# ROBUSTNESS OF ETHERNET-BASED REAL-TIME NETWORKED CONTROL SYSTEM WITH MULTI-LEVEL CLIENT/SERVER ARCHITECTURE

A Thesis

by

NAVEEN KUMAR BIBINAGAR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2010

Major Subject: Mechanical Engineering

# ROBUSTNESS OF ETHERNET-BASED REAL-TIME NETWORKED CONTROL SYSTEM WITH MULTI-LEVEL CLIENT/SERVER ARCHITECTURE

A Thesis

by

NAVEEN KUMAR BIBINAGAR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

| | |
|---|---|
| Chair of Committee, | Won-jong Kim |
| Committee Members, | Bryan Rasmussen |
| | Hamid A. Toliyat |
| Head of Department, | Dennis O'Neal |

August 2010

Major Subject: Mechanical Engineering

# ABSTRACT

Robustness of Ethernet-Based Real-Time Networked Control System with Multi-Level Client/Server Architecture. (August 2010)

Naveen Kumar Bibinagar, B. Tech., Vellore Institute of Technology University, India

Chair of Advisory Committee: Dr. Won-jong Kim

The importance of real-time communication at the device level in a factory automation setup is a widely researched area. This research is an effort to experimentally verify if Ethernet can be used as a real-time communication standard in a factory automation setup, by observing the effects of packet delays, packet loss, and network congestion on the performance of a networked control system (NCS). The NCS experimental setup used in this research involves real-time feedback control of multiple plants like DC motors and a magnetic-levitation system connected to one or more controllers. A multi-client-multi-server architecture on a local area network (LAN) was developed using user datagram protocol (UDP) as the communication protocol. Key observations are as follows. (1) The multi-client-single-server system showed the highest packet delays compared to single-client-single-server architecture. (2) In the single-client-single-server system, as the Ethernet link utilization increased beyond 82%, the average packet delays and steady-state error of the DC motor speed-control system increased by 2231% and 304%, respectively. (3) Even under high link utilization, adding an additional server to the NCS reduced average packet delays considerably. (4) With large packet sizes, higher packet rates were automatically throttled by Ethernet's flow control mechanism affecting the real-time communication negatively. (5) In the multi-client-multi-server architecture, average packet delays at higher packet rates, and at higher packet lengths were found to be 40% lesser than the those of the single-client-single-server system and 87.5% lesser than those of the multi-client-single-server system.

# ACKNOWLEDGMENTS

I thank my advisor, Dr. Won-jong Kim for his supervision, advice, and guidance throughout my study at Texas A&M University; his crucial contributions made him the backbone of this research and also to this thesis. I thank my committee members Drs. Bryan Rasmussen and Hamid Toliyat for their guidance and support throughout the course of this research.

I thank Stephen C. Paschall II, Ajith Ambike, and Minhyung Lee for developing the closed loop real-time networked control system that was used as a test bed for the experiments.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience. Finally, thanks to my mother and father for their encouragement and love.

# NOMENCLATURE

| | |
|---|---|
| MC-MS | Multi-Client-Multi-Server |
| MC-SS | Multi-Client-Single-Server |
| SC-SS | Single-Client-Single-Server |
| SSE | Steady-State Error |

# TABLE OF CONTENTS

Page

ABSTRACT .................................................. iii

ACKNOWLEDGMENTS .................................................. iv

NOMENCLATURE .................................................. v

TABLE OF CONTENTS .................................................. vi

LIST OF FIGURES .................................................. viii

CHAPTER

I INTRODUCTION .................................................. 1

A. Evolution of Networked Control Systems .................................................. 1
B. Communication Networks .................................................. 4
C. Ethernet (IEEE 802.3) .................................................. 6
D. Objectives and Contribution of the Thesis .................................................. 12
E. Thesis Organization .................................................. 13

II EXPERIMENTAL SETUP .................................................. 15

A. Hardware Setup .................................................. 15
B. Software Setup .................................................. 17
C. Specifications of Communication Network .................................................. 17
D. Packet Structure .................................................. 22
1) Sensor Packet .................................................. 22
2) Control Packet .................................................. 22

III MULTI-CLIENT-MULTI-SERVER ARCHITECTURE .................................................. 23

A. Development of Multi-Client-Multi-Server Architecture .................................................. 23
B. Advantages of Multi-Server Architecture .................................................. 27
C. Multi-Client-Multi-Server Rejection Algorithm .................................................. 28
D. Calculation of Average Packet Delay ($T_{av}$) .................................................. 30
E. Packet Generation .................................................. 31

CHAPTER Page

IV RESULTS .......... 33

A. Single-Client-Single-Server Architecture .......... 33
1) A-1: Increasing the Packet Length with Keeping the Packet Rate Constant.......... 34
2) A-2: Increasing the Packet Rate with Keeping the Packet Length Constant.......... 37
B. Multi-Client-Single-Server Architecture .......... 39
1) B-1: Increasing the Packet Length with Keeping the Packet Rate Constant.......... 39
2) B-2: Increasing the Packet Rate with Keeping the Packet Length Constant.......... 43
C. Multi-Client-Multi-Server Architecture .......... 45
1) C-1: Increasing the Packet Length with Keeping the Packet Rate Constant.......... 46
2) C-2: Increasing the Packet Rate with Keeping the Packet Length Constant.......... 47

V PERFORMANCE COMPARISONS.......... 50

A. Increasing the Packet Length with Keeping the Packet Rate Constant.......... 50
B. Increasing the Packet Rate with Keeping the Packet Length Constant.......... 52

VI CONCLUSIONS.......... 55

A. Summary .......... 55
B. Conclusions .......... 56
C. Future Work.......... 57

REFERENCES.......... 58

APPENDIX A .......... 62

APPENDIX B .......... 76

VITA .......... 86

# LIST OF FIGURES


| | | Page |
|---|---|---|
| Figure 1 | Block diagram of direct digital control | 1 |
| Figure 2 | Block diagram of supervisory control system | 2 |
| Figure 3 | Block diagram of a single-client-single-server NCS | 3 |
| Figure 4 | Block diagram of feedback control over a network | 4 |
| Figure 5 | IEEE 802.3 Ethernet packet structure | 7 |
| Figure 6 | Backoff exponential algorithm | 8 |
| Figure 7 | Block diagram of a factory communication system [16] | 11 |
| Figure 8 | Comparison of (a) conventional Ethernet and (b) switched Ethernet | 12 |
| Figure 9 | Single actuator magnetic levitation system | 15 |
| Figure 10 | DC motor speed-control system | 17 |
| Figure 11 | Time delays in Ethernet communication [7] | 19 |
| Figure 12 | Sensor packet structure | 22 |
| Figure 13 | Control packet structure | 22 |
| Figure 14 | Block diagram of multi-client-single-server architecture | 24 |
| Figure 15 | Block diagram of multi-client-multi-server architecture | 25 |
| Figure 16 | Illustration of client rejection algorithm | 26 |
| Figure 17 | Block diagram of store and forward switch | 28 |
| Figure 18 | Flowchart of multi-client-multi-server architecture | 29 |
| Figure 19 | Average packet delay calculation | 30 |
| Figure 20 | PackETH: Creating a packet | 32 |

Page

Figure 21 PackETH: Generating a packet ........................................................ 32

Figure 22 Single-client-single-server architecture........................................... 33

Figure 23 A-1: (a) Average delay (b) Standard deviation of
Steady-state error.................................................................... 35

Figure 24 A-1: Packet length (byte) vs. (a) Average delay (b) Standard
deviation of steady-state error (rps) (c) Link utilization, at
constant packet rate of 14000 packets/s ........................................... 36

Figure 25 A-2: Packet rate vs. (a) Average delay (ns) (b) Standard deviation of
steady-state error (rps), at constant packet length of 64 bytes .......... 38

Figure 26 Multi-client-single-server architecture............................................ 39

Figure 27 B-1: (a) Average delay (b) Standard deviation of
steady-state error .................................................................... 41

Figure 28 B-1: Packet length (byte) vs. (a) Average delay (b) Standard
deviation of steady-state error (rps) (c) Link utilization, at
constant packet rate of 14000 packets/s ........................................... 42

Figure 29 B-2: Packet rate vs. (a) Average delay (ns) (b) Standard deviation of
steady-state error (rps), at constant packet length of 64 bytes .......... 44

Figure 30 Multi-client- multi-server architecture............................................ 45

Figure 31 C-1: (a) Average delay (b) Standard deviation of
steady-state error .................................................................... 47

Figure 32 C-2: Packet rate vs. (a) Average delay (ns) (b) Standard deviation of
steady-state error (rps), at constant packet length of 64 bytes .......... 49

Figure 33 Performance comparisons of experiments A-1, B-1, and C-1 .......... 51

Figure 34 Performance comparisons of experiments A-2, B-2, and C-2 .......... 53

# CHAPTER I

# INTRODUCTION

In this chapter, the different modes of control and the evolution of a networked control system is briefly described. It is followed by a discussion on the advantages and disadvantages of different communication networks. As the main focus of this research is on the Ethernet-based network, an in-depth literature survey on the working principle of Ethernet and its recent advancements is presented here.

## *A. Evolution of Networked Control Systems*

Technological advancements in the field of electronics and computers brought a paradigm shift in the way control systems were implemented; analog controllers were gradually replaced by digital control, giving birth to the direct digital control [1]. As the name suggests, the setup consists of a computer/processor as the brain of the system with a direct connection between the plant and the controller. The sensors and actuators generate required signals for the plant and the controller as shown in Figure 1.

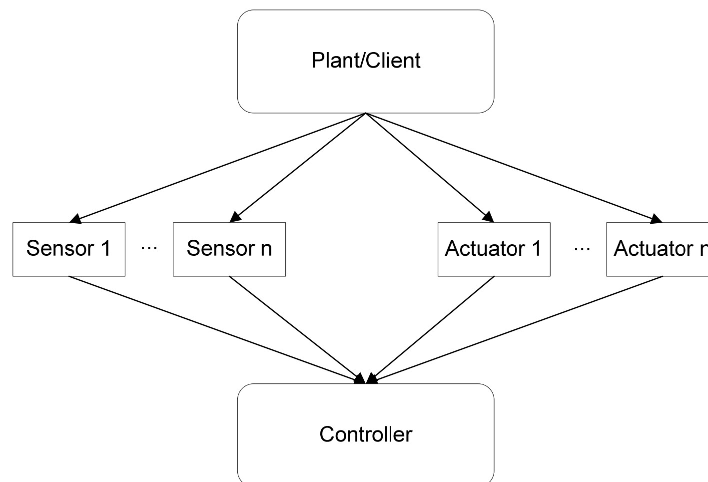



Figure 1. Block diagram of direct digital control

---

This thesis follows the style of *IEEE Transactions on Automation Science and Engineering.*

As the scale of control systems increased with the requirement of sensors, actuators, and controllers to be geographically distributed, it gave rise to a new era of control architecture known as distributed control system (DCS). In a DCS [2], majority of real-time tasks like sensing, actuation, and control are done at the process stations whereas supervisory tasks like monitoring, caution alarms, and on/off signals are done at the operator stations.

In a supervisory control system, the client/user system stays outside the control loop and is used to mainly monitor the control experiment. Srivatsava [3] designed an Internet-based supervisory control system where the user can monitor the process, from anywhere on the Internet. User can send corrective commands to the controller that will get implemented during the next sampling period. As shown in Figure 2, while the control loop between sensor and controller is closed locally, the client/operator computer is outside the control loop. Due to the local connection between the sensor and the controller, the delay element between the sensor and the controller is negligible.

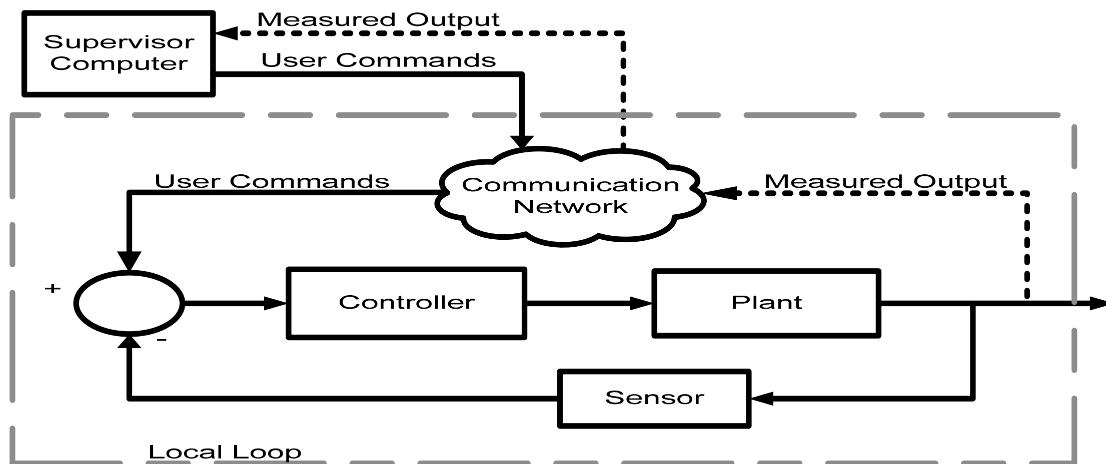



Figure 2. Block diagram of supervisory control system

In complex dynamical systems like autonomous manufacturing plants and reconfigurable manufacturing systems, computer networking plays an important role as a means of communication between distributed systems. With further advances in

computer technology and networking, the operator is not just an observer but a full-fledged controller. When the components of a networked control system (NCS) like sensors, actuators, and controllers are distributed and the feedback control loop is closed via a common communication channel, it is defined as an integrated communication and control system (ICCS) [2].

The framework of an NCS with a single controller is shown in Figure 3. In this setup, the communication between the controller and plant is not dedicated like in the previous case and it has to compete with the traffic from other controllers and applications on the network

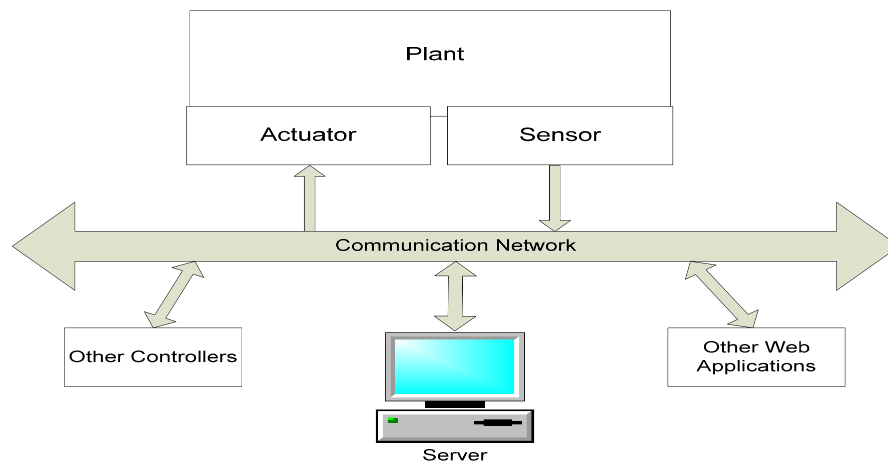



Figure 3. Block diagram of a single-client-single-server NCS

As shown in Figure 4, sharing of the communication medium between the controller and actuator results in time varying delays. The delay specified as $\tau_{ca}$ is the network-induced delay between the controller and the actuator, and $\tau_{sc}$ is the delay between the sensor and the controller.

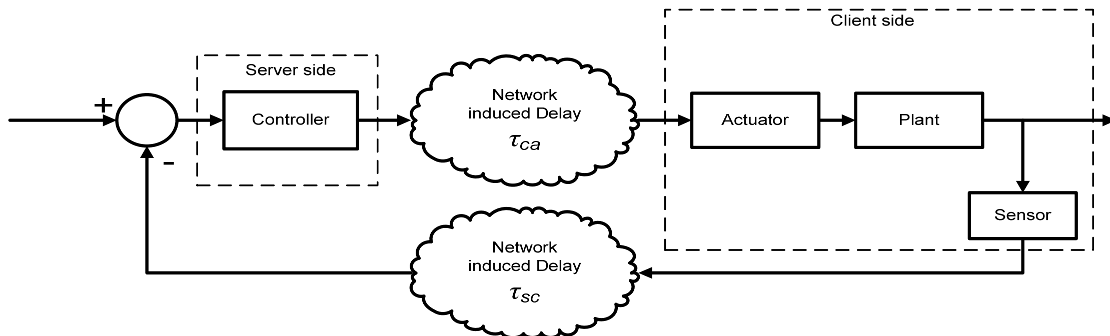



Figure 4. Block diagram of feedback control over a network

In the current scenario, the number of clients or plants that need to be served is constantly growing but there is always an issue of scaling the number of servers/controllers. As the number of requests to a single controller increases, there will be a degrading effect on the performance of the control system. The current research focuses on the performance analysis of an Ethernet-based multi-client-multi-server NCS in the presence of the network induced delays.

*B. Communication Networks*

On one hand, communication networks play a significant role in the feedback control of an NCS, and on the other hand they make the analysis and design of the control system even more complicated. Technological advancements in the field of automobile industry were the main stimulant for the use of communication networks in digital control systems. This setup turned out to be advantageous in terms of scalability, flexibility, modularity, and reduced cost of wiring.

There are two types of communication networks – data networks and control networks. In data networks there is usually a transmission of large data packets and the transmission rate is relatively infrequent. As the period of transmission is not so important, data rates are maintained considerably high to enable the transmission of large data frames. However in control networks there is a frequent transmission of small packets that must meet timing guarantees. It is important to differentiate between these two types of networks and employ them according to the real-time and nonreal-time

requirements. For example Raji [4] found that the throughput of Ethernet network drops considerably at 35% of bandwidth utilization.

Communication protocols were developed with respect to specific industries. Building automation control network (BACnet) developed by American society of heating, refrigeration and air conditioning (ASHRAE) [5] became a standard of communication for commercial and government buildings, and campus environments. Like Ethernet and Arcnet, BACnet can also be operated on a RS-485 twisted pair [5].

Controller Area Network (CAN) [6] is designed jointly by Bosch, Philips, and Intel primarily for the automotive industry. This standard mainly focuses on the data link layer, making it more of a serial communication bus than a network protocol. CAN protocol employs carrier sense multiple access (CSMA) /Arbitration on message priority (CSMA/AMP) medium access method. There is theoretically no limit on the number of nodes that can be connected on this network. When compared to Ethernet, bus arbitration is relatively straightforward because every message that gets transmitted has a specific priority attached to it. CAN-bus works as a multicast protocol in which, the message is received by all nodes on the network. Nodes accept or deny a message based on the unique identifier attached to it.

ControlNet is based on ring topology where a token is passed between the nodes in a sequential fashion. Node that needs to transmit retains the token and releases it either after transmitting its message or after a certain period of time. This protocol is mainly employed in time critical applications due to its deterministic nature of delays. The amount of time that a node has to wait before sending a packet is equal to the time that the token spends on traversing the ring. This deterministic nature makes the modeling of delays relatively easy when compared to Ethernet. However, it has been widely researched that only under heavy network load conditions, the average packet delays in ControlNet is lower than that of the contention protocols like Ethernet [7]. During low network traffic most of the time in ControlNet is wasted in transferring the token between nodes.

In industrial communication networks, Profibus and Factory instrumentation protocol (FIP) are categorized as field bus protocols. Profibus is a broadcast protocol operated in a Master/Slave architecture, and was developed by six German companies and five German institutes in 1987 [8]. It was first promoted by German department of education and research, and due to its deterministic nature, it went on to become the most widely used industrial field bus with more than 28 million devices installed by the end of 2008. Profibus is also based on token passing mechanism with an ability to support high-priority and low-priority messages enabling it to work with both time-critical and non-time-critical traffic [9]. To maintain hard real-time requirements of field devices like sensors, controllers, and actuators, various Fieldbus protocols have been developed by numerous organizations. Consequently, in the late 1990s, Profibus and FIP (International electrotechnical commission (IEC) 61158) were made an international standard in the field of industrial automation. But the high costs of hardware and incompatibility of multiple-vendor systems have become barriers in its acceptance. Recently, the computer network standard IEEE 802.3 Ethernet has come up as an alternative for real-time communication. It is widely believed that Ethernet's nondeterministic nature of delays would be a major hindrance in its acceptance.

*C. Ethernet (IEEE 802.3)*

Unlike those discussed above, Ethernet is a contention-based protocol that works on carrier sense multiple access with collision detection (CSMA/CD) for data transmission. In this mechanism there is no central bus controller that arbitrates the channel; every node acts as a self-arbiter, by listening to the channel for any ongoing transmissions and transmitting only when the channel is idle. Even after transmitting the packet, the sender keeps listening to the channel for any collisions and if detected, the colliding stations back off and wait for a random amount of time before trying to retransmit. This retransmission process is tried up to 16 times after which the packet is withdrawn. Under heavy traffic, CSMA/CD mechanism can make the communication

delay excessively large and in some extreme cases there can be packet losses if the transmission is unsuccessful for 16 times.

$$R = \{0 \text{ to } 2^{K-1}\} \text{ where } K = N \text{ and } K \leq 10; \qquad (1)$$

The structure of the Ethernet packet is shown in Figure 5. There is a mandatory requirement for Ethernet packet to be a minimum of 64 bytes due to the CSMA/CD protocol.

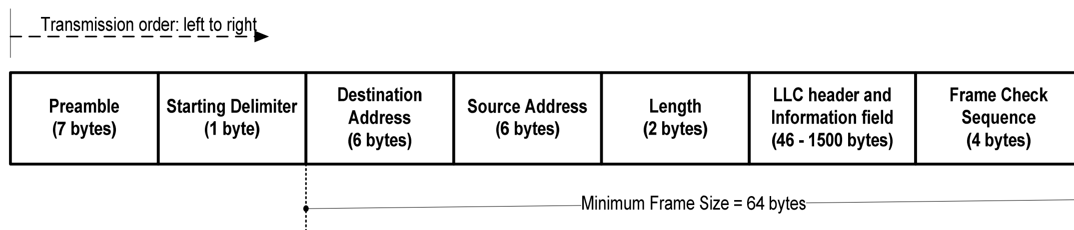



Figure 5. IEEE 802.3 Ethernet packet structure

In size calculations of a basic Ethernet packet, "Preamble" and "Starting Delimiter" are not included, so the minimum size of data field should be 46 bytes. If it is less than that, padding is added. A basic Ethernet packet of 64 bytes represents 512 bit times and therefore, for a 100-Mbps Ethernet, the slot time can be calculated as 5.12 µs. The amount of time that a node has to wait is a random selection based on back off propagation algorithm. It is explained as a flow chart in Figure 6. As the value of *'N'* increases from zero, the set containing the various random times increases by the order of $2^{K-1}$.

