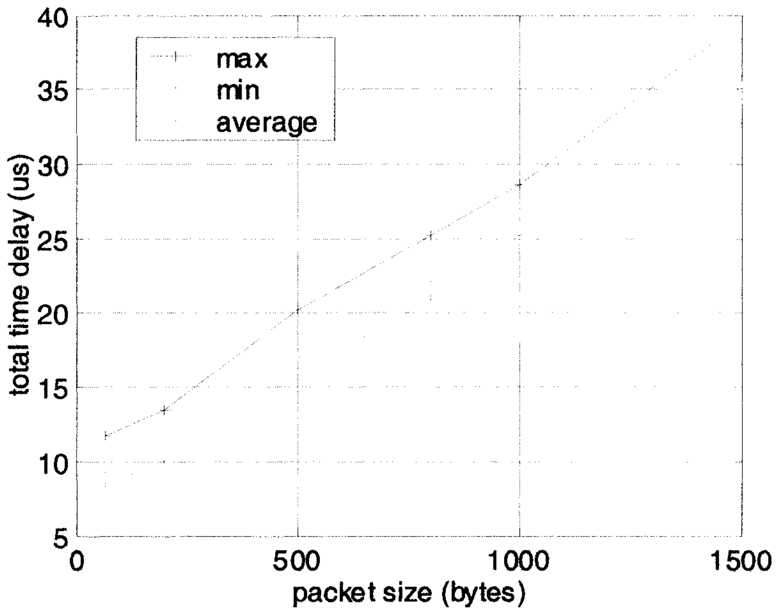


Figure 5-39 Test 12-d results of D_link DGE-550T gigabit NIC Tx time delay

Table 5-15 Test 12-d results of D_link DGE-550T gigabit NIC Tx time delay

Packet size (bytes)	64	200	500	800	1000	1500
$T_{dtest\,B}$ of Tx time delay average (μs)	9.22	11.57	17.01	22.13	25.89	35.15
Gigabit Ethernet T_{theory} = bits/1000Mbps (μs)	0.512	1.6	4.0	6.4	8.0	12.0
NIC Tx propagation time average (μs)	8.708	9.97	13.01	15.79	17.89	23.15
$T_{dtest\,A}$ of crossover cable average (μs)	15.6	18.0	26.0	32.7	37.5	48.1
$T_{dtest\,A} - T_{d\_p}$	12.93	15.33	23.33	30.03	34.83	45.43
NIC Rx propagation time average (μs)	3.71	3.76	6.32	7.9	8.94	10.28

The total time delay includes NIC Tx propagation time, packet-sending time, hub or switch propagation time, NIC Rx propagation time and parallel port feedback time.

Figure 5-40 shows the concept of the total time delay. Table 5-16 and Table 5-17 show the range of the numbers.

The NIC Tx propagation time is the time since the first signal came into the NIC until the first signal comes out of the NIC. From Test 7, demonstrated by Figure 5-21 and Table 5-3, the RTL8139 NIC Tx propagation time is from 8.9 μs to 22.1 μs according to the different packet size. Figure 5-39 and Table 5-15 show the results of the D_link DGE-550T gigabit NIC Tx propagation time delay, which varies from 9.22 ~ 35.15 μs.

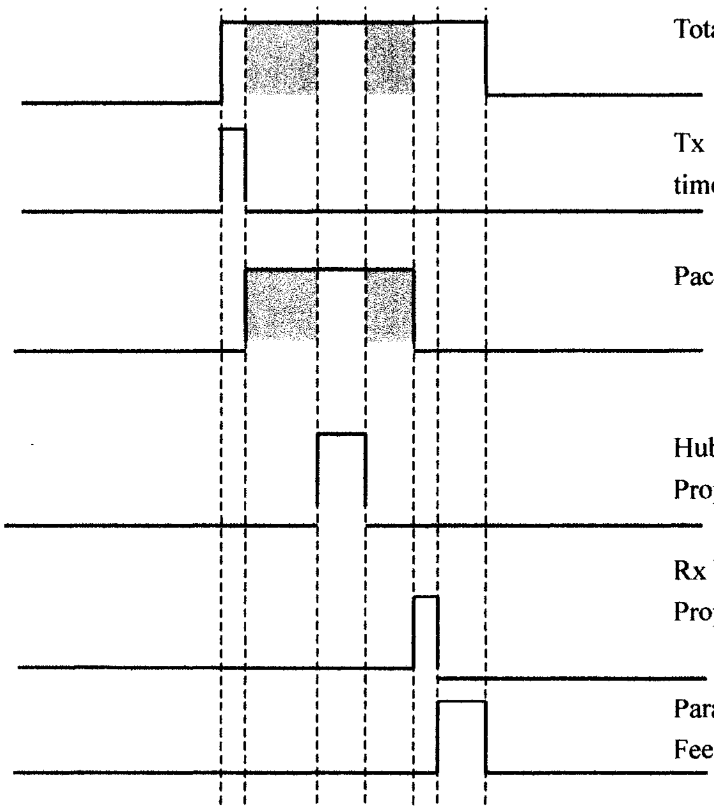


Figure 5-40 Diagram of total time delay analysis

Packet sending time is the time NIC spends to send the packet. This time can be theoretically calculated by the packet size and sending speed of the NIC. Table 5-17 shows the theoretical sending time.

Hub or switch propagation time indicates how long it takes the first signal to penetrate the hub or switch. This time has noticeable variations because the hubs and switches use different algorithms. Table 5-5 to Table 5-12, and Figure 5-26 to Figure 5-34 show the results of fast Ethernet RTL8139 NIC on different hubs and switches. Table 5-13 to Table 5-14 and Figure 5-37 to Figure 5-39 show the results of a gigabit Ethernet D_link DGE-550T on different hubs and switches.

The Rx NIC propagation time indicates how long it takes the last packet to penetrate the NIC. Because the total time delay is known and all of the other parts of time delay are known, this delay can be easily calculated. Table 5-3 shows the results of the Rx NIC propagation delay in Test 7.

Parallel port feedback time is measured in Test 4-c. Figure 5-13 and Figure 5-15 show the results of parallel port feedback time. The average is about 2.67 μs.

Table 5-16 Total time delay analysis

Ethernet	Total time delay (μs)	Tx NIC propagation time (μs)	Packet sending time (μs)	Hub or Switch Propagation time (μs)	Rx NIC Propagation time (μs)	Parallel port Feedback (μs)
Fast Ethernet	24.8 ~ 333.6	10.78 ~ 22.1	5.12 (64 bites) ~ 120 (1500 bites)	1.3 ~116.9	1.63 ~ 52.53	2.67
Gigabit Ethernet	17.3 ~ 61.4	8.7 ~ 23.15	0.512 (64 bites) ~ 12 (1500 bites)	2,21~25.3	3.69 ~ 10.26	2.67

Table 5-17 Theoretical packet sending time

Packet size	64	200	500	800	1000	1500
Fast Ethernet T_{theory} = bits/100Mbps (μs)	5.12	16	40	64	80	120
Gigabit Ethernet T_{theory} = bits/1000Mbps (μs)	0.512	1.6	4.0	6.4	8.0	12.0

6 TDMA Protocol Development

After the real-time driver is developed and tested, the next step is to develop a higher-level protocol on Ethernet. This chapter illustrates the TDMA protocol, the clock synchronization on Ethernet, the development of a TDMA protocol on Ethernet, and fault tolerance and communication with other processes. As demonstrated by the results of these tests, the clock synchronization and TDMA protocol work successfully and are capable of accomplishing the real-time control and simulation tasks.

6.1 Overview of TDMA Protocols

Time Division Multiple Access (TDMA) is a digital transmission technology that allows several users to access a single channel without interference by allocating unique time slots to each user within each channel as seen in Figure 6-1. In TDMA, time is cut into slots. Each node on the network has to send out messages during its own unique time slot. This technology was broadly used in the telecommunications industry, for example with satellite communications. Now TDMA will be implemented on the Ethernet for simulation and control purposes. Fig 6-1 shows the concept of TDMA on the Ethernet network.

Clocks determine time slots. However, the clock on each computer has a drift. To avoid the interference on the time slot, the clocks should be synchronized. One solution is to use a separate time synchronization channel such as a parallel port to synchronize the clock. The other solution is to use the packet on the network as a synchronization signal. This solution does not need an extra channel for synchronization. Both solutions will be implemented.

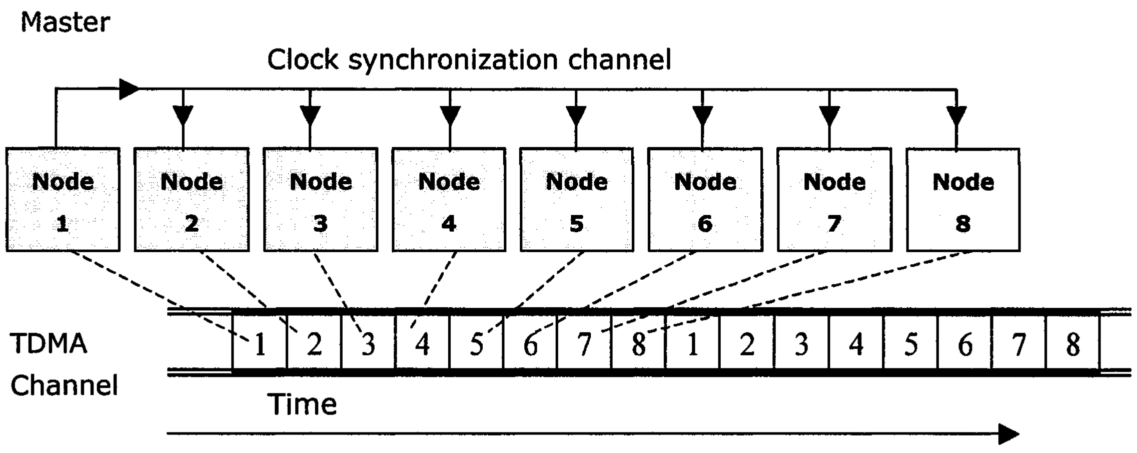


Figure 6-1 TDMA concept

6.2 Slot Size Determination

A TDMA period is the cycle time of the TDMA protocol. The slot size and the number of the slot determine the TDMA period. Three issues need to be considered when designing the slot size and are referred to in Figure 6-2. The first issue to consider is the packet transfer time. According to theory, the transfer time varies according to the packet size. From the experiments, in the same packet size there is a jitter in transfer time. To guarantee that there is no collision on the network, the maximum transfer time is used. Compared with the fast Ethernet, Gigabit Ethernet NIC can achieve a much smaller slot size for the same packet size. As witnessed in the experiments, the D-Link gigabit NIC spends less time sending and has much less jitter than the RTL8139 fast Ethernet NIC. Therefore, the slot size can be significantly reduced by using Gigabit Ethernet. Reducing packet size can also reduce the slot size. However, the header of the packet cannot be reduced, and the smaller the packet size, the less efficient the communication is.

The second issue is the operating system jitter. Because the operating system and the protocol program have jitters, a buffer time is needed to avoid the overlap on the time slot.

The third issue is clock drifting. Because clocks drift, the time is not exactly the same among the nodes. Even after the synchronization, there is still a small amount of drifting. Fig 6-2 shows the three issues of TDMA slots.

TDMA slot		
Max packet transfer time	Buffer time	Clock drift

Figure 6-2 TDMA slots

TDMA slot size = max packet sending time + buffer time + clock drift

Table 6-1 shows the slot size vs. packet size for RTL8139 NIC on a 3Com OfficeConnect hub. For example, when the packet size is 64 bytes, the actual maximum packet transfer time is 30 μs as shown in Figure 5-28. Now, we assume that the clock drift is 20 μs, and the buffer time is 20 μs. The slot size for this configuration is 70 μs. If we increase the packet size, the packet transfer time is increased, but the clock drift and buffer time remains the same.

Table 6-1 TDMA slot size with RTL8139 NIC

Packet size (bytes)	**64**	**200**	**1000**	**1500**
Slot size (μs)	70	90	200	270

6.3 Clock Synchronization

A TDMA protocol needs synchronized clocks because clock drift can damage protocol. There are many ways to synchronize the clocks through networks, including hardware modifications on the fast Ethernet NIC [30], or by using a network that has a time stamp feature [31]. In this project, our target is to use Ethernet NIC on the market without modifications to synchronize the nodes on the network.

The clock time on different computers has two kinds of differences. The first is offset difference, and the second is drifting rate difference. Figure 6-3 shows the offset and rate difference between two clocks. A clock drift test is designed to examine the drift between clocks on different computers. The clock drift can be observed in the following way.

