

© 2011 Sam Naghshinch

TELEOPERATION OF PASSIVITY-BASED MODEL REFERENCE ROBUST CONTROL OVER THE
INTERNET

BY

SAM NAGHSHINEH

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Systems and Entrepreneurial Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Adviser:

Associate Professor Dušan M. Stipanović

ABSTRACT

This dissertation offers a survey of a known theoretical approach and novel experimental results in establishing a live communication medium through the internet to host a virtual communication environment for use in Passivity-Based Model Reference Robust Control systems with delays. The controller which is used as a carrier to support a robust communication between input-to-state stability is designed as a control strategy that passively compensates for position errors that arise during contact tasks and strives to achieve delay-independent stability for controlling of aircrafts or other mobile objects. Furthermore the controller is used for nonlinear systems, coordination of multiple agents, bilateral teleoperation, and collision avoidance thus maintaining a communication link with an upper bound of constant delay is crucial for robustness and stability of the overall system. For utilizing such framework an elucidation can be formulated by preparing site survey for analyzing not only the geographical distances separating the nodes in which the teleoperation will occur but also the communication parameters that define the virtual topography that the data will travel through. This survey will first define the feasibility of the overall operation since the teleoperation will be used to sustain a delay based controller over the internet thus obtaining a hypothetical upper bound for the delay via site survey is crucial not only for the communication system but also the delay is required for the design of the *passivity-based model reference robust control*. Following delay calculation and measurement via site survey, bandwidth tests for unidirectional and bidirectional communication is inspected to ensure that the speed is viable to maintain a real-time connection. Furthermore from obtaining the results it becomes crucial to measure the consistency of the delay throughout a sampled period to guarantee that the upper bound is not breached at any point within the communication to jeopardize the robustness of the controller. Following delay analysis a geographical and topological overview of the communication is also briefly examined via a trace-route to understand the underlying nodes and their contribution to the delay and round-trip consistency. To accommodate the communication channel for the controller the input and output data from both nodes need to be encapsulated within a transmission control protocol via a multithreaded design of a robust program within the C language. The program will construct a multithreaded client-server relationship in which the control data is transmitted. For added stability and higher level of security the channel is then encapsulated via an internet protocol security by utilizing a protocol suite for protecting the communication by authentication and encrypting each packet of the session using negotiation of cryptographic keys during each session.

To my family for their limitless love and support.

ACKNOWLEDGEMENTS

Words can never adequately articulate my genuine benediction to the many individuals that have helped and contributed to the completion of my graduate study and my dissertation.

Foremost, I would like to express my sincere gratitude to my advisor Dr. Dušan M. Stipanović for the continuous support of my M.Sc. study and research, for his patience, motivation, enthusiasm, and immense knowledge. This thesis would not have been possible without his advocacy and support in believing in my capabilities. His guidance has substantially abetted me throughout my graduate study not only as an advisor but also as a great teacher.

I am also thankful to Dr. Erick J. Rodriguez for his help and support in writing this dissertation. Without his patience and help this dissertation would never have been made possible. He is without a doubt not only a great mentor but also undoubtedly a great friend.

I would also like to thank Dr. James J. Troy, Dr. Paul Murray and Dr. Charles A. Erignac from Boeing Research and Technology for their help and support in obtaining data and conducting experiments with their network facilities.

I am perpetually indebted to Dr. M. Moeinzadeh and family for their immense hospitality, kindness and moral support.

My profound gratitude goes to my colleagues and friends at the Coordinates Science Laboratory (CSL) and the department of Industrial Enterprise Systems Engineering for their friendship and support throughout my study. I would like to thank Chad for his welcoming hospitality which aided me to make Champaign feel at home during my first few days at the school. Moreover I would also like to thank my best friend Piyum Zonooz for providing me insightful empathy throughout my study period and encouraging me to daring heights. I would like to thank all my friends in Toronto for whom I revere to have after all the time we have spent apart.

Lastly, I would like to thank my family. I am immensely thankful for having such a wonderful, loving and encouraging sister who no matter the circumstances has always believed in me. I am greatly thankful to my parents, Hamid Naghshineh and Niloufar Saeedi. I thank them for sparking the creativity within me and always pushing me to my limits, expanding my engineering mind and teaching me to always ambitiously seek knowledge and to be the best at what I seek. Without their inspiration, love and support none of this would be made possible.

TABLE OF CONTENTS

List of figures.....	vii
List of abbreviations.....	viii
Chapter 1 Introduction.....	1
1.1 Control of time delay systems	2
1.2 Internet: an introduction	3
1.3 Protocols	3
1.4 Understanding network topology and distribution.....	5
1.5 Networked bilateral teleoperation.....	8
Chapter 2 Model reference robust control.....	9
2.1 Reference model	9
2.2 Scattering transformation.....	9
2.3 Stability analysis and state convergence	11
2.4 Design specifications.....	12
Chapter 3 Site survey.....	13
3.1 Geographical locations	13
3.2 Site survey server setup.....	14
3.3 Geolocational traceroute	16
3.4 Bandwidth test	18
3.5 Round-trip consistency.....	20
3.6 Trace-route	22
3.7 Results	22
Chapter 4 Controller applications.....	24
4.1 Bilateral teleoperation	24
4.2 Bilateral control devices.....	25
4.2.1 The haptic device	25
4.2.2 Coaxial helicopters.....	25
4.2.3 Mocap system.....	26
4.2.4 Communication.....	26
4.3 Network delay compensation using play-back buffer	27
Chapter 5 Network architecture	29
5.1 Understanding networking architecture.....	29

5.2	Transmission control protocol (TCP).....	30
5.2.1	Network function.....	30
5.2.2	TCP segment structure.....	30
5.2.3	Protocol operation.....	30
5.2.4	Connection establishment	31
5.2.5	Reliable transmission	31
5.2.6	Vulnerabilities	31
5.2.6.1	Denial of service	31
5.2.6.2	Connection hijacking	31
5.3	User datagram protocol.....	32
5.3.1	Reliability and congestion control	32
5.3.2	Vulnerabilities	32
5.4	Comparison of TCP and UDP.....	33
5.5	Simulation	34
Chapter 6	Pragmatic design.....	36
6.1	JAVA - UDP requester/sender program.....	36
6.2	C - UDP communication driver v1.0.....	36
6.3	C - TCP communication driver v2.0	38
6.4	C - TCP multi-threaded communication driver v2.1	39
6.5	Results	41
6.6	Robustness and security	42
Conclusion	44
References	45
Appendix A	– Java UDP communication driver code v1.0.....	47
Appendix B	– C TCP communication driver code V2.0	51
Appendix C	– C TCP multi-threaded communication driver code v2.1	53

LIST OF FIGURES

Figure 1.1 Networked control system with delays	2
Figure 1.2 Map of multicast internet in August 1996	5
Figure 1.3 Current internet topology	6
Figure 1.4 Teleoperation configuration	8
Figure 2.1 MRRC framework	10
Figure 3.1 Connection distance	13
Figure 3.2 Connection route via nodes from Champaign to Bellevue	16
Figure 3.3 iPerf Setup	18
Figure 3.4 Roundtrip consistency	20
Figure 3.5 Transfer speed vs delay	21
Figure 3.6 Trace-route table	22
Figure 4.1 Scheme of a bilateral teleoperation system	24
Figure 4.2 Master and slave agents	25
Figure 4.3 Rotational angles with respect to cartesian coordinates	25
Figure 4.4 A 3-D image generated by the MoCap system using Vicon iQ2.5 graphical display	26
Figure 4.5 Normalized PDFs for beta distribution	27
Figure 5.1 OSI model	29
Figure 5.2 CORE virtual simulation	34
Figure 5.3 Node configuration setup	35
Figure 6.1 Data loss due to UDP	37
Figure 6.2 Data comparison between transmitted and received data	38
Figure 6.3 Multi-threaded architecture	40

LIST OF ABBREVIATIONS

3DES	Triple Encryption Data Encryption Standard
AH	Authentication Header
ARIN	American Registry for Internet Numbers
BCP	Best Current Practices
CORE	Common Open Research Emulator
DCS	Distributed Control systems
DOS	Denial Of Service
ICANN	Internet Corporation for Assigned Names and Numbers
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPSEC	Internet Protocol Security
IPV4	Internet Protocol version 4
IPV6	Internet Protocol version 6
ISP	Internet Service Provider
MRRC	Model Reference Robust Control
MTU	Maximum Transmission Unit
NAT-T	Network Address Translation Traversal
NCS	Network Control systems
OSI	Open System Interconnection
OSPF	Open Shortest Path First
PID	Process Identifier
PPTP	Point-to-Point Tunneling Protocol
PSTN	Public Switched Telephone Network
QOS	Quality Of Service
RFC	Request for comments
RIP	Routing Information protocol
RIR	Regional Internet registry
RTT	Round Trip Time
RWIN	Receive Windows
SA	Security Association
SCTP	Stream Control Transmission Protocol
SDH	Synchronous Digit Hierarchy
SHA1	Secure Hash Algorithm
SONET	Synchronous Optical Networking
TCP	Transport Control Protocol
UAV	Unmanned Aerial Vehicle
VoIP	Voice over IP
VPN	Virtual Private Network
WAN	Wide Area Network
WWW	World Wide Web

CHAPTER 1 INTRODUCTION

Due to topical progressions within the field of controls and with increase in communication innovations there exists a rise in demand for providing a robust network control to bring stability to a system via Internet networks [1]. In the modern world of network controls measuring peripherals such as sensors, actuators, controllers, and processes (also known as agents within this context) are no longer circumscribed to be directly connected by physical means nor do they need to be within the vicinity of one another. A networked control system (NCS) or distributed control system (DCS) refers to a control system usually of a manufacturing system, process or any kind of dynamic system, in which the controller elements are not central in location (like the brain) but are distributed throughout the system with each component sub-system controlled by one or more controllers. The entire system of controllers is interconnected via various communication mediums such as local networks using wired Ethernet, wireless networks, and even wide area networks (WAN) in which the internet is consisted of. It is due to these advantages that DCSs are engaging within various applications of remote medical operations, underwater exploration, and military covert intelligence gathering mobile machines, and autonomous aerial and ground vehicles [2]. In fact one of the main applications of using control systems combined with teleoperation is for the control of unmanned aerial vehicles (UAV) which are autonomously controlled to carry out specific military tasks.

Granted DCS technology has considerably improved [3], one of the main challenges of guaranteeing their safe operations is to overcome communication challenges of maintaining a live connection and to ensure stability and robustness of the controller given a range of input tolerances [4]. One of the main factors that complicate the NCS technology in particular are time delays accumulated and developed due to large separation distances between agents, overhead within transmission protocols (discussed in Chapter 5 Network Architecture), congested communication networks and often times delay due to signal transmission between the controller and the actuator or plant. It is very important to analyze the delay within every step and find an integral time delay that represents the overall system and take it into consideration within the overall design, otherwise as a consequence the system will exhibit poor performance and in worse cases instability [5]. As a result in order to facilitate a control system especially one that is Passivity-Based Model Reference Robust Control over a networked connection, it is vital to first analyze the encompassing virtual topology for feasibility (discussed in Chapter 3 Site survey).

1.1 CONTROL OF TIME DELAY SYSTEMS

In order to understand the main control loop which facilitates the existence of delays within the controller a simple model of networked control system with delays will be visited as described in Figure 1.1 [6].

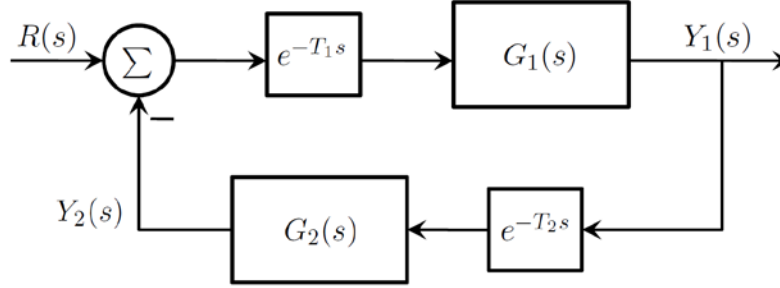


FIGURE 1.1 NETWORKED CONTROL SYSTEM WITH DELAYS

The primary objective in the control of most dynamic systems is to assume that the evolution of the states are not dependent on their previous state or simply information from their past. Although this assumption is suitable for many engineering processes, there exist a wide range of control systems for which the effects of delays cannot be ignored. As an example, let us consider the two interconnected linear systems depicted in Figure 1.1, where $G_i(s)$ and $T_i \geq 0$ represent the Laplace transfer functions and the associated delays for the first ($i = 1$) and second ($i = 2$) system, respectively [6]. The total transfer function from the input $R(s)$ to the output $Y_1(s)$ can be computed as

$$G(s) = \frac{Y_1(s)}{R(s)} = \frac{G_1(s)e^{-T_1 s}}{1 + G_1(s)G_2(s)e^{-(T_1+T_2)s}} \quad (1.1)$$

From the above equation and denominator, the dependence of the stable and unstable poles on the round-trip delay value becomes evident. More importantly, we have that for a positive round trip delay, i.e., $T_1 + T_2 > 0$, the closed-loop system has an infinite number of poles. Therefore, conventional control linear analysis is not sufficient to fully comprehend the behavior of (1.1). In order to show explicitly the effect of delays on stability, let $G_1(s) = k \frac{1}{s}$ and $G_2(s) = 1$ where k is a control parameter. Then, the characteristic equation of (1.1) is given by

$$s + ke^{-(T_1+T_2)s} = 0 \quad (1.2)$$

For no delay, we may easily check that the closed-loop system is stable for any $k > 0$. However, once there is a positive delay in the control loop; one can always find a positive value of k for which the system will become unstable. Indeed, the system is unstable for any round-trip delay satisfying [7] the following inequality:

$$T_1 + T_2 > \frac{\pi}{2k} \quad (1.3)$$

Thus it is crucial to consider delays within the system especially those accumulated by network roundtrip delays due to the teleoperative system discussed within this paper to ensure stability. Furthermore the inverse case is also possible and needs to be noted that unstable or marginally stable systems might be stabilized by delay output feedback [6] [8] or in the solution discussed in Chapter 5 Network Architecture by using a buffer to compensate for the variation within the network delay such that the overall delay can be stabilized by using a play-back buffer [9].

1.2 INTERNET: AN INTRODUCTION

The Internet is a global system of interconnected computer networks that use the standard Internet Protocol Suite (TCP/IP) to serve billions of users worldwide. It is a network of networks that consists of millions of private, public, academic, business, and government networks, of local to global scope, that are linked by a broad array of electronic, wireless and optical networking technologies. The Internet carries a vast range of information resources and services, such as the inter-linked hypertext documents of the World Wide Web (WWW) and the infrastructure to support electronic mail.¹

Most traditional communications media including telephone, music, film, and television are reshaped or redefined by the Internet, giving birth to new services such as Voice over Internet Protocol (VoIP). Newspapers, books and other print publishing materials are adapting to Web site technology, or are reshaped into blogging and web feeds. The Internet has enabled or accelerated new forms of human interactions through instant messaging, Internet forums, and social networking. Online shopping has boomed both for major retail outlets and small artisans and traders. Business-to-business and financial services on the Internet affect supply chains across entire industries.

The origins of the Internet reach back to research of the 1960s, commissioned by the United States government in collaboration with private commercial interests to build robust, fault-tolerant, and distributed computer networks. The funding of a new U.S. backbone by the National Science Foundation in the 1980s, as well as private funding for other commercial backbones, led to worldwide participation in the development of new networking technologies, and the merger of many networks. The commercialization of what was by the 1990s an international network resulted in its popularization and incorporation into virtually every aspect of modern human life. As of 2009, an estimated quarter of Earth's population used the services of the Internet.