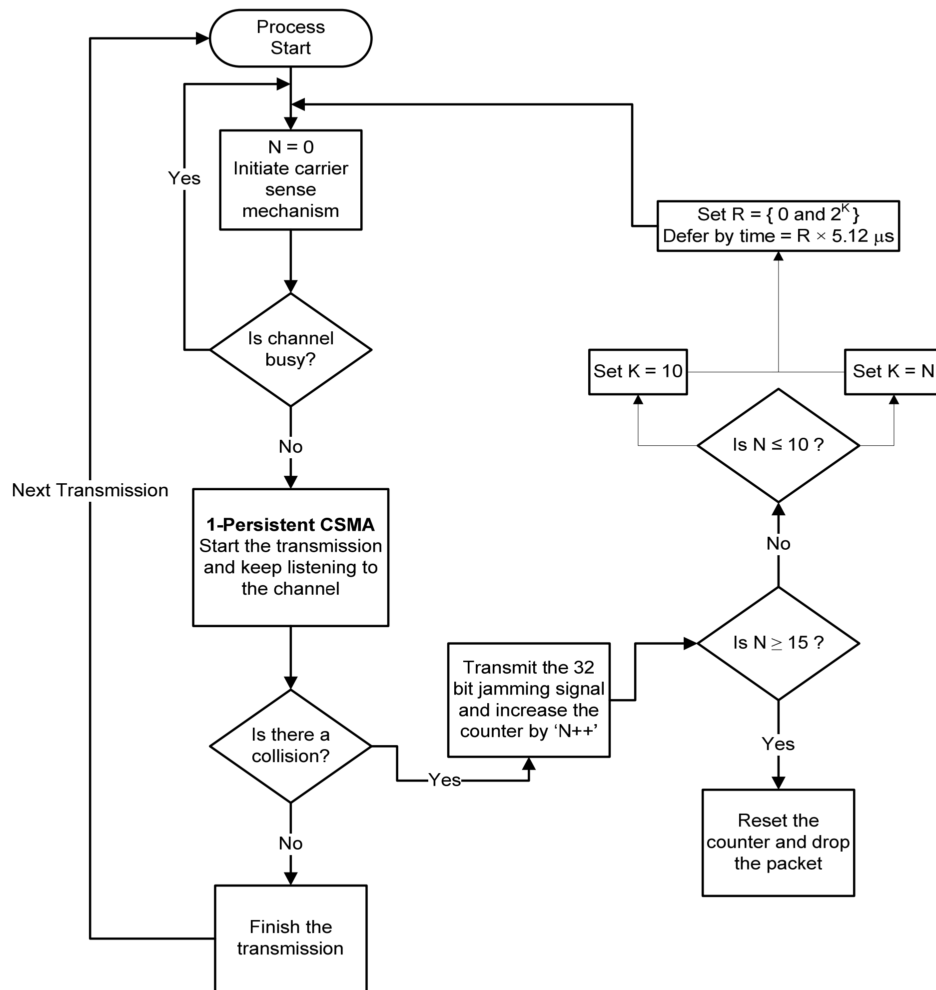



Figure 6. Backoff exponential algorithm

Ethernet was invented at Xerox PARC during 1974, Metcalfe and Boggs [10] did a simple analysis on this "experimental" 3 Mbps Ethernet. They observed that the throughput of a network is a function of packet length and network load (in terms of output of the buffer). They found that, for large packets of length 512 bytes, throughput remains near 100% but it drops to 37% (*1/e*) as the at minimum packet lengths of around 48 bits. Tobagi and Bruce [11] did an extensive analysis of throughput vs. delay characteristics of CSMA/CD networks. As the Ethernet used in their paper is either p-persistent or non-persistent, it cannot be directly compared with the 1-persistent Ethernet used in this research. Their results showed that as the ratio of long packets goes up, the

channel capacity available to short packets drops considerably. This increases the average delay of short packets not due to collisions, but majorly due to the time spent in waiting for the long packets to traverse the network. Moreover, as the ratio of long packets increases the average delay decreases, this is due to the fewer packets being transmitted and the minimum possibility of collisions.

Bux [12] did a comparative study of different LAN technologies like Ethernet and token ring systems. Bux found that even under high throughputs, packet delays were observed to be reasonable as long as the packet length was maintained at twice that of the propagation delay (measured in the units of bit times). Coyle and Liu [13] did analysis on a non-persistent CSMA/CD assuming finite number of hosts, each generating packets with an exponential distribution. They devised a stability measure term "Drift" that specifies the rate at which the number of backlogged packets is increasing. Drift depends on the number of users on the network and the shape of drift curve depends on the retransmission rate. In their analysis of CSMA/CD without a backoff algorithm, at large number of hosts and high retransmission rates, they found that the network went unstable, which emphasized the importance of the backoff mechanism in the Ethernet protocol.

Gonsalves and Tobagi [14] examined the performance of a network as a factor of distribution of hosts in the network. In most of the previous theoretical studies, a balanced star topology is assumed, but that does not occur often in reality. Thus in [14], they evaluated various configurations of a network. When the hosts are uniformly distributed as two equal sized clusters along a cable, their results showed that they experienced better delays and throughput than the hosts at the ends. Further, when the hosts are distributed in unequal clusters, the bigger cluster is better off in performance due to the local resolution of collisions. So, in real-time applications, positioning of different hosts is an important factor to consider.

Tasaka [15] researched on the performance of a slotted non-persistent CSMA/CD as a function of the network buffer size. It was found that, as the size of the buffers increases, more packets are retained in the queue leading to lesser congestion. Tasaka

found that under low loads, having large buffers resulted in a higher throughput but higher average delays.

Speeds of Ethernet have grown from the “experimental” 3 Mbps to 100 Mbps with 100BASE-T becoming the most widely used communication media as Fast Ethernet operating. The recent upgrade in Ethernet is termed as Gigabit Ethernet operating at 1000 Mbps. Although the Fieldbus has been a standard for device level automation since 1990’s , the advancements in Ethernet have made it possible to employ in factory automation systems at cell level and plant level [16] as shown in Figure 7. At the plant level, it acts as a factory backbone interconnecting the relevant stations, these stations mostly run Transport control protocol/ Internet protocol (TCP/IP) suite with application protocols like hypertext transport protocol (HTTP) [17], file transfer protocol (FTP) [18] and some World Wide Web functionalities. Although manufacturing messaging protocol (MMS) [19] is an international standard at the cell level of a factory communication system, it is often substituted with proprietary protocols that has further increased the incompatibility issues.

At the device level, different vendor specific protocols are currently being employed, and the introduction of Ethernet can bring an efficient interconnection between the three layers of communication. The main issue at this level is to maintain timing guarantees in real-time communication between the sensor and the controller and it is important to analyze the performance of Ethernet under such conditions.

Ethernet has been studied extensively with a Poisson traffic model in different simulation models. However, in a real-time scenario, the traffic is mostly bursty in nature. Mazraani and Parulkar [20] found that as long as the network utilization does not reach a particular threshold the behavior of the Ethernet remains the same under bursty conditions. They also observed that as the utilization increases beyond a threshold, packet delay, queue lengths and packet loss increase drastically. To address the issues of non-determinism, network architectures based on switching have gained significance. Switches are network devices that operate at the data-link layer interconnecting different hosts.

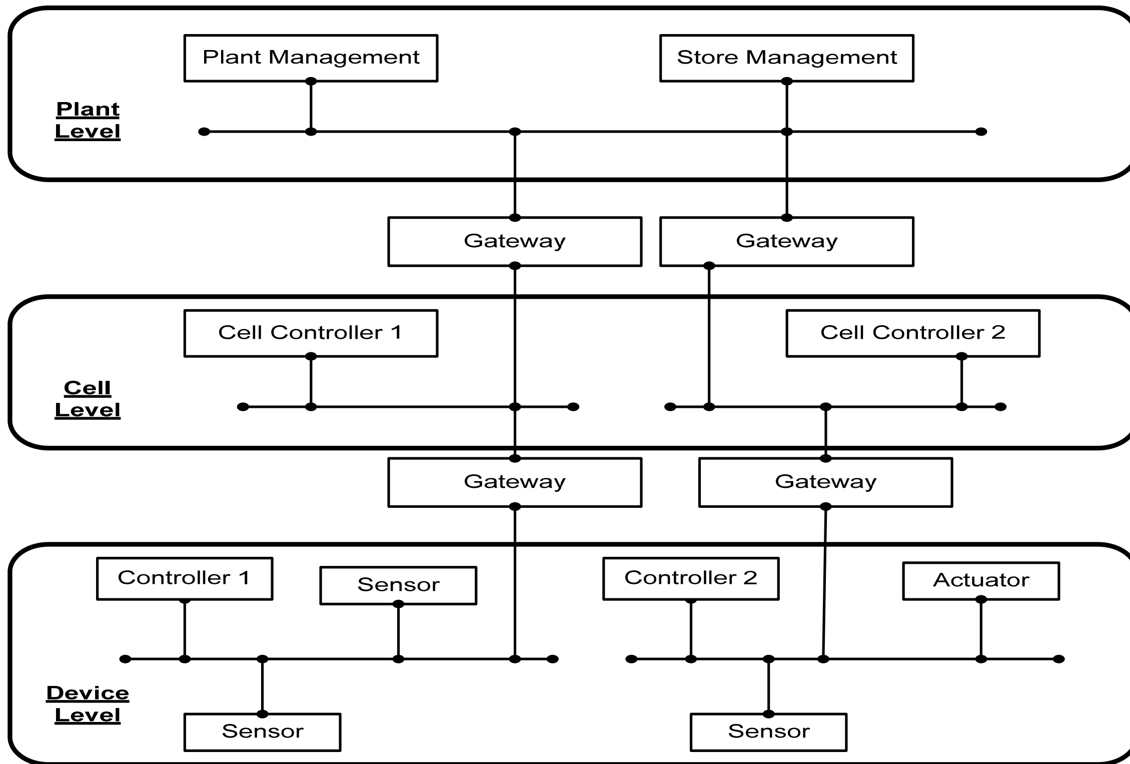



Figure 7. Block diagram of a factory communication system [16]

Contrary to a shared architecture, in switched network architectures, frames are sent only to the addressed nodes. This leads to a better handling of the traffic and considerable reduction in delays [21]. The difference between the two architectures of Ethernet is shown in Figure 8. In conventional Ethernet, when Node 1 tries to send a message to Node 3, all the other nodes on the network also receive the message even though the packet is not addressed to them. All nodes check for the destination address in the packet and if it is not addressed to them they discard the packet.  The hub in the conventional Ethernet acts like a repeater, broadcasting the packet to all the nodes on the network as shown in the figure. Whereas, in a switched Ethernet, the switch directs the packet only to the host that matches with the destination address. This shows that the number of collision domains in a switched Ethernet is considerably reduced.

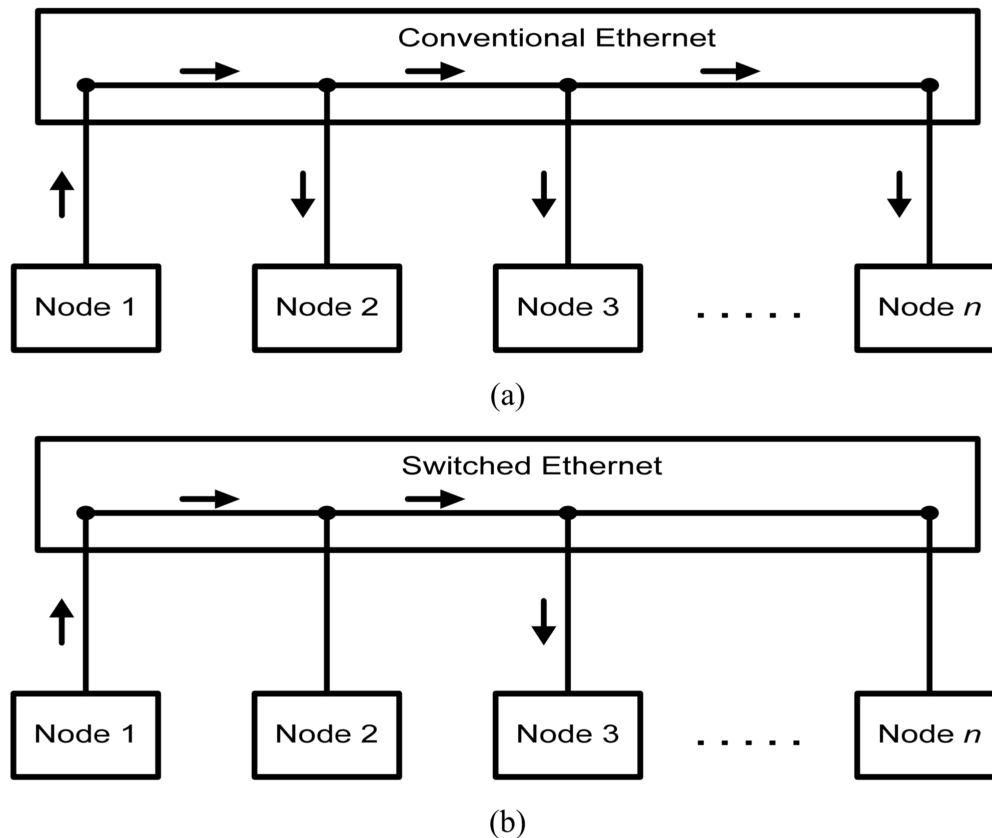



Figure 8. Comparison of (a) conventional Ethernet and (b) switched Ethernet

In this research, a similar switched Ethernet network is used. Although, previous researchers found switched architecture to be more effective than conventional Ethernet, a series of experiments are conducted under different load conditions to measure the performance of our switched Ethernet-based NCS. Various multi-client and multi-server architectures are developed and the performance of NCS is analyzed at maximum Ethernet link utilization.

*D. Objectives and Contribution of the Thesis*

The main objective of this thesis is to experimentally verify if Ethernet can be used as a real-time communication standard in a factory automation setup, by observing the effects of packet delays, packet losses, and network congestion. Although Ethernet has become the de facto standard of communication, its nondeterministic delays are a

major hindrance to the strict timing guarantees of a real-time system. To study a real-time communication network, a multi-client-multi-server (MC-MS) system is developed with a novel client-selection and client-rejection strategy. The client-rejection algorithm at the server enables it to identify a particular client and reject or accept it based on user-defined criteria. At the client the algorithm enables it to automatically connect to a new server when it receives a rejection from the first server. Although switched Ethernet is considered to be a significant improvement from the traditional hub standard, this MC-MS standard provided a good platform to empirically verify the effects of connecting more than one client to a single server and the potential performance improvements of an additional server.

Although the experimental setup is based on a high-speed switched Ethernet at 100 Mbps, it is important to understand that the performance of a real-time system can be affected by the communication from other unrelated applications and services. By loading the network with different packet sizes and packet rates, the maximum threshold of the link utilization is determined. It is beyond the threshold that the performance parameters of the control system started to decrease rapidly. By conducting various experiments, a relation between the average packet delay and the steady-state error of the system is empirically determined in this thesis.

### *E. Thesis Organization*

Chapter I provides a brief introduction of the evolution of a networked control system from a direct digital control, different types of real-time communication systems, and various modes of control using them. Advantages of and issues related to the different communication media are a part of the discussion. The literature review throws light on the recent advancements in the real-time communication networks and the evolution of the Ethernet as a real-time communication standard. This chapter also describes the objectives and contributions of this thesis.

Chapter II gives a brief study of the existing experimental setup used as the test-bed, explaining the architecture of NCS used for this research. It also describes the real-

time client-server software design, specifications of the current communication network and various network induced delays.

Chapter III explains the design and development of the multi-client-multi-server NCS architecture. The development of a novel client identification and rejection algorithm is also explained in detail. It also describes the theory on packet generation and the delay calculation.

Chapter IV describes the observations and results of various experimental scenarios ranging from single-client-single-server to multi-client-multi-server architectures.

Chapter V presents the performance comparisons of clients in the different experimental scenarios

Chapter VI summarizes and concludes this thesis. The conclusions are based on the results presented in Chapters III, IV, and V. The future work of the research in the area of NCSs is discussed at the end of this chapter.

# CHAPTER II

# EXPERIMENTAL SETUP

In this chapter, the existing experimental setup is discussed, including the hardware setup of the control system, software setup of the controller and sensor program, and the specifications of the current Ethernet communication network.

## *A. Hardware Setup*

The ball magnetic-levitation system (Maglev System) developed by Paschall as part of his senior Honors thesis [22] is used as one of the clients. The function of the maglev system is to levitate a steel ball at a predetermined reference position using an electromagnet. As shown in Figure 9, the test bed consists of an electromagnet actuator, pulse-width modulator (PWM) power amplifier, an optical position sensor and two DC power supplies. Position sensing is carried out by using a CdS photocell and an incandescent light source.

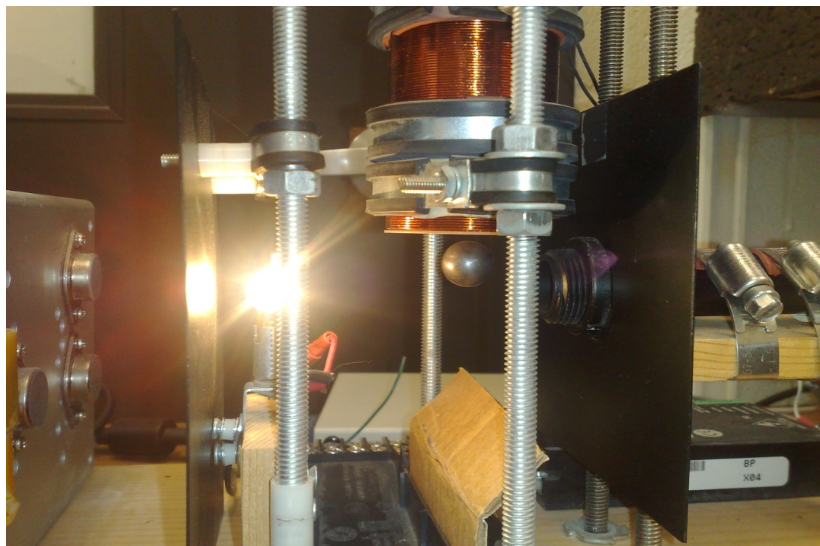


Figure 9. Single actuator magnetic levitation system

A 15-V DC power supply is used to run the CdS photocell and a 24-V DC power supply, to power the electromagnet. When the ball is placed in between the photocell

and the light source, the electrical resistance of the photocell varies based on the amount of light exposure. The voltage across the resistance placed in series varies as a function of the ball position, and this varying voltage is used to compute a control signal. The feedback loop is closed when the control input from the controller is given to the electromagnet through the power amplifier that generates the required force to levitate the ball. A digital lead-lag compensator implemented with the sampling frequency of 333 Hz, designed by Srivatsava [3] is used for feedback control.

$$D(z) = 4.15 \times 10^4 \, \frac{z^2 - 1.75z + 0.769}{z^2 - 0.782z - 0.13} \tag{2}$$

The maglev system is considered as a fast plant that requires a sampling frequency of 333 Hz and the actuation has to occur within 1.4 ms for system stability.

DC motor speed-control system (DC Motor System) developed by Youngchul Na is used as a test bed for simulating slow systems. Its sampling frequency is 50 Hz which is low when compared to Maglev System at 333 Hz. The function of the DC Motor system is to control the speed of the motor at a predetermined speed. As shown in Figure 10 the DC Motor system consists of five AMAX 26 DC motors connected to five HEDS 5540 Digital Encoders [23]. A National Instruments PCI 6221 board [24] is connected at the controller node for data acquisition purpose. The encoders act as the sensors that send the speed signal outputs in terms of pulse counts per unit time and the controller sends back the control input as a PWM signal. The digital controller for DC motor speed-control system is given by (2) [30].

$$u(k) = u(k - 1) + [K_P + 0.5K_I h]e(k) + [0.5K_I h - K_P]e(k - 1) \tag{3}$$

Figure 10. DC motor speed-control system

*B. Software Setup*

The software program by Ambike [26] was developed for the single-tier architecture. The algorithm is redesigned to make it work for multi-client and multi-server architecture. After a careful research [26] of different operating systems in a real-time environment, Ubuntu 6.10 is chosen as the operating system. To make Ubuntu real-time capable, an open source developed Real Time Application Interface (RTAI) 3.4 [27] is installed on it. RTAI modifies the Ubuntu kernel to make it fully preemptive. Linux Control and Measurement Device Interface (COMEDI) [6] is installed to act a data acquisition interface between the National Instruments' 6221 board and Ubuntu. The combination of Ubuntu, RTAI, COMEDI, and NI 6221 makes it a real-time data acquisition system.

*C. Specifications of Communication Network*

There are many factors that affect the performance of Ethernet [28] and the major ones are highlighted here:

1. Propagation delay: This is defined as the round-trip delay between any two hosts. The specification has a maximum round-trip delay of 464 bit times. In our Ethernet setup, the average propagation delay measured using the PING command is 0.388 ms [29].
2. Bit rate: The “experimental” Ethernet had a bit rate of 3 Mbps [10], and in our setup the bit rate of Ethernet is 100 Mbps.
3. Slot time: This is specified as the time elapsed for the host to acquire the network. It should be larger than the maximum propagation delay of 464 bit times and 48 bit times of a jamming sequence that comes to 512 bit times. For a 100-Mbps data rate the slot time is equal to 5.12 μs.
4. Packet length: Based on the specification of Ethernet, the packet length should not be shorter than the slot time which means the minimum packet length is 64 bytes, including a 4-byte frame check sequence and 14-byte header. The maximum packet length is 1518 bytes.
5. Number of hosts: Theoretically, Ethernet specifies 100 hosts per segment and 1024 total number of hosts in a multi-segment [28]. In our experiment setup, in our multi-segment subnet, the total number of hosts including the network control system is currently around 110.
6. Buffering: The hosts in the network control system are assumed to have fixed size of buffers. When the packets arrive at a rate that exceeds the transmission rate the arriving packets are assumed to be discarded. In reality Ethernet activated the flow-control mechanism to limit the arrival rate of the packet.

The performance of a network is measured in various factors like average delay, throughput, channel capacity, stability, and fairness. It is important to differentiate these performance measures with respect to different types of networks. For example, in a real-time system, average delay and network load are the most important performance

measures [30] whereas in a nonreal-time system throughput and channel capacity are important.

The performance measures studied in this research are average delay ($T_{av}$), and Ethernet link utilization is explained as follows:

Total Delay is measured as the time elapsed from when the sender tries to acquire a channel for transmitting data to the time that the receiver receives them. The total delay is measured as the time $T_d$ which is equal to the one way delay between the sender and the receiver. As shown in Figure 11 it is mainly made up of preprocessing time at the sender ($T_{pre}$), waiting time ($T_{wait}$), transmission time ($T_{tx}$), and post-processing time ($T_{post}$) at the receiver.