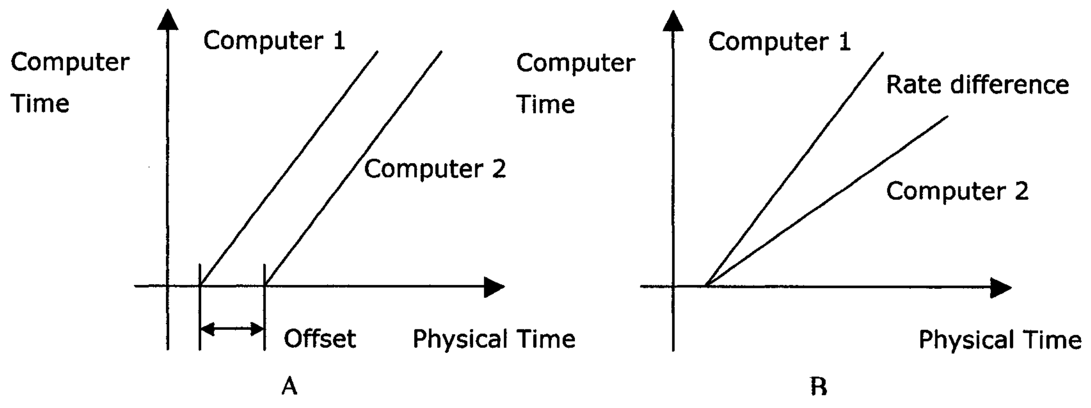


Figure 6-3 Clock drift

Test 13 Clock drift measurement: Recording 4 computer's clock drift rate

Operation:

Step 1, all of the computers send out a pulse to channel 1 of a parallel port per millisecond by their timer.

Step 2, one computer works as a parallel port monitor, which reads all of the pulses and records the time when a pulse comes according to its own clock.

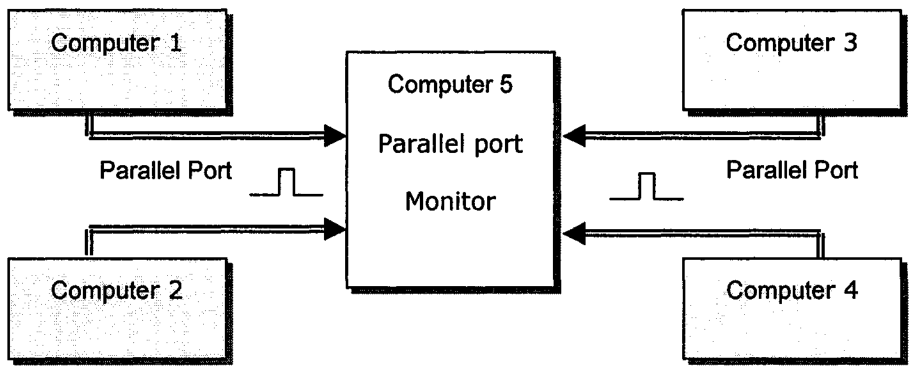


Figure 6-4 Clock drift rate test set up

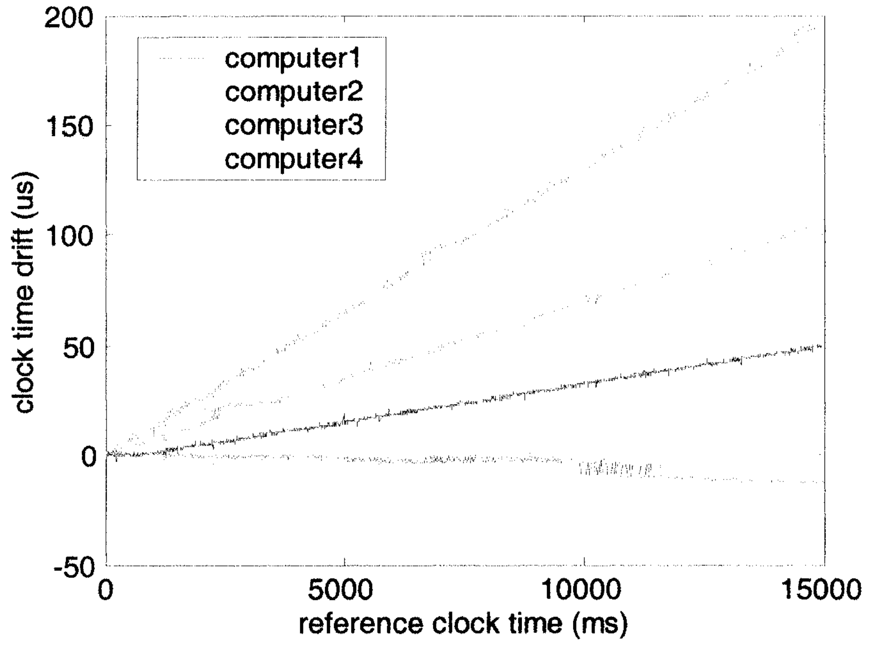


Figure 6-5 Clock drift after offset compensation on DELL Dimension 8250

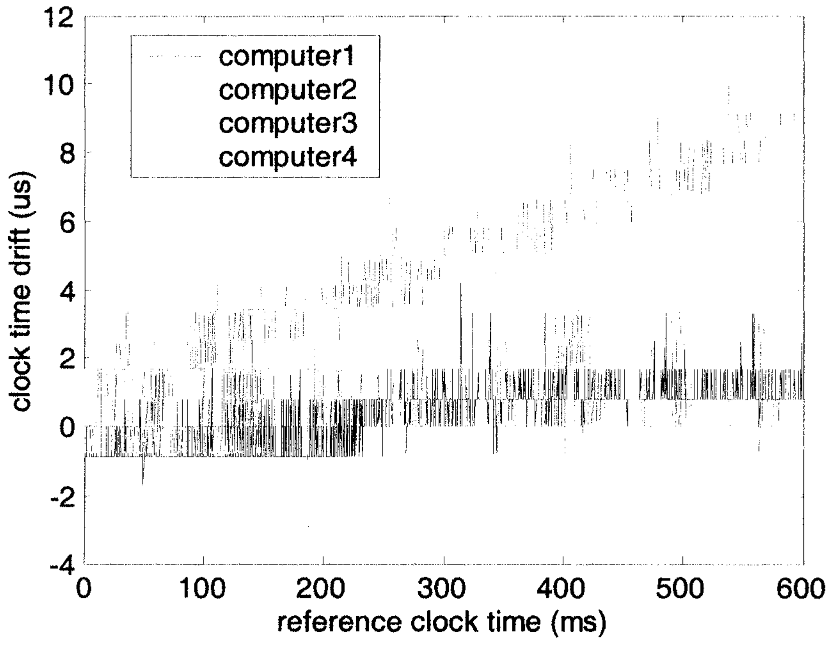


Figure 6-6 Zoom clock drift after offset difference compensation on DELL Dimension 8250

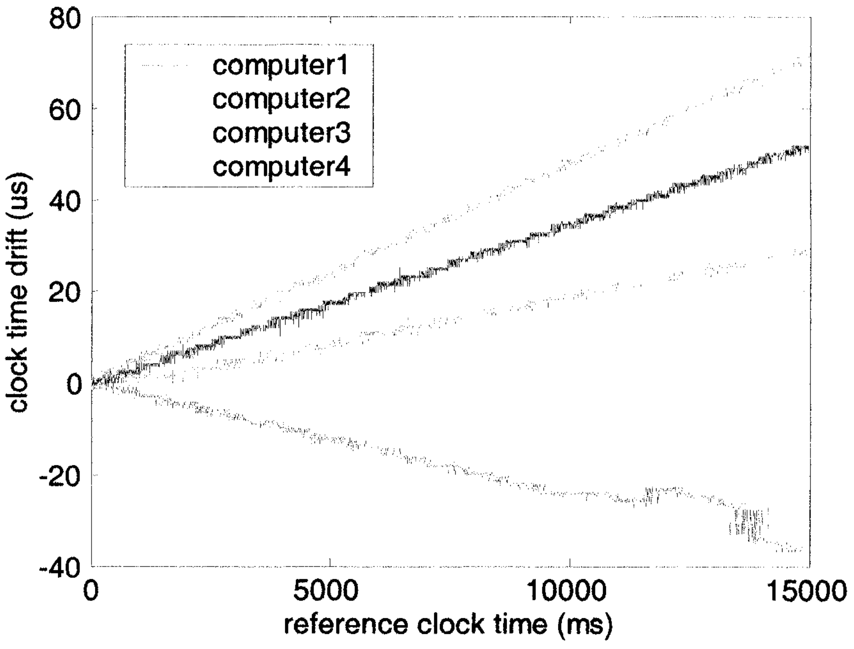


Figure 6-7 Clock drift after offset compensation on Dell PowerEdge 600SC

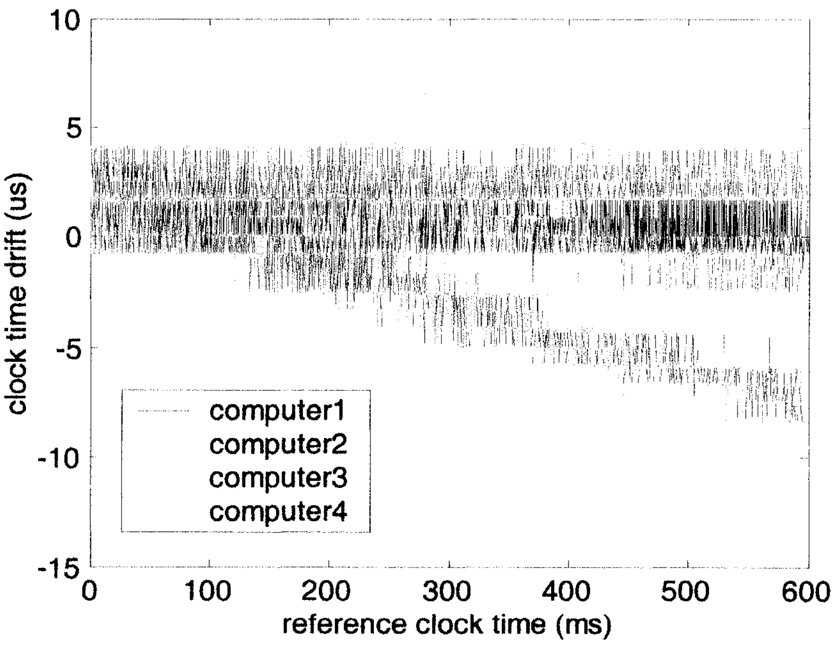


Figure 6-8 Clock drift after offset compensation on Dell PowerEdge 600SC

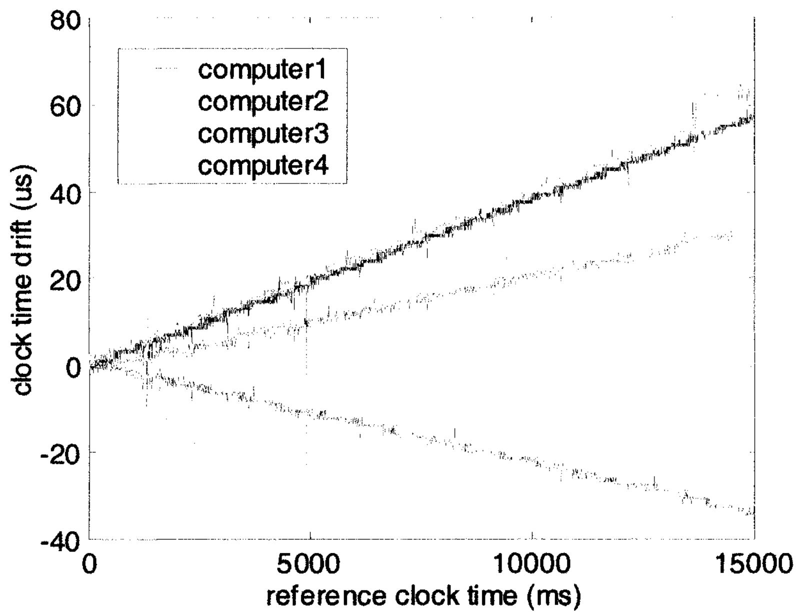


Figure 6-9 Clock drift after offset compensation on D_link DGE-550T gigabit NIC timer installed on Dell PowerEdge 600S

Figure 6-4 shows the test set-up. Figure 6-5 to Figure 6-9 show the drifting rate on different computers after offset compensation. In Figure 6-5, the four computers are all Dell dimension 8250s; please refer to appendix 2 for hardware configuration. As time passes, the computer clocks drift away. At time 15s, the maximum difference between computer 2 and computer 3 is about 205µs. The maximum drift rate is

205/15 = 13.6 µs/s or 13.6 ns/ms. 13.6e-6 = 0.0013.6%

The RTX system clock also has an error from the Windows clock. In an RTX clock, a 500 µs period is 499.504825 µs in actual [14]. When using an RTX timer to generate a one-millisecond square wave, this one millisecond square wave actual interval is 999.00965 µs. Because all of the nodes are running on the RTX system, this error is ignored.