The Internet has no centralized governance in either technological implementation or policies for access and usage; each constituent network sets its own standards. Only the overarching definitions of the two principal name spaces in the Internet, the Internet Protocol address space and the Domain Name System, are directed by a maintainer organization, the Internet Corporation for Assigned Names and Numbers (ICANN). The technical underpinning and standardization of the core protocols (IPv4 and IPv6) is an activity of the Internet Engineering Task Force (IETF), a non-profit organization of loosely affiliated international participants that anyone may associate with by contributing technical expertise.

1.3 PROTOCOLS

The complex communication infrastructure of the Internet consists of its hardware components and a system of software layers that control various aspects of the architecture. While the hardware can often be used to support other software systems, it is the design and the rigorous standardization process of the software architecture that characterizes the Internet and provides the foundation for its scalability and success. The responsibility for the architectural design of the Internet software systems has been delegated to the Internet Engineering Task Force (IETF). [10] The IETF conducts standard-setting work groups; open to any individual, about the various aspects of Internet architecture. Resulting discussions and final standards are published in a series of publications; each called a Request for Comments (RFC), freely available on the IETF web site. The principal methods of networking that enable the Internet are contained in specially designated RFCs that constitute the Internet Standards. Other less rigorous documents are simply informative, experimental, or historical, or document the best current practices (BCP) when implementing Internet technologies.

The Internet Standards describe a framework known as the Internet Protocol Suite. This is a model architecture that divides methods into a layered system of protocols (RFC 1122, RFC 1123). The layers correspond to the environment or

¹ Historical representation provided by living Internet and internet RFCs (originally invented by Steve Crocker).

scope in which their services operate. At the top is the Application Layer, the space for the application-specific networking methods used in software applications, e.g., a web browser program. Below this top layer, the Transport Layer connects applications on different hosts via the network (e.g., client-server model) with appropriate data exchange methods. Underlying these layers are the core networking technologies, consisting of two layers. The Internet Layer enables computers to identify and locate each other via Internet Protocol (IP) addresses, and allows them to connect to one-another via intermediate (transit) networks. Lastly, at the bottom of the architecture, is a software layer, the Link Layer, that provides connectivity between hosts on the same local network link, such as a local area network (LAN) or a dial-up connection. The model, also known as TCP/IP, is designed to be independent of the underlying hardware which the model therefore does not concern itself with in any detail. Other models have been developed, such as the Open Systems Interconnection (OSI) model, but they are not compatible in the details of description, nor implementation, but many similarities exist and the TCP/IP protocols are usually included in the discussion of OSI networking.

The most prominent component of the Internet model is the Internet Protocol (IP) which provides addressing systems (IP addresses) for computers on the Internet. IP enables internetworking and essentially establishes the Internet itself. IP Version 4 (IPv4) is the initial version used on the first generation of the today's Internet and is still in dominant use. It was designed to address up to ~ 4.3 billion (10^9) Internet hosts. However, the explosive growth of the Internet has led to IPv4 address exhaustion which is estimated to enter its final stage in approximately 2011. [11] A new protocol version, IPv6, was developed in the mid-1990s which provides vastly larger addressing capabilities and more efficient routing of Internet traffic. IPv6 is currently in commercial deployment phase around the world and Internet address registries (RIRs) have begun to urge all resource managers to plan rapid adoption and conversion. [12]

IPv6 is not interoperable with IPv4. It essentially establishes a "parallel" version of the Internet not directly accessible with IPv4 software. This means software upgrades or translator facilities are necessary for every networking device that needs to communicate on the IPv6 Internet. Most modern computer operating systems are already converted to operate with both versions of the Internet Protocol. Network infrastructures, however, are still lagging in this development. Aside from the complex physical connections that make up its infrastructure, the Internet is facilitated by bi- or multi-lateral commercial contracts (e.g., peering agreements), and by technical specifications or protocols that describe how to exchange data over the network. Indeed, the Internet is defined by its interconnections and routing policies.

1.4 UNDERSTANDING NETWORK TOPOLOGY AND DISTRIBUTION

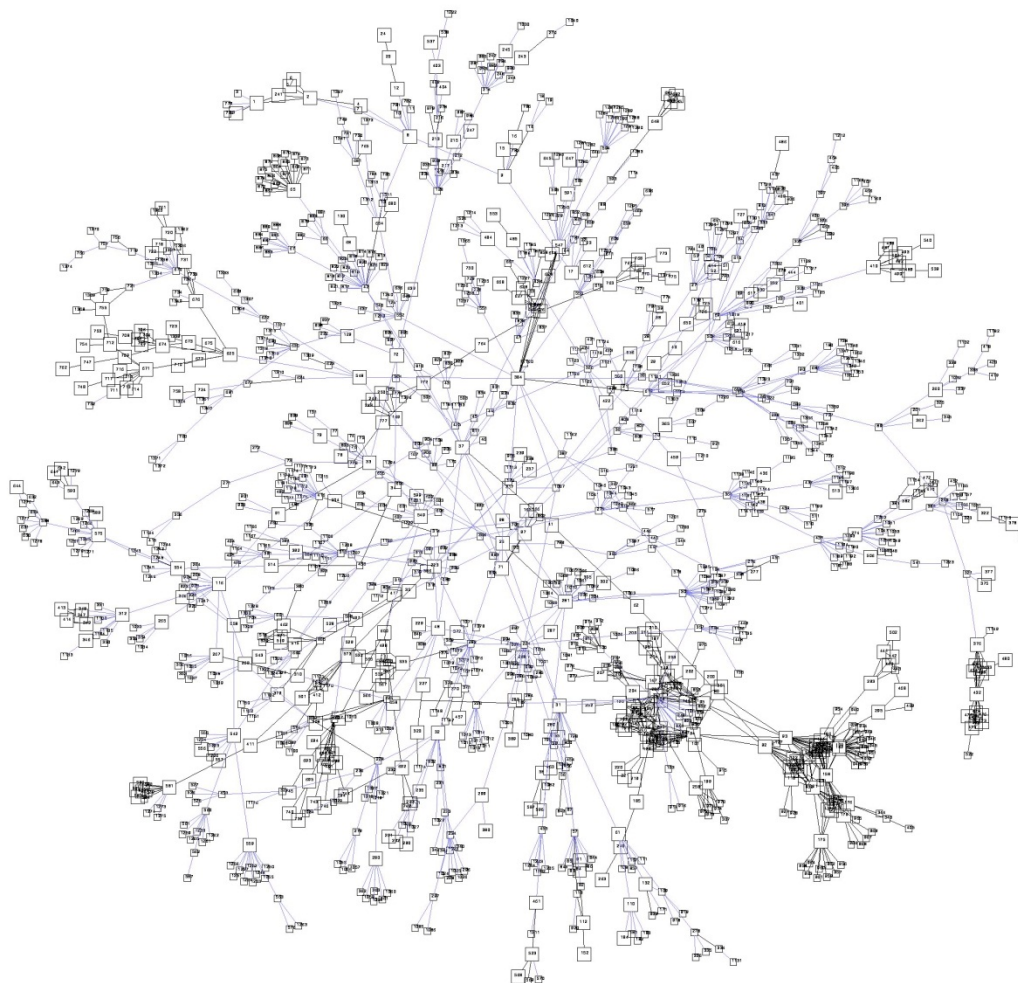


FIGURE 1.2 MAP OF MULTICAST INTERNET IN AUGUST 1996 (PRODUCED BY ELAN AMIR, UNIVERSITY OF CALIFORNIA AT BERKELEY)

To further build on the notion of understanding delay built up within networks, various network backbones will be visited to understand the building blocks of how routing occurs and how path optimization is at work with every packet travelled across the two nodes in which teleoperation will occur. One of the main methods of communications supporting domestic internet pipelines are based on optical fiber channels across United States [13] grounded by the synchronous optical networking (SONET) and synchronous digital hierarchy (SDH). When packets travel from one node to another they are almost never directly transferred across and pass through multiple Intranet service nodes to reach their destination. These nodes (mostly based on fiber optic networks) induce propagation delays due to hardware switches aiding network transmission and often cause delays by themselves aside from network congestion which also contributes to increase the roundtrip delay of packets. Furthermore certain networks aren't based on fiber optic networks and rather a slower less budget demanding technologies which often also are one of the leading cause of transmission delays, however these networks depending on topology, location, and congestion within that area can sometimes be avoided. A sample map of the multicast Internet backbone (MBone) topology in August 1996 is displayed

in Figure 1.2. This map shows just how complex the network topology was back in 1996 and another picture is provided in Figure 1.3 showing modern topology, a comparison can be made as to how fast and drastic the expansion is.

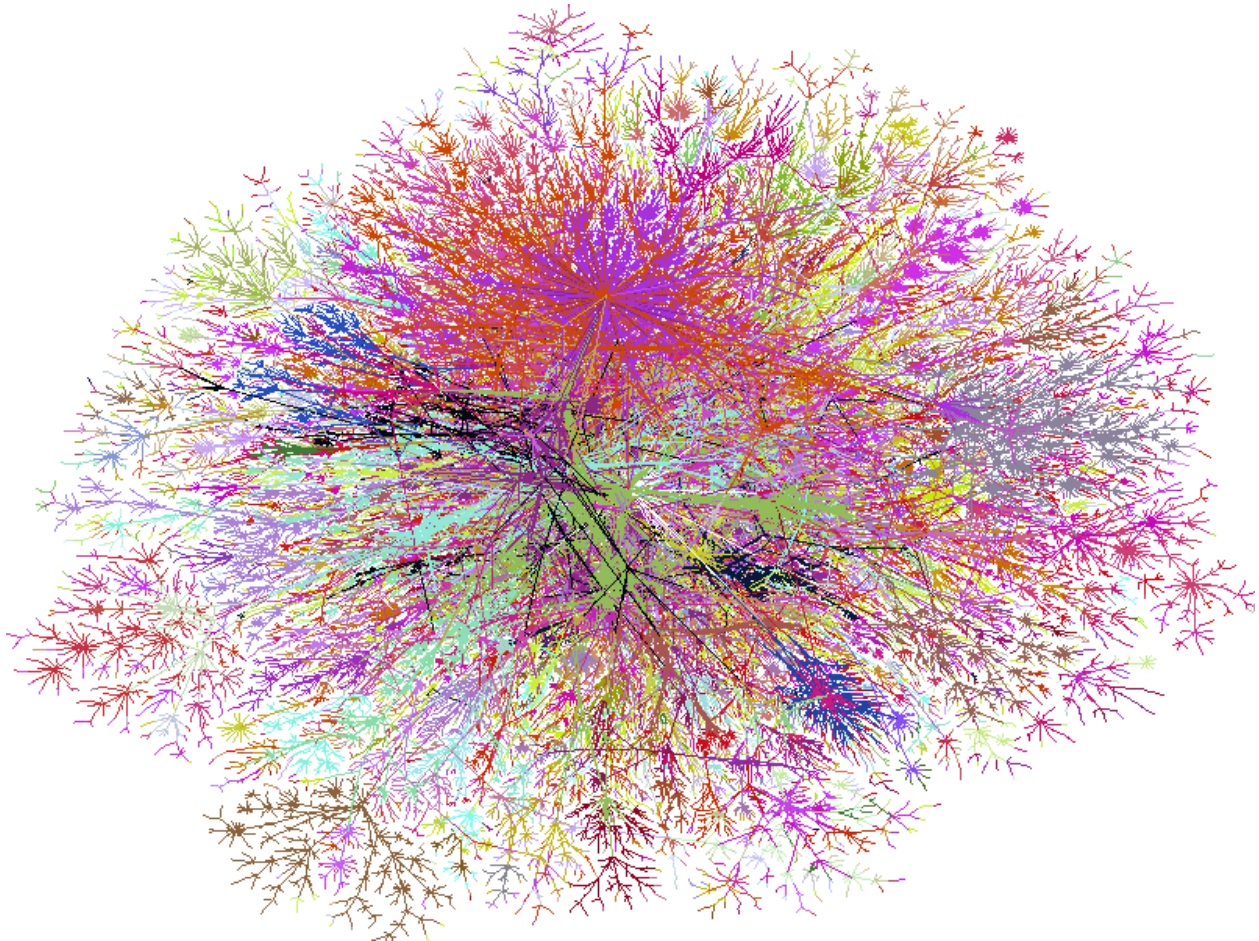


FIGURE 1.3 CURRENT INTERNET TOPOLOGY (CONNECTIONS OF ALL THE SUB-NETWORKS IN THE WORLD) CREDIT: BILL CHESWICK, LUMETA CORP

The Internet automatically chooses the average of best route path to take that leads utilizing a path with the least amount of concurrent congestion such that the delay is minimized, however the path can never be predicted due to the spontaneous initiation of devices that begin transmitting packets; thus the network congestion can never be predicted and is always modeled as a random variable. This randomness causes the packets to be sent from different nodes depending on their current congestion and causes variations within round trip delays thus causing instabilities within the overall design of the controller. Several solutions are discussed within this paper to address this issue aside from pragmatic tactics to create a real-time transmission (discussed in Chapter 6 Pragmatic Design). Some methods include utilizing play-back buffer briefly discussed earlier and more in detail in Chapter 4 Controller Applications.

In a practice known as static routing (or non-adaptive routing), small networks may use manually configured routing tables. Larger networks have complex topologies that can change rapidly, making the manual construction of routing tables unfeasible. Nevertheless, most of the public switched telephone network (PSTN) uses pre-computed routing tables, with fallback routes if the most direct route becomes blocked. Adaptive routing, or dynamic routing, attempts to solve this problem by constructing routing tables automatically, based on information carried by routing protocols, and allowing the network to act nearly autonomously in avoiding network failures and blockages.

Examples of adaptive-routing algorithms are the Routing Information Protocol (RIP) and the Open-Shortest-Path-First protocol (OSPF). Adaptive routing dominates the Internet. However, the configuration of the routing protocols often requires a skilled touch; networking technology has not developed to the point of the complete automation of routing.

Distance vector algorithms use the Bellman-Ford² algorithm. This approach assigns a number, the cost, to each of the links between each node in the network. Nodes will send information from point A to point B via the path that results in the lowest total cost (i.e. the sum of the costs of the links between the nodes used).

The algorithm operates in a very simple manner. When a node first starts, it only knows of its immediate neighbours, and the direct cost involved in reaching them. (This information, the list of destinations, the total cost to each, and the next hop to send data to get there, makes up the routing table, or distance table.) Each node, on a regular basis, sends to each neighbour its own current idea of the total cost to get to all the destinations it knows of. The neighbouring node(s) examine this information, and compare it to what they already 'know'; anything which represents an improvement on what they already have, they insert in their own routing table(s). Over time, all the nodes in the network will discover the best next hop for all destinations, and the best total cost.

When one of the nodes involved goes down, those nodes which used it as their next hop for certain destinations discard those entries, and create new routing-table information. They then pass this information to all adjacent nodes, which then repeat the process. Eventually all the nodes in the network receive the updated information, and will then discover new paths to all the destinations which they can still "reach." This is the main reason in variation of round trip delays at every instant.

² The Bellman-Ford algorithm computes single-source shortest paths in a weighted digraph. For graphs with only non-negative edge weights, the faster Dijkstra's algorithm also solves the problem. Thus, Bellman-Ford is used primarily for graphs with negative edge weights. The algorithm is named after its developers, Richard Bellman and Lester Ford, Jr.