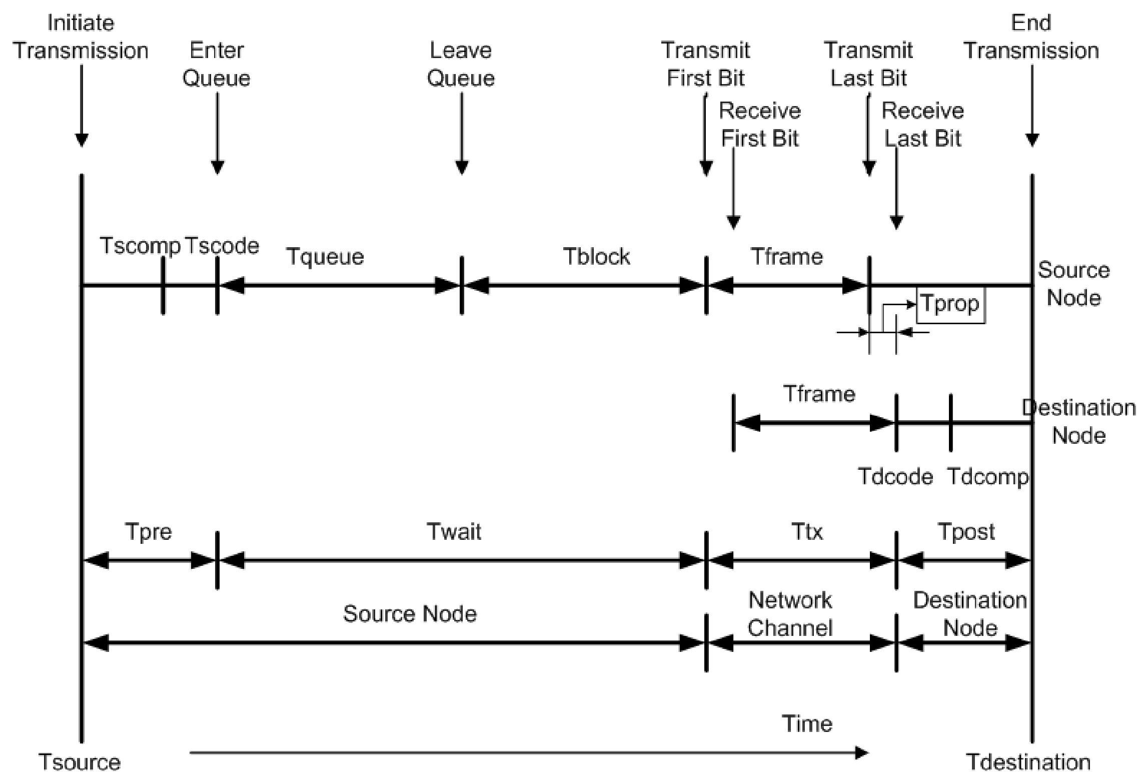



Figure 11. Time delays in Ethernet communication [7]

$$T_{\mathrm{d}} = T_{\mathrm{destination}} - T_{\mathrm{source}}$$
$$T_{\mathrm{d}} = T_{\mathrm{pre}} + T_{\mathrm{wait}} + T_{\mathrm{tx}} + T_{\mathrm{post}} \quad (4)$$
$$T_{\mathrm{d}} = 2 \times T_{\mathrm{proc}} + T_{\mathrm{wait}} + T_{\mathrm{tx}}$$
$$\text{Assumption : } T_{\mathrm{pre}} = T_{\mathrm{post}}$$

The preprocessing time ($T_{\mathrm{pre}}$) is a minor component of the average delay that includes the time taken by the sender to prepare the message and it mainly depends on processing power of the computer. It can be assumed as a fixed component. The post-processing time ($T_{\mathrm{post}}$) is another minor component that includes the time taken by the receiver to decode and process the message. It also depends on processing power of the receiver computer and it can be assumed as a fixed component. It can be assumed that $T_{\mathrm{pre}}$ is equal to $T_{\mathrm{post}}$ as the sender and the receiver are identical computers.

The waiting time ($T_{\mathrm{wait}}$) forms the major component of the average delay and it is responsible for the nondeterministic nature of the Ethernet communication. It is made of two components, queuing time ($T_{\mathrm{queue}}$) and blocking time ($T_{\mathrm{block}}$). The queuing time is the time the packet spends in the queue or the buffer at the sender computer before it gets on the network for transmission. It depends on the network load, arrival rate of packets and the blocking time of the previous packets already present in the queue. The blocking time ($T_{\mathrm{block}}$) is defined as the time that the sender has to wait based on the probabilistic waiting times deduced from the BEB algorithm described in Figure 6. $T_{\mathrm{block}}$ is the main component that introduces the nondeterministic delays, according to [7] the blocking time can be expressed as shown in the (4) where $\mathrm{E}(T_{\mathrm{k}})$ is the expected time of the $k^{th}$ collision.

$$E(T_{block}) = \sum_{k=1}^{16} E(T_k) + T_{resid} \quad (5)$$

$\mathrm{E}(T_{\mathrm{k}})$ depends on the number of nodes that are trying to send a packet and the arrival rate of packets on the network making it non-deterministic and unbounded.

Transmission time *($T_{tx}$)* is defined as the time taken to transmit a frame or a packet across the communication channel. It is a function of the frame length, bit rate of the channel and propagation delay ($T_{prop}$). In the experimental setup, the hosts are going to be at a fixed position and so, the propagation delay can be assumed as constant. Hence, the variable component of $T_{\text{tx}}$ is frame transmission delay ($T_{frame}$) that is a function of the frame length.

There are many ways in which network utilization is defined. In the case of half-duplex Ethernet, it is either 100% or 0% based on whether it carries a packet or not. In this research, the experimental NCS is based on a switched Ethernet that can communicate in full-duplex [31]. As shown in Figure 8, when computers are connected to a switch, the communication between the two computers can be assumed to be occurring over an imaginary dedicated link. The measure of utilization is termed as either Ethernet link utilization or link utilization. It is measured in terms of the capacity of the channel, calculated using the expression as shown in (5).

$$\text{Utilization (Mbps)} = \text{packet rate } \left(\frac{\text{pkts}}{\text{s}}\right) \times \text{packet length} \left(\frac{\text{bytes}}{\text{pkt}}\right) \times 8 \left(\frac{\text{bits}}{\text{byte}}\right) \qquad (6)$$

As the network control system plants operate at a specific sampling frequency, there is a specific rate at which packets are exchanged between the client and the server. As the sampling period in the experimental architectures is maintained at a constant 3 ms so, the rate at which the sensor-control packets are generated is always the same. The packet rate that is used as a load measure in the various experiments conducted is that of rate generated by the packet generator. The length of the sensor-control packet is also kept constant at specific values as explained in the Section *II.D*. The packet length that is used as a load measure in the various experiments conducted is that of length specified in the packet generator.

*D. Packet Structure*

*1) Sensor Packet*

The sensor packet structure is shown in Figure 12; the total length of the packet is 130 bytes with the protocol headers taking 42 bytes. The time stamp is used to calculate the average delay ($T_{av}$) of the packet and the fields' $y_0$ to $y_{t-6}$ are used to carry sensor data. The field $y_{t-7}$ is used as an identification number to run different control loops for different clients. The same identification number is also used to accept or reject clients in the multi-client-multi-server scenario to be explained in Section *III.F.*

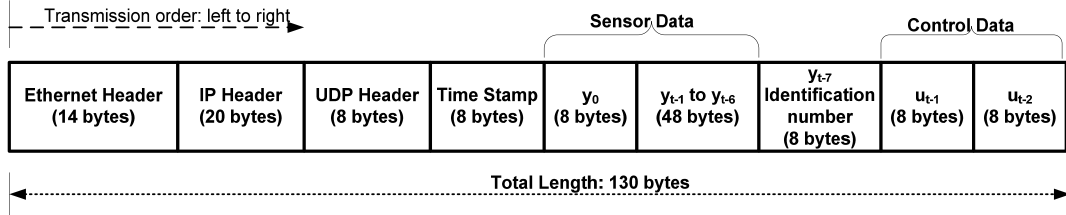



Figure 12. Sensor packet structure

*2) Control Packet*

The control packet structure is shown in Figure 13; the total length of the packet is 122 bytes with the protocol headers taking 42 bytes. Server receives the time stamp from the client packet and copies it in the corresponding time stamp field in control packet. The field $u_0$ is used to send the actual control signal and the rest $u_{1e}$ to $u_{4e}$ are predicted control values. $u_{5e}$ to $u_{8e}$ are reserved for future use, they can be used by increasing the order of prediction.

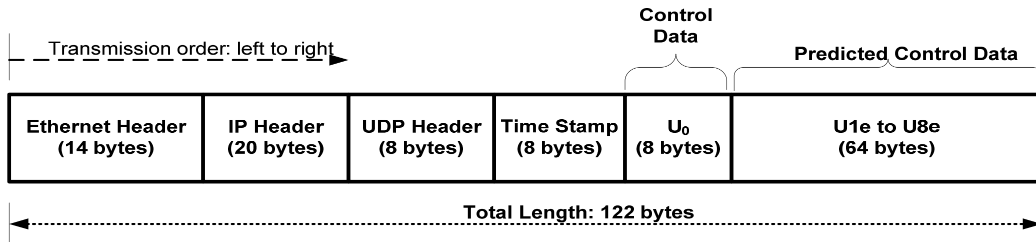



Figure 13. Control packet structure

# CHAPTER III

# MULTI-CLIENT-MULTI-SERVER ARCHITECTURE

In this chapter, the existing network architecture is discussed and the motivation for the development of the multi-client-multi-server architecture is presented. The client selection and rejection algorithm that will be employed in this architecture is also discussed here. The working principle and advantages of the algorithm in different scenarios is explained. A short theory on creating and generating Ethernet packets using "packeETH", a packet generator tool is presented.

## *A. Development of Multi-Client-Multi-Server Architecture*

Ambike [26] designed a single-client-single-server closed-loop real-time architecture that had the ball maglev system attached to the client computer and the controller program running on the server computer. A general block diagram of Ambike's test setup is shown in Figure 3. The motivation behind the research was to evaluate the performance of a closed-loop real-time control system that has network-induced timing delays. Artificial packet losses were introduced in the program to simulate the effects of network delays and losses on the performance of the ball maglev system. Performance degradation during packet losses was minimized by employing *p*-step ahead predictors to predict control data and use it when required.

Lee [29] further researched and extended the test bed to a multi-client-single-server architecture in which three different plants, DC motor speed-control system, ball maglev system and autonomous robot were connected to client computers. The block diagram of Lee's experimental setup is shown in Figure 14. Increasing the number of clients but maintaining one server led to different challenges in the system and Lee proposed different sampling periods based on bandwidth utilization and performance of the plant. Lee used an identification number to differentiate between the various clients and direct it to the appropriate control loop.

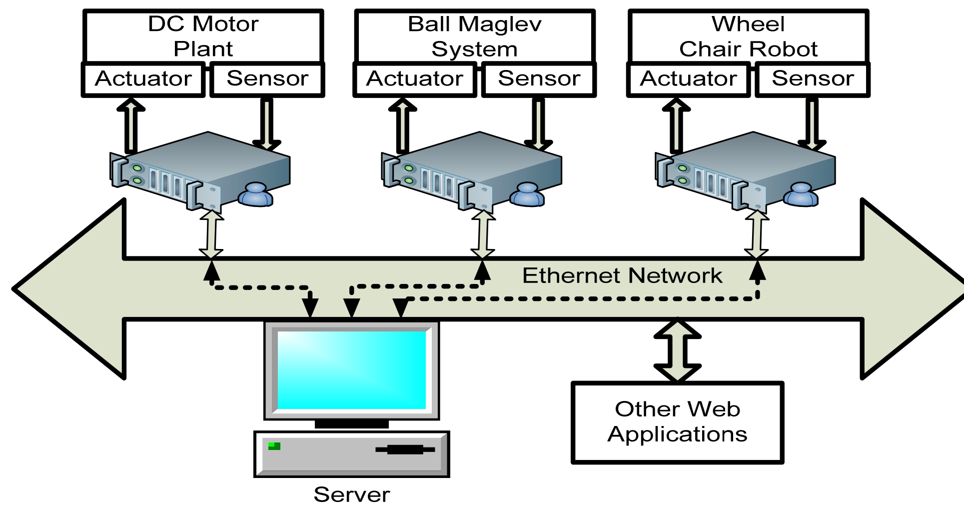



Figure 14. Block diagram of multi-client-single-server architecture

As mentioned previously, Ethernet allows up to 1024 hosts per multi-segment [30] but there is always a physical limitation on the number of clients that a single server can handle. As the number of clients being served by a single server is increased, the communication is not only affected by network induced delays but also by the delays caused due to the processing time at the server. A UDP-based client-server communication requires the server to create a socket and listen on a particular port. To accommodate more than one client, multi-threading was used to create a new thread for each clients-server communication. If the plants connected to the clients have a high sampling frequency, it burdens the server with heavy timing requirements. For example in the ball maglev system the actuation has to occur at every 1.4 ms to maintain the system stability. Under such condition the effectiveness of client-server communication gets limited by the processing power of the server computer. To overcome these challenges, a multi-server architecture as shown in Figure 15 was developed.

A multi-server architecture improves the scalability of an NCS by giving the flexibility for a client to connect to an appropriate server based on how heavy a server is loaded. In most of the client-server communications, server is in waiting mode and the

client initiates a new request for service. Even if the client connects to a heavily loaded server, the server can make a decision to serve or not.

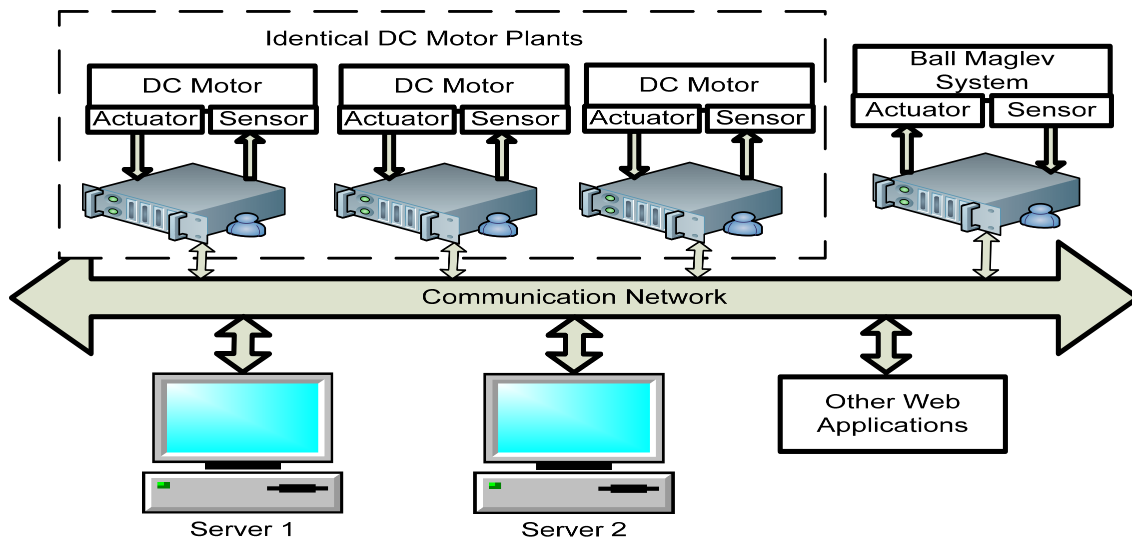


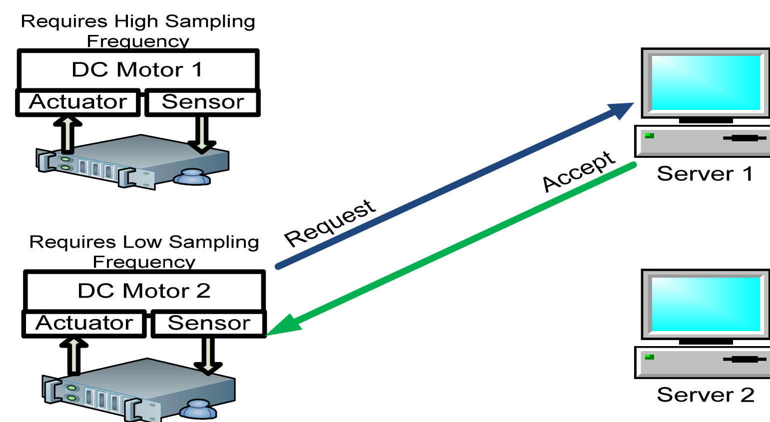

Figure 15. Block diagram of multi-client-multi-server architecture

Based on the architecture of a multi-client-multi-server system, the communication can be handled in the following ways:

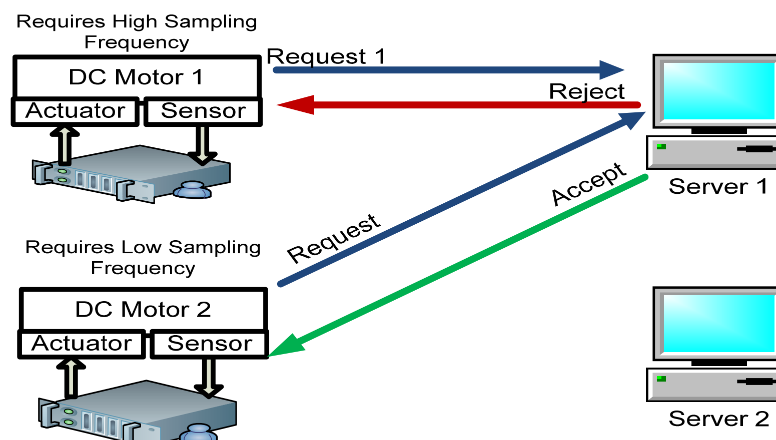
1. Each client can be programmed to always connect to a specific server and the connection is always a success.
2. Client sends a request to a server and the server can either serve it or send a rejection message notifying that it cannot serve at this moment. The rejected client can then send a new request to a different server.

In this research, the architecture is developed based on the second approach. The server can be programmed to make a decision based on factors like the number of clients already being handled, sampling frequency requirements of the plants, and processing power of the server. Currently, the process is user determined where the type of clients that the server can handle is predetermined by a user, the server decides solely based on the client's identification number. The above developed architecture satisfies our needs because we are mainly concerned with the performance evaluation of an NCS under varying loads in a multi-server scenario.

A general overview of the client rejection process is illustrated in Figure 16. As shown in Figure 16 (a), DC motor 1 is attached to Client 1 and DC motor 2 is attached to Client 2. In this setup, the Server 1 is programmed to accept only slower clients like Client 2 that is operating at low sampling frequency. So, when the Client 2 sends a request, it gets accepted by Server 1. But when the Client 1 sends a request to Server 1 as shown in Figure 16 (b) it gets rejected. Based on the client-selection-rejection algorithm, the Client 1 sends a new request to Server 2. Because Server 2 is programmed to accept faster clients like Client 1, the request gets accepted as shown in Figure 16 (c).

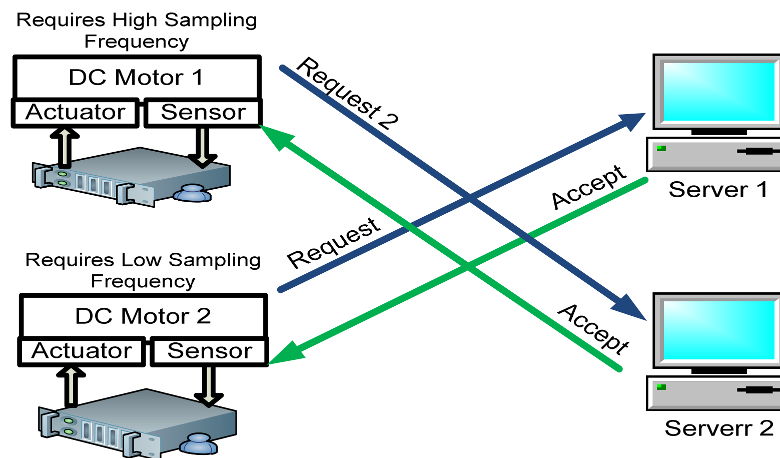



(a)




(b)

Figure 16. Illustration of client rejection algorithm




(c)

Figure 16. (Continued)

*B. Advantages of Multi-Server Architecture*

The major advantages of the multi-server architecture developed for this research are explained below:

1. Reduced Delays due to Processing Time: From the previously explained timing diagram in Figure 11, it can be noted that a small portion of overall packet delay ($T_d$) is due to preprocessing and post-processing of the packet at both the client and the server. When there are multiple clients connected to a single server, processing delays become significant and might become a major component of $T_d$ if the server is a low-end processor. The multi-server architecture proved to be an effective solution in such scenarios. For example in Figure 16, Server 1 is restricted to only slow response plants because of its processing limitations and Server 2 is capable of handling both slow and fast response plants. When Client 2 requests for a service from Server 1, the connection gets accepted and served because the control program in Server 1 is programmed to accept only slow response plants like Client 2. When Client 1 sends a request to Server 1, it receives a reject message from the Server upon which Client 1 sends a new request to Server 2 and gets served. This approach of directing clients to specific servers resulted in lower packet delays than single-server scenarios.

2. Reduced Delays due to Minimum Ethernet Link Utilization: As mentioned previously, the experimental test bed is connected to a switched Ethernet LAN. As already shown in Figure 8, the communication between a client and a server can be imagined to be occurring on a virtual link between the two computers. The switch used in our NCS is a store and forward switch as show in Figure 17. When Client 1 wants to send a packet to Server 2, the switch will first check for any previous packets waiting in Queue 2 to be sent to Server 2. If yes, then the packet is stored in Queue 2, else it is routed immediately. Hence, as more clients try to connect to a single server, the Ethernet link utilization starts to increase and the packets from different clients have to wait in a queue before being sent to that particular server. By providing an additional server as shown in Figure 17, Client 2 can connect to Server 1 instead of waiting in Queue 2 for Server 2.

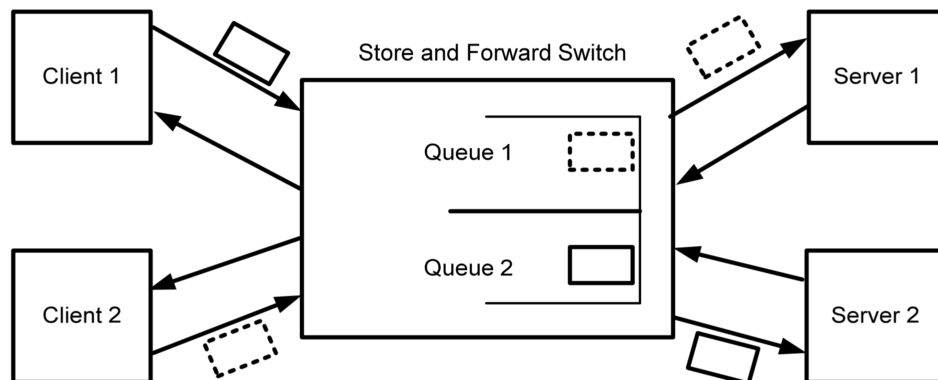



Figure 17. Block diagram of store and forward switch

*C. Multi-Client-Multi-Server Rejection Algorithm and Flowchart*

The flowchart of the multi-server rejection is as shown in and the algorithm is explained as follows:

1. The server program starts and waits for requests from clients.
2. Client initiates a connection by sending a sensor packet with sensor signal values and identification number (ID).
3. The server checks for the ID and decides if it has to serve the client or not.

4. If it is found eligible, the server program selects the control loop relevant to the client and executes the sensor-control communication.
5. If it is found not eligible, server sends a control packet with control signal equal to ‘0’ that resembles a “reject” signal.
6. In this particular experimental setup, the control signal is always found to be offset from zero, so when the client receives a perfect ‘0’ control signal, it considers it as the “reject” packet. The client then selects a different server and proceeds on with step 2.