The clock synchronization is part of the TDMA Ethernet protocol. The purpose of it is to prevent clocks from continually drifting away. In this TDMA Ethernet, the period of a TDMA cycle is around several ms. In such a short time, the clock drift among computers is very tiny, about 5 to 20 ns/ms . However, if clocks continually keep drifting, eventually the drift will become large enough to break the protocol. Clock synchronization in TDMA Ethernet is to limit the drift to a specified amount, which can be tolerated in TDMA protocols.

In the RTX operating system, the minimum adjustment unit for an RTX timer is 100 μs. Any change less than 100 μs will be considered as if it is 100 μs. For example, an interval of 10001 μs will be considered 10100 μs, and an interval of 1099 will be also considered 10100 μs. The adjustment between 1 and 99 has no effects on the timer interval. On the other hand, the clock synchronization adjustment is as small as 20ns/ms. Therefore, the RTX timer cannot be used for clock synchronization purposes. One of the solutions is to poll the clock to get the accurate time. However from the test, the clock reading time in RTX is an average 3.4 μs, which is also a large number for clock synchronization.

The clock synchronization concept is simple. After some TDMA cycles, a clock synchronization slot is added, as shown in Figure 6-10. All clocks are synchronized at this slot. After this synchronization slot, a synchronization cycle is finished. A synchronization cycle has n TDMA cycles. N is an integer. With this concept, we use two ways to synchronize the clock. The first way is by using a parallel port; the second is by using packets on the network.

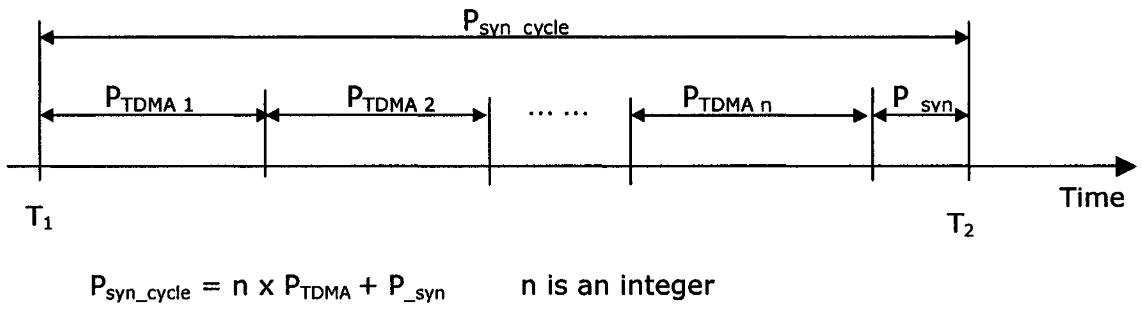


Figure 6-10 Clock synchronization in TDMA cycle

Clock synchronization using a parallel port is easier to implement. From the parallel port test, we found that it has a constant and fast performance. In parallel port synchronization, the server (master node) generates a synchronization square wave through the parallel port. The raising edge of the square wave is the time signal. The clients synchronize their clocks by this signal. In this design, a client does not adjust a real clock. It simply adjusts a virtual clock, which is a variable. The client reads the computer clock, and compares it with the square wave. Then the client calculates the time between the two signals and uses this time as the interval of synchronization cycle. Figure 6-11 shows the result of parallel port synchronization. Observing from the oscilloscope, the two signals are static, indicating that they are synchronized.

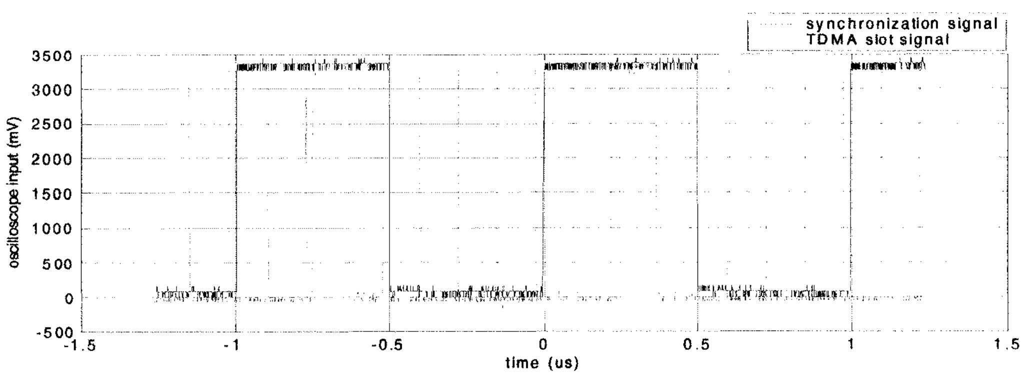


Figure 6-11 Result of parallel port synchronization.

The second way to synchronize clocks is by using the Ethernet and packet, shown in Figure 6-12. The idea is similar to the parallel port synchronization. The server sends out a time synchronization packet at the synchronization slot. The client receives this packet, stamps the time, and uses that time as the starting time for a new period. When this TDMA cycle finishes, the client is waiting for the coming synchronization packet and adjusts the clock according to the arrival time of the synchronization packet. This method does not need extra parallel port connections on the computers. Compared to the parallel port, the error in this packet synchronization is coming from the jitter of sending and receiving packets. From the previous test, NIC has a bigger sending jitter than a parallel port. This jitter makes the period varying.

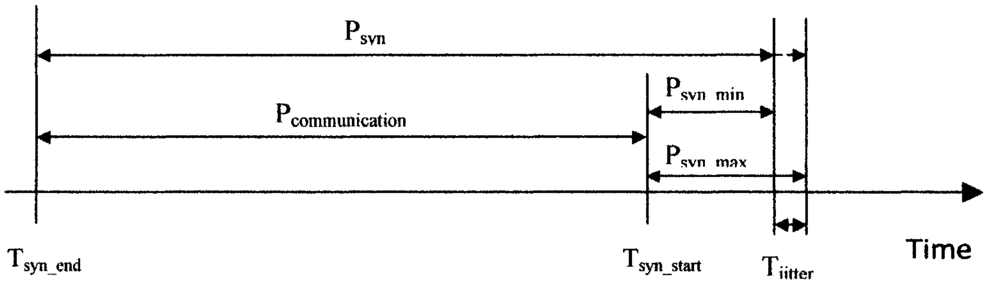


Figure 6-12 Clock synchronization by network packet

Adding a buffer time T_{buffer} can solve this problem. The buffer time is fixed and is slightly longer than the maximum synchronization packet sending time. After this fixed buffer time, the period is finished and fixed. The client should also wait until the buffer time is complete, and then check whether the packet has been received. This check may cost a little time, which also acts as an adjustment period for clock drift. As shown in Figure 6-12, a client's $P_{\_syn}$ is a little bit shorter than the server's because the receiving packet also has a jitter. The P_{syn_client} is not accurate. However, it prevents the clock from continually drifting.

Let's use two scenarios to illustrate how this method works. Please refer to Figure 6-13. In scenario A, the client's drift advances, which means that the client's tick is shorter than the server's. In this situation, the client will wait for the synchronization packet before it goes to the next cycle. The signal prevents the client from starting the TDMA cycle too early to follow the protocol.

In scenario B, the client's drift lags, which means that the client's 1 unit is longer than the server's. Because the P_{syn_client} is shorter than the P_{syn_server}, the client still has to wait for the synchronization signal, with which the client is synchronized. With this buffer time, the synchronization should perform very well. During implementation, this method has to be put into service very carefully. In the real world, the server's synchronization cycle time and the client's are very close. There is no room for any unnecessary statement. Even a clock reading statement can ruin the synchronization because in RTX, the time delay for a clock reading is an average of 3.4 μs, which is much longer than the clock drift in a synchronization cycle. Any extra statements between the end of a cycle and next clock reading could make TDMA unsynchronized in the packet synchronization.

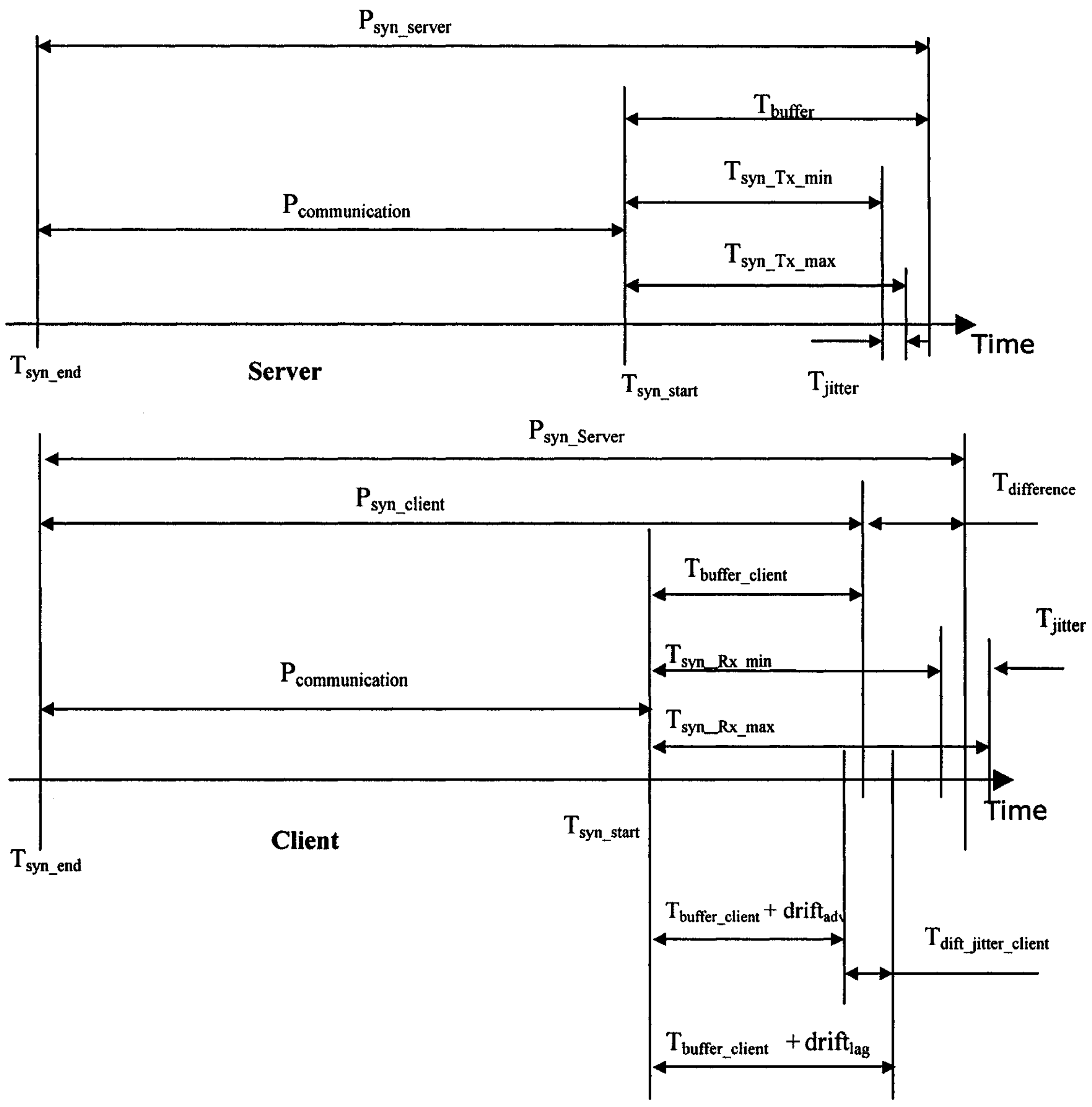


Figure 6-13 Analysis of clock synchronization by network packet

Measuring, observing and testing:

To measure a real-time system, real-time instruments and programs are needed. In this project, the Tektronix TDS 210 two-channel real-time oscilloscope [33] is one of the main instruments used, and has a 1GS/s sampling rate. As discovered in Chapter 5, the combination of RTX and a parallel port has good performance in terms of accuracy, small delays and consistency, which make them capable to measure other programs. The parallel ports are used to send out signals when the event happens, the oscilloscopes are used to observe, and a spare computer is used to store the data.