1.5 NETWORKED BILATERAL TELEOPERATION

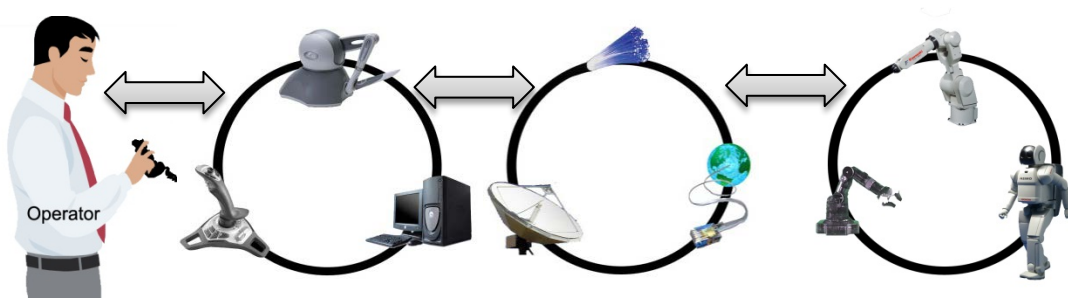


FIGURE 1.4 TELEOPERATION CONFIGURATION

Teleoperation indicates operation of a machine at a distance. It is similar in meaning to the phrase "remote control" but is usually encountered in research, academic and technical environments. It is most commonly associated with robotics and mobile robots but can be applied to a whole range of circumstances in which a device or machine is operated by a person from a distance. Teleoperation is also standard term in use both in research and technical communities and is by far the most standard term for referring to operation at a distance. This is opposed to "telepresence" which is a less standard term and might refer to a whole range of existence or interaction that include a remote connotation. In most cases a teleoperation system includes many processes that begin with local sensing for the human operator to interface from the human movements using a joystick or sensing device. Following sensing and locally tracking the motion of the human operator the data is collected by a device and encapsulated and transmitted via some connection medium across to the remote data collector which translates the data into a robot movement as real-time as possible, thus mimicking the human movements in a remote location via some remote robot. This sort of behavior can also be bilateral as seen in Figure 1.4. In other words feedback from the robot can also be transmitted back to the local human operator for better sense and feel. In both cases the local operator's control input is captured by a robot namely master robot while the remote robot local is referred to as slave robot. There exist various configurations of such master-slave hierarchy, one of which is the single master, multiple slave configurations which allow a single human operator to control multiple machines simultaneously.

Transparency often plays a key role in such master-slave relationships; it allows the system to feel more real-time with minimized delay and high accuracy among the two nodes. This is generally attained by transmitting remote slave information (e.g., position, velocity, and force) to the master robot in what is called a bilateral connection. Achieving transparency (commonly measured in terms of motion coordination, impedance matching, and force reflection) and stability of bilateral teleoperation systems has proved to be difficult and more than often, a conflicting task due to time delays in the control loop [14]. As a result of attempting to create transparency one of the main factors include time delays which arise from the distance between the master and slave robots and the network factors involved (congestion, speed, link quality etc.) Regardless of size of time delay which in the case of typical domestic internet range anywhere between 100ms ~ 500ms, time delays degrade performance and are always a negative factor that can often bring instability to a system.

CHAPTER 2 MODEL REFERENCE ROBUST CONTROL

2.1 REFERENCE MODEL

Due to diverse natural factors (e.g., propagation and transport phenomena) and implementation requirements (e.g., discretization and networking), time delays often appear in control systems. For instance, control of chemical processes, such as chemical reactors [15] and heat exchangers [16], typically experience time delays in the control loop as the result of mass transport and heat transfer phenomena. Similarly, data transmission in analog and digital communication-based NCSs inherently suffer from positive propagation delays due to the time it takes for the transmitted signal to travel from one end-point to another. In these scenarios, the presence of time delays in the control loop can degrade the performance of the control process and even lead to instability. Therefore, it is of great significance to formulate control algorithms conformed to time delay models. Following the research line of [17] [18] [19] [20], we now present the design of a model reference robust control (MRRC) framework that combines the use of the wave-based scattering transformation [21] to guarantee asymptotic stability of nonlinear dissipative Lagrangian systems¹ with dynamic uncertainties and arbitrary large input and state measurement constant delays. The proposed control law assumes that the unforced (i.e., zero input control) system is exponentially stable or, equivalently, output strictly passive in order to establish delay-independent stability of the controlled system. The design of the controller is comprised of two parts: a linear reference model and a scattering transformation block. The first is designed according to a desired input-to-output property that the delayed system must mimic, while the latter is used to stabilize the delayed coupling between the plant and the controller. In addition, the outputs of the scattering transformation are passively modified to enable explicit full state tracking between controller (i.e., reference model) and plant independently of dissimilar and unknown initial conditions as well as losses in the transmission lines, a recurring problem with scattering transformation based of motion.

We design, for simplicity, an asymptotically stable linear reference model as

$$\ddot{q}_m(t) = A_m \dot{q}_m(t) + u_m(t) + r_m(t) \quad (2.1)$$

$$y_m(t) = \dot{q}_m(t) \quad (2.2)$$

Where $q_m(t), \dot{q}_m(t) \in \mathbb{R}^n$ are the state vectors, $y_m(t) \in \mathbb{R}^n$ is the output vector, $u_m(t) \in \mathbb{R}^n$ is the control input, $A_m(t) \in \mathbb{R}^{n \times n}$ is a symmetric Hurwitz matrix. The reference signal $r_m(t) \in \mathbb{R}^n$ is given by

$$r_m(t) = K_d(q_d - q_m(t)) \quad (2.3)$$

Where $K_d(t) \in \mathbb{R}^{n \times n}$ is a positive-definite constant matrix and $q_d(t) \in \mathbb{R}^n$ is the desired state constant vector.

2.2 SCATTERING TRANSFORMATION

If the reference model and the time delay nonlinear system are to be directly coupled through their delayed outputs $q_m(t - T_2)$ and $q(t - T_1)$ and/or $\dot{q}_m(t - T_2)$ and $\dot{q}(t - T_1)$, it can be shown that the communication channel may act as a non-passive coupling element (i.e., may generate energy), potentially leading the system to instability [21]. In order to passify the communication channel and avoid instability, we propose the use of the wave-based scattering transformation. The wave variables $w_m(t)$ and $v(t)$, and the new control inputs $u_m(t) = -\tau_m(t)$ and $u(t) = \tau(t)$ are then computed as

$$\tau_m(t) = b\dot{e}_m(t) + K_m e_m(t) \quad (2.4)$$

$$w_m(t) = \sqrt{\frac{2}{b}} \tau_m(t) - v_m(t) \quad (2.5)$$

$$\dot{q}_{md}(t) = \frac{1}{b} (\tau_m(t) - \sqrt{2b} v_m(t)) \quad (2.6)$$

$$q_{md}(t) = \int_0^t \dot{q}_{md}(\theta) d\theta \quad (2.7)$$

$$e_m(t) = q_m(t) - q_{md}(t) \quad (2.8)$$

For the reference model and

$$\tau(t) = \sqrt{2b} w(t) \quad (2.9)$$

$$v(t) = w(t) - \sqrt{2b} \dot{q}(t) \quad (2.10)$$

for the nonlinear system; where the wave impedance b is a positive constant, K_m is a symmetric positive definite matrix, and

$$v_m(t) = v(t - T_1) \quad (2.11)$$

$$w(t) = w_m(t - T_2) \quad (2.12)$$

The implementation of the scattering transformation and the reference model is schematized in Figure 2.1. The importance of the scattering transformation lies on the passivation of the communication channel independently of any arbitrary large constant round-trip delays. To demonstrate this statement, let us verify that the communication channel is, in fact, passified. Manipulating (2.4)-(2.12) we can easily show that

$$\dot{q}_{md}^T \tau_m - \dot{q}^T (\tau - b \dot{q}) = \frac{1}{2} (w_m^T w_m - w^T w + v^T v - v_m^T v_m) \quad (2.13)$$

Then, integrating (2.13) with respect to time yields.

$$\int_0^t (\dot{q}_{md}^T \tau_m - \dot{q}^T (\tau - b \dot{q})) d\theta = \frac{1}{2} \int_{t-T_2}^t w_m^T w_m d\theta + \frac{1}{2} \int_{t-T_1}^t v^T v d\theta \geq 0 \quad (2.14)$$

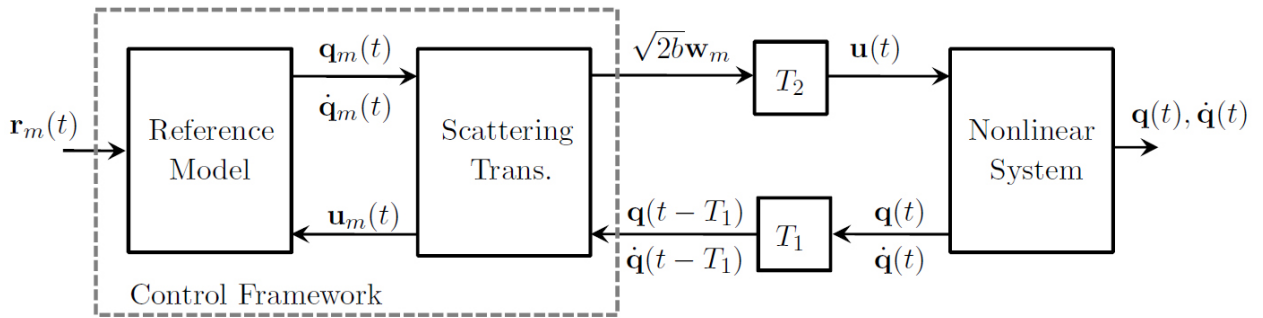


FIGURE 2.1 MRRC FRAMEWORK

which confirms the passivity claim for a small and constant delay³. The lower bound in (2.14) implies that the energy is temporary stored in the transmission lines and therefore, the communication channel is passified independently of the size of T_1 and T_2 so long as they are relatively small and constant⁴.

2.3 STABILITY ANALYSIS AND STATE CONVERGENCE

The equations of motions of an n-DOF (Degree of freedom) Lagrangian system are given by

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} = u - \frac{\partial \mathcal{F}(\dot{q})}{\partial \dot{q}} \quad (2.15)$$

Where gravitational effects have been either neglected or cancelled through constant control (i.e. gravitational forces are constant). Where M is the inertia matrix C is the Coriolis matrix while the control input u is assumed to be a delayed state feedback function depending on $q(t - T_1 - T_2)$ and $\dot{q}(t - T_1 - T_2)$, where $T_1 \geq 0$ and $T_2 \geq 0$ correspond to the measurement and plant-to-controller communication delay and the controller-to-plant communication delay, respectively. Having established the control framework and the passivation of the communication channel, we now proceed to claim asymptotic stability of (2.15) and state convergence independently of arbitrary large input and state measurement delays. The following theorem represents one of the main results of this chapter.

Theorem: Consider the time delay nonlinear system (2.15) coupled to the reference model (2.1) via the scattering transformation (2.4) to (2.12) and let $b < p$. Then, for all initial conditions we have the following results [6]

- i. All signals $q_m(t), q(t), e_m(t), \dot{q}_m(t), \dot{q}(t), \dot{e}_m(t), \ddot{q}_m(t)$, and $\ddot{q}(t)$ are bounded $\forall t \geq 0$ and the velocities $\dot{q}_m(t), \dot{q}(t), \dot{e}_m(t)$ converge to zero.
- ii. The error signals $e_m(t)$ and $q_m(t) - q_d$ converge asymptotically to zero.

Proof: Consider the following Lyapunov candidate function

$$\begin{aligned} V(t) &= V(q_m(t), e_m(t), \dot{q}_m(t), \dot{q}(t)) \\ &= \frac{1}{2} \dot{q}^T M(q) \dot{q} + \frac{1}{2} (q_d - q_m)^T K_d (q_d - q_m) + \frac{1}{2} \dot{q}_m^T \dot{q}_m + \frac{1}{2} e_m^T K_m e_m \\ &\quad + \int_0^t (\dot{q}_{md}^T - \dot{q}^T (\tau - b\dot{q})) d\theta \end{aligned} \quad (2.16)$$

Its time derivative is given by

$$\dot{V} = -\rho \dot{q}^T \dot{q} + \dot{q}_m^T A_m \dot{q}_m + b \dot{q}^T \dot{q} + \dot{e}_m^T K_m e_m - \dot{q}_m^T (b \dot{e}_m + K_m e_m) + \dot{q}_{md}^T (b \dot{e}_m + K_m e_m) \quad (2.17)$$

Since $b < \rho$ and A_m is Hurwitz, we have that

³ From experiments it can be approximate deduced that a round-trip delay of ~ 500 ms is the threshold until the control systems starts showing reduced performance

⁴ The definition of the scattering transformation proposed here differs from its typical implementation [21] in the sense that the current states of the time delay nonlinear plant are assumed to be inaccessible to the local plant, and therefore, cannot be used when computing the transformation variables. Consequently, all scattering transformation variables are computed at the same location in the network (see Figure 2.1 MRRC Framework), as opposed to their conventional bisected (or mirror) implementation

$$\dot{V} \leq -(\rho - b)\|\dot{q}\|^2 - \mu\|\dot{q}_m\|^2 - b\|\dot{e}_m\|^2 \leq 0 \quad (2.18)$$

Where $\mu > 0$ is the smallest eigenvalues of $-A_m$. Therefore, the overall system is stable in the sense of Lyapunov.

2.4 DESIGN SPECIFICATIONS

In regards to the mathematical deductions stated above the controller's stability and performance is very much dependent on the input network delay analysis. The controller can withstand input delays however there must be an upper bound to the delay no more than 500ms and a particular delay that does not fluctuate and is as constant as possible. Following this design specification the network delay within the system can affect the controller performance since the delay is never constant and may be higher than the accepted controller stability threshold. Equation (2.10) defines this delay used for the controller.

CHAPTER 3 SITE SURVEY

3.1 GEOGRAPHICAL LOCATIONS

The time delay between two nodes is naturally higher when the distance among the two nodes is increased. The medium used to currently carry information among two nodes depends on the distance separating the two nodes. Often times when two nodes are very close perhaps within one room the network carrying the workstations (nodes) will most likely be an Ethernet setup within a small proximity and the network is known as LAN (Local Area Network). Once expanding beyond LAN and trying to establish a connection between two workstations the data packet travels via various nodes to reach its destinations. These nodes are often hosted by ISP's (Internet Service Providers) and serve a connection point purpose within the internet topology.