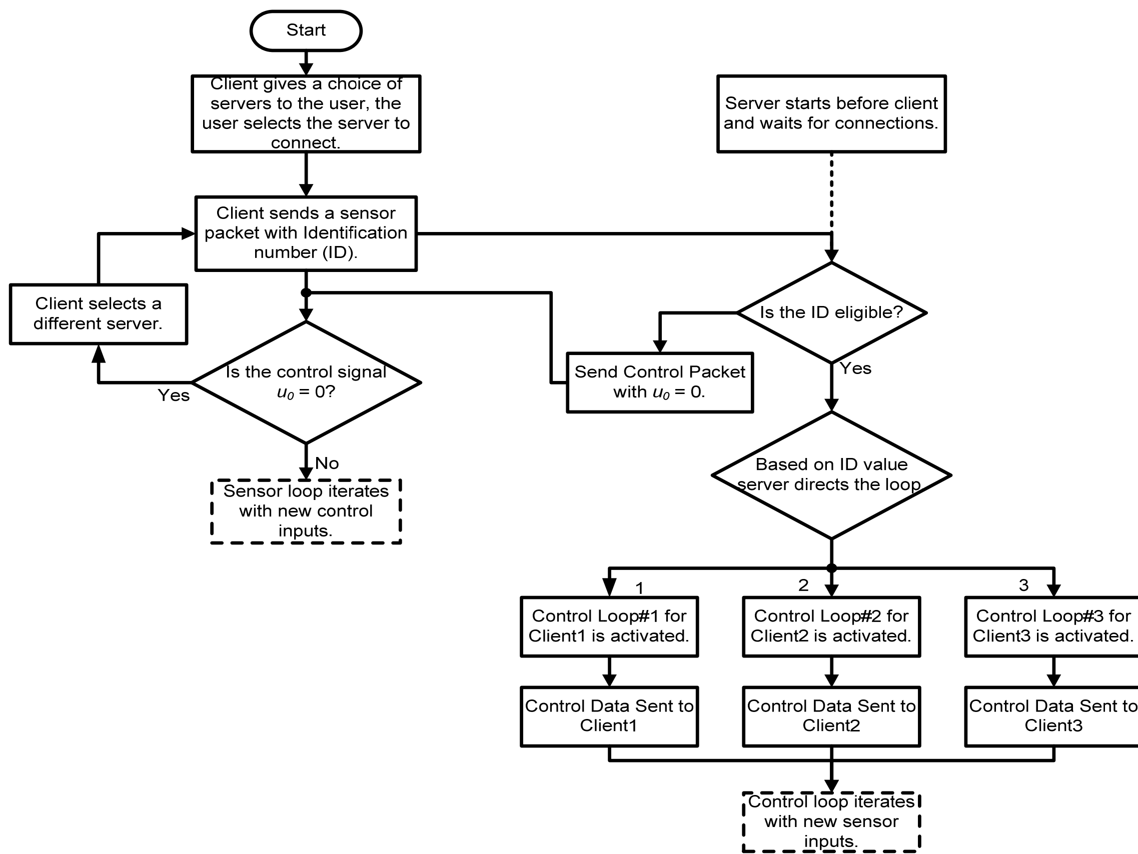



Figure 18. Flowchart of multi-client-multi-server architecture

*D. Calculation of Average Packet Delay ($T_{av}$)*

Average packet delay ($T_{av}$) is measured from the time stamp field. It is originally introduced in the sensor packet to identify if the packet received is an expected one or a delayed one. At the client, during the preparation of the sensor packet, the time stamp is initialized with the current time in nanoseconds using the RTAI function rt_get_time(). When the packet traverses the communication link, it experiences a delay a variable delay that is equal to $T_d$, as explained in the Figure 11. At the server the controller program copies the time stamp from the sensor packet to the control packet and sent back to the client experiencing another variable delay equal to $T_d$.

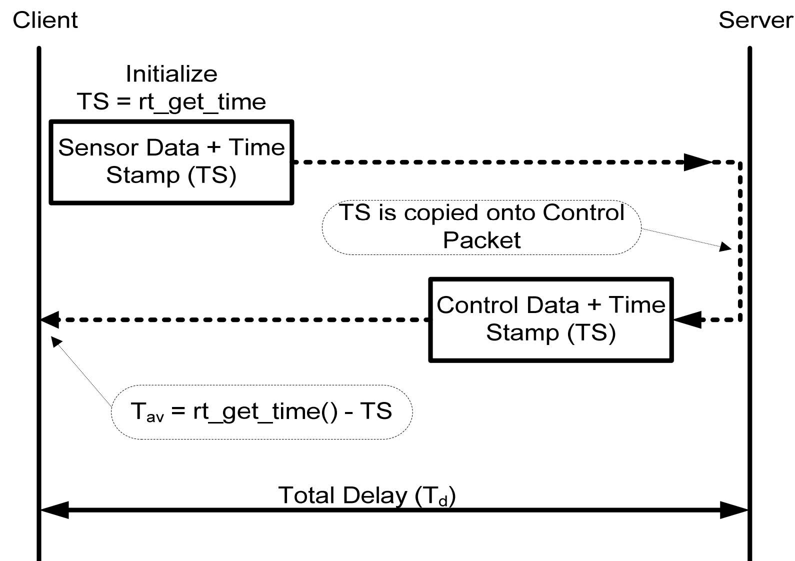



Figure 19. Average packet delay calculation

At the client the average delay ($T_{av}$) is calculated by comparing the time stamp field with the current time. From Figure 19, it can be deduced that the average delay can be assumed to be twice of the total delay ($T_{av} = 2 \times T_d$). It should also be noticed that the delay experienced from client to server need not be equal to that experienced from server to client because of the probabilistic nature of waiting times generated by Ethernet's

backoff algorithm. So, the calculated round-trip delay gives a more accurate figure of the average delay ($T_{av}$).

*E. Packet Generation*

"PackETH" is a Linux-based graphical user interface tool (GUI) tool to generate Ethernet packets at different lengths and varying packet rates. It is an open-source software with General Public License, as published by the Free Software Foundation. It is also available for windows operating system, but a Linux tool is used throughout this research.

To create a packet, the "Builder" tab is clicked to open the builder window Figure 20. Based on the type of protocol, the corresponding fields are chosen as shown in the. In this scenario, a very basic Ethernet packet is created by selecting "User defined payload" option without specifying any higher protocols like TCP or UDP.  The destination medium access control (MAC) address refers to the server's address and the source refers to the packet generator computer's. The pattern field is an arbitrary number that is used to generate a user-defined payload of specified length as highlighted in Figure 20.

To create a sequence of packets, the "Gen-b" tab is clicked that opens up a generator window. As we are mainly concerned with the packet rate, the "number of packets to send" option is selected as infinite. The packet rate is specified in terms of delay between the packets, say for a packet rate of 5000 packets/s the delay between the packets is 200 μs as entered in the highlighted box in Figure 21.

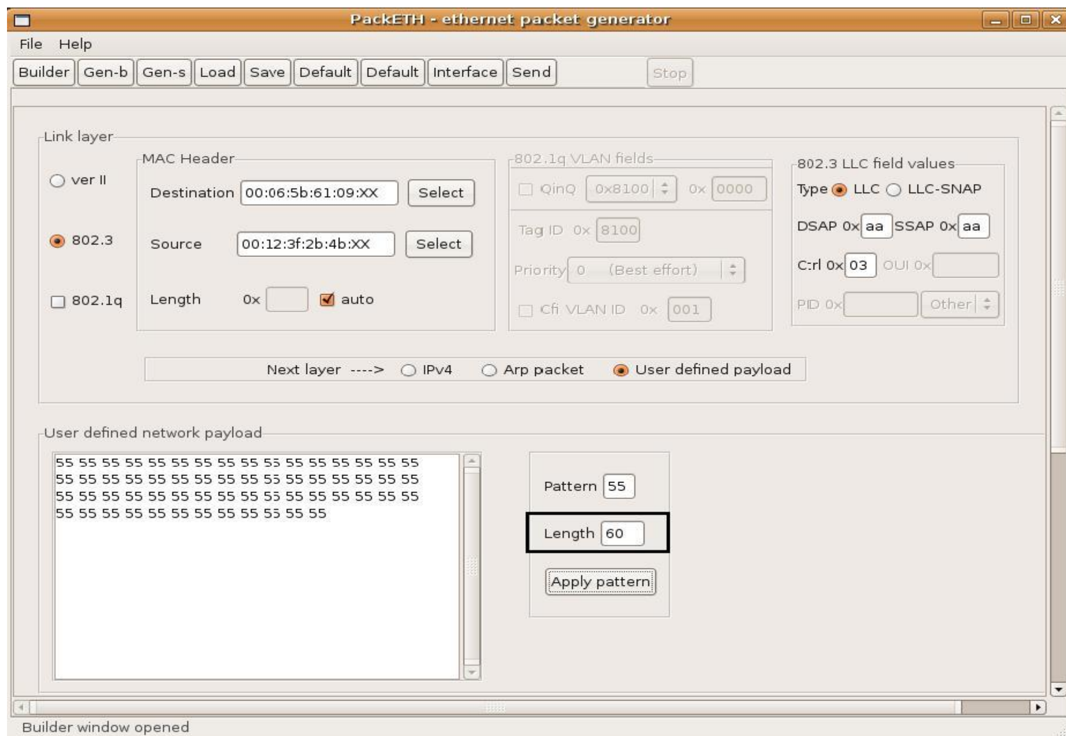



Figure 20. PackETH: Creating a packet

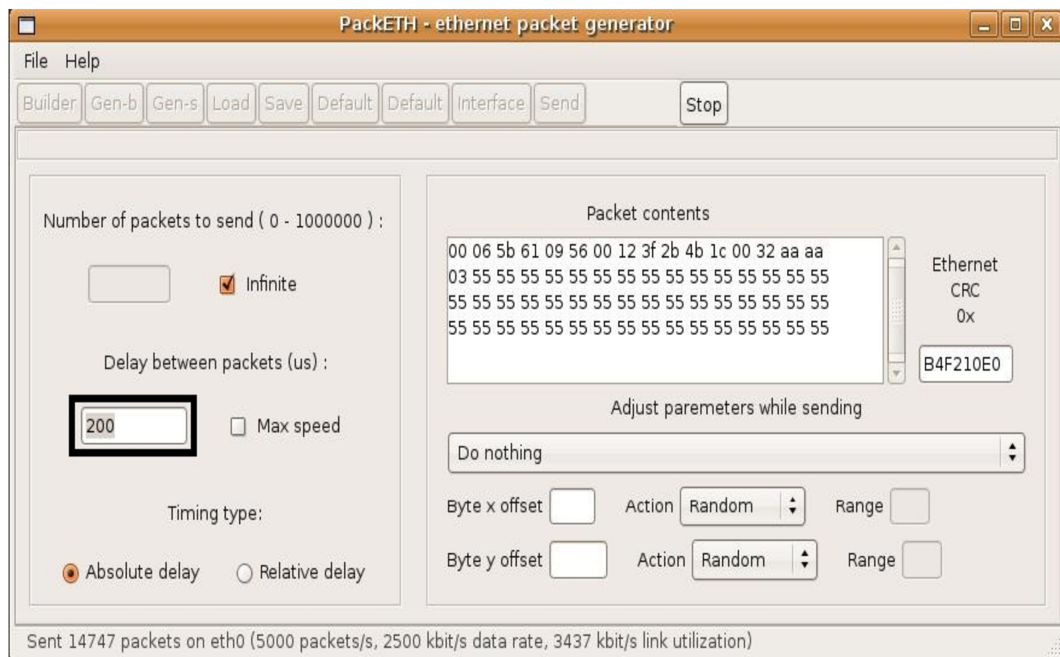



Figure 21. PackETH: Generating a packet

# CHAPTER IV

# RESULTS

In this chapter, the design of three different experimental architectures including single-client-single-server, multi-client-single-server, and multi-client-multi-server is explained. The NCS is loaded using packets generated from a packet generator and, the observations of average packet delay ($T_{av}$) and SSE are presented with graphs.

## *A. Single-Client-Single-Server Architecture*

In this architecture, the setup consists of a single client and single server connected to the network as shown in Figure 22. The client can be either a DC motor or a ball maglev system. A packet generator is also connected to load the network based on the test conditions. Connected to the same LAN, there are other computers that are not part of the networked control system but are communicating with the World Wide Web.

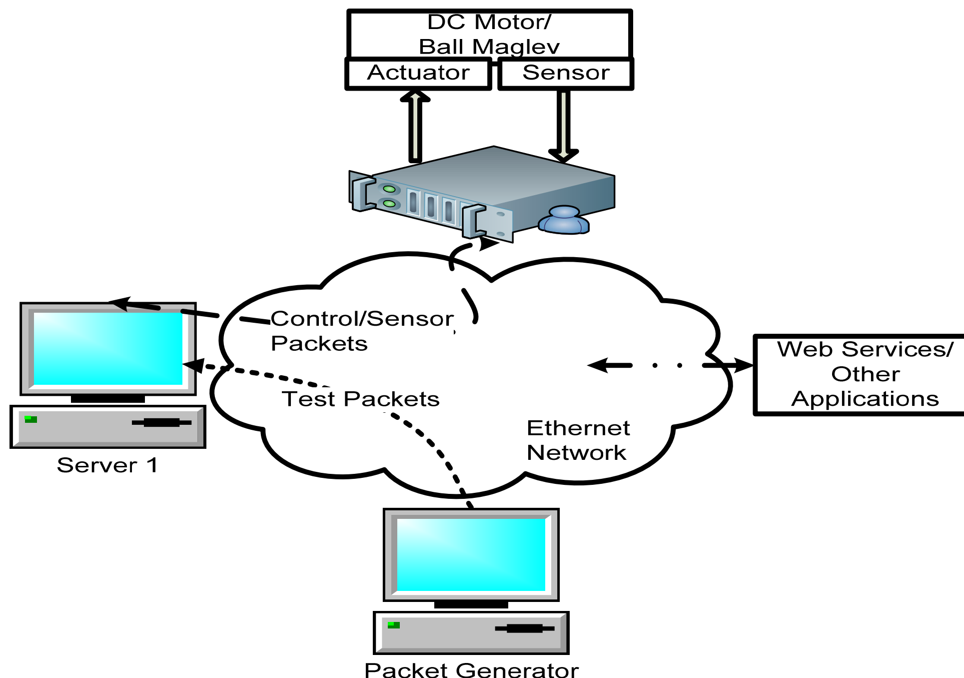



Figure 22. Single-client-single-server architecture

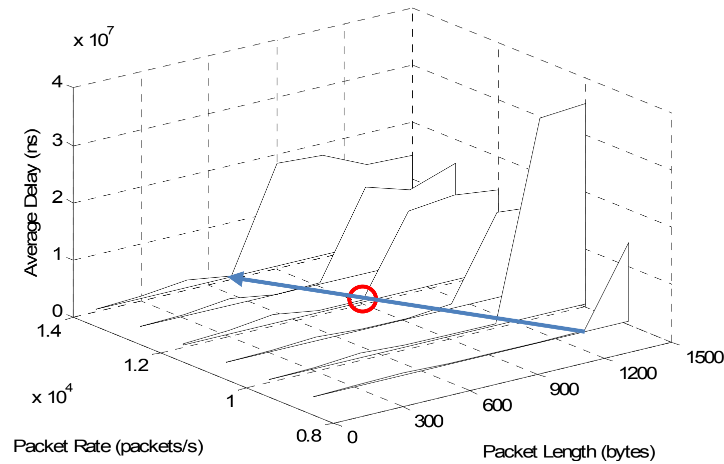
1) *A-1: Increasing the packet length with keeping the packet rate constant*

- Architecture: Single-Server-Single-Client (SC-SS)
- Client: One DC Motor speed-control system
- Methodology: Constant packet rate with variable packet length

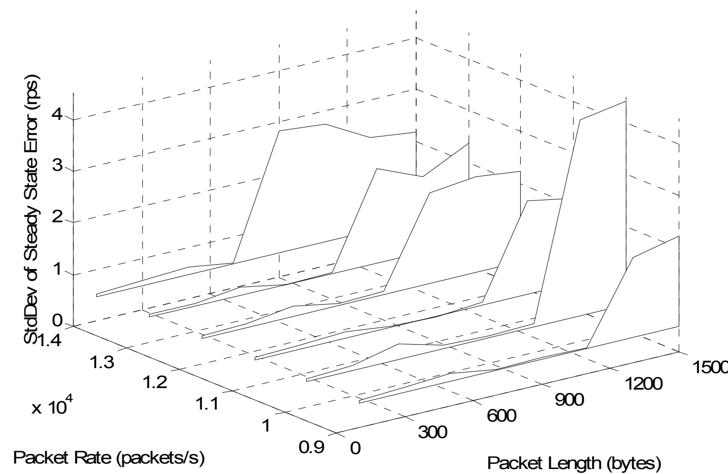
The client is connected to a DC motor speed-control system running at a 3 ms sampling period. It generates control and sensor packets at a rate of 333.3 packets/s in each direction, totaling to 666.6 packets/s between the client and the server. The packet generator is used to send packets at varying packet lengths for different packet rates. In each experiment, the packet rate is fixed and the packet length is increased for every consecutive iteration. The packet length is increased from 100 bytes to 1500 bytes in steps of 300 bytes generating a total of eight data points for every experiment. A total of six experiments are conducted by increasing the packet rate from 9000 packets/s to 14 000 packets/s in steps of 1000 packets/s.

As shown in Figure 23 (a–b), the change in standard deviation of the steady-state error is proportional to the change in average delay. All curves except that of 10 000 bytes packet length, showed a consistent increase in average delay as the packet length increased, the 10 000 packets/s curve shows unusual spikes and can be ignored. Ethernet link utilization is a function of packet rate and packet length, and it can be observed that the spikes in average delay occurred at lesser packet lengths as the packet rate is increased. For example, the highlighted part in Figure 23 (a) shows that at 12 000 packets/s the spike starts to occur at 900 bytes packet length because at that point the corresponding Ethernet link utilization is 86.6%. At 14 000 packets/s the spike starts to occur at 700 bytes packet length because at that point the corresponding Ethernet link utilization is 79.6%. It is interesting to note that for every packer rate, as the packet length approaches the maximum 1500 bytes, the final value of average packet delay is similar for all packet rates. This phenomenon can be explained as, when the link utilization reaches a maximum, the link becomes very congested after which any increase in the packet rate or the packet length doesn't seem to have a effect. The average delay values at the initial packet lengths and at the maximum packet lengths are

observed to be similar. And the total increase in average delay from the smallest packet length to the largest maximum packet length is observed to be 2480%. Correspondingly, the total increase in standard deviation of SSE is 311.2%.




(a)

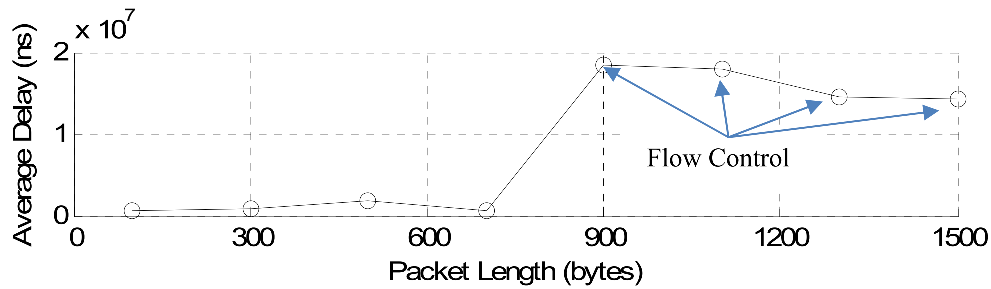



(b)

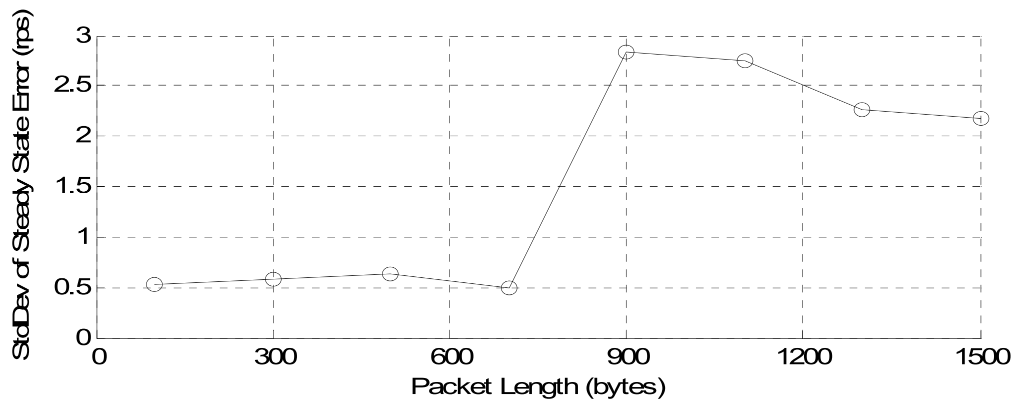
Figure 23. A-1: (a) Average delay (b) Standard deviation of steady-state error

Figure 24 shows that at 14 000 packets/s, as the packet length is increased from 700 bytes to 900 bytes, the average delay increases by more than 3073.8% and the standard deviation of the steady-state error increases by 483.3%. Even though the packet length is increased further, there is no increase in average delay or SSE due to the decreased packet rate. Figure 24 (c) shows that, link utilization reaches 97.4% at 900

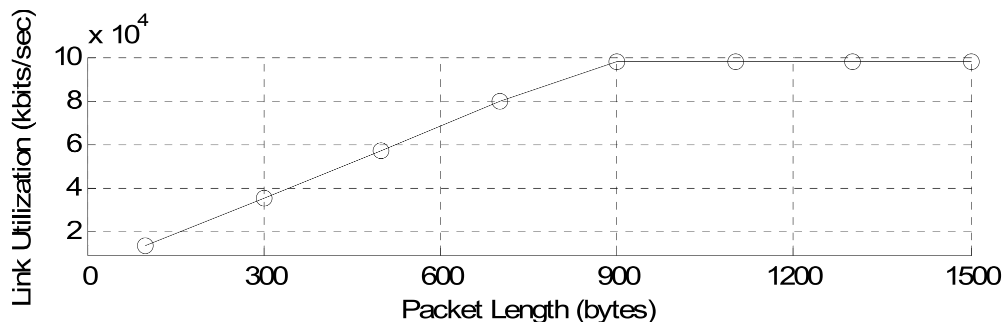
bytes and the flow control mechanism [32], as highlighted in Figure 24(a), forces the hosts to decrease the packet rate to maintain the link utilization under the limit. Though the flow control mechanism is activated to reduce the packet loss and delay, it also restricts the real-time communication between the plant and controller, drastically increasing the packet delays and steady-state error of the system.