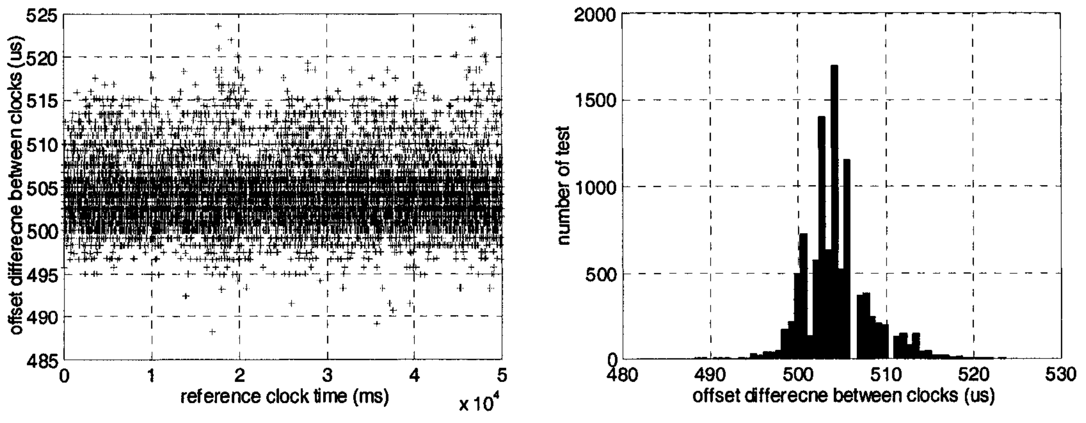


Figure 6-14 Offset difference between server and synchronized nodes clocks.

Figure 6-14 shows the time difference between the server and synchronized nodes. The test hardware is as follows: Dell Dimension 8250 desktop for nodes, RTL8139 Ethernet NICs and a NETGEAR DS108 Hub for the network. From the figure, the clock synchronization limits the clock's drift. This means that the packet synchronization works well. The synchronization jitter from the figure is 38 μs, which includes several parts: 1) the parallel port jitter of 7.5μs (from Figure 5-12 and Figure 5-14), 2) the test's double clock reading jitter of 10μs (from Figure 5-3), 3) the one-way NIC Tx, Rx, and hub jitter (from Figure 5-29) of 12.5 μs, and 4) the TDMA protocol clock reading jitter of 5 μs. In these jitters, items 1 and 2 are the test jitters while items 3 and 4 are the synchronization jitters. Therefore, the actual synchronization jitter is 38 -7.5 - 10 = 20.5 μs.

6.4 Implementation of TDMA Protocol

6.4.1. TDMA Protocol

After the clock is synchronized, TDMA protocol is easy to implement. Each node is configured by a file that includes information for the TDMA cycle time, slot number, maximum slot number, and synchronization buffer time. The node sends and receives packets according to its clock.

In pure and perfect TDMA protocol and applications, the application knows when and where the packet comes from. There is no need to use an ID to mark the packet. However, for the fault tolerance and some complex enhancements, source and destination ID is necessary. Figure 6-15 is a suggestion for a TDMA packet frame. The destination and source ID can be used in bit or byte type and the type section in the TDMA frame is reserved for some special packets proposed for future development.

Number of the octets	2	2	1	41~1495
	TDMA Destination ID	TDMA Source ID	Type	Data/Pad

Figure 6-15 TDMA frame

<u>Testing on a multi-node system (up to 8 computers on a switch/hub)</u>

A real-time TDMA Ethernet network with 8 computers is tested. It has two groups of results on hub or switch connections. The results show that the TDMA protocol and clock synchronization work very well.

6.4.2. Communication With Other Programs

TDMA protocol has to communicate and exchange data with the upper layer protocol and program. Shared memory is used to communicate between a driver and a higher-level protocol. Two pieces of shared memory are used as a sending (Tx) buffer and a receiving (Rx) buffer, as shown in Figure 6-16 and Figure 6-17. They work as a full-duplex channel.

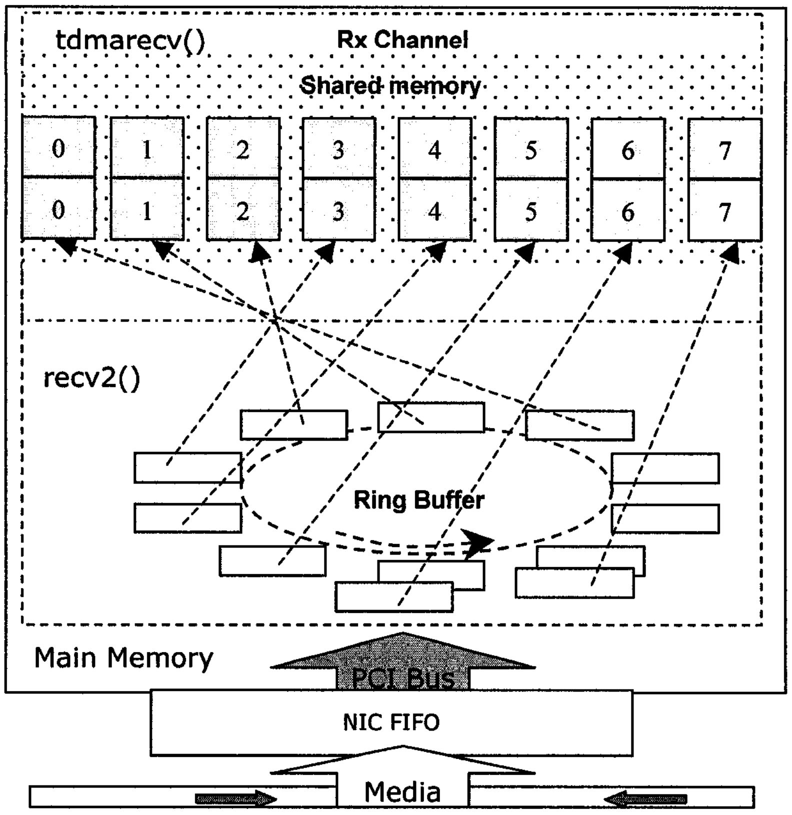


Figure 6-16 Rx channel

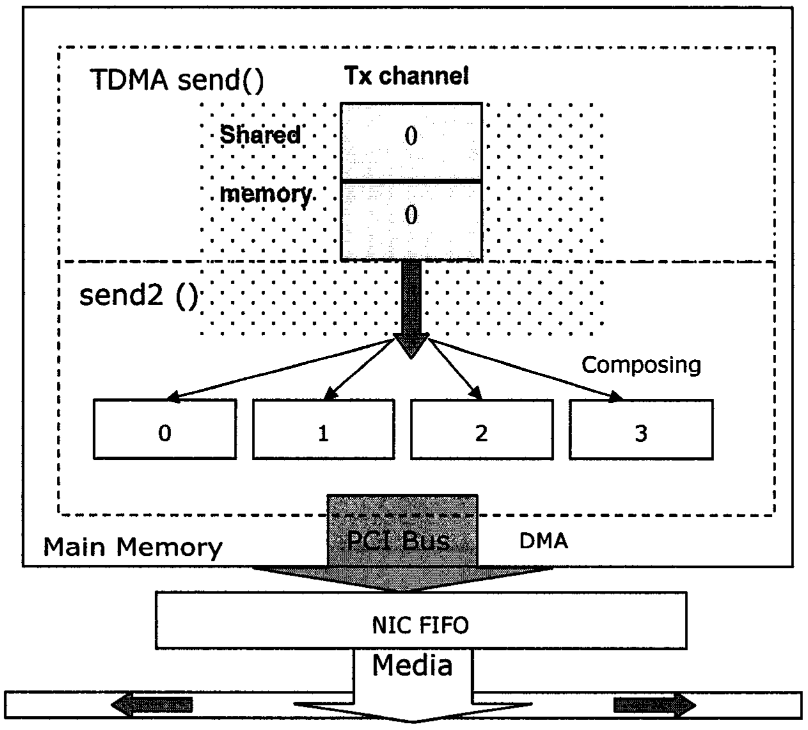


Figure 6-17 Tx channel

6.4.3. Fault Tolerance

This TDMA protocol has to tolerate or detect many faults that occur from erroneous operations, hardware breakdowns or software crashes. The following are some possible faults and how this TDMA protocol deals with them.

a) Ethernet collision

The goal of the TDMA protocol is to eliminate Ethernet collisions where all the nodes are scheduled to send packets at different times. If the TDMA works normally, there should be no collision. In some cases, even if one of the nodes prolongs its sending time to the next slot, the next node will still listen to the network before sending packets in order to avoid collision. If an Ethernet collision happens, the server will sound an alarm, and the network and TDMA configuration will need to be checked.

b) Missing node

A simple fault detection method is implemented and helps to find missing nodes easily. Each node has to send a message in its time slot. If it has no message to send, it sends a 64 byte dummy message to tell the other nodes that it is alive. The server listens to the network, and if there is no message in a time slot for several cycles, the server will consider the node in this time slot dead, and sound an alarm to the operator.

c) Missing synchronization

A missing synchronization could occur if the master node breaks down or simply if a faulty sending or receiving of a packet occurs. If a client does not receive a synchronization packet in time, it will record it and continue. Because the drift is very small, all nodes are still synchronized. If after several synchronization cycles the client still does not receive the synchronization packets, it will assume that the server is down, and it will sound an alarm to the operator. During this time, it shifts the network from TDMA protocol to normal mode to maintain the fault diagnosis.

d) Transmission out of turn

If clock synchronization accuracy is lost, or the clock in one of the node faults, the node could send packets into a wrong slot. Using node ID could easily detect this situation. Any node (especially the server node) receiving a TDMA packet will check the TDMA ID. If the ID is not supposed to be in this time slot, the node will sound an alarm to the server and ask to reset the network.

e) Missing packet

There is a possibility that there will be missing packets because of the network hardware. If this happens, protocol should detect the problem. However, in a real-time system, there are fewer opportunities to ask for a resend. Detection could be accomplished by adding a serial number or counting the packet numbers. A redundant network could recover the missing packet.

f) Overloaded network

Because TDMA protocol schedules all of the sending times, and the maximum packet size is limited, the network cannot be overloaded when the TDMA and clock synchronization works well. When the network is overloaded, it will be because of some other phenomena. The server will watch the network. If some strange packets appear, it will sound an alarm. Also, because the network flow is controlled in TDMA, network flow control is unnecessary. However, a node can be overwhelmed because it gets more data than its processing ability. This can only be adjusted by the application.

6.5 Performance

The TDMA protocol takes CPU time to process. The CPU overhead test will show us the percentage of the CPU time for TDMA protocol and driver. This test uses a low priority process to occupy all the time that is left. By comparing the result with and without TDMA protocol, the TDMA overhead can be calculated. Table 6-2 shows that the larger the slot sizes, the less overhead it takes. When the slot size is 100 μs, the CPU is almost fully used by TDMA protocol. The main reason for this is that the protocol has to constantly poll the clock for any time less than 100 μs, as discussed in the previous chapter. In this case, a TDMA slot size must be large than 100μs. However, there is a way to reduce the overhead. Instead of the polling clock, a timer can be used that generates an interval of less than 10 μs without CPU utilization (needs operating system support). In this way the overhead on clock polling will be reduced dramatically. Table 6-3 shows the estimated transfer rate based on the RTL8139 NIC and 3ComOfficeConnect 10/100 hub. Table 6-4 shows the matrix time delay on a number of nodes and slot sizes.

Table 6-2 CPU overhead

	Slot size (μs)			
TDMA Cycle time (us)	**100**	**125**	**250**	**500**
1000	96.7% (10 slots)	64.7% (8 slots)	48.7% (4 slots)	33.9% (2 slots)
2000	99.0% (20 slots)	64.3% (16 slots)	45.2% (8 slots)	32.2% (4 slots)

Table 6-3 Total data transfer rate (100 Mbps media)

	Slot size (μs)			
TDMA Cycle time (us)	**100** **(200 bytes)**	**125** **(500 bytes)**	**250** **(1500 bytes)**	**500** **2X(1500 bytes)**
1000	16.1 Mbps	32Mbps	48 Mbps	96 Mbps
2000	16.1 Mbps	32Mbps	48 Mbps	96 Mbps

Table 6-4 Network delay

	Slot size (μs)			
Number of slots	**100**	**125**	**250**	**500**
2	200	250	500	1000
4	400	500	1000	2000
8	800	1000	2000	4000

7 Application to Distributed Systems

In Chapter 7, two application examples are developed and tested on this real-time Ethernet based TDMA network. The first example is a pure software distributed simulation system, which compares a mass-spring system simulation working on a single computer, and working with it on two synchronized computers connected by the real-time Ethernet based TDMA network. The second example is a hardware-in-the-loop simulation system, which compares a local motor speed PID controller to a distributed PID controller.

In the two applications, the Ethernet based real-time network provides a deterministic communication among the computers. If the two processes are synchronized perfectly, there should be no difference in the result. If the two processes are not synchronized, the network changes some characteristics of the entire system. As a result, the network characteristic has to be considered in the control system.