In the site survey which was performed for the setup of the controller the two ends of the nodes were the same locations in which the client and server programs (discussed in Chapter 5 Network Architecture) resided. Geographically one was located at the University of Illinois in Urbana-Champaign, Illinois, the other at Boeing in Bellevue, Washington.⁵

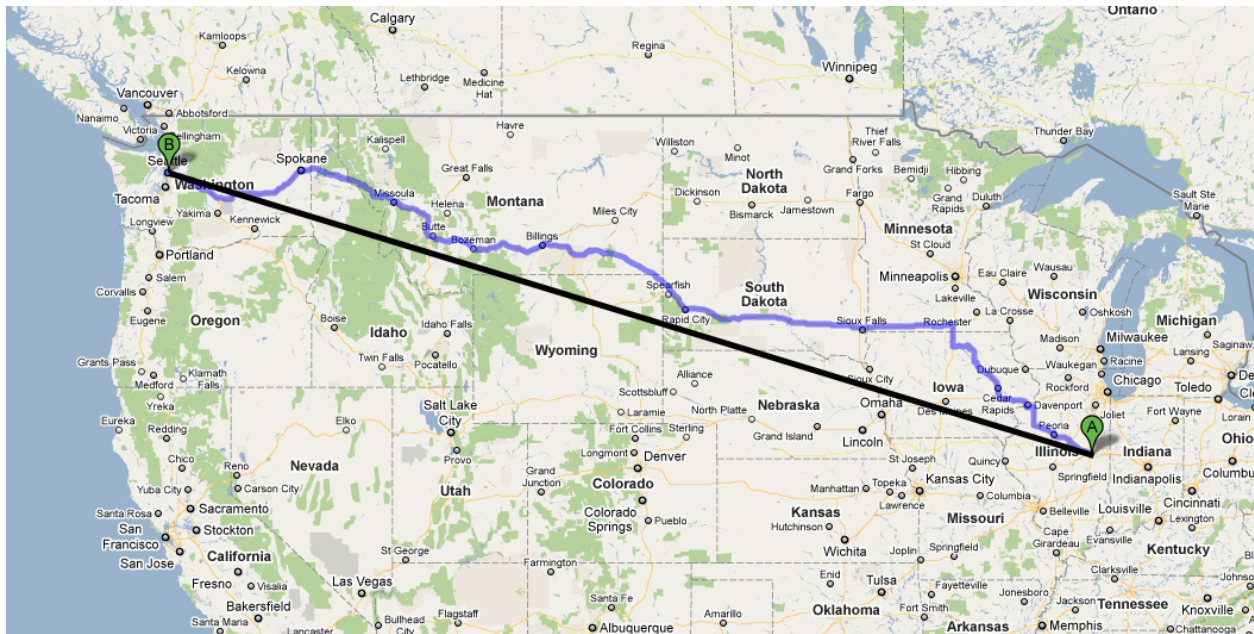


FIGURE 3.1 CONNECTION DISTANCE

From the Figure 3.1 it can be seen that although technically the straight route cannot be traversed on land there exists a state route which traverses city to city via various types of roads. This is very similar to the path the packets will be travelling. The nodes visited by each packet resembled the cities and are in fact located within cities connected via various mediums of connection (Fiber optic, wireless, coaxial etc.) The two nodes taken into account are separated by 2'091miles and will be put through rigorous tests to analyze the connection at hand.

⁵ IP addresses and ports conducted within this survey are deliberately masked as to not expose any details of Boeing and UIUC's underlying network architecture

3.2 SITE SURVEY SERVER SETUP

Following the setup discussed in previous the section two workstations were setup, one in University of Illinois – Coordinated Sciences Lab and the other in Boeing research building in Bellevue Washington. The server setup included a pair of dedicated Linux servers. To obtain connection analysis results the servers would begin rigorous testing on the network layout separating them to obtain information about connection bandwidth, round trip delay, delay consistency, bidirectional bandwidth test, and trace route. The entire results of the tests will then be taken into consideration for obtaining information regarding not only an upper bound on the round trip delay however the nature of the delay and its consistency. The latter information will be used to design the playback buffer (discussed in Chapter 4 Controller Applications).

Both servers ran two separate Linux architectures, one utilized Gentoo Linux while the other was equipped with Ubuntu Linux architecture. Each workstation was loaded with both a client and a server program. The server hosts the session to transmit packaged data to measure connection information while the client transmits the data. The same procedure would be repeated however in the opposite direction. Once both results are obtained a brief overview of the network delay and bandwidth can be measured via various package sizes and parameters transmitted between the two nodes. Following the measurement the same process can be repeated with full duplex mode (multithreaded operation) to host the same exchange of data however simultaneously to obtain a bidirectional connection.

The client server program utilized is a variation of iPerf. iPerf is an efficient program for measuring throughput, jitter and datagram loss. It has both client and server pieces, so it requires installation at both ends of the connection being measured. Three terminals can be laid out to view three activities at once, one for the client, one for the server, and one running tcpdump just to see all those packets zoom by. iPerf can be initiated by:

```
sam@corvinus:~$ iperf -s
sam@eos.cs.boeing:~$ iperf -c xena
```

By default iperf uses TCP/UDP port 5555. This is the result of a run without tcpdump running:

```
-----
Server listening on TCP port 5555
TCP window size: 85.3 KByte (default)
-----
[ 4] local 192.168.20.200 port 5555 connected with 192.168.20.200 port 5555
[ 4] 0.0-10.0 sec 112 MBytes 93.8 Mbits/sec
```

While tcpdump is running:

```
[ 5] 0.0-10.0 sec 56.5 MBytes 47.3 Mbits/sec
```

By default, iperf sends TCP packets over wires as fast as possible. A bi-directional test, which is the -d option, forces the system to run both ways

```
sam@eos.cs.boeing:~$ iperf -c xena -d
[ 4] 0.0-10.0 sec 109 MBytes 91.3 Mbits/sec
[ 5] 0.0-10.0 sec 84.5 MBytes 70.8 Mbits/sec
```

iPerf can also be used to test for full bandwidth test to measure jitter⁶

```
sam@corvinus:~$ iperf -su
sam@eos.cs.boeing:~$ iperf -c xena -u -b 100m
```

⁶Jitter in technical terms is the deviation in or displacement of some aspect of the pulses in a high-frequency digital signal. As the name suggests, jitter can be thought of as shaky pulses.

[ID]	Interval	Transfer	Bandwidth	Jitter	Lost/Total
	Datagrams				
[4]	0.0-10.0 sec	113 MBytes	95.0 Mbits/sec	0.008 ms	544/81389 (0.67%)
[4]	0.0-10.0 sec	1 datagrams received out-of-order			

The above represents a sample data communication between two very fast and close networks. The same tests will then be applied for the connection at hand and the results will be analyzed.

Another tool used for analysis of site survey was Bwping. Bwping is a tool to measure bandwidth and response times between two hosts using Internet Control Message Protocol (ICMP) echo request/echo reply mechanism. It does not require any special software on the remote host. The only requirement is the ability to respond on ICMP echo request messages. Also to measure throughput of network for problem specification purposes the TCP receive window and RTT (Round trip time) for the path are required. Furthermore to measuring bandwidth the network throughput is also measured using TTCP (Test TCP) utility.

The requirements for calculating network throughput are given using the tools mentioned earlier. As defined:

$$Throughput \leq \frac{RWIN}{RTT} \quad (3.1)$$

where RWIN is the TCP Receive Window and RTT is the round-trip time for the path. The Max TCP Window size in the absence of TCP window scale option is 65,535 bytes however in the case of the testing a predefined window size of 1.43Mbytes was picked (discussed later for unidirectional bandwidth test).

$$Max\ Bandwidth = \frac{1.43\ MBytes}{11s} \times 8 = \frac{1.09Mbits}{s} \quad (3.2)$$

We multiply the Byte per second times 8 to get the Bit per second rate. Over a single TCP connection between those endpoints, the tested Bandwidth will be restricted to 1.09 Mbit/s or less even if the contracted Bandwidth is greater.

3.3 GEOLOCATIONAL TRACEROUTE

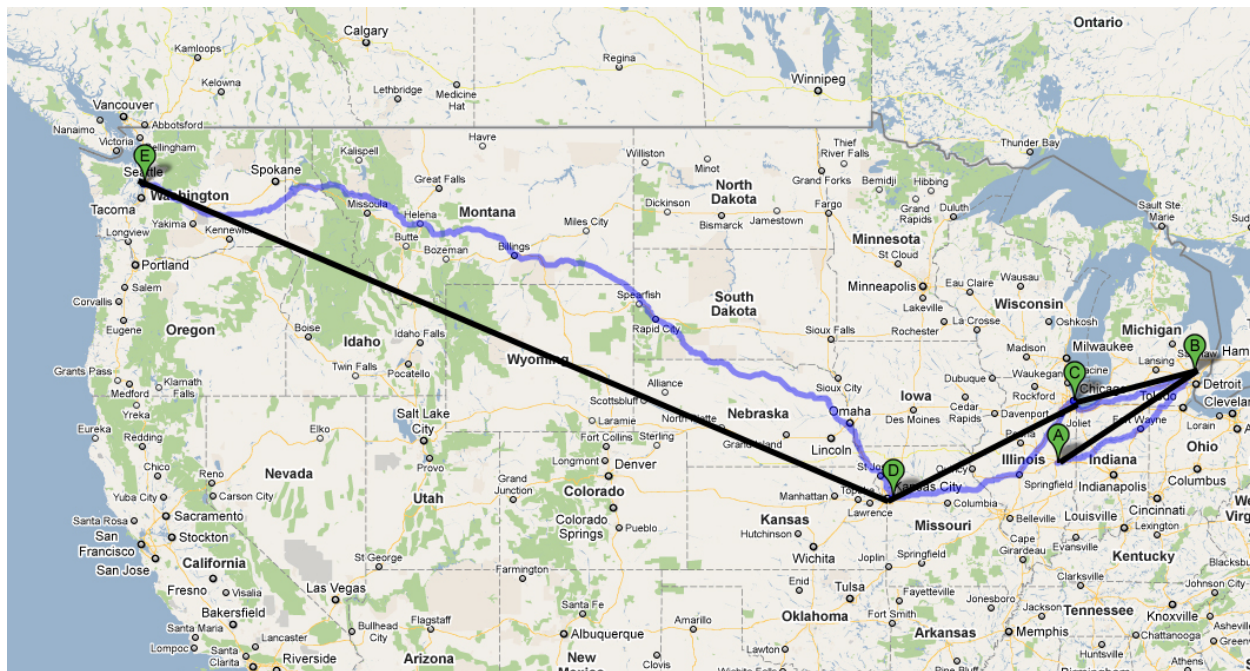


FIGURE 3.2 CONNECTION ROUTE VIA NODES FROM CHAMPAIGN TO BELLEVUE

The analysis of packet communication behavior between the two nodes resulted in packets departing from Champaign and visiting various nodes until reaching its final destination in Bellevue. The nodes visited are each identified by their IP addresses and are later analyzed. The resulting IP's traced during site survey resulted in the following public IP addresses:

```

01 eos.csl.illinois.edu (130.126.138.69) 0.206 ms
02 24.12.200.1 (24.12.200.1) 13.679 ms
03 te-5-1-ur01.champaign.il.illinois.edu (68.85.178.65) 199.867 ms
04 68.85.177.225 (68.85.177.225) 22.227 ms
05 pos-1-14-0-0-ar01.area4.il.chicago.comcast.net (68.87.231.17) 27.459 ms
06 pos-1-15-0-0-cr01.chicago.il.ibone.comcast.net (68.86.93.181) 26.427 ms
07 xe-9-3-0.edge1.Chicago2.Level3.net (4.71.248.21) 47.630 ms
08 vlan51.ebr1.Chicago2.Level3.net (4.69.138.158) 43.234 ms
09 ae-3-3.ebr2.Denver1.Level3.net (4.69.132.61) 56.159 ms
10 ae-2-2.ebr2.Bellevue1.Level3.net (4.69.132.53) 101.770 ms
11 ae-22-52.car2.Bellevue1.Level3.net (4.68.105.35) 99.135 ms
12 PACIFIC-NOR.car2.Bellevue1.Level3.net (4.53.146.142) 84.094 ms
13 ae0--4010.iccr-sttlwa01-03.infra.pnw-gigapop.net (209.124.188.134) 79.056 ms
14 209.124.188.134 (209.124.188.134) 79.382 ms
15 209.124.188.134 (209.124.188.134) 76.919 ms
16 209.124.188.134 (209.124.188.134) 117.601 ms
17 209.124.188.134 (209.124.188.134) 107.685 ms
18 209.124.188.134 (209.124.188.134) 101.739 ms7

```

⁷ Note: The addresses are blurred as to not expose the details of the Boeing internal network

The above list provides the node number (located on the far left) then the address or host, then the round trip delay (ping) in which the hop has responded to. This method is referred to as traceroute and will be considered further in detail in the Trace-Route section.

Following the testing, once the packet is sent, the connection traverses through various nodes notably nodes located within the cities indicated in Figure 3.2 obtained from recent site survey done from University of Illinois to Boeing. Although it must be noted that there are 5 nodes displayed in the map and 18 nodes from the route result, most of these nodes are placed in the same location and can be grouped in such way. The reason for this is due to server hops that may occur for processing, for example a packet may travel between several substations within the same location or server host, thus appearing as one node on the geographical map however show as several nodes on the list of nodes above. As displayed this shows that internet packets do not follow shortest path but utilize bellman Ford algorithm to calculate shortest cost path. In this context cost refers to network traffic or congestion. Due to this the path taken may change and will almost always have a fluctuating round trip delay.

The results above are obtained from doing a traceroute from a client located in Champaign, IL to Bellevue, WA. Furthermore the geographical data of the hosts are obtained from visual route specifications of each IP. This process is known as geolocation, which is the mapping of an IP address or MAC address to the real-world geographic location of an Internet connected to a computing device or mobile device. Geolocation involves in mapping IP address to the country, region (city), latitude/longitude, ISP and domain name among other useful things.

There are a number of commercially available geolocation databases, and their pricing and accuracy may vary. Ip2location, MaxMind, Tamo Soft and IPLigence offer a fee based databases that can be easily integrated into an web application. Most geolocation database vendors offers APIs and example codes (in ASP, PHP, .NET and Java programming languages) that can be used to retrieve geolocation data from the database. We use Ip2Location database to obtain the above locations from the hosts.

Accuracy of geolocation database varies depending on which database you use. For IP-to-country database, some vendors claim to offer 98% to 99% accuracy although typical Ip2Country database accuracy is more like 95%. For IP-to-Region (or City), accuracy range is anywhere from 50% to 75% if neighboring cities are treated as correct. Considering that there is no official source of IP-to-Region information, 50+% accuracy is pretty good [22].

ARIN (American Registry for Internet Numbers) Whois database provides a mechanism for finding contact and registration information for IP resources registered with ARIN. The IP whois information is available for free, and determining the country from this database is relatively easy. When an organization requires a block of IP addresses, a request is submitted and allocated IP addresses are assigned to a requested ISP [23].

3.4 BANDWIDTH TEST

As explained in Section 3.2 Site Survey Server Setup using the setup provided the bandwidth test can be done to measure the network throughput. The test was done by placing one node in Champaign, IL while the other in Bellevue, WA with a client/server relationship to have a test buffer packet be sent and measure the attributes of the transmission. The results were obtained using JPerf in combination with iPerf which are diagnostic tools for measuring bandwidth and quality of a network link. JPerf in particular is simply the graphical version of iPerf written in Java.