(a)




(b)




(c)

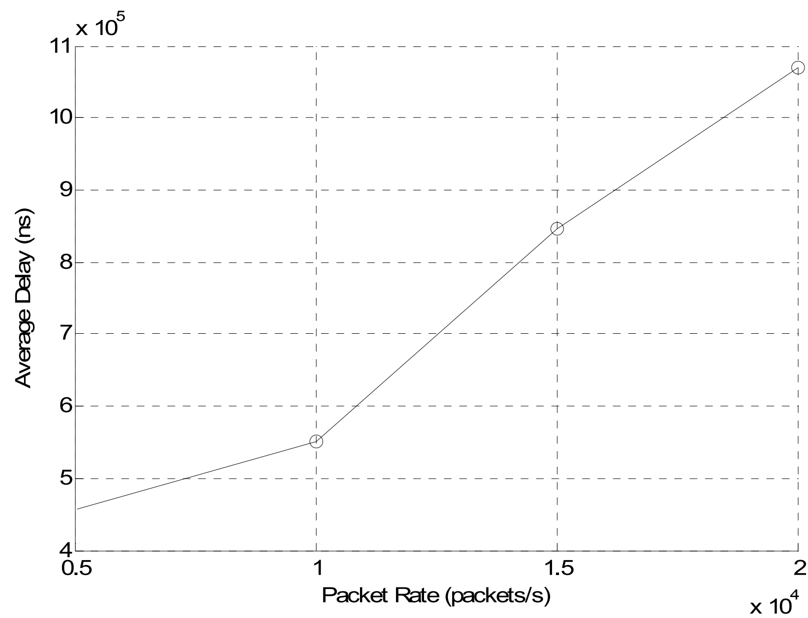
Figure 24. A-1: Packet length (byte) vs (a) Average delay (b) Standard deviation of the steady-state error (C) Link utilization, at constant packet rate of 14 000 packets/s

2) *A-2: Increasing the packet rate with keeping the packet length constant*

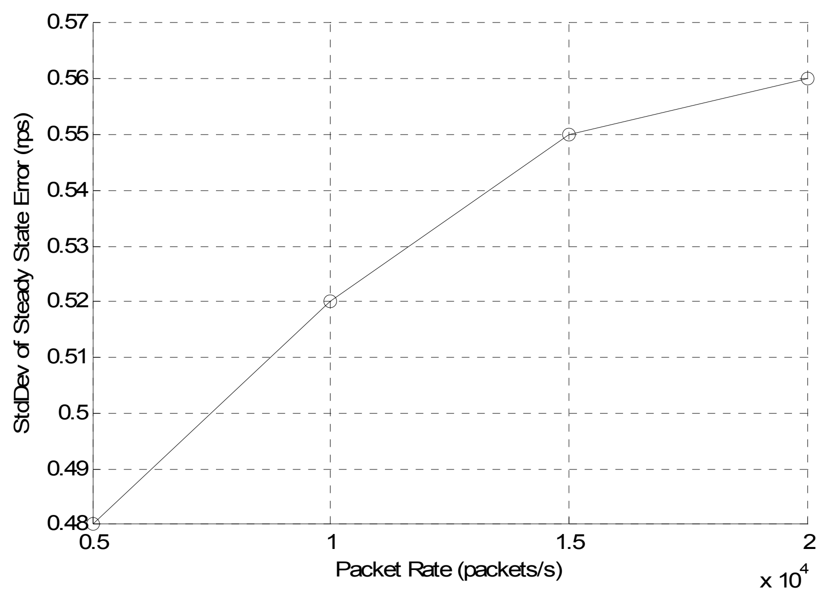
- Architecture: Single-Server-Single-Client (SS-SC)
- Client: One DC Motor speed system
- Methodology: Constant packet length with variable packet rate

The client is connected to a DC motor speed-control system running at a 3 ms sampling period. It generates control and sensor packets at a rate of 333.3 packets/s in each direction, totaling to 666.6 packets/s between the client and the server. The packet generator is used to send packets at an increasing packet rate but a constant length of 64 bytes per packet is maintained. On every consecutive iteration, the packet rate is increased from 5000 to 20 000 packets/s in steps of 5000. During the experiments it is observed that as the packet rate is increased beyond the upper bound at 20 000 packets/sec, the sensor program on the client gets stalled. This is due to the physical limitation of the sending and receiving computers. So, the maximum packet rate is fixed at 20 000 packets/s.

Figure 25 (a) presents a correlation between the packet rate and the average delay. As the packet rate is increased there is an almost linearly proportional increase in the average packet delay. As opposed to the observations in test A-1, the average packet delay and stead-state error is always showing an increasing trend in this scenario. It is because even at 20 000 packets/s, the link utilization is still below 15 Mbps, which represents 15% of the 100 Mbps Ethernet. Theoretically, If the packet rate is increased beyond 20 000 packets/s, the average packet delay and standard deviation in SSE will always show an increase till the link utilization hits the threshold of 80%. Figure 25 (a – b) shows that for a 274% increase in average delay from the initial packet rate to final packet rate, the standard deviation of the steady-state error increased by 16.5%.




(a)




(b)

Figure 25. Packet rate vs. (a) Average delay (ns) (b) Standard deviation of the steady-state error (rps), at constant packet length of 64 bytes

*B. Multi-Client-Single-Server Architecture*

In this scenario, the setup consists of two clients and one server connected to the network as shown in Figure 26. The two clients are connected to two identical DC motor systems. A packet generator is also connected, to load the network based on the test conditions. Connected to the same LAN, there are other computers that are not part of the NCS but are communicating with the World Wide Web.

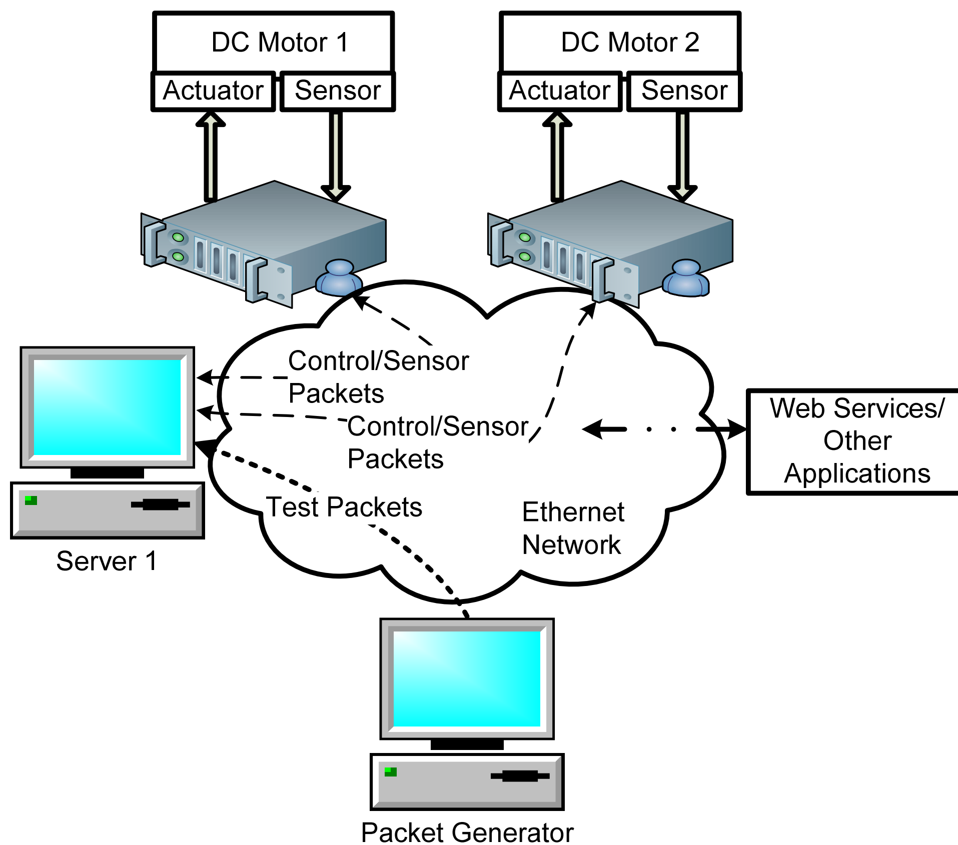


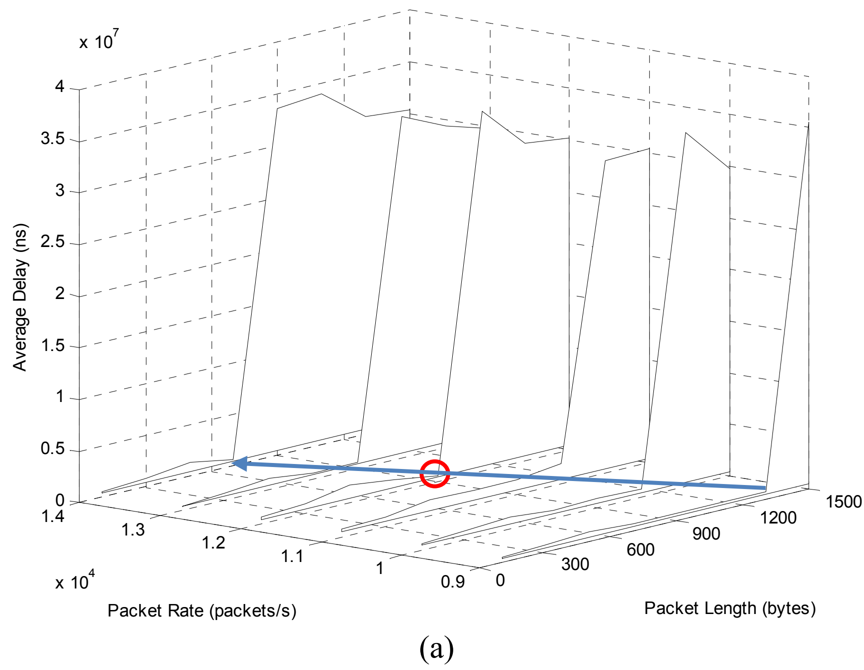

Figure 26. Multi-client-single-server architecture

1) *B- 1: Increasing the packet length with keeping the packet rate constant*

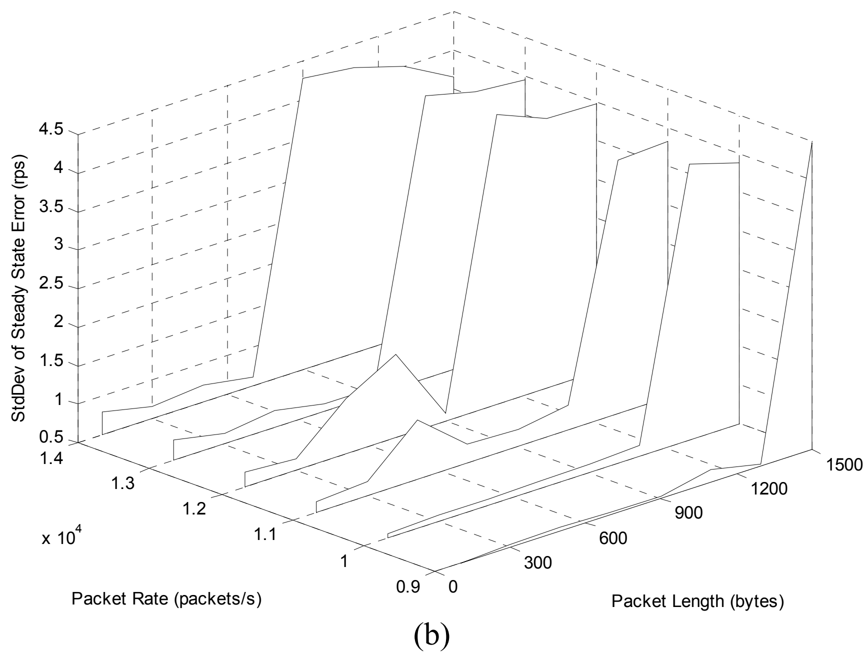
- Architecture: Multi-Client -Single-Server (MC-SS)
- Clients: Two DC Motor speed-control systems
- Methodology: Constant packet rate with variable packet length

The client is connected to a DC motor speed-control system running at a 3 ms sampling period. It generates control and sensor packets at a rate of 333.3 packets/s in each direction, totaling to 666.6 packets/s between the client and the server. As there are two DC motors connected to the network, the total sensor-control packets generated are 1333.2 packets/s. The packet generator is used to send packets at varying packet lengths for different packet rates. In each experiment, the packet rate is fixed and the packet length is increased for every consecutive iteration. The packet length is increased from 100 bytes to 1500 bytes in steps of 300 bytes generating a total of eight data points for every experiment. A total of six experiments are conducted by increasing the packet rate from 9000 packets/s to 14 000 packets/s in steps of 1000 packets/s.

As shown in Figure 27 (a–b), the change in the standard deviation of the steady-state error is proportional to the change in average packets delay. The spikes in average delay occur at the same juncture as in the previous single-client architecture. For example, the highlighted part in Figure 27 (a) shows that at 12 000 packets/s the spike starts to occur at 900 bytes packet length whereas at 14 000 packets/s the spike starts to occur at 700 bytes packet length. The main difference between the single and multi-client scenarios is the magnitude of the delay, the total increase in average delay from the smallest packet length to largest maximum packet length is observed to be 4984%. Correspondingly, the total increase in standard deviation of SSE is 488.7%. These values are almost twice that of those observed in the SC-SS architecture. This drastic increase can be attributed to the rapid increase in queues at the switch shown in Figure 17. A detailed comparison of performance of these two architectures is to be explained in Chapter V.




(a)




(b)

Figure 27. B-1: (a) Average delay (b) Standard deviation of steady-state error

Considering the example of 14 000 packets/s, Figure 28 shows that, as the packet length is increased from 700 bytes to 900 bytes the average delay increases by 5532%

and the standard deviation of the steady-state error increases by 604.8%. Though the packet rate is kept fixed at 14 000 packets/s, the actual packet rate decreases because of the Ethernet's flow control mechanism [32], as highlighted in figure. Figure 28 (c) shows that the link utilization reaches the maximum allowable limit at 900 bytes and flow control mechanism forces the hosts to decrease the packet rate to maintain the link utilization under limits.

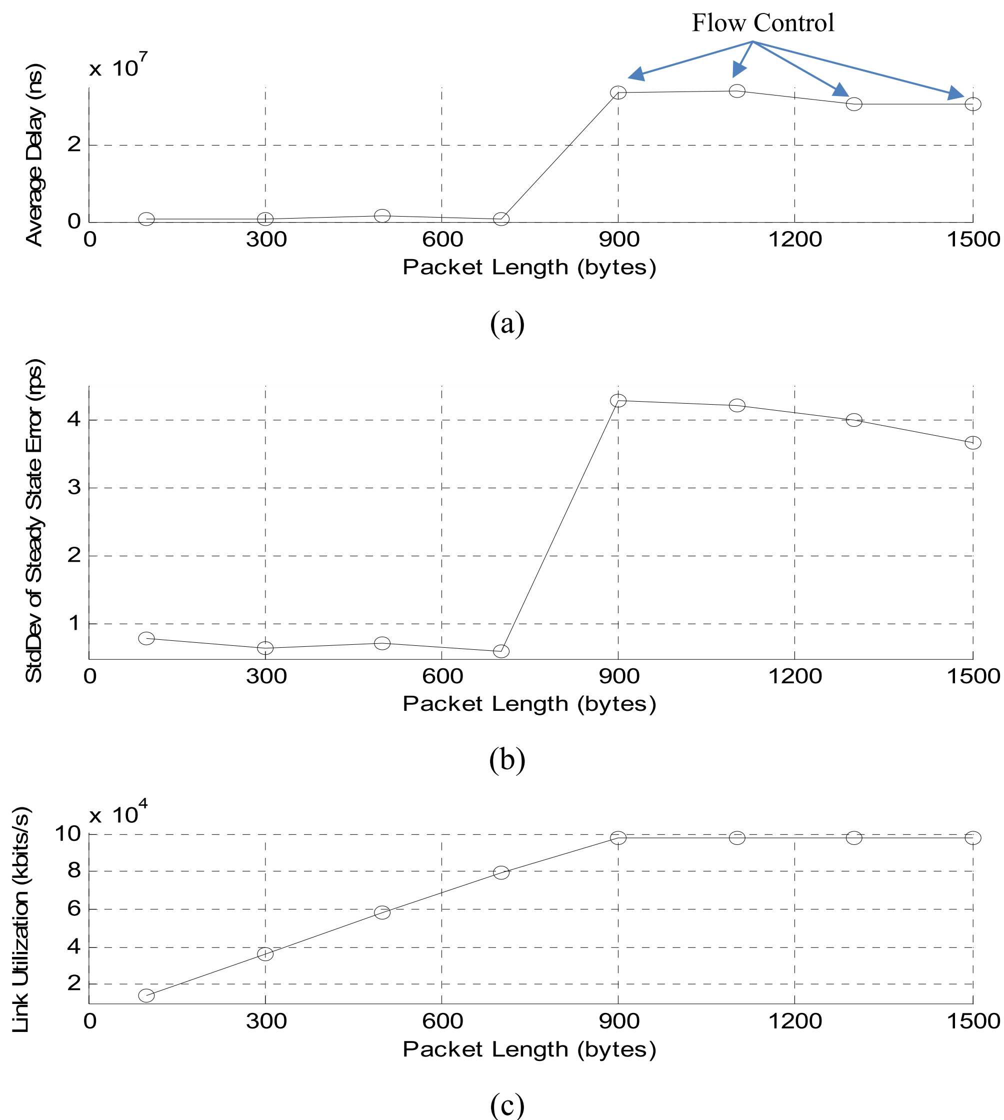


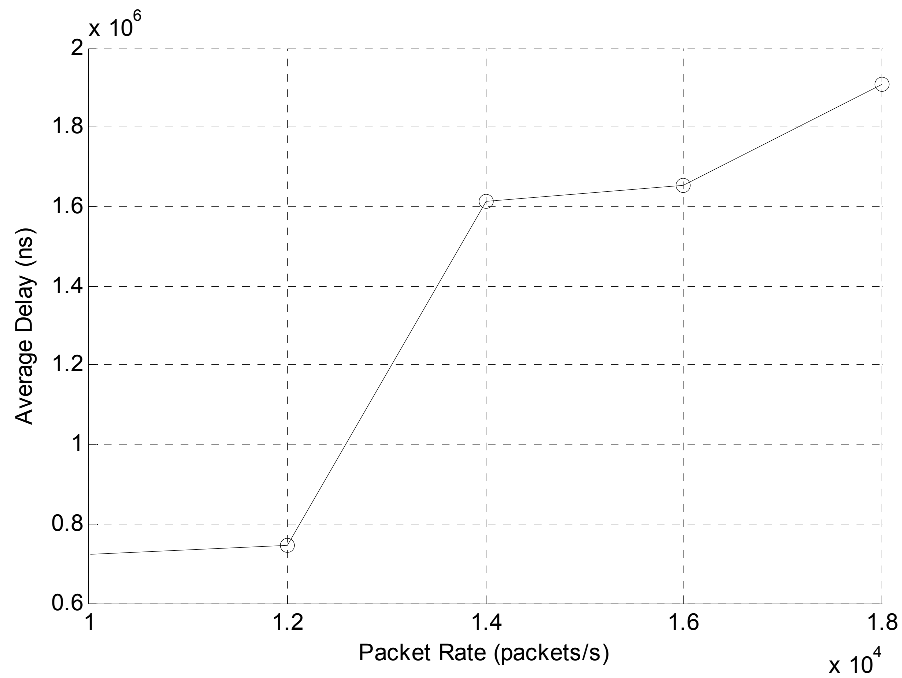

Figure 28. B-1: Packet length (byte) vs (a) Average delay (b) Standard deviation of steady-state error (c) Link utilization, at constant packet rate of 14000 packets/s

*2) B-2: Increasing the packet rate with keeping the packet length constant*

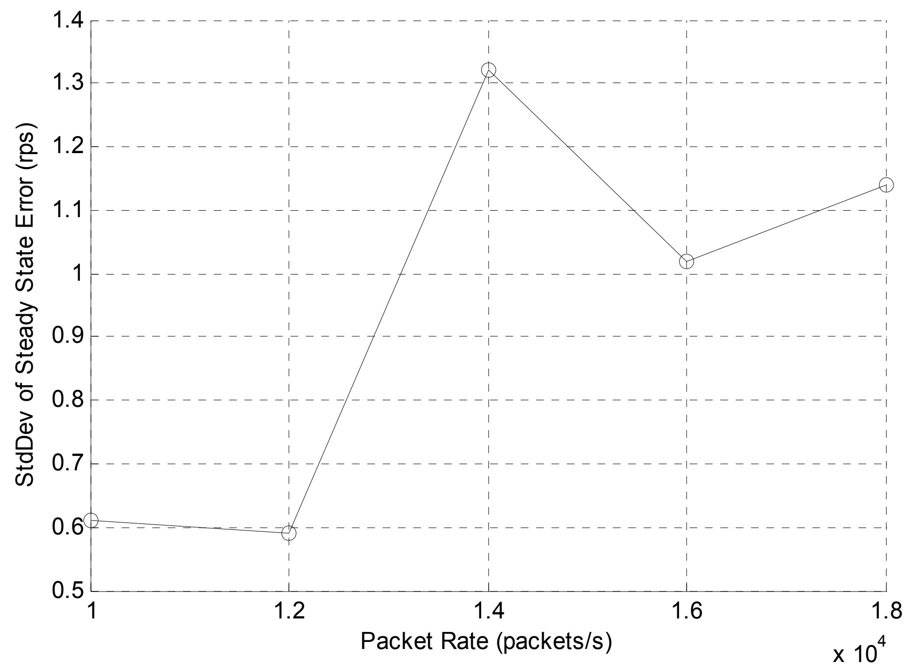
- Architecture: Multi-Client-Single-Server (MC-SS)
- Client: Two DC Motor speed-control systems
- Methodology: Constant packet length with variable packet rate

The client is connected to a DC motor speed-control system running at a 3 ms sampling period. It generates control and sensor packets at a rate of 333.3 packets/s in each direction, totaling to 666.6 packets/s between the client and the server. As there are two DC motors connected to the network, the total sensor-control packets generated are 1333.32 packets/s. The packet generator is used to send packets at an increasing packet rate but packet length is maintained at a constant length of 64 bytes per packet. On every consecutive iteration, the packet rate is increased from 10 000 to 18 000 packets/s in steps of 2000. Contrary to the single-client scenario, the maximum packet generation is limited to 18 000 packets/s in multi-client scenario. During the experiments it is observed that, as the packet rate is increased beyond 18 000 packets/sec, the sensor-control communication stalled due to the excessive swamping of packets. This is due to the physical limitation of the sending and receiving computers. So, the maximum packet rate is fixed at 18 000 packets/s.

Figure 29(a) presents a correlation between the packet rate and the average delay. As the packet rate is increased the average delay also increases. As opposed to the observations in scenario B-1, the average delay is always increasing in this case because even at 18 000 packets/s, the link utilization is still below 15 Mbps, which represents 15% of the 100 Mbps Ethernet. Theoretically, if the packet rate is increased beyond 18 000 packets/s, the average packet delay and standard deviation in SSE will always show an increase till the link utilization hits the threshold of 80% after which the flow control gets activated. Figure 29(b) shows that from 10 000 packets/s to 18 000 packets/s, an increase of 164% in average delay led to an increase of 85.3% in the standard deviation of steady-state error.




(a)




(b)

Figure 29. B-2: Packet rate vs. (a) Average delay (ns) (b) Standard deviation of steady-state error (rps), at constant packet length of 64 bytes

*C. Multi-Client-Multi-Server Architecture*

In this scenario, the setup consists of two clients and two servers connected to the network as shown in Figure 30. Each client is connected to a DC motor speed-control systems that are identical to each other. As in the previous architectures, the packet generator is swamping Server 1 with packets at different rates and sizes. In this architecture, the client rejection algorithm explained in Section *III.A* is employed, where the Server 1 will serve only one client and rejects all new requests. Client 2 is allowed first to send a request to Server 1 and it gets accepted but when Client 1 sends a connection request, Server 1 rejects it. In accordance with the multi-server algorithm, Client 1 tries to connect to another server and Server 2 accepts it. So, we have a networked control system that has Server 1 serving Client 2 and Server 2 serving Client 1. It should be noted that the packet generator is swamping packets only on Server 1.To evaluate the effects of adding an additional server in the NCS, the analysis is mainly focused on Client 1. The following sections explain the different test scenarios on this architecture.