7.1 Distributed Real-time Simulation

Mass - spring models are one of the simplest type of mechanical systems. In the single computer system, the equations are computed on one computer to study the behaviour of the mass-spring system. On the real-time Ethernet network, the equations can be put onto two computers. If the two computers are synchronized well, the result should be the same as on one computer. This architecture can be extended to large-scale systems. Figure 7-1 shows the mass spring system and Figure 7-2 shows the forces on mass 1 and mass 2.

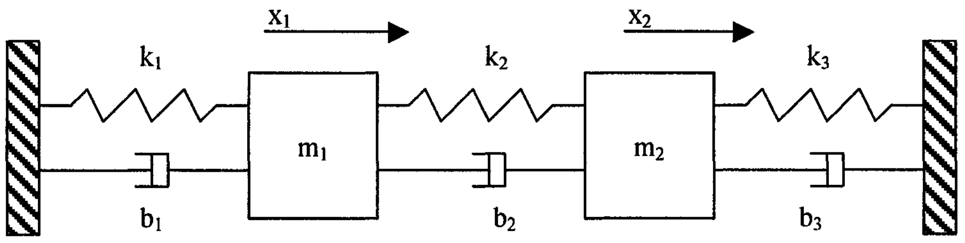


Figure 7-1 Mass spring system

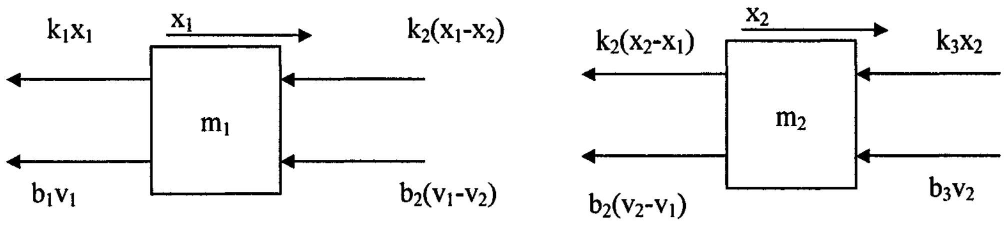


Figure 7-2 Force analysis of mass spring system

The equation of the system is the following.

$$m_1\dot{v}_1 = -k_1x_1 - b_1v_1 - k_2(x_1 - x_2) - b_2(v_1 - v_2) \qquad \textbf{(1)}$$

$$\dot{x}_1 = v_1 \qquad \textbf{(2)}$$

$$m_2\dot{v}_2 = -k_3x_2 - b_3v_2 - k_2(x_1 - x_2) - b_2(v_2 - v_1) \qquad \textbf{(3)}$$

$$\dot{x}_2 = v_2 \qquad \textbf{(4)}$$

For the purpose of comparison, we use the Euler method (Equation 5 and Equation 6) to solve the equations on both simulations. The parameters are listed in Table 7-1.

Table 7-1 Simulation parameters

Name	**m1**	**x1**	**v1**	**M2**	**x2**	**v2**
Value	0.1 kg	5 m	0 m/s	0.1 kg	0 m	0 m/s
Name	**k1**	**k2**	**k3**	**b1**	**b2**	**b3**
Value	1.0 N/m	1.0 N/m	1.0 N/m	0.1 Ns/m	0.01 Ns/m	0.01 Ns/m
Name	**ts**	**te**	**Δt**			
Value	0 s	1 s	0.001 s			

In this real time simulation, one simulation step is 1ms, which means **Δt** = 1ms. The total simulation time is 1 second. The result is sampled in every simulation cycle of 1ms. The number of the total sample is 1000 points. The single computer simulation results are in Figure 7-3 and Figure 7-4.

$$x(t+\Delta t) = x(t) + \dot{x}\Delta t \qquad \textbf{(5)}$$

$$v(t+\Delta t) = v(t) + \dot{v}\Delta t \qquad \textbf{(6)}$$

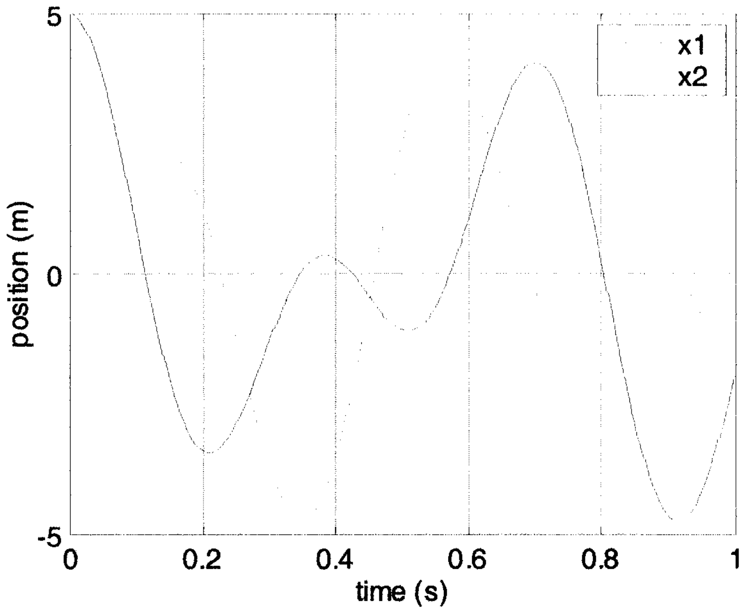


Figure 7-3 Position of single computer simulation

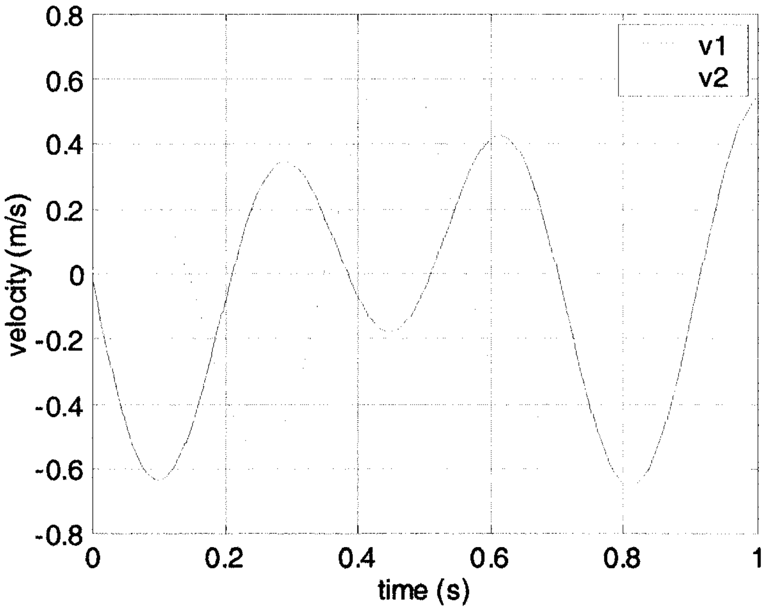


Figure 7-4 velocity of single computer simulation

Distributed control system simulation design:

In this distributed simulation, mass-spring system 1 is simulated on node 1; mass-spring system 2 is simulated on node 2. Node 1 and node 2 communicate with each other through the Ethernet based real-time TDMA network. In the distributed simulation, the simulation steps remain the same as the single computer simulation.

Two experiments are designed to test the impact with the communication cycle change. Experiment A keeps the same simulation cycle time of 1ms, and the TDMA cycle time is also 1ms. This setting will ensure that there is no difference from the single computer simulation. Experiment B sets the simulation step time to 350 μs, $\mathbf{\Delta t} = 0.35\text{ms}$. The TDMA communication cycle is 2 ms, which means the two nodes will exchange data every 2 ms. The results of both experiment A and B are sampled in every simulation cycle in node 1. The number of the total sample is 1000 points in each experiment.

Figure 7-5 demonstrates the simulation distribution. Node 1 simulates equation 1 and equation 2. Node 2 simulates equation 3 and equation 4. Figure 7-6 shows the simulation loop sequence. Figure 7-7 to Figure 7-12 show the simulation results.

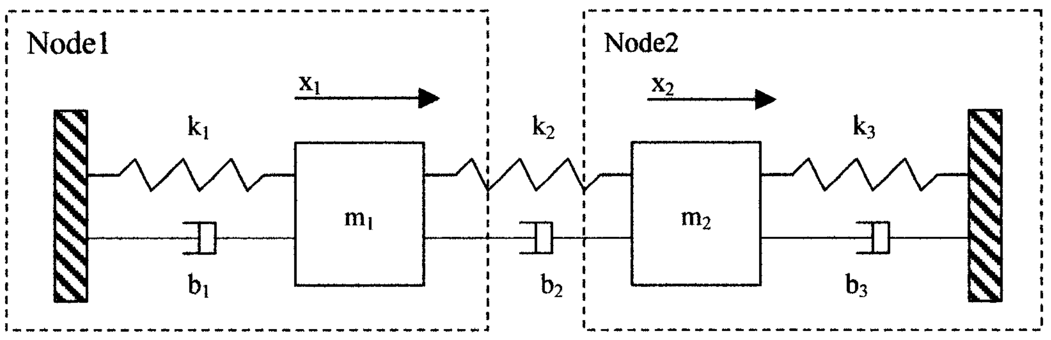


Figure 7-5 Mass spring model on distributed system

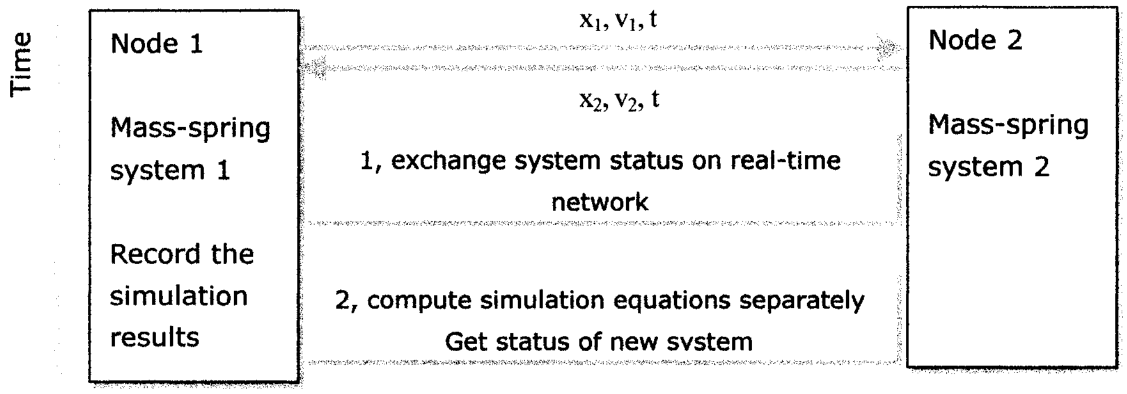

Time
Node 1
Mass-spring system 1
Record the simulation results
x_1, v_1, t
x_2, v_2, t
1, exchange system status on real-time network
2, compute simulation equations separately Get status of new svstem
Node 2
Mass-spring system 2