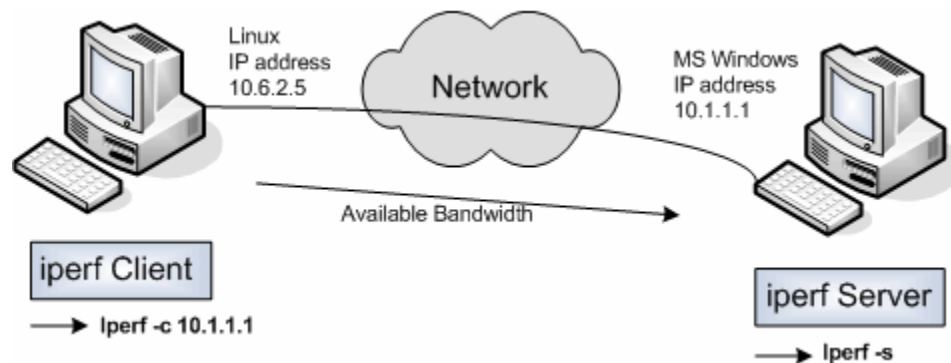


FIGURE 3.3 IPERF SETUP

Using the setup mentioned above (depicted in Figure 3.3) we can then obtain network data regarding the virtual link connecting the two nodes and the resulting output from iPerf reveals the following:

```
Node1: UIUC <—> Node2: Boeing
TCP Window Size: 16.0 Kbyte
Port: 4444

Server (Boeing):
uiuc@scorpio:~$ ./iperf -s -p 4444

Server listening on TCP port 4444
TCP window size: 85.3 KByte (default)

[ 4]
local 10.1.1.20.000 port 4444 connected with 10.1.1.20.000 port 4444
[ 4] 0.0-12.6 sec 1.43 MBytes 954 Kbits/sec

Client(UIUC):
sam@eos:~$ iperf -s -p 4444

Server listening on TCP port 4444
TCP window size: 85.3 KByte (default)

sam@eos:~$ iperf -c 10.1.1.20.000 -p 4444

Client connecting to 10.1.1.20.000, TCP port 4444
TCP window size: 16.0 KByte (default)

[ 3]
local 10.1.1.20.000 port 4444 connected with 10.1.1.20.000 port 4444
[ 3] 0.0-11.0 sec 1.43 MBytes
1.09 Mbits/sec
```

Before analyzing the above output, iPerf's command line parameters need to be described in further detail. Simply put the client server relationship are setup using the “-c” parameter which distinguishes the particular instance of iPerf to be the client instance, while the “-s” parameter signifies the server. The server is started on port⁸ 4444 using “-p” parameter and the client is connected to the respective port. Following the connection a predefined TCP window size signifies the size in which data is transmitted during each transmission of every packet⁹. However it must be noted that due to the wide area network being used the settings will be temporarily be changed to match the configuration of the hops utilized between the two nodes however the transmission configuration resumes once the packet reaches the server. This can be noted when the incoming port is 4444 is different from the port in while the server is listening to, this is due the remote router accepting connections on random ports and allocating that port to the designated server this is also known as the difference between incoming port and trigger port. iPerf transmits a default set amount of data 1.43Mbytes and clocks the time the transmission was begun and finalizes and provides a bandwidth of the transmission calculated using (3.1) yielding 1.09Mbit/sec. This bandwidth will be further discussed in 3.7 Results section.

Following the bandwidth test done above, it can be noted that it is only measuring the test going one way, while most communication especially those done with the MRRC are bidirectional. Thus the bandwidth test above simply notes the single threaded communication of sending a packet or simply transmitting using unidirectional connection. As a result iPerf will also be used to conduct a bidirectional test which can communicate both separately and simultaneously to obtain measurement of simultaneous bandwidth tests. In this case the simultaneous bandwidth test is conducted to denote a full-duplex behavior and not the separate bidirectional which simply conducts separate unidirectional tests but each of them testing opposite directions.

```

Node1: UIUC <—> Node2: Boeing
TCP Window Size: 16.0 Kbyte
Port: 4444

Server (Boeing):
uiuc@scorpio:~$ ./iperf -s -p 4444

Server listening on TCP port 4444
TCP window size: 85.3 KByte (default)

[ 4]
local 194.144.24.001 port 4444 connected with 24.12.200.92 port 4444

Client connecting to 194.144.24.001, TCP port 4444
TCP window size: 16.0 KByte (default)

[ 6]
local 194.144.24.001 port 4444 connected with 194.144.24.001 port 4444
[ 6] 0.0-12.1 sec 1.09 MBytes 752 Kbits/sec
[ 4] 0.0-17.9 sec 1.58 MBytes 741 Kbits/sec

Client(UIUC):
sam@savage:~> iperf -c 194.144.24.001 -p 4444 -d -L
4444

Server listening on TCP port 4444
TCP window size: 85.3 KByte (default)

Client connecting to 194.144.24.001, TCP port 4444
TCP window size: 16.0 KByte (default)

[ 5]

```

⁸ Although the port number is masked out to not expose too much detail the port is selected above port range of 1-1024 to ensure it is out of the router's scope of server port list and to also avoid loop back denial with certain routers

⁹ This also defines the PDU (Packet datagram unit) size within the TCP window

```

local 194.144.24.0/24 port 4567 connected with 194.144.24.0/24 port 4567
[ 4]
local 194.144.24.0/24 port 4567 connected with 194.144.24.0/24 port 4567
[ ID] Interval Transfer Bandwidth
[ 5] 0.0-11.0 sec 1.58 MBytes
1.21 Mbits/sec
[ ID] Interval Transfer Bandwidth
[ 4] 0.0-15.7 sec 1.09 MBytes 581 Kbits/sec

```

Following the above output from iPerf for bidirectional test it can be noted that the server observed two simultaneous connections of 752Kbits/sec and 741Kbits/sec. It can be noted that these values are smaller than the 1.43Mbytes/sec in fact, they are almost half of the unidirectional test which is quite justified since the pipeline is being used to transmit two simultaneous data rather than one thus a slower bandwidth is expected.

3.5 ROUND-TRIP CONSISTENCY

For a more robust connection the designed MRRC requires a constant delay. However since this cannot be made possible due to modern internet design and the fact that a network congestion is a random variable the delay will fluctuate. As a result it becomes crucial to measure the fluctuation of the delay not only for measurement purposes but also for the playback buffer which will be used to maintain a constant delay to give performance to the MRRC.

Measuring round trip consistency depends on the small sample consistency [24]. If the small sampling varies within 5% of each iteration of round trip delay then the round trip delay needs to be measured over a larger sample to obtain an average delay and upper bound that represents the current network state.

The round trip consistency is measured using the ping command which is readily available using any computer architecture. The ping is based on Internet Control Message Protocol (ICMP) which is one of the core protocols of the Internet Protocol Suite. It is chiefly used by the operating systems of networked computers to send error messages indicating, for example, that a requested service is not available or that a host or router could not be reached. ICMP can also be used to relay query messages. ICMP [25] differs from transport protocols such as TCP and UDP in that it is not typically used to exchange data between systems, nor is it regularly employed by end-user network applications (with the exception of some diagnostic tools like ping and traceroute).

Using ping the delays are then compared to see if the variance is higher than 5%.

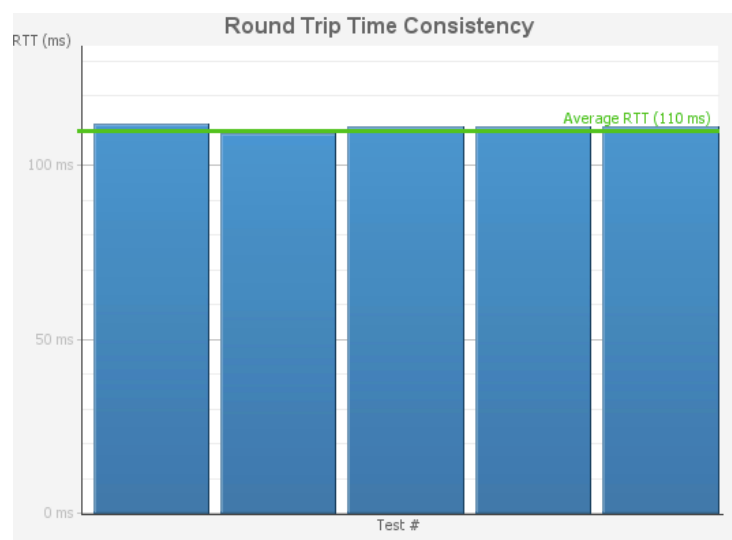


FIGURE 3.4 ROUNDTrip CONSISTENCY

Following Figure 3.4 the resulting round trip times are $< 5\%$ of one another thus the current small sample suffices according to [24]. The samples are taken with 1min delay between each other and have resulted in an average round trip delay of 110ms.

Furthermore the delay affects the transmission speed as well, the higher the delay the lower the transmission speed. Wireshark (formerly known as Ethereal) is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development. Moreover tcpdump is a common packet analyzer that runs under the command line. It allows the user to intercept and display TCP/IP and other packets being transmitted or received over a network to which the computer is attached. These tools can be used to analyze the raw data being transmitted by the ICMP and also comparing the transfer speeds with respect to the change in network delay.

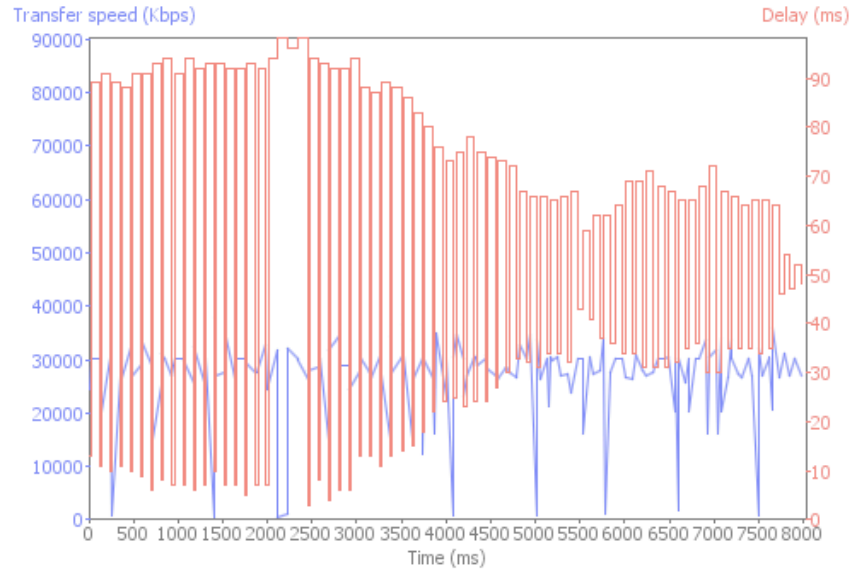


FIGURE 3.5 TRANSFER SPEED VS DELAY

The relationship discussed above can be observed in Figure 3.5. The delay is now more visualized in this graph as the transfer is in progress and packets are sent back and forth in a bidirectional manner. Each bidirectional packet transfer observes a delay denoted by the red which the blue denotes the resulting transfer speed. From the above graph it can be seen that although the connection seems stable the speed varies constantly due to the various network settings such as MTU (discussed later) and packet sizes. Furthermore it can be seen from the symmetry of the graph that the packets trip times were similar to one another. Meaning if one packet took 50ms to travel from Client to server, the same packet would also take 50ms from the Server back to the Client.

The maximum transmission unit (MTU) of a communications protocol of a layer is the size (in bytes) of the largest protocol data unit that the layer can pass onwards. MTU parameters usually appear in association with a communications interface (NIC, serial port, etc.). Standards (Ethernet, for example) can fix the size of an MTU; or systems (such as point-to-point serial links) may decide MTU at connect time [26].

A larger MTU brings greater efficiency because each packet carries more user data while protocol overheads, such as headers or underlying per-packet delays, remain fixed; the resulting higher efficiency means a slight improvement in bulk protocol throughput. A larger MTU also means processing of fewer packets for the same amount of data. In some systems, per-packet-processing can be a critical performance limitation. Although the MTU is preset by the underlying networks it will be discussed on how to optimize MTU to obtain an optimized network designed specifically for short burst transmissions utilized mainly for the MRRC. Large packets can occupy a slow link for some time, causing greater delays to following packets and increasing lag and minimum latency. For example, a 1500-byte packet, the largest

allowed by Ethernet at the network layer (and hence over most of the Internet), ties up a 14.4k modem for about one second.

Large packets are also problematic in the presence of communications errors. Corruption of a single bit in a packet requires that the entire packet be retransmitted. At a given bit error rate larger packets are more likely to be corrupted. Naturally retransmission of a larger packet takes longer.

3.6 TRACE-ROUTE

By analyzing the various nodes within the path from Client to Server, further statistical data can be found such as node consistency, paths taken by the packets (Naturally the optimal path is taken the shortest one with the least cost to take). A tabular format of the nodes traveled within the path is shown below followed by their respective delay.




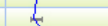




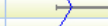

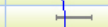

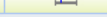


Hop	%Loss	IP Address	Node Name	Location	Tzone	ms	Graph	Network
0		130.126.138.69	eos.csl.illinois.edu			0		[Local Network]
1		130.126.138.69	eos.csl.illinois.edu			0		[Local Network]
2		24.12.200.1	-	Cherry Hill, usa		27		[Network for 24.12.200.1]
3		68.85.178.65	te-5-1-ur01.champaign.il.c	Chicago, IL, USA	-06:00	17		Comcast Cable Communicatio
4		68.85.177.225	-	Mt Laurel, usa		16		Comcast Cable Communicatio
5		68.87.231.17	pos-1-14-0-0-ar01.area4.i	Chicago, IL, USA	-06:00	26		[Network for 68.87.231.17]
6		68.86.90.49	pos-1-13-0-0-cr01.chicag	Chicago, IL, USA	-06:00	23		[Network for 68.86.90.49]
7		68.86.86.78	pos-1-2-0-0-pe01.350ece	Mt Laurel, usa		26		[Network for 68.86.86.78]
8		192.205.37.117	-	Indianapolis, IN, USA		26		AT&T Services, Inc.
9		12.122.86.42	cr2.cgclil.ip.att.net	Chicago, IL, USA	-06:00	32		[Network for 12.122.86.42]
10		12.122.2.22	cr2.sl9mo.ip.att.net	St. Louis, MO, USA	-06:00	40		[Network for 12.122.2.22]
11		12.123.140.170	cr84.sl9mo.ip.att.net	St. Louis, MO, USA	-06:00	31		[Network for 12.123.140.170]
12		12.122.142.141	-	Middletown, NJ, USA	-05:00	33		[Network for 12.122.142.141]
13		12.87.47.110	-	Morristown, NJ, USA	-05:00	35		[Network for 12.87.47.110]
14								
15		192.168.1.1	boeing.com	Chicago, IL, USA	-06:00	33		The Boeing Company

FIGURE 3.6 TRACE-ROUTE TABLE

The above traceroute summarizes the connection attempt made to connect to the server located in Boeing, Bellevue. The graph on the right side signifies the delays associated with each node. As a result it can be deduced that the delay depends on more than just one connection rather it depends on the delay induced by every node within the path.

3.7 RESULTS

From the above trace-route it can be seen that certain nodes may behave in an erratic way causing a peak in the overall delay of the communication. As a result depending on how fast and at what rate the communication is required to be achieved better results can be tuned by switching the application layer or the Transport protocol of the data to achieve a higher priority within the transmission thus achieving a higher QoS (Quality of Service). To conclude the tests a standard ping test was also done confirming the above tests and providing a better resolution on choosing optimum delay average and upper bound.

```

PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=52 time=117 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=52 time=117 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=52 time=116 ms
...
64 bytes from 192.168.1.1: icmp_seq=92 ttl=52 time=117 ms
64 bytes from 192.168.1.1: icmp_seq=93 ttl=52 time=118 ms
64 bytes from 192.168.1.1: icmp_seq=94 ttl=52 time=117 ms
- 192.168.1.1 ping statistics -
94 packets transmitted, 94 received, 0% packet loss, time 93136ms

```

Thus from the resulting tests the following round trip times can be deduced:

```

Average: 110.073ms
Minimum: 105.033ms
Maximum: 143.993ms (excluding TCP Delay)
Standard deviation: 3.966ms
Download QOS: 92%
Upload QOS: 97%
TCP delay: 48ms
Download test type: Socket
Average download pause: 6ms
Route concurrency: 6.444185
Download TCP forced idle: 87%

```

As a result the system can be modeled by assuming an average of 110ms delay for transport and 48ms for application layer giving a 158ms block delay for communication (round trip).