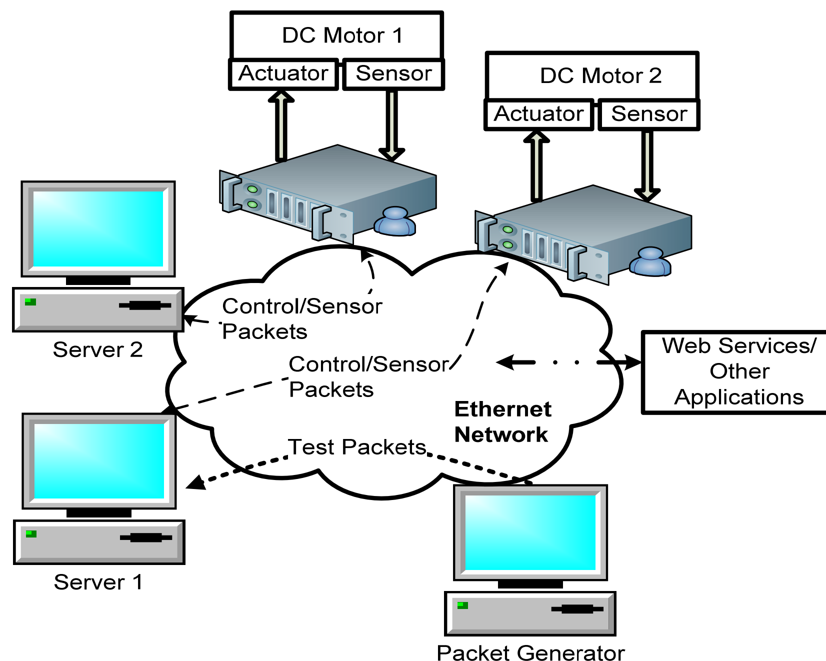


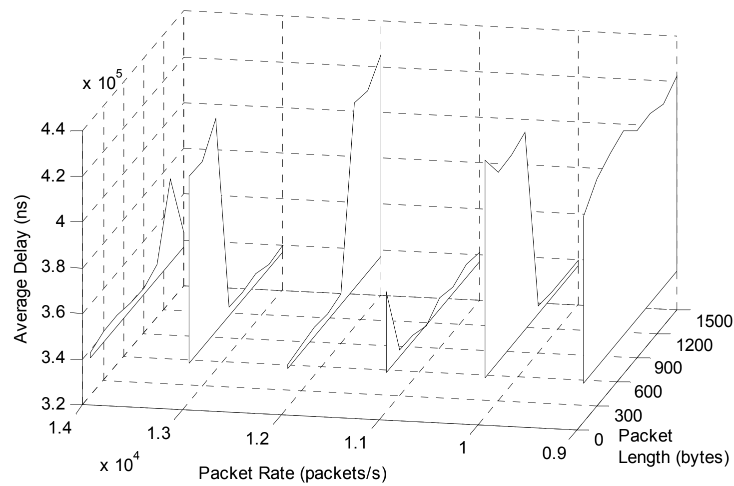

Figure 30. Multi-client-multi-server architecture

*1) C-1: Increasing the packet length with keeping the packet rate constant*

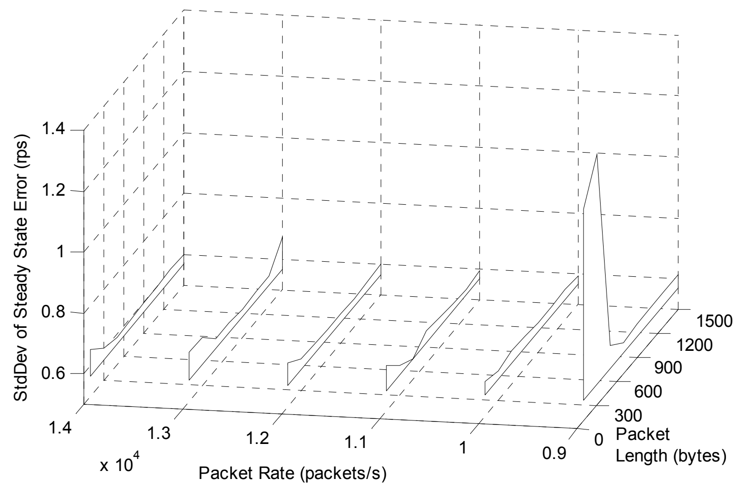
- Architecture: Multi-Client- Multi-Server (MC-MS)
- Clients: DC Motor speed-control system
- Methodology: Constant packet rate with variable packet length

The client is connected to a DC motor speed-control system running at a 3 ms sampling period. It generates control and sensor packets at a rate of 333.3 packets/s in each direction, totaling to 666.6 packets/s between the client and the server. As there are two DC motors connected to the network, the total sensor-control packets generated are 1333.32 packets/s. The packet generator is used to send packets at varying packet lengths for different packet rates. In each experiment, the packet rate is fixed and the packet length is increased for every consecutive iteration. The packet length is increased from 100 bytes to 1500 bytes in steps of 300 bytes generating a total of eight data points for every experiment. A total of six experiments are conducted by increasing the packet rate from 9000 packets/s to 14 000 packets/s in steps of 1000 packets/s. So, a total of 48 data points are generated in this test.

Figure 31(a–b) shows the average delay and SSE observations of client 1. It can be clearly observed from Figure 31(a) that there is no particular relation between average delay and packet length. This is because; Server 2 had to deal with only the control packets and had a non-congested communication link to Client 1. The packet generator is swamping packets only on Server 1 and the link between Server 1 and Client 2 is the one that is highly congested. The 9000 packet/s curve can be ignored from analysis because it shows unusually higher delays and correspondingly, unusually higher SSE. The average delay experienced at different packet rates is almost similar and inconclusive, but the average delay at the highest packet rate 14 000 packets/s is observed to be 34 5409.1 ns. The second graph also corroborates the fact that the link utilization overload between Client 2 and Server 1 had no effect on the performance of Client 1. On average, the standard deviation of the SSE is observed to be at 0.63 rps.




(a)




(b)

Figure 31. C-1: (a) Average delay (b) Standard deviation of steady-state error

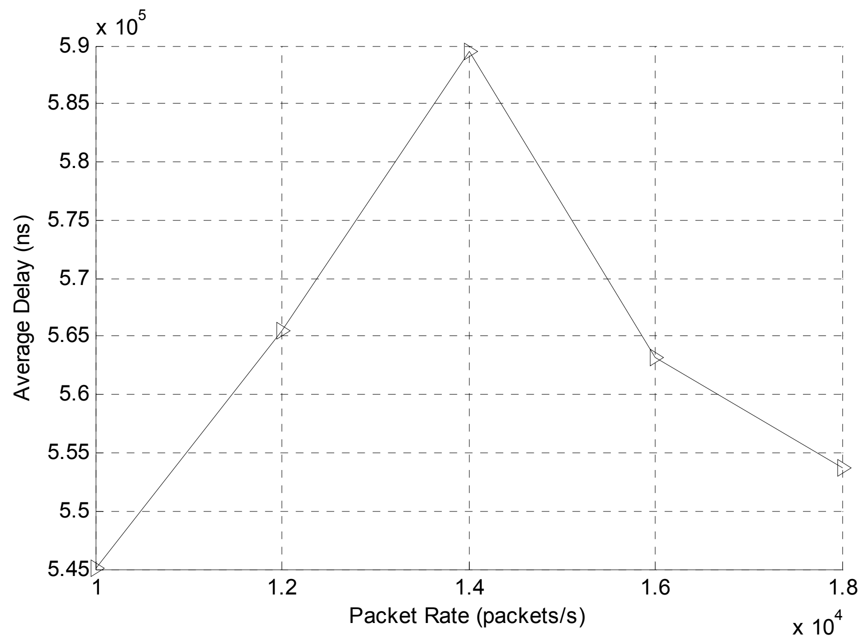
2) *C-2: Increasing the packet rate with keeping the packet length constant*

- Architecture: Multi Client and Multi Server (MC-MS)
- Clients: Two DC Motor speed system
- Methodology: Constant packet length with variable packet rate

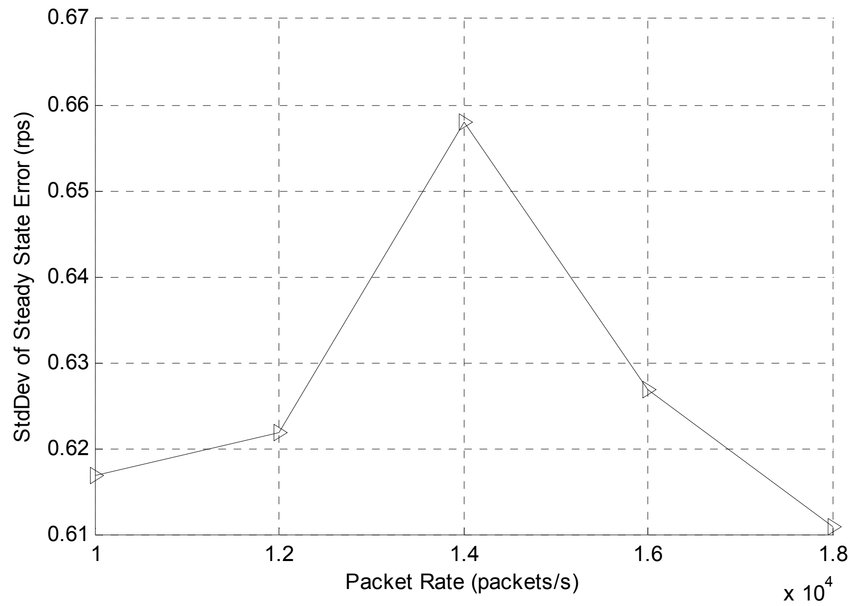
The client is connected to a DC motor speed-control system running at a 3 ms sampling period. It generates control and sensor packets at a rate of 333.3 packets/s in

each direction, totaling to 666.6 packets/s between the client and the server. As there are two DC motors connected to the network, the total sensor-control packets generated are 1333.32 packets/s. The packet generator is used to send packets at an increasing packet rate but packet length is maintained at a constant length of 64 bytes per packet. On every consecutive iteration, the packet rate is increased by 2000 packets/s going from 10 000 to 18 000 packets/s.  Contrary to single-client architecture, the maximum packet generation is limited to 18 000 packets/s in this architecture. During the experiments it is observed that, as the packet rate is increased beyond 18 000 packets/sec, the sensor-control communication stalled due to the excessive swamping of packets. This is due to the physical limitation of the sending and receiving computers. So, the maximum packet rate is fixed at 18 000 packets/s.

Figure 32(a) presents a correlation between the average delay and standard deviation in SSE. It is clearly shown that the average packet delay and standard deviation of the SSE are no more a function of the packet rate. Contrary to previous architectures, the packet rate has no effect on the average delay because, the link between Server 2 and Client 1 is not congested. The packet generator is swamping packets only on Server 1 and the link between Server 1 and Client 2 is the one that is highly congested. The average delays observed in Client 1 are the similar to the ones observed in a normal scenario without any packet generator. In accordance with our previous observations and published theory, the standard deviation of the SSE varies with the delays experienced. The average packet delay increased by 4.9% from the initial value, and the standard deviation of the SSE value showed no corresponding variation to this. It is because, previous observations confirmed that the delay variations need to much higher to have an appreciable effect on the SSE.




(a)




(b)

Figure 32. C-2: Packet rate vs. (a) Average delay (ns) (b) Standard deviation of steady-state error (rps), at constant packet length of 64 bytes

# CHAPTER V

# PERFORMANCE COMPARISONS

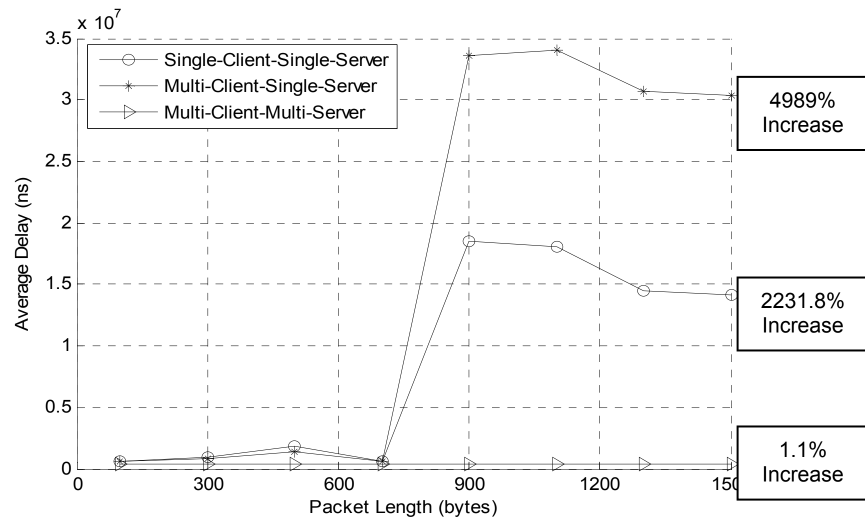
In this chapter, the performance of the network control system in SC-SS, MC-SS, and MC-MS architectures are compared with each other. The performance measures used in this research are average delay ($T_{av}$) and standard deviation in steady-state error (SSE).

The speed-control system reaches a steady state after about 50 samples, which represents150 ms in time. Then, the difference between the desired speed and the actual speed is measured as the SSE. The standard deviation of SSE is calculated and plotted against the packet length and packet rate.

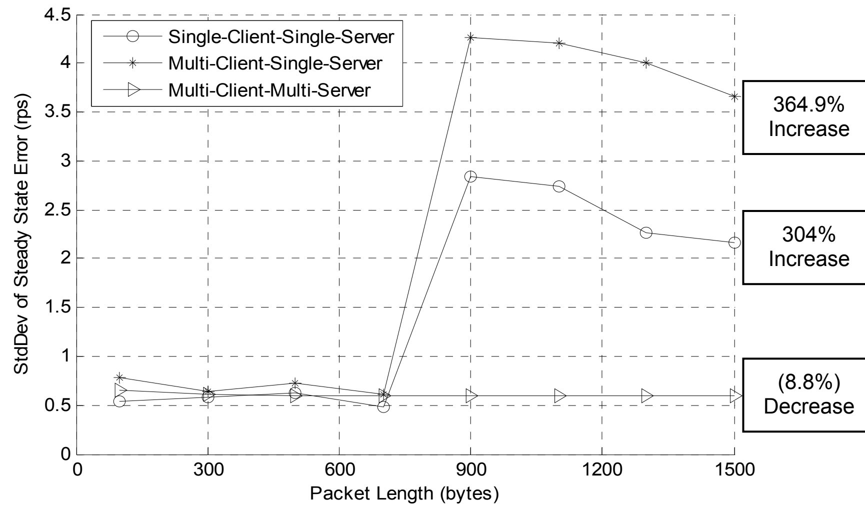
## *A. Increasing the Packet Length with Keeping the Packet Rate Constant*

Figure 33 shows the observed average delay and SSE at an increasing packet length from 100 bytes to 1500 bytes with constant packet rate of 14 000 packets/s. In both SC-SS and MC-SS architectures, the average delay and SSE show a similar pattern till 700 bytes packet length. As the packet length reaches 900 bytes, the average packet delay ($T_{av}$) of Client 1 in the MC-SS architecture increases rapidly. At the maximum packet rate and packet length, the average packet delay in the MC-SS architecture shows an increase of 4989% from its initial delay. The SC-SS architecture also exhibited a similar drastic increase at 900 bytes but as it had only one client, its delay at 900 bytes is 82% lesser than that of the MC-SS architecture. The packet delay in the SC-SS architecture showed an increase of 2231.8% from its initial value.

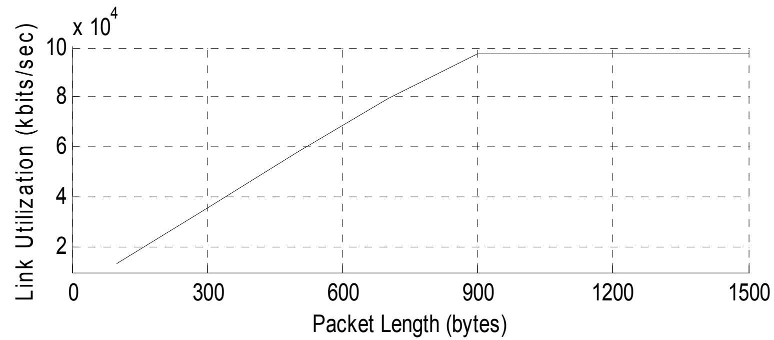
As shown in Figure 33(a) packet delays in the MC-MS architecture remained almost constant, showing an increase of 1.1%. When Client 1 found that Server 2 was already serving Client 2 and was also loaded with the packet generator, it connected to Server 1 which was not serving any clients at that time. Therefore, the link between Client 1 and Server 2 was less congested and exhibited very low packet delays and SSE.




(a)




(b)




(c)

Figure 33. Performance comparisons of experiments A-1, B-1, and C-1

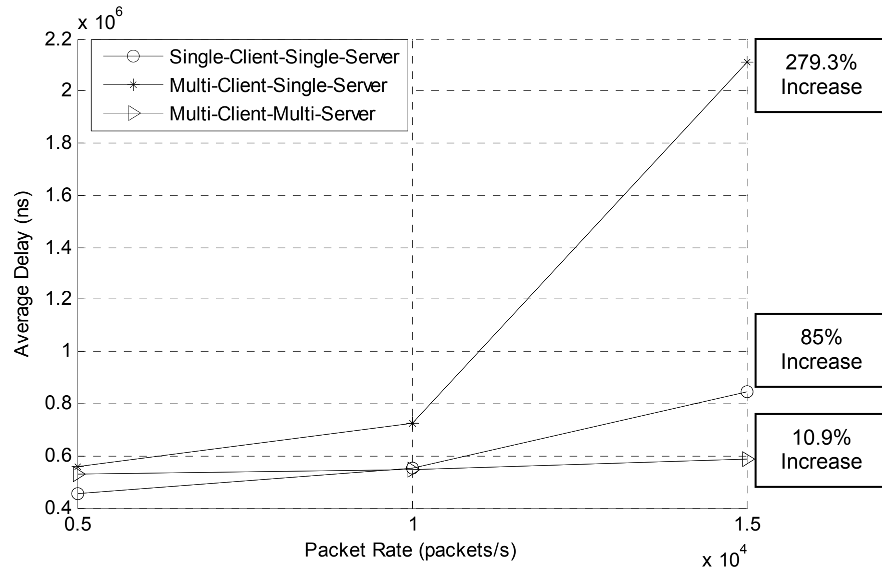
In the SC-SS architecture, as there was only one server, the link between the server and client had high traffic due to the sensor packets from the client and packets from the generator tool. In the MC-SS architecture, an extra client on the network further increased the number of packets that are trying to communicate with the server. As all these computers are connected to a switch at the back-end, there will be a rapid increase of the queue associated with the single server and due to the queuing delay increase average packet delay also increases rapidly. Although it is unfair to compare the MC-MS architecture with the other two architectures, it proves that even though one client-server communication is highly stressed, real-time communication between other servers and plants can be guaranteed.

In accordance with increase in average packet delays, the standard deviation of the SSE also varied. As expected, standard deviation if the SSE showed highest increase in MC-SS architecture with a rise of 364% from its initial value. Client 1 in the SC-SS architecture performed a little better than that of the MC-SS architecture, where it showed an increase of 304% from its initial value. However, in the MC-MS architecture, the Client 1 showed no increase in SSE because of the minimal increase (1.1%) in average packet delay.

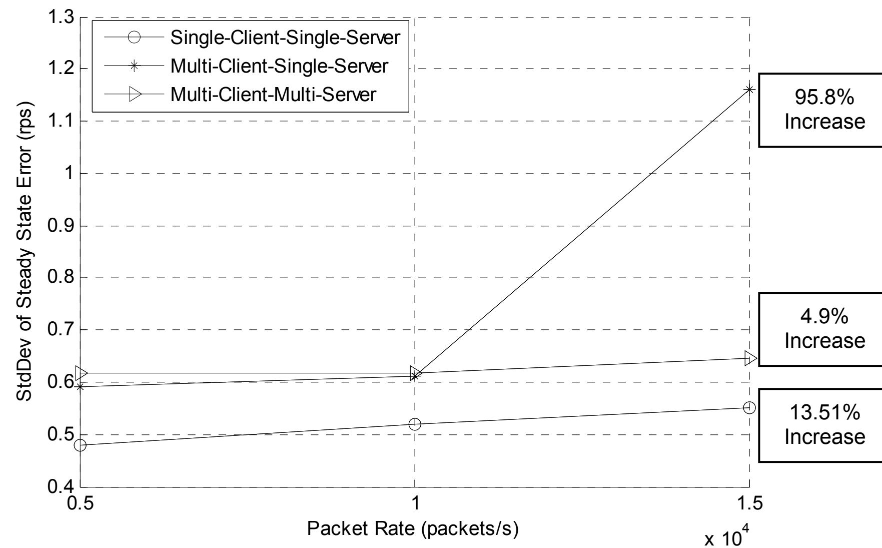
*B. Increasing the Packet Rate with Keeping the Packet Length Constant*

Figure 34 shows the observed average packet delay and SSE at an increasing packet rate from 5 000 packets/s to 15 000 packets/s with constant packet length of 64 bytes. To compare the performance of three architectures, three sets of packet rates (5000, 10 000 and 15 000) are considered. It is clear from the observations that as the packet rate is increased, the average delay also showed an increase in both the SC-SS and MC-MS architectures. At the final packet rate of 15 000 packets/s, the average packet delay of Client 1 in a MC-SS architecture shows an increase of 279.3 from its initial delay value. This is very high when compared to the packet delay observed in the SC-SS architecture; it showed an increase of 85% from its initial value. It is notable that the packet delays in the MC-MS architectures are even lesser than SC-SS architecture

because the client-server communication link had minimum congestion due to which there was minimum queuing delay at the switch.




(a)




(b)

Figure 34. Performance comparisons of experiments A-2, B-2, and C-2

Correspondingly, Figure 34 (b) shows that the plant in MC-SS architecture experienced the highest increase in the standard deviation of the SSE at 95.8% when

compared to the other two architectures. The client in the SC-SS architecture showed an increase of 13.5% from it is initial SSE. Although the client in the MC-MS architecture had the least increase in SSE at 4.9%, the initial value of SSE is still higher than that of the SC-SS architecture. It is because, in the SC-SS architecture there is only one client on the network while in both the MC-SS and MC-SS architectures, there are two clients on the network. The initial average packet delay in MC-MS architecture is higher than that of the SC-SS architecture and so, the standard deviation of SSE is also observed to be at a higher level to that of the SC-SS architecture.

# CHAPTER VI

# CONCLUSIONS

In this chapter, the key findings and results of this research are summarized. Based on these key results, some important conclusions regarding the application of switched Ethernet in a real-time NCS are derived.

## *A. Summary*

This research is an effort to experimentally verify if Ethernet can be used as a real-time communication standard in a factory automation setup. The existing single-server-multi-client architecture was redesigned and extended to a multi-client-multi-server architecture. An Ethernet-based multi-client-multi-server networked control system is designed and developed. As the factory communication systems transport both nonreal-time and real-time data, it is important to evaluate if Ethernet (IEEE 802.3) can be used as a real-time standard for communication. The major drawback of Ethernet is the nondeterministic nature of the delays that makes it difficult to enforce timing guarantees. To evaluate this, the NCS is used as a test bed and the network is loaded using a packet generator. The experiments are designed to simulate different kinds of load by varying packet sizes and packet rates. Key results from this research are summarized as follows:

1. When under heavy load, the client rejection algorithm in the MC-MS architecture successfully rejected additional clients.
2. The rejected client automatically connected to a new server and was able to successfully complete the control communication.
3. Although the introduction of an additional client to a SC-SS architecture did raise the average packet delays, it was not until Ethernet's link utilization reached the maximum network capacity that the performance started degrading rapidly.