Figure 7-6 Distributed simulations via real-time Ethernet based TDMA network

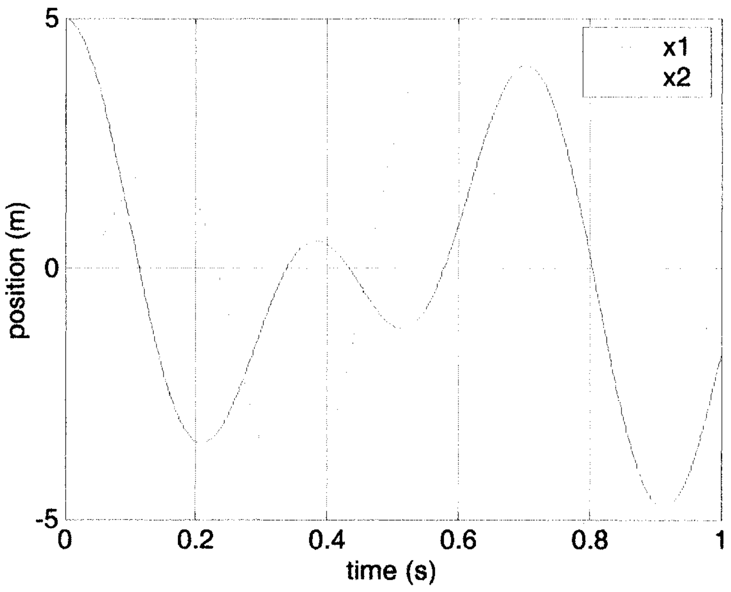


Figure 7-7 Experiment A - position of distributed simulation

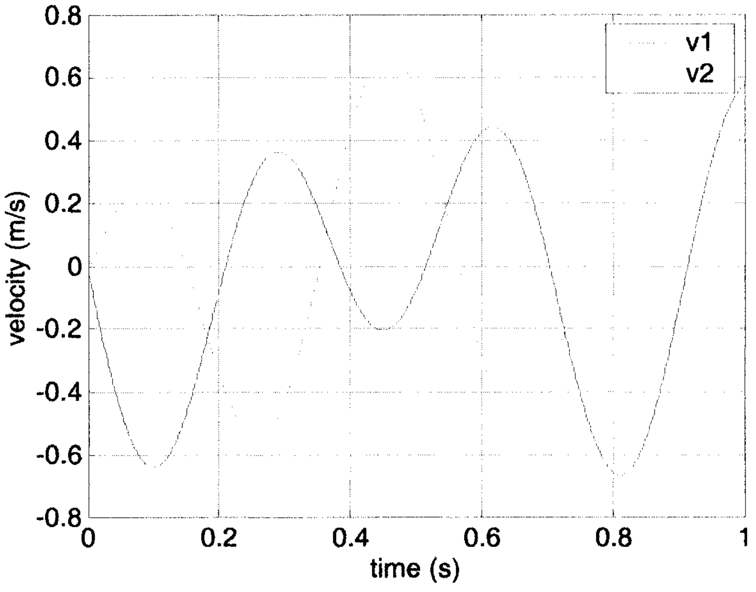


Figure 7-8 Experiment A - velocity of distributed simulation

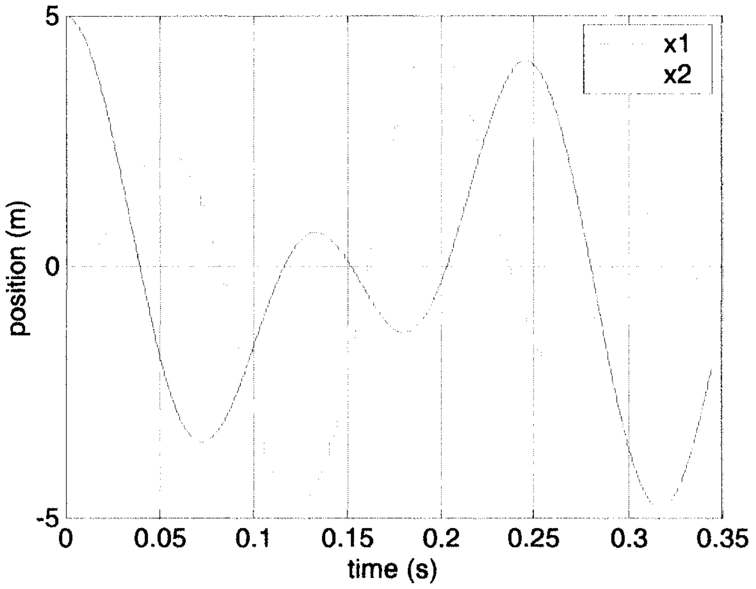


Figure 7-9 Experiment B - position of distributed simulation

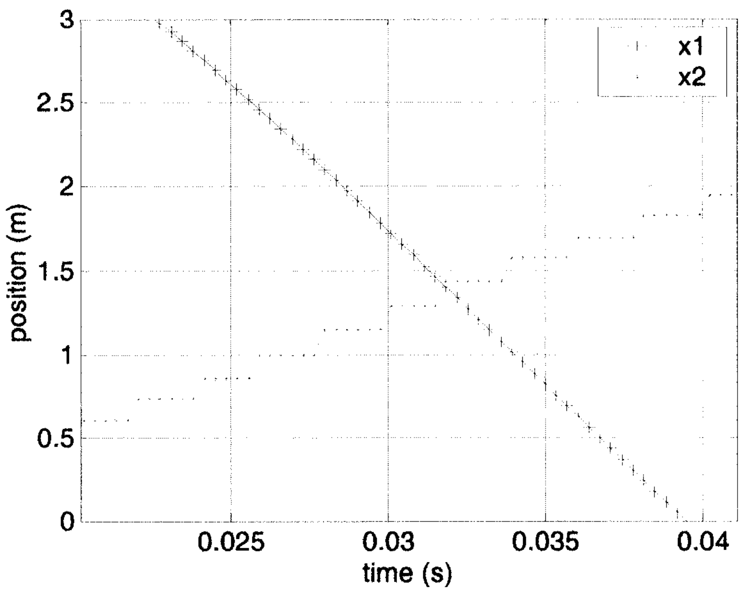


Figure 7-10 Experiment B – position of distributed simulation (zoom)

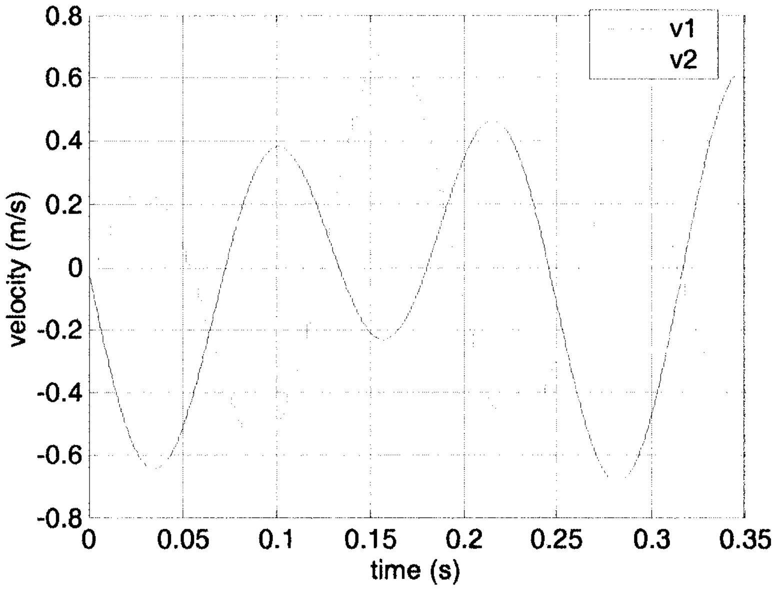


Figure 7-11 Experiment B - velocity of distributed simulation

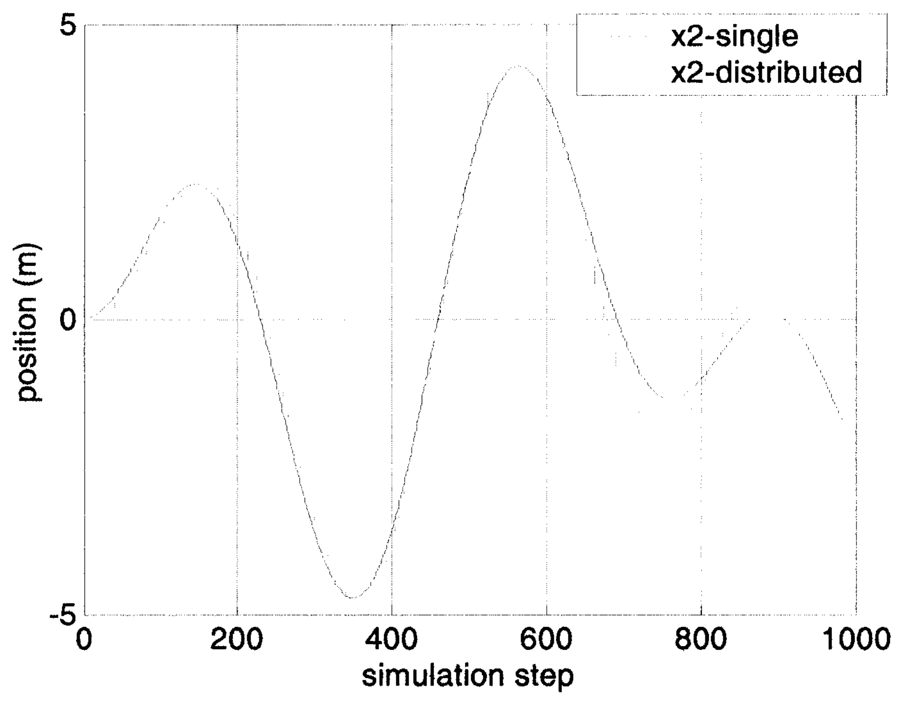


Figure 7-12 Experiment B x2 and single computer simulation x2

Conclusion: differing from the results of the single computer simulation, the distributed simulation's results are affected not only by the simulation step, but also by the communication frequency. In experiment A, the simulation cycle and TDMA communication cycle match each other. The simulation result is the same as in the single computer simulation. In experiment B, the simulation cycle is smaller than the communication cycle. From Figure 7-12, we find that the result of x2 is discrete according to the communication cycle. This makes an impact on both the accuracy and the stability of the simulation.

7.2 Distributed Control Systems

This application example is designed to show the ability of the real-time Ethernet. In the general application, the motor speed PID controller is local. Making this PID controller remote demonstrates the stability and response time of the real-time Ethernet. In this example, the sampling time is T = 1 ms. The motor type is PITTMAN GM9236C534-R2. Figure 7-13 illustrates the model of the motor.

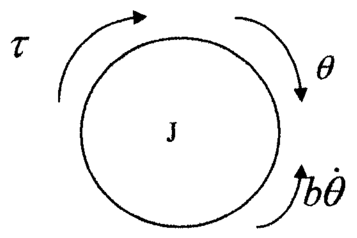


Figure 7-13 Motor model

The motor model equations are:

$$J\dot{\omega} = -b\omega + \tau(t) \qquad \textbf{(7)}$$

$$\dot{\theta} = \omega \qquad \textbf{(8)}$$

Because the system is a digital system, the PID controller difference equation is used:

$$\tau_{t+1} = e_t\left[\frac{K_d}{T} + K_p + K_i T\right] - e_{t-1}\left[\frac{2K_d}{T} + K_p\right] + e_{t-2}\left[\frac{K_d}{T}\right] + \tau_t \qquad \textbf{(9)}$$

$$e = \omega_d - \omega \qquad \textbf{(10)}$$

Local PID controller implementation

A local PID controller is implemented as a reference for the distributed PID controller. The sampling time is T = 1ms. Figure 7-14 and Figure 7-15 show the results of local PID control. The gains are obtained from tuning, where:

Kp = 0.1, Ki = 50, Kd= 0.0

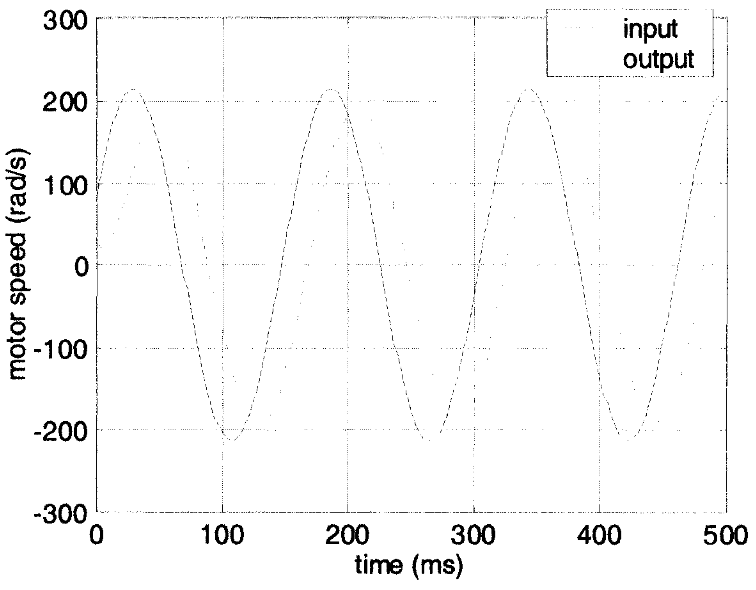


Figure 7-14 Local motor speed PID control

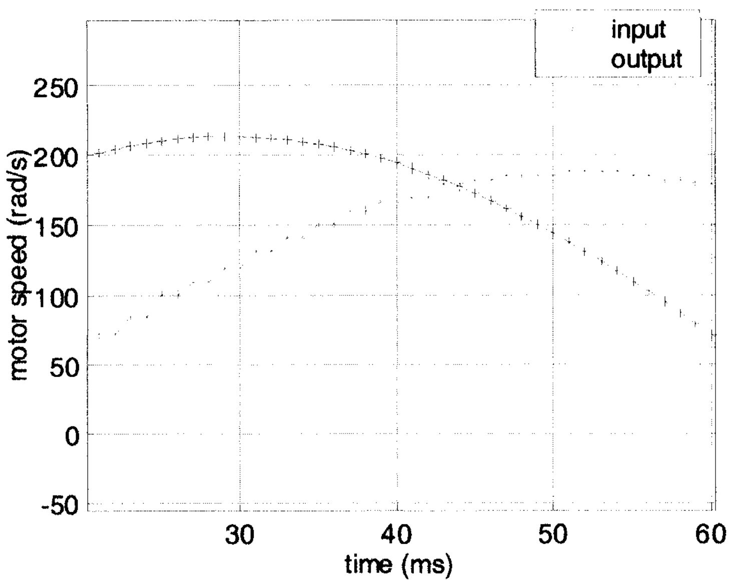


Figure 7-15 Local motor speed PID control (zoom)

Distributed PID control design

The distributed PID control system design considers the TDMA protocol communication rate as a factor not included in the local PID controller. Figure 7-16 shows the design.