Simple bidirectional calculations produces the following:

$$\frac{\gamma_t + \gamma_r}{2} = \gamma_{avg} \quad (3.3)$$

Where γ_t denotes the bidirectional transmit rate and γ_r denotes the bidirectional receive rate (since bidirectional refers to simultaneous connections, the transmit and receive in this case refer to point of view from client and server respectively). Thus the bidirectional average transmission rate is given by: $\gamma_{avg} = 661\text{Kbps}$.

Furthermore, In order to assume a seamless connection the maximum delay witnessed is calculated to be 143.9ms, also once the connection is closed the socket pair associated with the connection is placed into a state known as Time-wait, which prevents other connections from using that protocol. Since multithreaded operation is used (discussed later) the TCP time delay needs to also be accounted for to account for the maximum delay visible. Thus:

Round Trip Delay: 191.9ms

<Client Side> ——— [Delay 95.9ms] ——— <Server Side>

The MRRC described above can function up to 500ms of delay until instability is observed. However the smaller the delay the better the MRRC will perform as a result 191.9ms round trip delay is well within the design specifications. Furthermore the controller will be transferring coordinate data and receiving position-velocity attributes back which require much less than 661Kbps transfer speed to operate. As a result the site survey confirms that the given operation is feasible and within margin.

CHAPTER 4 CONTROLLER APPLICATIONS

4.1 BILATERAL TELEOPERATION

Following the site survey done to assure feasibility of having a remote controller, we now center our attention to the special problem of time delay bilateral teleoperation. In principle, a teleoperation system is a dual (or multi) robotic set that enables a human operator to manipulate, sense, and physically interact with a distant environment. In such system, the desired manipulation or task is performed remotely by a slave robot which tracks the motion of a locally human-controlled master robot. The master and slave robot are coupled through a communication channel that, ideally, should be transparent to the operator, meaning that he or she should feel as if being directly active in the remote location [14]. This is generally achieved by transmitting remote slave information (e.g., position, velocity, and force) to the master robot in what is called a bilateral connection. Unfortunately, bilateral configurations can potentially yield a teleoperation system unstable due to delays [27] and data losses [28] experienced in the communication network [6].

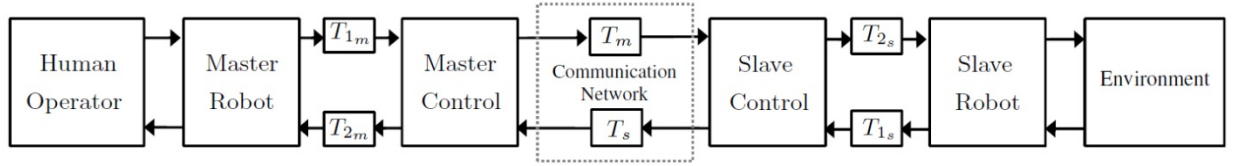


FIGURE 4.1 SCHEME OF A BILATERAL TELEOPERATION SYSTEM WITH LOCAL DELAYS T_1 AND T_2 , AND INTERCONNECTION DELAYS T_m AND T_s

A bilateral teleoperator is a NCS with a *human-in-the-loop*. Therefore, network-induced delays associated with NCSs are also of concern in a bilateral teleoperator (see Figure 4.1). However, in a teleoperation system, time delays in the master's and slave's local control loop are typically less significant than interconnection delays between the local and remote site (i.e., $T_{ji} \ll T_i$ for $i \in \{m, s\}, j \in \{1, 2\}$). Thereby, it is standard to ignore local delays (i.e., $T_1 = T_2 = 0$) and consider only the presence of interconnection delays between master and slave. From now on, we will follow this convention when addressing bilateral teleoperators.

From section 3.5 Round-trip consistency, it was deduced that the delay of the packet being transmitted and received were symmetrical and so in this case $T_m \approx T_s$.

4.2 BILATERAL CONTROL DEVICES

4.2.1 THE HAPTIC DEVICE

The haptic device, used as the master robot and illustrated in Figure 4.2, is the commercially available PHANTOM by SensAble Technologies, Inc. with 6-DOF positional and rotational input and 6-DOF force and torque output. Position and velocity commands to/from the slave agents are relative to the base of the PHANTOM's end-effector and are properly scaled to match with the mobility range of the haptic device. Rotational movements around the base of the end-effector are ignored, leaving the Cartesian coordinates, x , y , and z as the only controllable DOF.

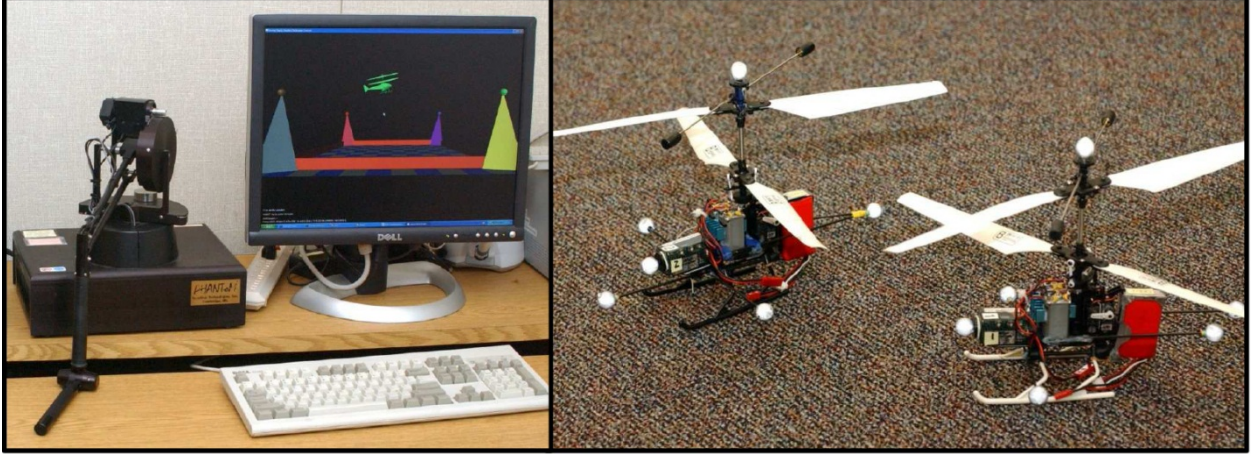


FIGURE 4.2 MASTER AND SLAVE AGENTS. THE LEFT AND RIGHT PHOTOS ILLUSTRATE THE PHANTOM HAPTIC DEVICE AND THE COAXIAL HELICOPTERS, RESPECTIVELY. COPYRIGHT © 2010 BOEING. ALL RIGHTS RESERVED.

4.2.2 COAXIAL HELICOPTERS

The slave agents, shown in Figure 4.2, are two modified E-Flite Blade CX2 coaxial helicopters with multiple spherical retro-reflective markers for identification/localization purpose and cover removed to lower weight. Each vehicle weighs 220g and measures 340mm of rotor diameter. We assume that the CX2 helicopters have three controllable DOF corresponding to Cartesian x , y , and z motion, while control of the yaw ψ angle is ignored. A pictorial representation of the relation between the rotational angles and the Cartesian coordinates is given in Figure 4.3.

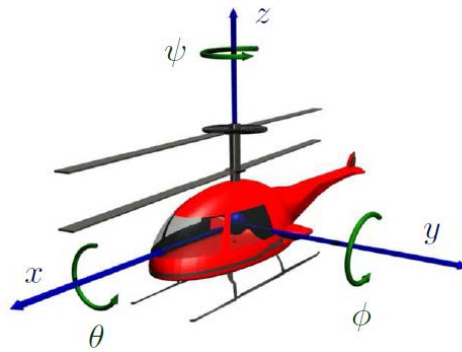


FIGURE 4.3 ROTATIONAL ANGLES WITH RESPECT TO CARTESIAN COORDINATES

4.2.3 MOCAP SYSTEM

Position tracking of the helicopters is performed off-board, meaning that the helicopters lack of self-contained position and velocity sensors. Instead, the test bed employs a MoCap system [29] that consists of multiple high speed cameras located in the remote environment and capable of tracking position and orientation of the slave agents in real-time by collecting two-dimensional visual data and constructing a three-dimensional representation through a photogrammetry-based technique. The cameras are able to sense and track unique configuration patterns of retro-reflective markers placed on the tracked vehicle or obstacle with sub-millimeter accuracy at a sampling rate of 120Hz. The position and orientation of all vehicles and obstacles are then transmitted to each helicopter's control computer within less than 10ms such that every vehicle knows its own location and the location of nearby obstacles. Velocities of the agents are then computed locally by differentiation.

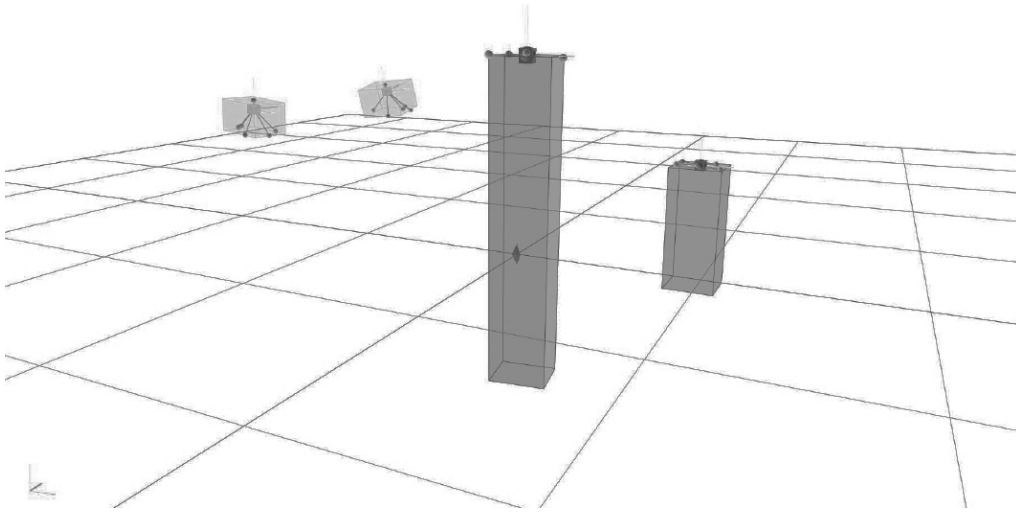


FIGURE 4.4 A 3-D IMAGE GENERATED BY THE MOCAP SYSTEM USING VICON IQ2.5 GRAPHICAL DISPLAY. THE SMALL PURPLE CUBES REPRESENT THE POSITION AND ORIENTATION OF THE TWO HELICOPTERS WHILE THE TWO GRAY RECTANGULAR PRISMS REPRESENT THE POSITION OF THE OBSTACLES. COPYRIGHT © 2010 BOEING. ALL RIGHTS RESERVED.

4.2.4 COMMUNICATION

Communication between agents and haptic device is achieved through the multithreaded TCP socket connection established by the communication software discussed in Chapter 6 Pragmatic Design. Each agent transmits its Cartesian coordinates and velocities (q_i, \dot{q}_i) to the virtual environment via the pipeline command within linux and receives from the virtual environment the coordinates and velocities of the virtual helicopter (q_v, \dot{q}_v) and the corresponding offset for the desired formation. The pipeline simply links a set of processes chained by their standard streams, so that the output of each process (stdout) feeds directly as input (stdin) to the next one. Using this setup the server/client's output from the internet stream would feed directly into the Master/Slave controller while using the program's multithreaded design the opposite is done using the same program via pipeline to channel the Master/Slave controller's output as input into the communication software which in turn feeds the information to the remote side to be received by the Server/ Client. Although the above were all tested, this actual setup was never tested with the distant remote client/server relationship with the SMMS (Single Master Multi Slave). However the communication software was tested separately using rigorous cases resembling that of the communication parameters of the current setup (discussed in Chapter 6 Pragmatic Design).

4.3 NETWORK DELAY COMPENSATION USING PLAY-BACK BUFFER

Play-back buffers were originally designed for multimedia play-back [14]. In [30], Liberatore proposed an algorithm to integrate a play-back buffer with networked control for the control algorithm, actuator, and sensor. The main feature is a buffer located at the actuator which delays the application of a control signal until a specified play-back time is reached. The play-back time is determined at the controller and is paired with the appropriate control signal in a single packet. Control signals which arrive after the play-back time are applied immediately.

While network delays are usually modeled by a semi-infinite, heavy-tailed distribution defined on $[\tau_{min}, +\infty)$, we begin by studying the case when all uncertainty in the loop delay can be removed. Therefore, according to [31] we generate random delays using a bounded-interval distribution defined on $[\tau_{min}, \tau_{max}]$. We use the beta distribution, whose *probability density function* (PDF) on the interval $[\tau_{min}, \tau_{max}]$ is given by:

$$f(x) = \frac{\left(\frac{x - \tau_{min}}{\tau_{max} - \tau_{min}}\right)^{\alpha-1} \left(1 - \left(\frac{x - \tau_{min}}{\tau_{max} - \tau_{min}}\right)\right)^{\beta-1}}{(\tau_{max} - \tau_{min}) \int_0^1 u^{\alpha-1} (1-u)^{\beta-1} du} \quad (4.1)$$

It should be noted that this distribution was not chosen because it specifically matched any real network data. However, we chose this distribution for qualitative reasons: in most real network delay distributions, most delays will be close to the minimum delay and the system will be subject to less frequent, long delay spikes, this can also be confirmed that the average delay is closer to the minimum delay than the maximum delay. If we set $\alpha = 1$, as β increases from 1 to ∞ , the beta distribution shifts from uniform to an impulse at the minimum value. Therefore, as β increases and the play-back delay stays at τ_{max} .

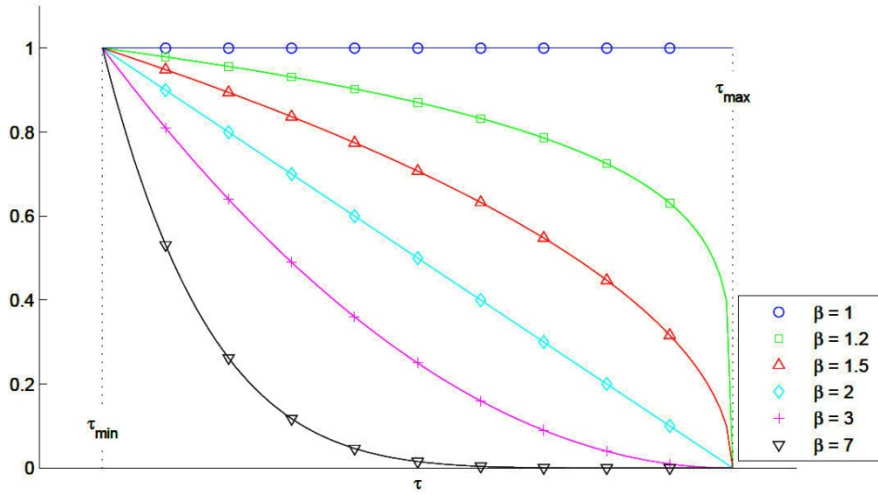


FIGURE 4.5 NORMALIZED PDFS FOR BETA DISTRIBUTION FOR SEVERAL β AND $\alpha = 1$

Here, we study varying values of β , τ_{min} , and τ_{range} , where $\tau_{range} = \tau_{max} - \tau_{min}$, and we always use $\alpha = 1$. We start with $\beta = 1$, which is the case of uniformly distributed delays, and increase β to 7. From previous site survey results the value of the minimum delay was 105ms, so we consider values of the minimum delay in that neighborhood. Specifically, we consider $\tau \in [105, 192]ms$.