4. The threshold at which the average packet delays starts to increase drastically and performance degrades rapidly is found to be at around 80% of link utilization.
5. The average packet delays and performance degradation is far higher in the case of MC-SS than MC-MS.
6. In the experimental setup, all the clients and servers were residing on the same LAN. Even if a communication between one pair of client-server is overloaded with packets and the corresponding plant shows performance degradation, newer clients can connect to a different server on the same network and show a good performance.
7. Based on the various packet size and packet rate experiments it was observed that at very high packet rates, as the Ethernet packet length is increased beyond 50% of the maximum transmission unit (MTU), 1500 bytes, the plant's performance started to degrade rapidly.
8. The real-time sensor-control packets used in this research are less than 1% of the MTU and the remaining capacity can handle large nonreal-time packets without any performance degradation of the plants.

*B. Conclusions*

A series of experiments were conducted to analyze the network at different load conditions, and the conclusions derived from this research are as follows:

1. From the experimental evaluation of the steady-state error in the SC-SS and MC-MS architectures, it is found that even at the maximum possible packet rates, the NCS showed minimum performance degradation. Therefore, a switched Ethernet-based network can be employed in the real-time communication architecture without any performance degradation. This is contingent upon the fact that additional servers should be introduced under heavy loads.

2. Until the load on the network is kept below the maximum threshold of link utilization, the NCS showed good performance. It is only beyond the threshold that the performance of NCS started to degrade rapidly. However, in a real-time scenario a 100-Mbps high speed-Ethernet network is seldom loaded to its maximum capacity [30].
3. Based on the above observations, it is evident that switched Ethernet holds a great potential in factory communication systems. The adoption of switched Ethernet can overcome the existing issues of high cost of wiring, proprietary protocols, and incompatibility.

*C. Future Work*

The communication architectures used in this research rely on the Ethernet flow control mechanism to handle congestion. However, it was found that under heavy loads real-time communication experienced excessive packet delays. It can be further researched to differentiate between real-time and nonreal-time communication and handle the flow control mechanism in an efficient manner.

The plants used in this research had identical sampling periods, and they are maintained at the highest to simulate high load conditions. Further study can be done to determine the optimal sampling period at different load conditions. As load on the network increases, average packet delays also increase. The sampling period can be made to dynamically change based on the different loads on the network.

# REFERENCES

[1] A. Leon-Garcia, and I. Widjaja, *Communication Networks*, New York, NY: McGraw-Hill, Inc, 2004.

[2] Y. Halevi, and A. Ray, “Integrated Communication and Control Systems: Part I-Analysis,” *ASME Journal of Dynamic Systems, Measurement and Control,* vol. 110, no. 4, pp. 367−373, December, 1988.

[3] A. Srivatsava, “Distributed Real-time Control via Internet,” M.S Thesis, Texas A&M University, 2003.

[4] R. S. Raji, “Smart Networks for Control,” *IEEE Spectrum,* vol. 31, no. 6, pp. 49−55, June, 1994.

[5] J. Nilsson, “Real-time Control Systems with Delays,” Ph.D. Thesis, Lund Institute of Technology, Lund, Sweden, 1998.

[6] J. P. Thomesse, “Fieldbus Technology in Industrial Automation,” *Proceedings of the IEEE,* vol. 93, no. 6, pp. 1073−1101, June, 2005.

[7] F. L. Lian, J. R. Moyne, and D. M. Tilbury, “Performance Evaluation of Control Networks: Ethernet, ControlNet, and DeviceNet,” *IEEE Control Systems Magazine,* vol. 21, no. 1, pp. 66−83, February, 2001.

[8] G. Olsson, and G. Piani, *Computer Systems for Automation and Control*, Englewood Cliffs, NJ: Prentice-Hall, 1992.

[9] E. Tovar, and F. Vasques, “Real-Time Fieldbus Communications Using Profibus Networks,” *IEEE Transactions on Industrial Electronics,* vol. 46, no. 6, pp. 1241−1251, December, 1999.

[10] R. Metcalfe, and D. R. Boggs, “Ethernet: Distributed Packet Switching for Local Computer Networks,” *ACM Communications,* vol. 19, no. 7, pp. 395−404, July, 1976.

[11] F. A. Tobagi, and H. V. Bruce, “Performance Analysis of Carrier Sense Multiple Access with Collision Detection,” *Computer Networks (1976),* vol. 4, no. 5, pp. 245−259, November, 1980.

[12] W. Bux, “Local-Area Subnetworks: A Performance Comparison,” *IEEE Transactions on Communications,* vol. 29, no. 10, pp. 1465−1473, October, 1981.

[13] E. Coyle, and B. Liu, “Finite Population CSMA/CD Networks,” *IEEE Transactions on Communications,* vol. 31, no. 11, pp. 1247−1251, November, 1983.

[14] T. A. Gonsalves, and F. A. Tobagi, “On the Performance Effects of Station Locations and Access Protocol Parameters in Ethernet Networks,” *IEEE Transactions on Communications,* vol. 36, no. 4, pp. 441−449, April, 1988.

[15] S. Tasaka, “Dynamic Behavior of a CSMA-CD System with a Finite Population of Buffered Users,” *IEEE Transactions on Communications,* vol. 34, no. 6, pp. 576−586, June, 1986.

[16] S. Vitturi, “On the Use of Ethernet at Low Level of Factory Communication Systems,” *Computer Standards & Interfaces,* vol. 23, no. 4, pp. 267−277, September, 2001.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext Transfer Protocol–HTTP/1.1”, RFC 2616, 1999.

[18] J. Postel, and J. Reynolds, “File transfer protocol”, RFC 959, 1985.

[19] M. Brilla, and U. Gramm, “MMS: MAP Application Services for the Manufacturing Industry,” *Computer Networks and ISDN Systems,* vol. 21, no. 5, pp. 357−380, July, 1991.

[20] T. Y. Mazraani, and G. M. Parulkar, “Performance Analysis of the Ethernet under Conditions of Bursty Traffic,” in *IEEE Global Telecommunications Conference, GLOBECOM '92*, Orlando, FL, 1992, pp. 592−596 vol.1.

[21] K. Lee, and S. Lee, “Performance Evaluation of Switched Ethernet for Real-Time Industrial Communications,” *Computer Standards & Interfaces,* vol. 24, no. 5, pp. 411−423, November, 2002.

[22] S. C. Paschall, “Design, Fabrication, and Control of a Single Actuator Magnetic Levitation System,” B.S. Honors Thesis, Texas A&M University, 2002.

[23] Agilent Technologies "Datasheet: HEDS-5540 Quick Assembly Two and Three Channel Optical Encoder", 2002. [Online]. http://www.alldatasheet.com/datasheet-pdf/pdf/88208/HP/HEDS5540.html [May, 2010]

[24] National Instruments "Datasheet: PCI-6221 (37-Pin)", 2004. [Online]. http://sine.ni.com/nips/cds/view/p/lang/en/nid/202004 [May, 2010]

[25] Department of Mechanical Engineering "DC Motor Closed-Loop Speed Control, MEEN 667 Laboratory Manual", Texas A&M University, TX, 2008.

[26] A. Ambike, “Closed Loop Real-Time Control on Distributed Networks,” M.S. Thesis, Texas A&M University, 2004.

[27] L. Dozio, and P. Mantegazza, "Real Time Distributed Control Systems Using RTAI," *in Proceedings of 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Hokkaido, Japan, pp. 11−18, 2003.

[28] Digital Equipment Corporation, Intel, Xerox, “The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications (Version 1.0)”, 1980.

[29] M. Lee, “Real-Time Networked Control with Multiple Clients,” M.S. Thesis, Texas A&M University, 2009.

[30] D. R. Boggs, J. C. Mogul, and C. A. Kent, “Measured Capacity of an Ethernet: Myths and Reality,” *ACM SIGCOMM Computer Communication Review,* vol. 18, no. 4, pp. 222−234, August, 1988.

[31] E. Jasperneite, and P. Neumann, "Switched Ethernet for Factory Communication," *in Proceedings of 8th IEEE International Conference on Emerging Technologies and Factory Automation*, Antibes – Juan les Pins, France, pp. 205−212 vol.1, October, 2001.

[32] O. Feuser, and A. Wenzel, "On the Effects of the IEEE 802.3x Flow Control in Full-Duplex Ethernet LANs," *in Proceedings of 24th Annual IEEE Conference on Local Computer Networks*, Tampa, FL, pp. 160, October, 1999.

# APPENDIX A

## CLIENT.C

```
// client.c

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>
#include <signal.h>
//#include <comedilib.h>
#include "/home/comedilib-0.8.0/comedilib/include/comedilib.h"
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/ioctl.h>
#include "defines.h"

#include <sched.h>
#include <errno.h>

#define KEEP_STATIC_INLINE

#include <rtai_sem.h>
#include <rtai_usi.h>
#include <rtai_lxrt.h>
//#include <rtai_fifos.h>
#include <time.h>
//#include <rtai_types.h>

#define PERIOD 1000000
#define LOOPS 1000
#define NTASKS 2
#define taskname(x) (1000 + (x))
//Specifies the Number of Samples -  509
#define MAXLOOP 5000

RTIME time_stamp;
double u0;
double u1e;
double u2e;
double u3e;
double u4e;
double u5e;
double u6e;
double u7e;
double u8e;
```

```
double y_0;
double y_1;
double y_2;
double y_3;
double y_4;
double y_5;
double y_6;
//Added by  on 2nd
//to check for rejection of clients
double y_7=3;//This is DC Motor clientat 20rps
double u_1;
double u_2;
// Mar7_2010 for samplingperiod
float samplingperiod=3000000;

RTIME current_time_stamp;

// new
int j = 0;
int m = 0;
int b0 = 0;
int b1 = 0;
int cnt = 0;
double speed = 0;
int p = 0;
// new

pthread_t task[NTASKS];
int ntasks = NTASKS;
RT_TASK *mytask;
SEM *sem; //added static
static int cpus_allowed;

SEM *sock_sem; //socket semaphoere, used by all the threads.
int sockid;
RTIME start_instant;
int server_sock_size = 0;
struct sockaddr_in my_addr, server_addr;

comedi_t *it;
int in_subdev = 2;      //digital input
int out_subdev = 1;     //analog output
int in_chan = 0;
int out_chan = 0;
int in_range = 0;
int out_range = 0;
int aref = AREF_GROUND;
int i=0;

//comedi declarations
lsampl_t in_data;
lsampl_t out_data;
float volts = 0.0;
int in_maxdata = 0, out_maxdata = 0;
```

```
comedi_range *in_range_ptr, *out_range_ptr;
int endme_int = 0;
//Added by Nav  30 - global variable
int reject_flag=0;//is set when zero control signal is sent
int server_cnt=0;//increases when a reject happens


void terminate_normally(int signo);
void endme(int sig)
{
      printf("You want to kill me?\n");
      endme_int = 1;
      rt_sem_delete(sem);
      comedi_close(it);
      stop_rt_timer();
      rt_task_delete(mytask);
      signal(SIGINT, SIG_DFL);
      exit(1);
}




void *send_thread_fun(void *arg)
{
      RTIME start_time, period, end_time, difference;
      RTIME t0;
      SEM *sem;
      RT_TASK *mytask;
      unsigned long mytask_name;
      int mytask_indx;
      double *buffer = NULL;
      int iRet = 0;
      struct recv_data *send_msg = NULL;
      int send_msg_size;
      FILE *fp = NULL; // - 509
      float print_data[MAXLOOP];
      int loop_count = 0;

      fp = fopen("res3ms20_10v_may1810_m5mcms14kp_.txt","w");
      if(fp == NULL)
      {
            printf("could not open file");
            exit(0);
      }

        pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
      mytask_indx = 0;
      mytask_name = taskname(mytask_indx);
      cpus_allowed = 1 - cpus_allowed;
      if (!(mytask = rt_task_init_schmod(mytask_name, 1, 0, 0,
SCHED_FIFO, 1 << cpus_allowed))) {
            printf("CANNOT INIT send_thread TASK\n");
            exit(1);
```

```
	}
	//printf("THREAD INIT: index = %d, name = %lu, address = %p.\n", 
mytask_indx, mytask_name, mytask);
	printf("send thread pid = %d\t master pid = %d\n", getpid(), 
getppid());
	mlockall(MCL_CURRENT | MCL_FUTURE);

	//rt_make_hard_real_time();

	rt_receive(0, (unsigned int*)&sem);

	send_msg_size = sizeof(struct recv_data);
	if(( send_msg = (struct recv_data *)calloc(1, sizeof(struct 
recv_data))) == NULL)
	{
		printf("cannot allocate message memory\n");
		exit(4);
	}

	period = nano2count(PERIOD);
	start_time = rt_get_time() + nano2count(10000000);
	t0 = start_instant;
	printf("send: t0 = %lld\t", t0);
	printf("This period = %lld\t", rt_get_time());
	printf("actual start = %lld\n", t0 + nano2count(500000000));
	rt_task_make_periodic(mytask, (t0 + nano2count(500000000)), 
nano2count(samplingperiod));
	// 3ms sampling time
	//rt_task_make_periodic(mytask, rt_get_time(), 
nano2count(samplingperiod));
	//start_time = rt_get_cpu_time_ns();
	printf("starting the send_thread while loop\n");


// DC motor speed measure
	while(1)
	//for(;;)
	{
		if(endme_int == 1)
		{
			break;
		}

	// Counting encoder pulses
	start_time = rt_get_cpu_time_ns();
	comedi_dio_config(it, in_subdev, in_chan, COMEDI_INPUT);

		//* Encoder Pulse Counting
		for(j=0;j<500;j++)	//* for loop
		    {
			m = comedi_data_read(it, in_subdev, in_chan, 
in_range, aref, &in_data);

			//fprintf(fp,"in_data: %d\n", m);
```

```
                  if(in_data == 1)  //* high or low?
                  {b1 = 1;}
                  else
                  [30]
                  if(b1 != b0)      //* count turn-over (H to L or L to
H)
                  {cnt++;}

                  b0 = b1;
            //    printf("cnt#: %d\n", cnt);
                } // for loop

      end_time = rt_get_cpu_time_ns(); //* End of the FOR loop



      current_time_stamp = rt_get_cpu_time_ns();

            //y_0 = comedi_to_phys(in_data, in_range_ptr, in_maxdata);

            //printf("cnt#: %d\n", cnt);
            //new
            //speed = (cnt*0.234)+1.026; //* AI counting

            speed = (cnt*0.214)+1.322; //* DI counting w/ j=1000, PCI-
6025E
            //speed = (cnt*0.070)+0.504; //* DI w/ j=500, PCI-6221
            y_0 = speed;
            //new

            //fprintf(fp,"y_0: %f\n ", y_0);

            print_data[loop_count] = y_0;
            send_msg->y_0 = y_0;
            send_msg->y_1 = y_1;
            send_msg->y_2 = y_2;
            send_msg->y_3 = y_3;
            send_msg->y_4 = y_4;
            send_msg->y_5 = y_5;
            send_msg->y_6 = y_6;
            send_msg->y_7 = y_7;
            send_msg->u_1 = u_1;
            send_msg->u_2 = u_2;
            send_msg->time_stamp = current_time_stamp;

            rt_sem_wait(sock_sem);
            iRet = sendto(sockid, (const void *)send_msg,
send_msg_size, 0, (struct sockaddr*)&server_addr, server_sock_size);
            rt_sem_signal(sock_sem);
            if(iRet <= -1)
            {
                  perror("sendto() failed\n");
                  break;
```

```
            }

            //y_7 = y_6;
            y_6 = y_5;
            y_5 = y_4;
            y_4 = y_3;
            y_3 = y_2;
            y_2 = y_1;
            y_1 = y_0;

            //new
            cnt = 0;
            //new

            if(loop_count < 5)
            {
                  printf("s: %d\n", loop_count);
            }

                if(loop_count > MAXLOOP - 5)
                {
                        printf("s: %d\n", loop_count);
                }

                loop_count++;

            if(loop_count == MAXLOOP)
            {
                  break;
            }
            //Added by Nav  30
            if(reject_flag==1)
            {
                  //printf("Breaking from Send Function");

                  break;
            }
            //printf("volts=%f\t time_stamp=%lld\n", volts,
current_time_stamp);
            rt_task_wait_period();
      }     //END of FOR (WHILE) LOOP

      end_time = rt_get_cpu_time_ns();
      difference = end_time - start_time;
      printf("difference = %lld\n", difference);
      endme_int++;
      //printf("send() endme_int++ line300");
      rt_sem_signal(sem);
      rt_make_soft_real_time();
      for(i=0;i<MAXLOOP;i++)
      {
            fprintf(fp, "%f\n", print_data[i]);
      }
      fclose(fp);
```

```
	free(send_msg);
	rt_task_delete(mytask);
	//printf("Reject flag==%i",reject_flag);
	printf("send_thread ENDS\n");
	return 0;
}	// End of send_thread


void *recv_thread_fun(void *arg)
{
	RTIME start_time, period, end_time, difference, delay;
	RTIME t0;
	SEM *sem;
	RT_TASK *mytask;
	unsigned long mytask_name;
	int mytask_indx;
	struct data *buffer = NULL;
	int iRet = 0;
	int recv_msg_size;
	struct send_data *recv_msg = NULL;
	int loop_count = 0;
	char temp;
	//added  509
	FILE *fp1 = NULL;
	//FILE *fp2 = NULL;
	fp1 = fopen("pktime3ms20_10v_may1810_m5mcms14kp_.txt","w");
	//fp2 = fopen("cputime.txt","w");
	RTIME packettimestamp[MAXLOOP], cputime[MAXLOOP];

	recv_msg_size = sizeof(struct send_data);
	if(( recv_msg = (struct send_data *)calloc(1, sizeof(struct 
send_data))) == NULL)
	{
		printf("cannot allocate message memory\n");
		exit(4);
	}

	  pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
	mytask_indx = 1;
	mytask_name = taskname(mytask_indx);
	cpus_allowed = 1 - cpus_allowed;
	if (!(mytask = rt_task_init_schmod(mytask_name, 1, 0, 0, 
SCHED_FIFO, 1 << cpus_allowed))) {
		printf("CANNOT INIT recv_thread TASK\n");
		exit(1);
	}
	//printf("THREAD INIT: index = %d, name = %lu, address = %p.\n", 
mytask_indx, mytask_name, mytask);
	printf("recv thread pid = %d\t master pid = %d\n", getpid(), 
getppid());
	mlockall(MCL_CURRENT | MCL_FUTURE);

	//rt_make_hard_real_time();
```

```
	rt_receive(0, (unsigned int*)&sem);

	period = nano2count(PERIOD);
	start_time = rt_get_time() + nano2count(10000000);
	t0 = start_instant;
	printf("recv: t0 = %lld\t", count2nano(t0));
	printf("This period = %lld\t", count2nano(rt_get_time()));
	printf("actual start = %lld\n", count2nano(t0 +
nano2count(500500000)));
	//rt_task_make_periodic(mytask, (RTIME)*((RTIME *)arg) +
nano2count(1001200000), nano2count(samplingperiod));
	rt_task_make_periodic(mytask, (t0 + nano2count(500500000)),
nano2count(samplingperiod));

	start_time = rt_get_time();

	printf("starting the recv_thread while loop\n");
	for(;;)
	{
		if(endme_int == 1)
		{
			//printf("recv() endme_int=1 line377");

			break;
		}

		rt_sem_wait(sock_sem);
		iRet = recvfrom(sockid, (void *)recv_msg, recv_msg_size, 0,
(struct sockaddr *)&server_addr, &server_sock_size);
		rt_sem_signal(sock_sem);

		if(iRet <= -1)
		{
			endme_int = 1;
			perror("recvfrom() failed\n");
			break;
		}


		if(loop_count < MAXLOOP)
		{
		u0 = recv_msg->u0;
		u1e = recv_msg->u1e;
		u2e = recv_msg->u2e;
		u3e = recv_msg->u3e;
		u4e = recv_msg->u4e;
		u5e = recv_msg->u5e;
		u6e = recv_msg->u6e;
		u7e = recv_msg->u7e;
		u8e = recv_msg->u8e;
		time_stamp=recv_msg ->time_stamp;
		packettimestamp[loop_count]=time_stamp;
		cputime[loop_count]=rt_get_cpu_time_ns();
```

```
                //printf("%lld\t", packettimestamp[loop_count]);
                //printf("%lld\n", cputime[loop_count]);
                }
//              delay=rt_get_cpu_time_ns() - time_stamp;
                //printf("after recv_msg 409");
                if(loop_count < MAXLOOP-2)
                {
                      volts = u0*1.014-0.005;
                      //volts=4.99;
                      //printf(" %f\n", volts);
                }
                /*else if(loop_count == MAXLOOP-2) //every 2 seconds data
missing
                {
                      volts = 0.0;
                //    printf(" speed 0 @ %d\n", loop_count);
                }*/

                if(loop_count < 5)
                {
                      printf("r: %d\n", loop_count);
                }

                    if(loop_count > MAXLOOP-5)
                    {
                            printf("r: %d\n", loop_count);
                      //Added to send 0 voltage at the end/to cancel the
residual voltage - 29,09
                      volts = 0.0;
                    }


                if(volts > 5.0)   // new: voltage limit change 10 -> 5
                {
                      volts = 4.99999;
                }
                if(volts < 0.0)
                {
                      volts = 0.0;
                }
                //Added by   2nd
                //To close the recv thread and call main() again
                //printf("before forloop u0=0 448");
                if(u0==0)
                {
                      printf("Server sent a NULL control signal\n");
                      //scanf("%c",&temp); //just to notice the printf msg
                      reject_flag=1;
                      break;
                }
                //printf("before out_data 456");
```

```
            out_data = comedi_from_phys(volts, out_range_ptr,
out_maxdata);

            comedi_data_write(it, out_subdev, out_chan, out_range,
aref, out_data);

            //u_2 = u_1;
            //u_1 = u0;

            if(loop_count == MAXLOOP-1)
            {
                  //printf("recv() loop_count == MAXLOOP 467");

                  break;

            }

            loop_count++;

            //printf("before rt_task_wait_period 472");
            rt_task_wait_period();
            //printf("AFTER rt_task_wait_period 474");
      }
      //printf("before forloop 473");
      //printf("going in print for loop\n");
      for(i=0;i<MAXLOOP;i++)
      {
            //printf("in pkttime loop\n");
            if(reject_flag==1) { break;}
            fprintf(fp1,"%lld\t\n",cputime[i]-packettimestamp[i]);
            //fprintf(fp2, "%lld\n", cputime[i]);
            //printf("just out\n");
      }

      fclose(fp1);
      //fclose(fp2);
      //end_time = rt_get_cpu_time_ns();
      //difference = end_time - start_time;
      //printf("difference = %lld\n", difference);
      endme_int++;

      rt_make_soft_real_time();

      free(recv_msg);
      rt_task_delete(mytask);
      printf("recv_thread ENDS\n");
      if(reject_flag==1)
      {
            printf("Server Rejected the request\n Please connect to a
different client\n");
            //main();
      }
      return 0;
}
```

```
int main(void)
{
      int i;
      unsigned long mytask_name = nam2num("MASTER");
      struct sigaction sa;
      int select_int;
      //Added by Nav  30
      char serverip[20][20];
      strcpy(serverip[0],"165.91.95.116");
      strcpy(serverip[1],"165.91.95.122");

   /*************************Selecting the 
server********************************
   char server_str[20];
   int select_int;
   printf("Please select the server \n""****************Server 
Menu**************\n""1 - Maglev Server(Maglev 27) \n""2 - DC Motor 
Server(Maglev 21)\n""Please enter your choice in number :");
   scanf("%d",&select_int);
   if (select_int==1)
      strcpy(server_str,"165.91.95.122");
   else
      strcpy(server_str,"165.91.95.116");

***********************************************************************
**********/


//char * server_ip = "165.91.95.120"; //maglev1
//char * server_ip=server_str; //added  May 22 2008
//unsigned short my_port, server_port;

   char * server_ip=serverip[server_cnt]; //added by Nav on  30,08
   printf("Trying to connect to server ip: %s\nEnter a character to 
continue:",serverip[server_cnt]);
   //scanf("%d",&select_int);//just a dummy var

   unsigned short my_port, server_port;
      my_port = 4445;
   server_port = 4444;

   /* -- Create client side socket -- */
   printf("creating socket\n");
   if( (sockid = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
   {
     perror("socket() failed ");
     exit(2);
   }

   /* -- Initialize client side socket address -- */
   memset((void *) &my_addr, (char) 0, sizeof(my_addr));
```

```
   my_addr.sin_family = AF_INET;                     /* Internet
Address Family */
   my_addr.sin_addr.s_addr = htonl(INADDR_ANY);  /* I can receive from
any host */
   my_addr.sin_port = htons(my_port);
   if ( (bind(sockid, (struct sockaddr *) &my_addr, sizeof(my_addr)) <
0) )
   {
     perror("bind() failed ");
     exit(3);
   }

   /* -- Initialize server side socket address -- */
   server_sock_size = sizeof(server_addr);
   memset((void *) &server_addr, (char) 0, server_sock_size);
   server_addr.sin_family = AF_INET;
   server_addr.sin_addr.s_addr = inet_addr(server_ip);
   server_addr.sin_port = htons(server_port);

      sa.sa_handler = endme;
      sa.sa_flags = 0;
      sigemptyset(&sa.sa_mask);

      if(sigaction(SIGINT, &sa, NULL))
      {
            perror("sigaction");
      }
      if(sigaction(SIGTERM, &sa, NULL))
      {
            perror("sigaction");
      }

      it = comedi_open("/dev/comedi0");
      if(it == NULL)
      {
            printf("Could not open comedi\n");
            exit(1);
      }

      in_maxdata = comedi_get_maxdata(it, in_subdev, in_chan);
      out_maxdata = comedi_get_maxdata(it, out_subdev, out_chan);

      in_range_ptr = comedi_get_range(it, in_subdev, in_chan,
in_range);
      out_range_ptr = comedi_get_range(it, out_subdev, out_chan,
out_range);

      if (!(mytask = rt_task_init(mytask_name, 1, 0, 0))) {
            printf("CANNOT INIT main TASK \n");
            exit(1);
      }
      printf("MASTER INIT: name = %lu, address = %p.\n", mytask_name,
mytask);
```

```
      sem = rt_sem_init(10000, 0);
      sock_sem = rt_sem_init(nam2num("SOCK"), 1);
      //rt_set_oneshot_mode();
      rt_set_periodic_mode();
      start_rt_timer(nano2count(25000));

      start_instant = rt_get_time();
      printf("main: start_instant = %lld\n", start_instant);

      if (pthread_create(&task[0], NULL, send_thread_fun,
&start_instant))
      {
            printf("ERROR IN CREATING send_thread\n");
            exit(1);
      }

      //printf("After send thread before recv thread \n");

      if (pthread_create(&task[1], NULL, recv_thread_fun,
&start_instant))
      {
            printf("ERROR IN CREATING recv_thread\n");
            exit(1);
      }

      //printf("After recv thread creating???\n");
      for (i = 0; i < ntasks; i++)
      {
      //    printf("1111\n");

            while (!rt_get_adr(taskname(i)))
            {
                  rt_sleep(nano2count(20000000));
      //          printf("2222\n");
            }
      }

      //printf("ntasks 0,1 or 0 only \n");

      for (i = 0; i < ntasks; i++)
      {
            rt_send(rt_get_adr(taskname(i)), (unsigned int)sem);
      }

      printf("Start waiting for sem\n");
      while(endme_int == 0)
      {
            rt_sem_wait_timed(sem, nano2count(50000000));   //
5,000,000,000
      }
      printf("Stop waiting for sem\n");

      for (i = 0; i < ntasks; i++)
      {
```

```
		while (rt_get_adr(taskname(i)))
		{
			rt_sleep(nano2count(20000000));
		}
	}

	rt_sem_delete(sem);
	rt_sem_delete(sock_sem);
	stop_rt_timer();
	comedi_close(it);
	rt_task_delete(mytask);
	printf("MASTER %lu %p ENDS\n", mytask_name, mytask);
	for (i = 0; i < ntasks; i++) {
		pthread_join(task[i], NULL);
	}
	//printf("Entering reject_flag loop\n");
	if (reject_flag==1)
		{
			//printf("Inside Main()....reject flag is set\n");
			//printf("Resetting the reject_flag\n\n");
			reject_flag=0;
			//calling the main function again
			close(sockid);
			//Added by Nav on  30
			server_cnt++;//To connect to the next server
			if (server_cnt<2)
			{
				main();
			}
			else
			{
				printf("All available servers rejected..Try
again!!\n");
			}
		}
	return 0;
}
```