Node 1:

A MultiQ I/O board [35] is plugged in the PCI slot in node 1. The motor analog control signal (-5v~+5v) is sent to the motor amplifier. A motor is connected to the amplifier. The encoder of the motor is connected to the MultiQ encoder port. Node 1 samples the encoder and calculates the speed of the motor ω, then sends ω to the remote PID controller through a real-time network with a communication frequency of 250 Hz. Node 1 also receives the motor control voltage command from the remote PID controller node 2, and then sends this voltage to the MultiQ I/O board.

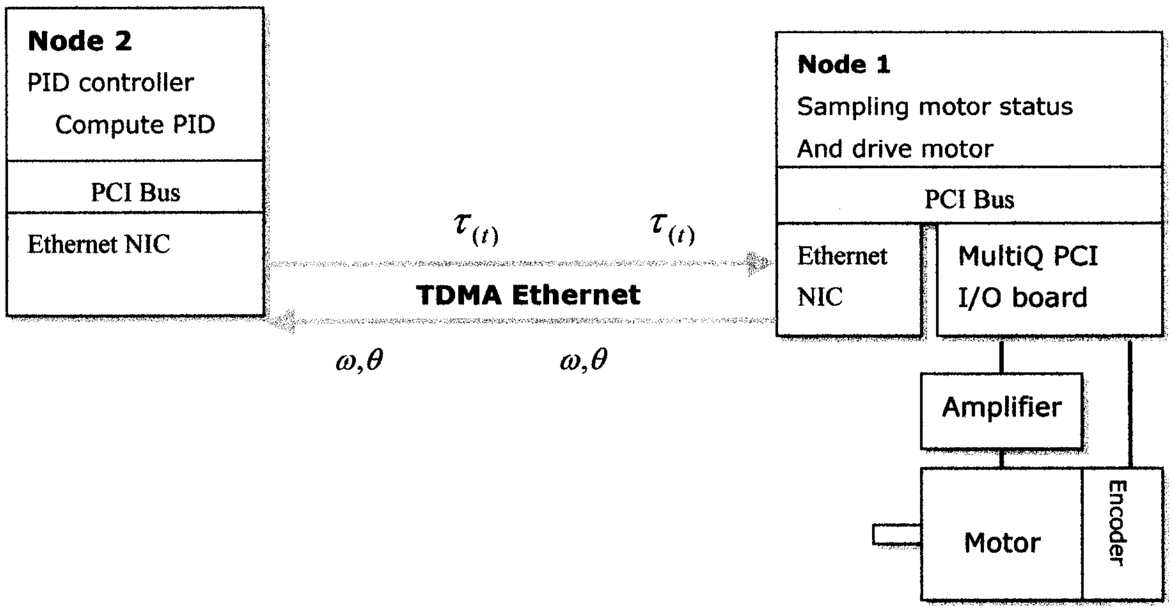


Figure 7-16 Distributed PID motor control system structure

Node 2:

A PID controller is running on node 2. Through the real-time TDMA network, it receives the motor status from node 1, and then uses this value to do the PID controller computation. It then sends the result to node 1. The data is collected from the remote controller node.

Two results are obtained with different combinations of sampling time and communication time. Figure 7-17 and Figure 7-18 show result 1, in which the sampling time is T = 1ms on node 1, the controller step is $T_{control}$ = 1ms on node 2, and the communication frequency is f_{TDMA} = 1kHz. The gains are Kp = 0.1, Ki = 50, Kd= 0.0 on node 2.

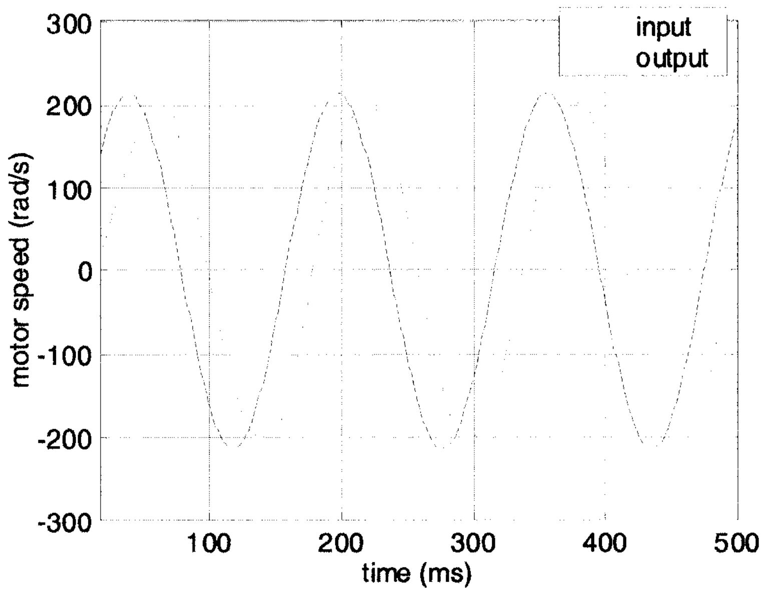


Figure 7-17 Distributed motor speed PID control result 1

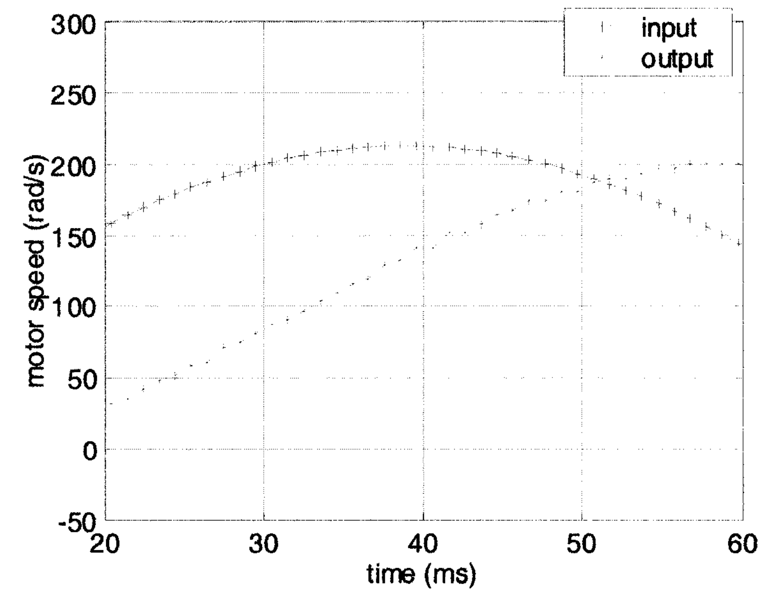


Figure 7-18Distributed motor speed PID control result 1 (zoom)

Figure 7-19 and Figure 7-20 show result 2, in which the sampling time is T = 0.5ms on node 1, the controller step is $T_{control}$ = 0.5 ms on node 2, and the communication frequency is f_{TDMA} = 0.5kHz. The gains are Kp = 0.1, Ki = 50, Kd= 0.0 on node 2.

From these figures we can see that the lower communication frequency has the same effects on the result as the lower sampling rate does, although sampling time and controller step time are less.

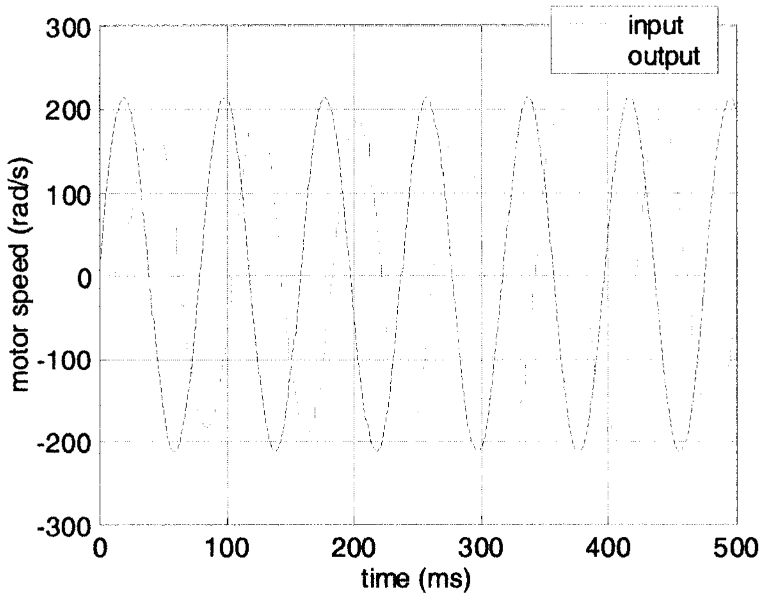


Figure 7-19 Distributed motor speed PID control result 2

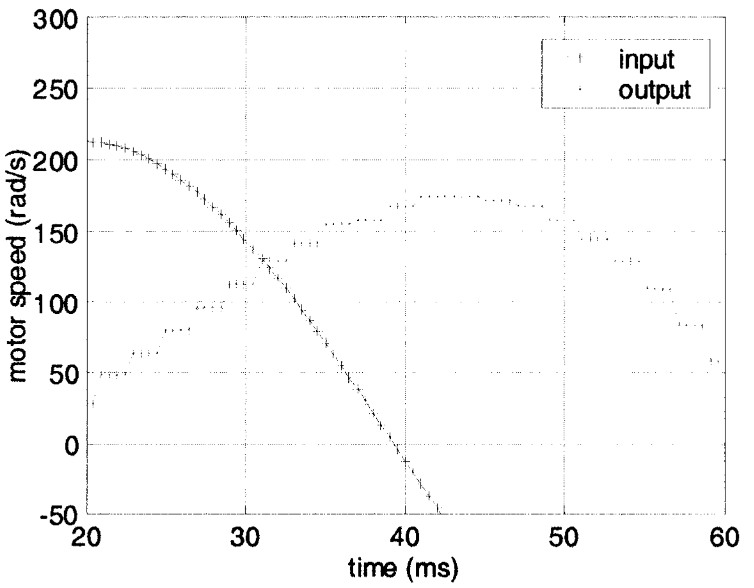


Figure 7-20 Distributed motor speed PID control result 2 (zoom)

Conclusion:

These examples implement and test the real-time distributed system on the real-time Ethernet based TDMA network. The results show that in some circumstances, the result of a distributed system is very close to a centralized system. However, the communication frequency among controllers and actuators affects the results. More tests and research need to be done to discover the network's impacts on the system's characteristics and how to make the networked system more robust.

8 Conclusions

A new approach for real-time communication using Ethernet and TDMA protocol is implemented and discussed in detail. The main results and contributions of this thesis are summarized as follows:

1) Driver software for real-time Ethernet communication

It is necessary to develop an Ethernet NIC driver for the real-time operating system (RTX) because it is not generally supplied with the operating system. This driver passes packets and manages the sending and receiving process between programs and hardware. The driver provides real-time response to the sending and receiving commands. It also provides fault detection and recovery functions.

2) Performance testing of Ethernet hardware

To study the characteristics of Ethernet hardware in a real-time environment, the performance tests must start with the time delay test of every single function and equipment component. The process ends with the total system time delay test of all the functions and hardware. The tests are designed using relatively cheap equipment to get a reasonably high degree of accuracy. From the test results, the tested gigabit NIC has a better performance than the fast Ethernet NIC tested in both average time delay and jitter.

Testing different hubs and switches showed a significant variance of performance. The switches showed larger time delays than some of the hubs as expected.

3) Real-time TDMA protocol for Ethernet including accurate clock synchronization

The real-time TDMA protocol developed manages communication on the network by controlling the driver and hardware to avoid collisions and transmit simultaneously on the switches. Several accurate clock synchronization methods on the network are implemented to achieve the essential criteria for the TDMA protocol. The protocol is very flexible with room for expansion and more development in the future.

4) Experimental applications of the real-time Ethernet system

Applications including a distributed mass-spring simulation system and a remote PID motor speed control system are developed and tested. All of the applications demonstrate deterministic network communication with consistent time delays.

The developed Ethernet based real-time communication system is one of the first implementations with the flexibility available necessary for advanced distributed simulation and control. Currently there is no other available system that has the same features and performance level.

In the future, this new approach will be expanded to allow for more flexibility and real-time control over the TDMA protocol parameters. This will allow a larger class of distributed simulation and control systems to be investigated for control systems research. More sophisticated fault detection/recovery and redundant networks will also be included. Finally, the approach will be applied to many industrial applications including distributed hardware-in-the-loop simulation and x-by-wire control systems.