The playback delay τ_{pb} can be naively selected as $\tau_{pb} = \tau_{max}$ however performance degrading may occur with high delay. Thus to optimize this selection by design we can obtain a β which provides the lowest τ_{pb} distribution, in other words β can be tuned to result in

the best playback delay which will result in the smallest constant delay representation of the network delay transparent to the controller.

From analytical study and equation (3.1) Figure 4.5 can be deduced which shows the effects of tuning β for obtaining nominal τ_{pb} .

CHAPTER 5 NETWORK ARCHITECTURE

5.1 UNDERSTANDING NETWORKING ARCHITECTURE

Network architecture is the design of a communications network. It is a framework for the specification of a network's physical components and their functional organization and configuration, its operational principles and procedures, as well as data formats used in its operation. The Open Systems Interconnection model (OSI model) is a product of the Open Systems Interconnection effort at the International Organization for Standardization. It is a way of sub-dividing a communications system into smaller parts called layers. A layer is a collection of similar functions that provide services to the layer above it and receives services from the layer below it. On each layer, an instance provides services to the instances at the layer above and requests service from the layer below. Figure 5.1 describes this hierarchy:

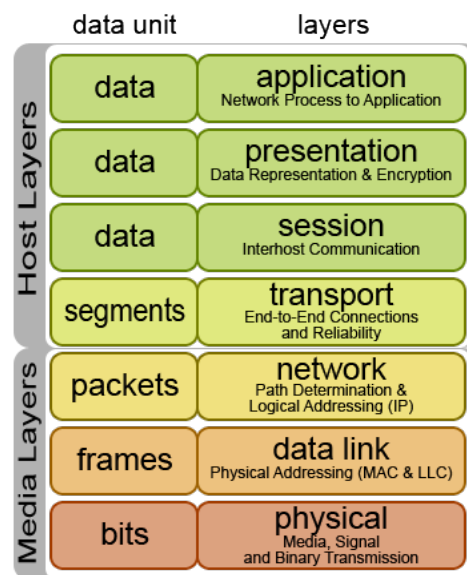


FIGURE 5.1 OSI MODEL

In order to establish a connection from the Master control to the slave control for the MRRC a multithreaded program will be utilized which is better described in Chapter 6 Pragmatic Design. This “communication driver” program is located in the application layer of the host layer within the OSI model. This program can establish a connection to another workstation using similar program using all the layers and can define attributes to these layers. One of the layers affected by this method of connection establishment is the transport layer.

The Transport Layer provides transparent transfer of data between end users, providing reliable data transfer services to the upper layers. The Transport Layer controls the reliability of a given link through flow control, segmentation/desegmentation, and error control. Some protocols are state and connection oriented. This means that the Transport Layer can keep track of the segments and retransmit those that fail. The Transport layer also provides the acknowledgement of the successful data transmission and sends the next data if no errors occurred. Some Transport Layer protocols, for example TCP, but not UDP, support virtual circuits provide connection oriented communication over an underlying packet oriented datagram network. Where it assures the delivery of packets in the order in which they were sent and assure that they are free of errors. The datagram transport delivers the packets randomly and broadcast it to multiple nodes¹⁰.

¹⁰ The transport layer multiplexes several streams on to 1 physical channel. The transport headers tell which message belongs to which connection

5.2 TRANSMISSION CONTROL PROTOCOL (TCP)

TCP is the protocol that major Internet applications rely on, applications such as the World Wide Web, e-mail, and file transfer. Other applications, which do not require reliable data stream service, may use the User Datagram Protocol (UDP) which provides a datagram service that emphasizes reduced latency over reliability [32]. TCP provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer. Several key features of the TCP are noted below.

5.2.1 NETWORK FUNCTION

TCP provides a communication service at an intermediate level between an application program and the Internet Protocol (IP). That is, when an application program desires to send a large chunk of data across the Internet using IP, instead of breaking the data into IP-sized pieces and issuing a series of IP requests, the software can issue a single request to TCP and let TCP handle the IP details.

IP works by exchanging pieces of information called packets. A packet is a sequence of octets and consists of a header followed by a body. The header describes the packet's destination and, optionally, the routers to use for forwarding until it arrives at its destination. The body contains the data IP is transmitting.

Due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets can be lost, duplicated, or delivered out of order. TCP detects these problems, requests retransmission of lost data, rearranges out-of-order data, and even helps minimize network congestion to reduce the occurrence of the other problems. Once the TCP receiver has reassembled the sequence of octets originally transmitted, it passes them to the application program. Thus, TCP abstracts the application's communication from the underlying networking details.

5.2.2 TCP SEGMENT STRUCTURE

TCP segment consists of a segment header and a data section. The TCP header contains 10 mandatory fields, and an optional extension field (Options, pink background in table) [33].

The data section follows the header. Its contents are the payload data carried for the application. The length of the data section is not specified in the TCP segment header. It can be calculated by subtracting the combined length of the TCP header and the encapsulating IP segment header from the total IP segment length (specified in the IP segment header).

5.2.3 PROTOCOL OPERATION

TCP protocol operations may be divided into three phases. Connections must be properly established in a multi-step handshake process (connection establishment) before entering the data transfer phase. After data transmission is completed, the connection termination closes established virtual circuits and releases all allocated resources.

A TCP connection is managed by an operating system through a programming interface that represents the local endpoint for communications, the Internet socket. During the lifetime of a TCP connection it undergoes a series of state changes [33]:

1. LISTEN : waiting for a connection request from any remote client.
2. SYN-SENT : waiting for the remote peer to send back a TCP segment
3. SYN-RECEIVED : waiting for the remote peer to send back an acknowledgment
4. ESTABLISHED : the port is ready to receive/send data from/to the remote peer.
5. FIN-WAIT-1
6. FIN-WAIT-2
7. CLOSE-WAIT
8. CLOSING
9. LAST-ACK
10. TIME-WAIT : represents waiting for enough time to pass to be sure the remote peer received the acknowledgment of its connection termination request.
11. CLOSED

5.2.4 CONNECTION ESTABLISHMENT

To establish a connection, TCP uses a three-way handshake. Before a client attempts to connect with a server, the server must first bind to a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may initiate an active open. To establish a connection, the three-way (or 3-step) handshake occurs:

SYN: The active open is performed by the client sending a SYN to the server. It sets the segment's sequence number to a random value A .

SYN-ACK: In response, the server replies with a SYN-ACK. The acknowledgment number is set to one more than the received sequence number ($A + 1$), and the sequence number that the server chooses for the packet is another random number, B .

ACK: Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value i.e. $A + 1$, and the acknowledgement number is set to one more than the received sequence number i.e. $B + 1$.

At this point, both the client and server have received an acknowledgment of the connection.

5.2.5 RELIABLE TRANSMISSION

TCP uses a sequence number to identify each byte of data. The sequence number identifies the order of the bytes sent from each computer so that the data can be reconstructed in order, regardless of any fragmentation, disordering, or packet loss that may occur during transmission. For every payload byte transmitted the sequence number must be incremented. In the first two steps of the 3-way handshake, both computers exchange an initial sequence number (ISN). This number can be arbitrary, and should in fact be unpredictable to defend against TCP Sequence Prediction Attacks. If the sender infers that data has been lost in the network, it retransmits the data, thus ensuring a reliable transmission of data within sequence.

5.2.6 VULNERABILITIES

Not all protocols are made perfect and they all contain some sort of security hole within their architecture that can allow unwanted access to packets being transferred. Namely two different types of vulnerabilities within the TCP architecture will be considered.

5.2.6.1 DENIAL OF SERVICE

By using a spoofed IP address and repeatedly sending purposely assembled SYN packets, attackers can cause the server to consume large amounts of resources keeping track of the bogus connections. This is known as a SYN flood attack. Proposed solutions to this problem include SYN cookies and Cryptographic puzzles. Overcoming these vulnerabilities will be discussed in Chapter 6 Pragmatic Design [34].

5.2.6.2 CONNECTION HIJACKING

An attacker who is able to eavesdrop a TCP session and redirect packets can hijack a TCP connection. The attacker has to guess correctly the sequence number to be used by the sending host. If they can do this, they will be able to send counterfeit packets to the receiving host which will seem to originate from the sending host, even though the counterfeit packets may in fact originate from some third host controlled by the attacker [35].

5.3 USER DATAGRAM PROTOCOL

The User Datagram Protocol (UDP) is one of the core members of the Internet Protocol Suite, the set of network protocols used for the Internet. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without requiring prior communications to set up special transmission channels or data paths [36].

UDP uses a simple transmission model without implicit handshaking dialogues for providing reliability, ordering, or data integrity. Thus, UDP provides an unreliable service and datagrams may arrive out of order, appear duplicated, or go missing without notice. UDP assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system.[1] If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

UDP's stateless nature is also useful for servers answering small queries from huge numbers of clients. Unlike TCP, UDP is compatible with packet broadcast (sending to all on local network) and multicasting (send to all subscribers) [25].

5.3.1 RELIABILITY AND CONGESTION CONTROL

Lacking reliability, UDP applications must generally be willing to accept some loss, errors or duplication. Some applications such may add rudimentary reliability mechanisms into the application layer as needed [25]. Most often, UDP applications do not employ reliability mechanisms and may even be hindered by them. Potentially more seriously, unlike TCP, UDP based applications don't necessarily have good congestion avoidance and control mechanisms. Congestion insensitive UDP applications that consume a large fraction of available bandwidth could endanger the stability of the internet, as they frequently give a bandwidth load that is inelastic.

5.3.2 VULNERABILITIES

UDP is very much vulnerable to a form of denial-of-service (DOS) attack using UDP flood attack. Using UDP for denial-of-service attacks is not as straightforward as with the Transmission Control Protocol (TCP). However, a UDP flood attack can be initiated by sending a large number of UDP packets to random ports on a remote host. Thus, for a large number of UDP packets, the victimized system will be forced into sending many ICMP packets, eventually leading it to be unreachable by other clients.

5.4 COMPARISON OF TCP AND UDP

From the previous section's detailed attributes of the two protocols a comparison is now done for analyzing the uses of such protocols for our communication driver program [25].

1. **TCP** (Transmission Control Protocol). TCP is a connection-oriented protocol, a connection can be made from client to server, and from then on any data can be sent along that connection.
 - **Reliable** - when you send a message along a TCP socket, you know it will get there unless the connection fails completely. If it gets lost along the way, the server will re-request the lost part. This means complete integrity, things don't get corrupted.
 - **Ordered** - if you send two messages along a connection, one after the other, you know the first message will get there first. You don't have to worry about data arriving in the wrong order.
 - **Heavyweight** - when the low level parts of the TCP stream arrive in the wrong order, resend requests have to be sent, and all the out of sequence parts have to be put back together, so requires a bit of work to piece together.
 - **Streaming** - Data is read as a "stream," with nothing distinguishing where one packet ends and another begins. There may be multiple packets per read call.
2. **UDP** (User Datagram Protocol). A simpler message-based connectionless¹¹ protocol. With UDP you send messages (packets) across the network in chunks.
 - **Unreliable** - When you send a message, you don't know if it'll get there, it could get lost on the way.
 - **Not ordered** - If you send two messages out, you don't know what order they'll arrive in.
 - **Lightweight** - No ordering of messages, no tracking connections, etc. It's just fire and forget. This means it's a lot quicker, and the network card / OS have to do very little work to translate the data back from the packets.
 - **Datagrams** - Packets are sent individually and are guaranteed to be whole if they arrive. One packet per one read call.

From the above comparison it can be easily deduced that for the nature of the communication being setup TCP would be the better choice since the MRRC requires that no packets are lost, the coordinates from the Master robot to slave robot must be in order (this is especially crucial for flight control). However TCP poses one threat and that is overhead. TCP is a heavyweight protocol and establishing the connection requires hand shaking and may cause degraded performance when delay is required to be minimal.

As a result the communication driver is tested using both protocols for test purposes to see the practicality of either protocol.

¹¹ Connectionless describes communication between two network end points in which a message can be sent from one end point to another without prior arrangement

5.5 SIMULATION

The setup proposed above can then be utilized by setting up TCP and UDP connection programs to be tested with the real-time network. Since these programs contain their own processing thread delay further testing in a smaller scale can be performed prior to setting up client/server between Bellevue, WA and Champaign, IL. These tests can be emulated using modern simulated packet errors within a controlled virtual environment. This environment can emulate TCP errors due to packet collisions while routing thus creating naturally occurring packet drops within networks. These packet drops increase the delay further than expected. The environment used to emulate such behavior used within this research is called Common Open Research Emulator (CORE).

The Common Open Research Emulator (CORE) is a tool that allows us to emulate entire networks on one or more machines. One can connect these emulated networks to live networks or to additional emulated networks. CORE consists of a GUI for easily drawing topologies that drives lightweight virtual machines, and various utilities. CORE uses virtualized network stacks in a patched FreeBSD kernel, or Linux virtual machines.

CORE has been developed by a Network Technology research group that is part of the Boeing Research and Technology division [37]. The Naval Research Laboratory is supporting further development of this open source project.

By setting up the proposed network offline within a control environment like CORE the same node specifications obtained before can be setup within the simulator. All nodes mimic the behavior of modern internet nodes (with delays, packet errors etc.)

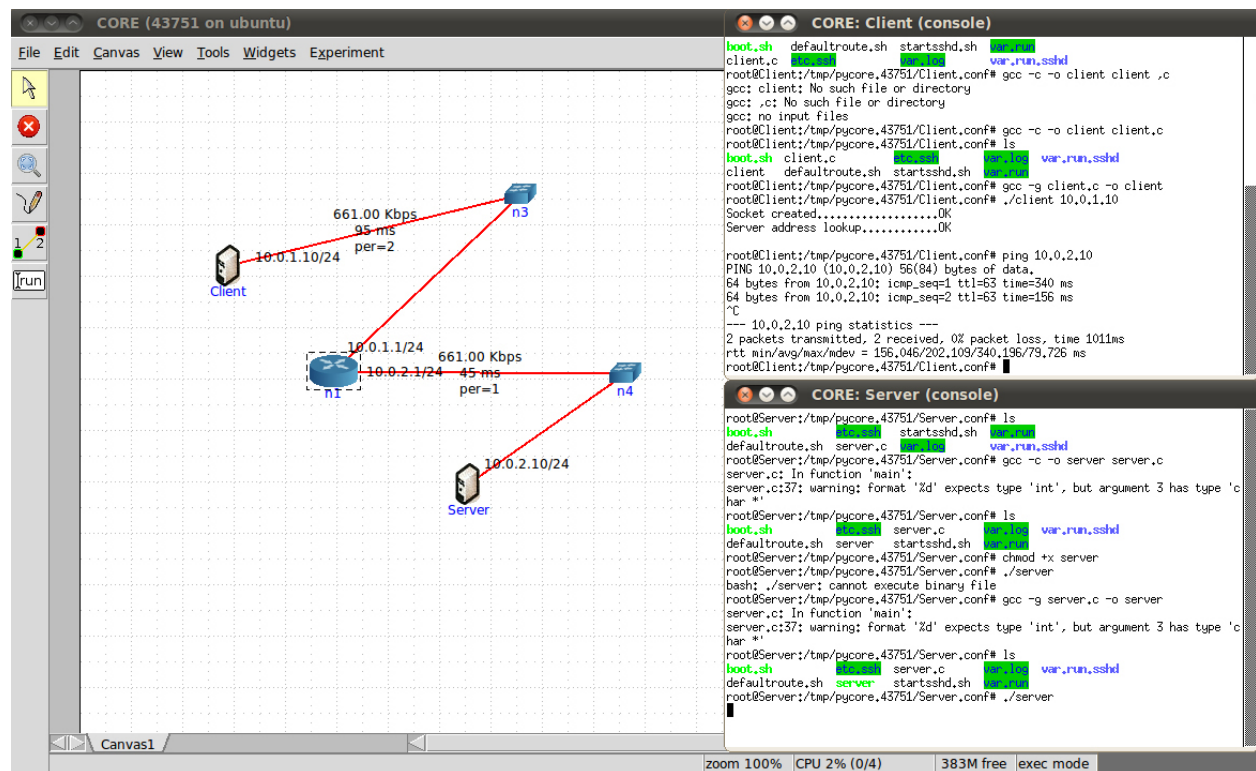


FIGURE 5.2 CORE VIRTUAL SIMULATION

From Figure 5.2 this setup can be seen, although the site survey suggested 18 nodes linking the source and the destination, the overall network behavior can be projected using several nodes. From the above figure it can be seen that the Client's link has network speed of 661Kbps (the same as site survey) which will limit the higher link speeds to that

bandwidth and a delay of 95ms, this delay with the delay of node 4 which has 45ms should provide a 140ms delay between the client and the server. This is how most of this paper's simulation is setup. Once the link specifications are setup the router linking the two hosts is configured to allow network discover of other hosts such that the two hosts are able to communicate. Each node, host, or router can be configured further to allow meet various specifications and provide a variety of network services. This configuration setup is depicted in Figure 5.3.