# APPENDIX B

## SERVER.C

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <asm/errno.h>
#include <sys/types.h>
#include <sys/user.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sched.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <errno.h>
#include <inttypes.h>
#include "defines.h"

#define KEEP_STATIC_INLINE
//#include <rtai_lxrt_user.h>
#include <rtai_lxrt.h>

RTIME time_stamp;
double u0;
double u1e;
double u2e;
double u3e;
double u4e;
double u5e;
double u6e;
double u7e;
double u8e;
double y_0;
double y_1;
double y_2;
double y_3;
double y_4;
double y_5;
double y_6;
double y_7;
double u_1;
double u_2;
double y_hat_1;
```

```
double y_hat_2;
double y_hat_3;
double y_hat_4;
double y_hat_5;
double y_hat_6;
double y_hat_7;
double y_hat_8;

// DC-motor (m5)
double y_dot_desi = 10.0;
double e0 = 0;    //* instead of long
double e1 = 0;    //* instead of long
double u0 = 0;    //* value0, instead of long
double u1 = 0;    //* value1, instead of long

// DC-motor (m23)
double y_dot_desi_b = 15.0;
double e0_b = 0;  //* instead of long
double e1_b = 0;  //* instead of long
double u0_b = 0;  //* value0, instead of long
double u1_b = 0;  //* value1, instead of long

// A Ball Maglev
double y_hat_desi = 0.005;    //* User input (desired set point)
double v_hat_err = 0.0;
//double k = 0.053;
double k = 0.083; //
//double k = 0.098-0.025; //* small k small vib.      current setting
(Gain parameter)
double c = 0.0;              //* Controller constant
//double v = 0.7;
//double v = 1.018;          //* For initial offset
double v = 0.975; //current setting
double er0 = 0.0;                     //* Input to controller at time
n
double er1 = 0.0;                      //* Intput to controller at
time n-1
double er2 = 0.0;                      //* Intput to controller at
time n-2
int i=0;


//----new for display
int p1 = 0;
int p2 = 0;
int p3 = 0;

//rtai declarations
unsigned long mtsk_name;
RT_TASK *mtsk;
struct sched_param mysched;

void terminate_normally(int signo)
{
```

```
        fflush(stdin);

        if(signo==SIGINT || signo==SIGTERM)
        {
              printf("Terminating the program normally\n");

              //make the process soft real time process
              rt_make_soft_real_time();

              printf("MASTER TASK YIELDS ITSELF\n");
              rt_task_yield();

              printf("MASTER TASK STOPS THE PERIODIC TIMER\n");
              stop_rt_timer();

              printf("MASTER TASK DELETES ITSELF\n");
              rt_task_delete(mtsk);

              printf("END MASTER TASK\n");
        }

        exit(0);
}


main(int argc, char *argv[])
{
   int sockid, nread, addrlen;
   char their_address;
   struct sockaddr_in my_addr, client_addr;
   int nw, nr;
   int send_buffer_size, recv_buffer_size;
   unsigned short server_port = 0;
   struct send_data *send_buffer = NULL;
   struct recv_data *recv_buffer = NULL;
   char temp;


        RTIME start_time = 0;
        RTIME end_time = 0;
        RTIME actual_period = 0;
        RTIME difference = 0;
        size_t iRet = 0;
        int esti_count = 0;
        double vhaterr_prev[5] = {0.0, 0.0, 0.0, 0.0, 0.0};
        int j=0;

        //signal handling
        struct sigaction sa;

        //Initialize the signal handling structure
        sa.sa_handler = terminate_normally;
        sa.sa_flags = 0;
        sigemptyset(&sa.sa_mask);
```

```
      if(sigaction(SIGINT, &sa, NULL))
      {
            perror("sigaction");
      }
      if(sigaction(SIGTERM, &sa, NULL))
      {
            perror("sigaction");
      }

   fprintf(stderr, "creating socket\n");
   if ( (sockid = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
   {
     perror("socket() failed ");
     fprintf(stderr, "%s: socket error: %d\n", argv[0], errno);
     exit(2);
   }

   fprintf(stderr, "binding my local socket\n");
   //sscanf(argv[1], "%hu", &server_port);
   server_port = 4444;

   memset((void *) &my_addr, (char) 0, sizeof(my_addr));
   my_addr.sin_family = AF_INET;
   my_addr.sin_addr.s_addr = htons(INADDR_ANY);
   my_addr.sin_port = htons(server_port);

   if ( (bind(sockid, (struct sockaddr *) &my_addr,
      sizeof(my_addr)) < 0) )
   {
     perror("bind() failed ");
     fprintf(stderr, "bind() errno = %d\n", errno);
     exit(4);
   }

   recv_buffer_size = sizeof(struct recv_data);
   if(( recv_buffer = (struct recv_data *)calloc(1, sizeof(struct
recv_data))) ==NULL)
   {
        fprintf(stderr, "cannot allocate memory for buffer!\n");
      exit(4);
   }

   send_buffer_size = sizeof(struct send_data);
   if(( send_buffer = (struct send_data *)calloc(1, sizeof(struct
send_data))) ==NULL)
   {
        fprintf(stderr, "cannot allocate memory for buffer!\n");
      exit(4);
   }

   addrlen = sizeof(client_addr);

   fprintf(stderr, "%s: starting blocking message read\n", argv[0]);
```

```
      mysched.sched_priority = 99;

      if( sched_setscheduler( 0, SCHED_FIFO, &mysched ) == -1 ) {
      puts(" ERROR IN SETTING THE SCHEDULER UP");
      perror( "errno" );
      exit( 0 );
      }

      mlockall(MCL_CURRENT | MCL_FUTURE);

      mtsk_name = nam2num("MTSK");
      if (!(mtsk = rt_task_init(mtsk_name, 0, 0, 0))) {
            printf("CANNOT INIT MASTER TASK\n");
            exit(1);
      }

      start_time = rt_get_cpu_time_ns();
      printf("main: start_time = %lld\n", start_time);

      printf("MASTER TASK STARTS THE ONESHOT TIMER\n");
      //rt_set_oneshot_mode();
      actual_period = start_rt_timer(nano2count(25000));

      printf("actual_period = %lld\n", actual_period);

      //make the process a hard real time process
      //rt_make_hard_real_time();

      printf("MASTER TASK MAKES ITSELF PERIODIC \n");
      rt_task_make_periodic(mtsk, rt_get_time()+ nano2count(3000000),
nano2count(3000000));


      FILE *fp;
      fp = fopen("result.txt", "w");

      if(fp == NULL)
        {
          printf("could not open file.\n");
        }


   while( 1 )
   {

      nr = recvfrom(sockid, (void *)recv_buffer, recv_buffer_size, 0,
(struct sockaddr *) &client_addr, &addrlen);
      if( nr <= -1 )
      {
            fprintf(stderr, "recvfrom() errno = %d\n", errno);
            exit(10);
      }
```

```
        printf("the client address is:
%s\n",inet_ntoa(client_addr.sin_addr));
        their_address=inet_ntoa(client_addr.sin_addr);


      start_time = rt_get_cpu_time_ns();

      y_0 = recv_buffer->y_0;
      y_1 = recv_buffer->y_1;
      y_2 = recv_buffer->y_2;
      y_3 = recv_buffer->y_3;
      y_4 = recv_buffer->y_4;
      y_5 = recv_buffer->y_5;
      y_6 = recv_buffer->y_6;
      y_7 = recv_buffer->y_7;
      u_1 = recv_buffer->u_1;
      u_2 = recv_buffer->u_2;

      if(y_7 == 1)
      {

      er0 = (y_hat_desi - (-0.0010108*y_0+0.0114970))*k;  /* Error
Calculation */
      er1 = (y_hat_desi - (-0.0010108*y_1+0.0114970))*k;
      er2 = (y_hat_desi - (-0.0010108*y_2+0.0114970))*k;
      u0 = ((0.782*(u_1-v)) + (0.13*(u_2-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
      send_buffer->u0 = u0;

      y_hat_1 = 0.8122*y_0 - 0.3479*y_1 - 0.0294*y_2 + 0.4605*y_3 +
0.0742*y_4 + 0.1042*y_5 + 0.1117*y_6;// - 0.3561*y_7;
      er0 = (y_hat_desi - (-0.0010108*y_hat_1+0.0114970))*k;  /* Error
Calculation */
      er1 = (y_hat_desi - (-0.0010108*y_0+0.0114970))*k;
      er2 = (y_hat_desi - (-0.0010108*y_1+0.0114970))*k;
      u1e = ((0.782*(u0-v)) + (0.13*(u_1-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
      send_buffer->u1e = u1e;

      y_hat_2 = 0.3117*y_0 - 0.3119*y_1 + 0.4366*y_2 + 0.4482*y_3 +
0.1645*y_4 + 0.1964*y_5 - 0.2653*y_6;// - 0.2892*y_7;
      er0 = (y_hat_desi - (-0.0010108*y_hat_2+0.0114970))*k;  /* Error
Calculation */
      er1 = (y_hat_desi - (-0.0010108*y_hat_1+0.0114970))*k;
      er2 = (y_hat_desi - (-0.0010108*y_0+0.0114970))*k;
      u2e = ((0.782*(u1e-v)) + (0.13*(u0-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
      send_buffer->u2e = u2e;

      y_hat_3 = -0.0587*y_0 + 0.3281*y_1 + 0.4390*y_2 + 0.3080*y_3 +
0.2195*y_4 - 0.2329*y_5 - 0.2544*y_6;// - 0.1110*y_7;
      er0 = (y_hat_desi - (-0.0010108*y_hat_3+0.0114970))*k;  /* Error
Calculation */
      er1 = (y_hat_desi - (-0.0010108*y_hat_2+0.0114970))*k;
```

```
      er2 = (y_hat_desi - (-0.0010108*y_hat_1+0.0114970))*k;
      u3e = ((0.782*(u2e-v)) + (0.13*(u1e-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
      send_buffer->u3e = u3e;

      y_hat_4 = 0.2804*y_0 + 0.4594*y_1 + 0.3097*y_2 + 0.1925*y_3 -
0.2372*y_4 - 0.2605*y_5 - 0.1176*y_6;// + 0.0209*y_7;
      er0 = (y_hat_desi - (-0.0010108*y_hat_4+0.0114970))*k;  /* Error
Calculation */
      er1 = (y_hat_desi - (-0.0010108*y_hat_3+0.0114970))*k;
      er2 = (y_hat_desi - (-0.0010108*y_hat_2+0.0114970))*k;
      u4e = ((0.782*(u3e-v)) + (0.13*(u2e-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
      send_buffer->u4e = u4e;

      y_hat_5 = 0.6872*y_0 + 0.2122*y_1 + 0.1842*y_2 - 0.1081*y_3 -
0.2397*y_4 - 0.0884*y_5 + 0.0523*y_6;// - 0.0999*y_7;
      er0 = (y_hat_desi - (-0.0010108*y_hat_5+0.0114970))*k;  /* Error
Calculation */
      er1 = (y_hat_desi - (-0.0010108*y_hat_4+0.0114970))*k;
      er2 = (y_hat_desi - (-0.0010108*y_hat_3+0.0114970))*k;
      u5e = ((0.782*(u4e-v)) + (0.13*(u3e-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
      send_buffer->u5e = u5e;

      y_hat_6 = 0.7703*y_0 - 0.0548*y_1 - 0.1283*y_2 + 0.0767*y_3 -
0.0374*y_4 + 0.1239*y_5 - 0.0231*y_6;// - 0.2447*y_7;
      er0 = (y_hat_desi - (-0.0010108*y_hat_6+0.0114970))*k;  /* Error
Calculation */
      er1 = (y_hat_desi - (-0.0010108*y_hat_5+0.0114970))*k;
      er2 = (y_hat_desi - (-0.0010108*y_hat_4+0.0114970))*k;
      u6e = ((0.782*(u5e-v)) + (0.13*(u4e-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
      send_buffer->u6e = u6e;

      y_hat_7 = 0.5708*y_0 - 0.3963*y_1 + 0.0541*y_2 + 0.3173*y_3 +
0.1810*y_4 + 0.0572*y_5 - 0.1586*y_6;// - 0.2743*y_7;
      er0 = (y_hat_desi - (-0.0010108*y_hat_7+0.0114970))*k;  /* Error
Calculation */
      er1 = (y_hat_desi - (-0.0010108*y_hat_6+0.0114970))*k;
      er2 = (y_hat_desi - (-0.0010108*y_hat_5+0.0114970))*k;
      u7e = ((0.782*(u6e-v)) + (0.13*(u5e-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
      send_buffer->u7e = u7e;

      send_buffer->u8e = 1;

      p1 = p1+1;
      printf("_m02_");
      fprintf(fp,"m02 %d\n", p1);
      //printf("m02 %d\n",p1);
      } //if client 1

      else if(y_7 == 2) //if client 2 (maglev5 DC motor)
```

```
{

//new
e1 = y_dot_desi - y_0;  /* Error Calculation */
//u1 = u0 + 2.0225*e1 - 0.9775*e0; //* 209ms sampling period
u0 = u1 + 1.5075*e1 - 1.4925*e0;   //* 3ms sampling period
//u0 = u1 + 1.5225*e1 - 1.4775*e0; //* 9ms sampling period
//u0 = u1 + 1.5250*e1 - 1.4750*e0; //* 10ms sampling period
printf("speed %f\n", y_0);
fprintf(fp,"speed %f\n ", y_0);
//new


send_buffer->u0 = u0;
u1e = 0;
send_buffer->u1e = u1e;
u2e = 0;
send_buffer->u2e = u2e;
u3e = 0;
send_buffer->u3e = u3e;
u4e = 0;
send_buffer->u4e = u4e;
u5e = 0;
send_buffer->u5e = u5e;
u6e = 0;
send_buffer->u6e = u6e;
u7e = 0;
send_buffer->u7e = u7e;
u8e = 2;
send_buffer->u8e = u8e;

p2 = p2+1;
//fgets(p2, sizeof line, fp);
printf("_m05 %d_\n", p2);
//fprintf(fp,"_m05_");
u1 = u0;
e0 = e1;
  }


else if(y_7 == 3) //if client 3 (maglev23 DC motor)
{

//new
e1_b = y_dot_desi_b - y_0;  /* Error Calculation */

//u1 = u0 + 2.0225*e1 - 0.9775*e0; //* 209ms sampling period

u0_b = u1_b + 1.5075*e1_b - 1.4925*e0_b;   //* 3ms sampling 
period



send_buffer->u0 = u0_b;
```

```
      u1e = 0;
      send_buffer->u1e = u1e;
      u2e = 0;
      send_buffer->u2e = u2e;
      u3e = 0;
      send_buffer->u3e = u3e;
      u4e = 0;
      send_buffer->u4e = u4e;
      u5e = 0;
      send_buffer->u5e = u5e;
      u6e = 0;
      send_buffer->u6e = u6e;
      u7e = 0;
      send_buffer->u7e = u7e;
      u8e = 3;
      send_buffer->u8e = u8e;

      p3 = p3+1;
      printf("_m23 %d_\n", p3);
      //fprintf(fp,"_m23_\n");
      u1_b = u0_b;
      e0_b = e1_b;
        }

//Naveen added this on  to test for rejection of MAGLEV clients
      else //When y_7 is not equal to 1,2 or 3
      {
      printf("Wrong Cient & Reject message sent to it");
      //scanf("%c",&temp);
      //new
      e1_b = y_dot_desi_b - y_0;  /* Error Calculation */

      //u1 = u0 + 2.0225*e1 - 0.9775*e0; //* 209ms sampling period

      u0_b = u1_b + 1.5075*e1_b - 1.4925*e0_b;   //* 3ms sampling
period

      send_buffer->u0 = 0; //Control signal is sent as "0"
      u1e = 0;
      send_buffer->u1e = u1e;
      u2e = 0;
      send_buffer->u2e = u2e;
      u3e = 0;
      send_buffer->u3e = u3e;
      u4e = 0;
      send_buffer->u4e = u4e;
      u5e = 0;
      send_buffer->u5e = u5e;
      u6e = 0;
      send_buffer->u6e = u6e;
      u7e = 0;
      send_buffer->u7e = u7e;
      u8e = 3;
      send_buffer->u8e = u8e;
```

```
      p3 = p3+1;
      printf("_m23 %d_\n", p3);
      u1_b = u0_b;
      e0_b = e1_b;
      }
      end_time = rt_get_cpu_time_ns();

     send_buffer->time_stamp = recv_buffer->time_stamp;

     nw = sendto(sockid, (const void *)send_buffer, send_buffer_size,
0, (struct sockaddr *)
             &client_addr, addrlen);
     if( nw <= -1 )
     {
       perror("sendto failed ");
       fprintf(stderr, "sendto() errno = %d \n", errno);
       exit(12);
     }

  } // END of while

   fclose(fp);

      //make the process soft real time process
      //rt_make_soft_real_time();

      printf("MASTER TASK YIELDS ITSELF\n");
      rt_task_yield();

      printf("MASTER TASK STOPS THE PERIODIC TIMER\n");
      stop_rt_timer();

      printf("MASTER TASK DELETES ITSELF\n");
      rt_task_delete(mtsk);



   close(sockid);
   free(send_buffer);
   free(recv_buffer);
 }
```

# VITA

Naveen Kumar Bibinagar received his Bachelor of Technology degree in mechanical engineering and information technology from Vellore Institute of Technology University, India in 2005. He worked as a Programmer Analyst at Cognizant Technology Solutions from 2005 to 2007. In 2007, he entered the graduate program in the Mechanical Engineering Department at Texas A&M University, College Station, and received his M.S. degree in 2010. He joined the Precision Mechatronics and Nanotechnology Laboratory in December 2007. His research interests include computer communication, networking and real-time control systems. He can be reached at Dept. of Mechanical Engineering, 3123 TAMU, College Station, TX 77840.