References

[1] Lawrenz, W. "CAN System Engineering From Theory to Practical Applications", Springer-Verlag New York, 1997

[2] Leen, G., Heffernan, D., "Expanding Automotive Electronic Systems", IEEE *Computer*, pp. 88~93, January 2002.

[3] Buyya, R., "High Performance Cluster Computing: Architectures and Systems", Prentice Hall, 1999

[4] Apte, M., Chakravarthi, S., Padmanabhan J., and Skjellum, A., "A synchronized Real-time Linux Based Myrinet Cluster for Deterministic High Performance Computing and MPI/RT", Ninth International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 2001), April 2001.

[5] IEEE, "IEEE Standard 802.3, 2000 Edition", www.ieee.com

[6] Lian, F., Moyne, J. R., and Tilbury, D. M., "Performance Evaluation of Control Networks: Ethernet, ControlNet, and DeviceNet", IEEE Control Systems Magazine, pp. 66~83 February 2001.

[7] Bard, S. "Real-Time 1394b Data Transfer for Consumer Electronics", Intel Developer UPDATE Magazine, October 2000.

[8] Myricom, Inc., http://www.myri.com.

[9] Koopman, P. J. Jr., Upender, B. P., "Time Division Multiple Access Without a Bus Master", United Technologies Research Center Technical Report RR-9500470, June 30, 1995.

[10] Shay, W. A. "Understanding Data Communications And Networks" 2nd edition, Brooks/Cole Publishing, 1999.

[11] Jork L¨oser, Hermann H¨artig, "Real-time on Ethernet using off-the-shelf hardware"

[12] Upender, B. P., Koopman, P. J. Jr., “Communication Protocols for Embedded Systems”, Embedded Systems Programming, 7(11), pp. 46-58, November 1994,

[13] Jain, R., "ATM Networks: Issues and Challenges Ahead", InterOp+Networld Engineering Conference, March 1995.

[14] VenturCom Inc., “RTX 5.0 user’s guide”, http://www.vci.com

[15] The MathWorks, Inc., http://www.mathworks.com

[16] Microsoft Corporation, http://www.microsoft.com/windows/directx/

[17] Wind River Systems, Inc., http://www.windriver.com

[18] QNX Software Systems Ltd. http://www.qnx.com

[19] Linux online, http://www.linux.org

[20] REALTEK Semiconductor Corp., http://www.realtek.com.tw

[21] Intel Corporation, http://www.intel.com

[22] 3Com Corporation, http://www.3com.com

[23] Becker, D., “Linux network drivers”, http://www.scyld.com/network

[24] Held, G., “Ethernet Networks: Design, Implementation, Operation and Management”, John Wiley & Sons Canada, 2002.

[25] REALTEK Semiconductor Corp., “RTL8139C (L) Preliminary”, 1999.

[26] REALTEK Semiconductor Corp., “RTL8100 programming guide”.

[27] William S. Davis, “Operating Systems – a systematic view 4th edition”, The Benjamin/Cummings Publishing, 1992.

[28] VenturCom Inc., “RTX Reference guide”, http://www.vci.com.

[29] McDaniel, G., “IBM Dictionary of Computing” 10th edition, Computing McGraw-Hill, September 1993.

[30] Horauer, M., Kerö, N., Schmid, U., "A network interface for highly accurate clock Synchronization", Proceedings AUSTROCHIP'00, Graz, Austria, October 2000.

[31] Manoj Apte, Srigurunath Chakravarthi, Jothi Padmanabhan and Anthony Skjellum "A Synchronized Real-Time Linux Based Myrinet Cluster for Deterministic High Performance Computing and MPI/RT"

[32] Bogenberger, F., Müller, B., Führer, T., "Protocol overview", FlexRay International Workshop, April 2002.

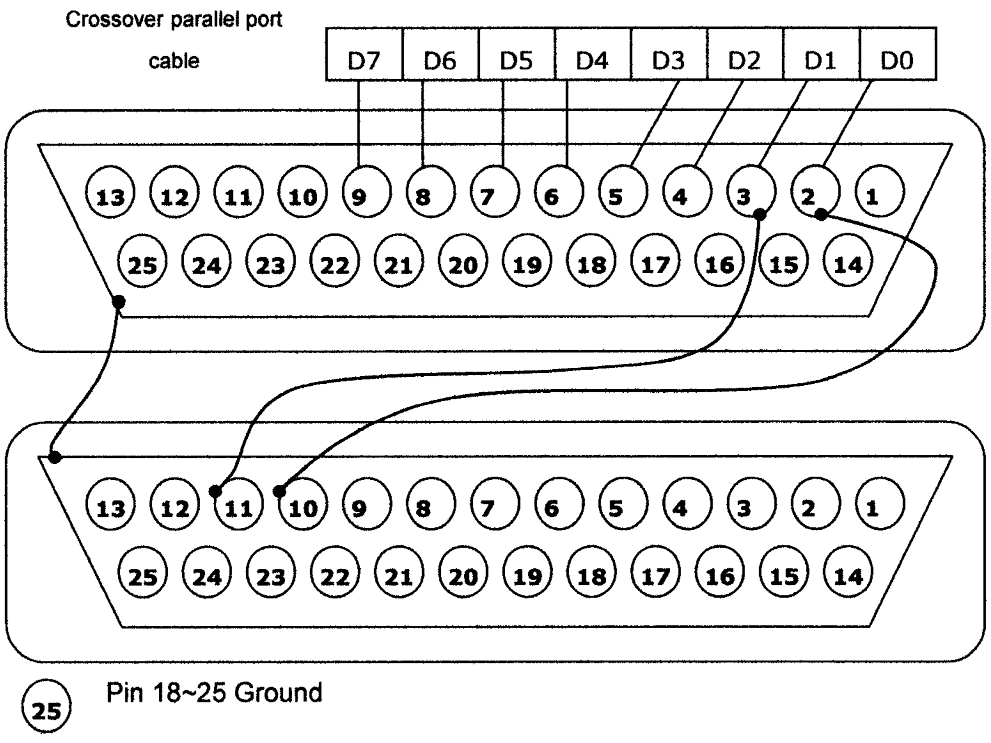
[33] Tekronix. Inc., "TDS 200-Series Digital Real-Time Oscilloscope user manual", http://www.tektronix.com

[34] Kopetz, H., Bauer, G., "The Time-Triggered Architecture", IEEE Special Issue on Modeling and Design of Embedded Software Conference, 2001

[35] Quanser Consulting, Inc., "MultiQ-PCI Data Acquisition System Programming Guide"

[36] MPI/RT Forum, http://www.mpirt.org/

Appendix 1 Crossover Parallel Port Cable



Output Signal	D0~D7	$\overline{C0}$	$\overline{C1}$	C2	$\overline{C3}$
Pin	2 ~ 9	1	14	16	17

Input Signal	S3	S4	S5	S6	S7
Pin	15	13	12	10	11

Figure appendix 1-1 Crossover parallel port cable

Appendix 2 Computer Hardware Configuration

Table appendix 2-1 Dell Dimension 8250 Desktop hardware configuration:

Item	Description
Processor	Intel Pentium 4 2.53Ghz clock speed 2.6 GHz
Main board	Intel 850E
Memory	256M RIMM RAM
Hard Disks	Westin Digital 60G
Video card	ASUS GeForce4 Ti 4200 AGP8X 64 Megabytes RAM
Sound card	Creative SB Live! Series (WDM)
CD-ROM	Lite-On LTN486S 48x Max & HL-DT-ST CD-RW GCE-8481B
Floppy drive	Standard floppy disk drives
Network Interface Card	Intel integrated PCI fast Ethernet Network Interface & test cards
Keyboard	USB Human Interface Device
Monitor	Plug and Play Monitor - (Standard monitor types)
USB port	USB 2.0

Table appendix 2-2 Dell PowerEdge 600sc server hardware configuration:

Item	Description
Processor	Intel Pentium 4 2.4Ghz
Main board	ServerWorks GC-SL
Memory	256M SDRAM
Hard Disks	7200 rpm IDE 60G
Video card	Integrated ATI-Rage XL controller with 8MB of SDRAM
Sound card	No
CD-ROM	Lite-On LTN486S 48x Max
Floppy drive	Standard floppy disk drives
Network Interface Card	Intel integrated gigabit NIC & test cards
Keyboard	Standard
Monitor	Plug and Play Monitor - (Standard monitor types)
USB port	USB 1.1

Appendix 3 Protocols category descriptions

Table appendix 3-1 Protocols category descriptions [12].

Protocol categories	Description
CSMA/CD	(Carrier Sense Multiple Access / Collision Detection) A node waits for an idle channel before transmitting. Collisions could happen if two or more nodes transmit simultaneously. If the sending nodes detect a collision, the nodes stop transmitting after waiting for a time by the random back off algorithm and try again.
Polling	A centrally assigned Master polls the other nodes (slaves). Non-master nodes transmit messages when they are polled. Inter-slave communication through the master.
Token Bus	A token signal is passed from node to node on a bus (virtual ring). Only the token holder has permission to access the media.
Token Ring	Nodes are connected in a ring using point-to-point links (this is not a circular bus, every wire is independent and operates concurrently). A token signal is passed from one node to another in a circular fashion. The token holder has the permission to access the media. This is a bit-at-a-time transfer protocol (bits are shifted around the ring, all other protocols discussed deal with whole messages).
CSMA/CA	When the media is idle, the active station will transmit the packet immediately. After each message, it reserves S slots for N nodes. If the media is busy, the station will wait for the slot reserve. If S is equal to N, there is no collision, which is known as Reservation CSMA. If S is smaller than N, some collision could happen, however, it is a statistical collision avoidance.
Binary Countdown	Each node is assigned a unique identification number. All nodes wishing to transmit compete for the channel by transmitting a binary signal based on their identification value. A node drops out the competition if it detects a dominant state while transmitting a passive state. Thus, the node with the *LOWEST (change picture)* identification value wins
TDMA	The master node sends out a sync frame to synchronize clocks, each node then transmits during its unique time slot. There is a Variation of TDMA: Variable Length TDMA (~Implicit Token). The unused time slices are truncated to save time. Compared with standard TDMA, the use of bandwidth is more efficient.

Glossary

API

Application Programming Interface (API) is an interface that allows an application to communicate with an operating system or another application.

Bandwidth

Range of frequencies passing through a given device. The broader the bandwidth, the faster the information sent or accessed through the circuit.

Bit

It is the smallest unit of information in a binary system. It represents either one or zero ("1" or "0").

Bugs

They are unintentional errors in programs.

Byte

8 bits compose a byte.

bps

Bits Per Second. bps is the number of bits sent every second.

Device driver

It is a program that sends and receives data. Typically it communicates with a hardware interface card, which receives device messages and maps their content into a region of memory on the card. The device driver then reads this memory and delivers the contents to the correspondent software programs.

Full duplex

A full duplex channel is a channel that provides transmission in both directions simultaneously.

Half duplex

A half duplex channel is a channel that provides signals in both directions, but not simultaneously.

ISO

International Standard Organization is responsible for a wide range of standards, including those relevant to networking.

OSI

Open Standards Interconnection; an international standardization program, facilitated by ISO and ITU to develop standards for data networking.

Propagation time delay

The time required for a signal transferring from one point to another.

TCP/IP

Transmission Control Protocol/Internet Protocol combines both TCP and IP protocols. They are widely used in applications such as Telnet, FTP and SMTP, to interface with TCP/IP.

Real-time

The determinism characteristic applied to hardware and/or software. A real-time process must finish a task in a bonded length of time even in the worst-case scenario. A real-time program is considered to be correct only if it gives the right result within a bonded deadline. The phrase "real-time" does not mean how fast the program responds, even though many people believe that real-time means real-fast. Determinism is more important than average speed for a real-time system.

RTOS

Real-time operating system is an operating system that offers deterministic responses. An RTOS has three characters: services, performance, and controllability. The less resources the services require (such as RAM to load the O/S, time to do a task switch, latency of message passing, etc.), the more deterministic the services are. At a minimum, this means priority-driven pre-emptive scheduling, with all system services capable of being satisfied in deterministic priority-driven FIFO order. This must be documented.

Real-time system

It is an application or group of applications with real-time requirements.

Rx

Receiving

TDMA

Time Division Multiple Access, the technique used by the digital network to squeeze more information from different sources onto one channel by dividing a channel into a few "discontinuous" pieces.

Timer

It is the feature of a real-time system that allows for starting or stopping a process at a fixed or relative time. Three important attributes of timers are:

Accuracy is how closely the timers' notion of time corresponds to the "real" time.

Jitter refers to the variation of timer events around a requested periodicity.

Resolution is an interval used to measure the time. Timers in real-time systems always measure time in integral multiples of a time interval, which is called the resolution.

Tx

Sending

Winsock

It is a Windows subroutine library that provides access to the Internet TCP/IP.