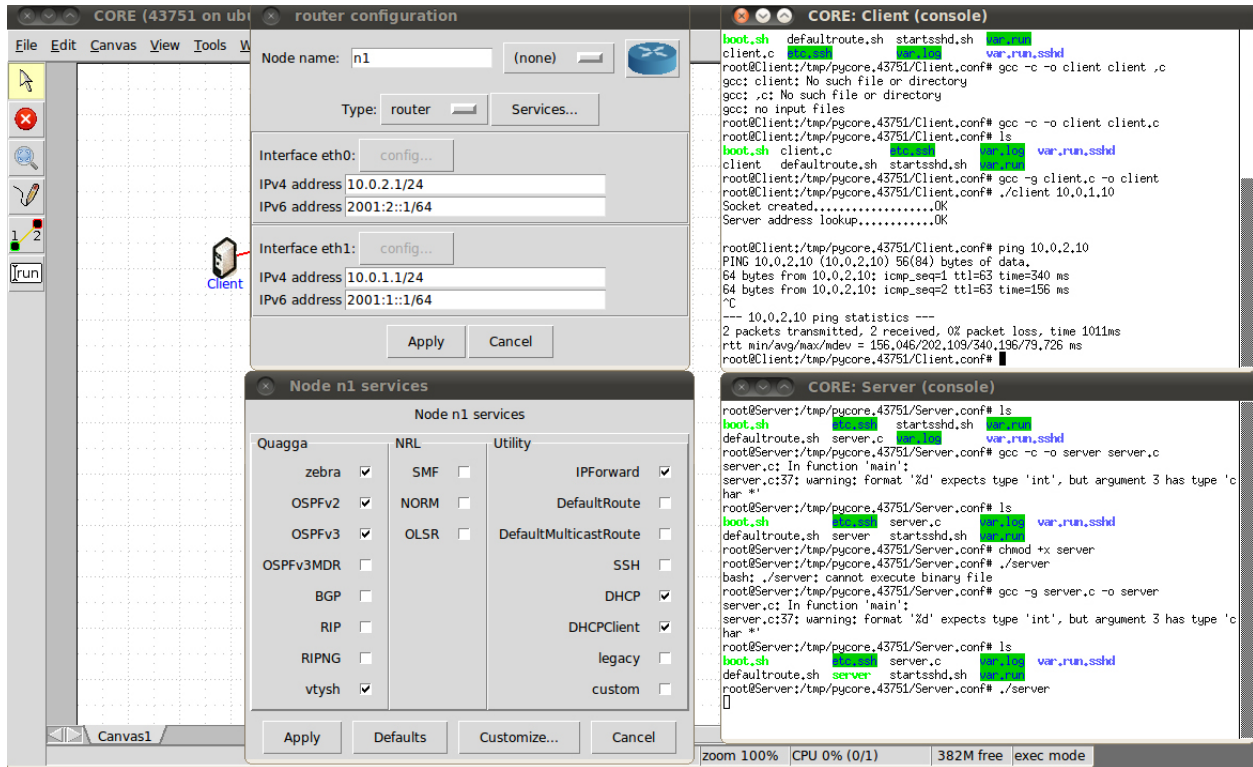


FIGURE 5.3 NODE CONFIGURATION SETUP

Once the overall network is setup the client and server program's discussed in C - TCP Multi-threaded Communication Driver v2.1 section can be loaded on to the hosts. Each host can be accessed via a terminal for command line interfacing and allowing a runtime environment in which the client and server programs will be hosted. As seen in Figure 5.2 the two terminal's on the right side represent the consoles of the client and server, each running their own respective communication driver program to initiate data transfer. This simulation environment is later used to test various iterations of the communication driver development discussed later.

CHAPTER 6 PRAGMATIC DESIGN

6.1 JAVA - UDP REQUESTER/SENDER PROGRAM

Since in Comparison of TCP and UDP from previous section it was deduced that UDP and TCP were to be both tested the pragmatic design began with the design of a simple UDP program that would open a connection on a default port and await a sender program to start sending data to that port. The two programs could be run inversely on the machines such that a requester program and transmitter program can both reside on the same machine thus creating a bidirectional negotiation. The program utilized the `ServerSocket` Class, which provides easy implementation of Socket programming within Java.

The program worked flawlessly however Java is known to be a slower runtime component than native C programs. As a result the notion of developing the program using Java was rejected by Boeing.

6.2 C - UDP COMMUNICATION DRIVER V1.0

Considering Java's lack of agility the pragmatic design was scratched and rewritten in C. The UDP driver utilized the socket programming libraries of C namely `<sys/socket.h>` library. Using struct based object oriented programming an instance of the connection attempt would be created and the input stream being a file could be flushed into the cache and passed into the socket connection's buffer to be segmented into packets and sent via UDP protocol by simply sending the data packets to the address at the given point.

The operation would work by invoking the following commands on the server side:

```
root@n4:/tmp/pycore.43751/n4.conf> ./receiverprog
```

The receiver program is now running and waiting for any UDP packets. Following that, on the client side the following commands are invoked:

```
root@n2:/tmp/pycore.43751/n2.conf> cat sample | ./senderprog 10.0.0.11
```

Following this command the sample data packet (which can also be a stream input of data from a COM device such as joystick or PHANTOM device) will then be piped into the sender program which then processes the data and sends to the server (receiver program). Once the data is received they can be channeled into the screen (stdout) or simply piped into another COM component that may provide control inputs for a controller or simply helicopter. In this case the data is simply output to the server side's screen and is seen in the following example:

```
A10.227874332631472
B34.5903722471121
C47.8188418440368
A33.826766526272
B88.8618727061839
C47.2337239126318
```

Although the control input data could be in any delimited format however in this we have transmitted roll, pitch, and yaw movements each distinguished by their respected ID and following coordinate. The reason for the ID's are that UDP packets may be lost during transmission thus it is crucial to know which ID the incoming coordinates are referring to.

Following the transmission, the client side confirms that the data has been sent (but cannot confirm their safe arrival):

```
Client-gethostname() is OK...
Client-socket() sockfd is OK...
Using port: 4950
sent 121571 bytes to 10.0.0.11
Client-sockfd successfully closed!
```

However problems began to rise as network performance was degraded within the simulation environment until the attributes were matching that of the modern network link between the two main nodes. As performance degraded more packages were lost as a result the data sent and that of the data being received began to differ. As data loss occurs the specific ID whom did not receive coordinates would be skipped (thus updated with a delay). As a quick fix the previously successfully received data can be replaced for the current skipped coordinates. Once the changes in bandwidth, delay and packet drop percentage were done within the CORE simulation program the problem began to show itself more clearly:

<u>Transmitted</u>	<u>Received</u>
B89.1824117852593	B89.1824117852593
C35.9219708975616	C35.9219708975616
A99.2925314640161	99.2925314640161B
B75.7348415793786	75.7348415793786C
C89.0716297401028	89.0716297401028A
A28.3513214918003	.3513214918003B57
B57.9651240266685	9651240266685C32.
C32.4015765606418	4017656648A44.026
A44.0265127896055	512789605B82.4490

FIGURE 6.1 DATA LOSS DUE TO UDP (LEFT DATA IS THE TRANSMITTED DATA, RIGHT: DATA RECEIVED. SHADED DATA INDICATE MISSING DATA WITHIN THE RECEIVED END)

By comparing the above data it can be seen that even within the simulation environment data loss occurred. From 162bytes that were transmitted 6 bytes were not transmitted (these missing characters represented in Figure 6.1 (left) are highlighted). This represents a 3.7% error rate, in which may result in omitting the rest of the bits of one of the axis thus raising the final error rate to 14.8%, this can be confirmed from the 'C' ID's main two digits which resulted in discarding of the whole C coordinates (which refer to the yaw axis). Although the compensator at the client end can try to do error correction however the lost data header for the ID cannot be recovered if the first digits following the ID (which correspond to the most significant digits of the coordinates) are missing. As a result the particular axis coordinates needs to be skipped. This is only the simplest example, one of the worse cases is a combination of these errors, especially one with missing delimiter like '.' Once this happens there is no way of knowing where the data begin and where it ended. This can be seen from the above data and that if this continues to occur, a data skew may occur for a particular axis which can jeopardize the robustness of the MRRC. Thus it has become evident that UDP is clearly not the right choice for real-time control data transmission. The code for this Java edition of the communication driver is provided in Appendix A – Java UDP Communication driver Code v1.0.

6.3 C - TCP COMMUNICATION DRIVER V2.0

Following the failed attempt of UDP to act as the simple and fast transport layer, focus is now turned to development of a communication driver which takes advantage of the TCP features of ensuring reduced data loss.

Like the other designs this method also involves creating a server and a client to connect to one another. However as it can be noted from the Comparison of TCP and UDP, that TCP connections lack the speed and agility of the UDP transport however they are connection oriented as a result the handshaking process needs to be initiated prior to transmitting data.

Due to this once again the <sys/socket.h> library is preloaded for use with creating a new socket connection. However this time the socket performs a “bind” operation to the listening socket by listening for an incoming connection. Following the connection that the client program requests to the server – using the “connect” directive from <netinet/in.h> library – the client can initiate a connection to the “listening” server that awaits a request as described in Protocol Operation. Following the handshaking process (using SYN-ACK) the connection is then established using a predefined port and now data streaming can begin. Like UDP the data is split into groups and sent off to the server. The server end confirms the safe arrival of every packet and the client proceeds to send more data as they become available either via input or once again a COM device like joystick or PHANTOM device. The implementation of such setup is described in further detail within the Appendix. The particular setup described utilizes port 3000 and a buffer length of 256bytes¹². The buffer length indicates the size of each packet being transmitted.

For test purposes the same sample data used for UDP is once again utilized here to ensure data loss does not occur. The CORE simulation environment is used with the same parameters that were obtained during Chapter 3 Site survey. Thus the delay and bandwidth were setup within CORE between 18 nodes as described in the Simulation section. The input and output are once again compared to ensure no data loss occurred.

At the command line on the server machine the server program is initiated:

```
root@n4:/tmp/pycore.43751/n4.conf> ./server
```

Once the server is up and running the client side can now begin invoking the transmission of sample data:

```
root@n2:/tmp/pycore.43751/n2.conf> cat sample | ./client 10.0.0.11
Socket created.....OK
Server address lookup.....OK
Connection established.....OK
Transmitting...
Close connection.....OK
```

By using the pipe command and the cat sample command the sample data is output to the screen and the output is channeled to the input of the transmitter program (client in this case) which then sends the data off from the socket to the internet address argument¹³. The input/output data are compared below:

<u>Transmitted</u>	<u>Received</u>
C19.8254250244645	C19.8254250244645
A32.3872629433026	A32.3872629433026
B68.0662945334007	B68.0662945334007
C89.1824117852593	C89.1824117852593
A35.9219708975616	A35.9219708975616
B99.2925314640161	B99.2925314640161

FIGURE 6.2 DATA COMPARISON BETWEEN TRANSMITTED AND RECEIVED DATA

¹² The port must be open to the server within the remote host’s router’s port forwarding section

¹³ The internet address used during this experiment is a virtual node address predefined within the CORE simulation

Following TCP's successful data transmission more rigorous tests are conducted. The CORE simulation is then rigged by introducing connection interruptions that cause packets to be dropped. Following the modification the same test as above is done however this time more data is transmitted (231.4KB) and the input/output are compared using the diff¹⁴ command within Linux. The test successfully passes.

Since the MRRC system previous described involves a bidirectional connection, the program can be updated to support such feature demand. Thus instead of invoking the client server programs twice inversely within each workstation such that a client and a server both are running within each workstation to establish a bidirectional connection, one program is designed which can mimic such behavior. This design can also be further expanded to allow multiple connections, thus a SMMS (single master multiple slave) setup can be established to allow multiple receivers to take command from the master. This gives rise to a multithreaded design discussed in the section. The code for this iteration is provided in Appendix B – C TCP Communication driver Code V2.0

6.4 C - TCP MULTI-THREADED COMMUNICATION DRIVER V2.1

To further build on the notion of multithreaded operation to provide bidirectional support as well as dealing with reading and writing to various types of standard inputs the previous version will used as a starting point for socket connections however with added hierarchy of parent and child processes. This can be achieved using “forking”. In computing, when a process forks, it creates a copy of itself. More generally, a fork in a multithreading environment means that a thread of execution is duplicated, creating a child thread from the parent thread. Under Unix and Unix-like operating systems, the parent and the child processes can tell each other apart by examining the return value of the fork() system call. In the child process, the return value of fork() is 0, whereas the return value in the parent process is the PID (Process Identifier) of the newly-created child process.

The fork operation creates a separate address space for the child. The child process has an exact copy of all the memory segments of the parent process, though if copy-on-write semantics are implemented actual physical memory may not be assigned (i.e., both processes may share the same physical memory segments for a while). Both the parent and child processes possess the same code segments, but execute independently of each other.

Using this tool we can build on the Bilateral Teleoperation design that was discussed earlier. By creating child processes via forking we can achieve a multithreaded program which can do several tasks in parallel. As a result the previous notion of client/server changes in the sense that they are only client and server by name, the programs are quite identical since they are now multithreaded and can now both act as “transceivers”, In other words they can both transmit and receive at the same time.

The master robot which has the human operator providing input control via a haptic device such as Joystick or Phantom device can provide input via a serial COM device to the MRRC controller application [6] (developed in Python) which also receives feedback from the actual slave robot's position and generates a control input to be sent off to the slave robot via the multithreaded client application. The multithreaded client application (in this case the client communication driver) creates child processes both to handle the MRRC controller's request to transmit coordinates to the slave robot (helicopter) as well as receive robot position information from the slave robot (Vicon system) and channel it back to the MRRC controller.

On the Slave side the multithreaded server applications receives helicopter coordinates via its parent process and channels the given data and places it into a buffer to be sent to the helicopter which is interfaced via a serial COM devices (such as USB). Here the buffer can also be extended via Network delay compensation using play-back buffer to stabilize the communication delay for better MRRC performance. Furthermore the multithreaded server application obtains Vicon's position information about the helicopter and sends it back to the Master robot. This modular design is depicted in Figure 6.2.

¹⁴ Diff compares two files for any difference among them and displays the differences between the two files

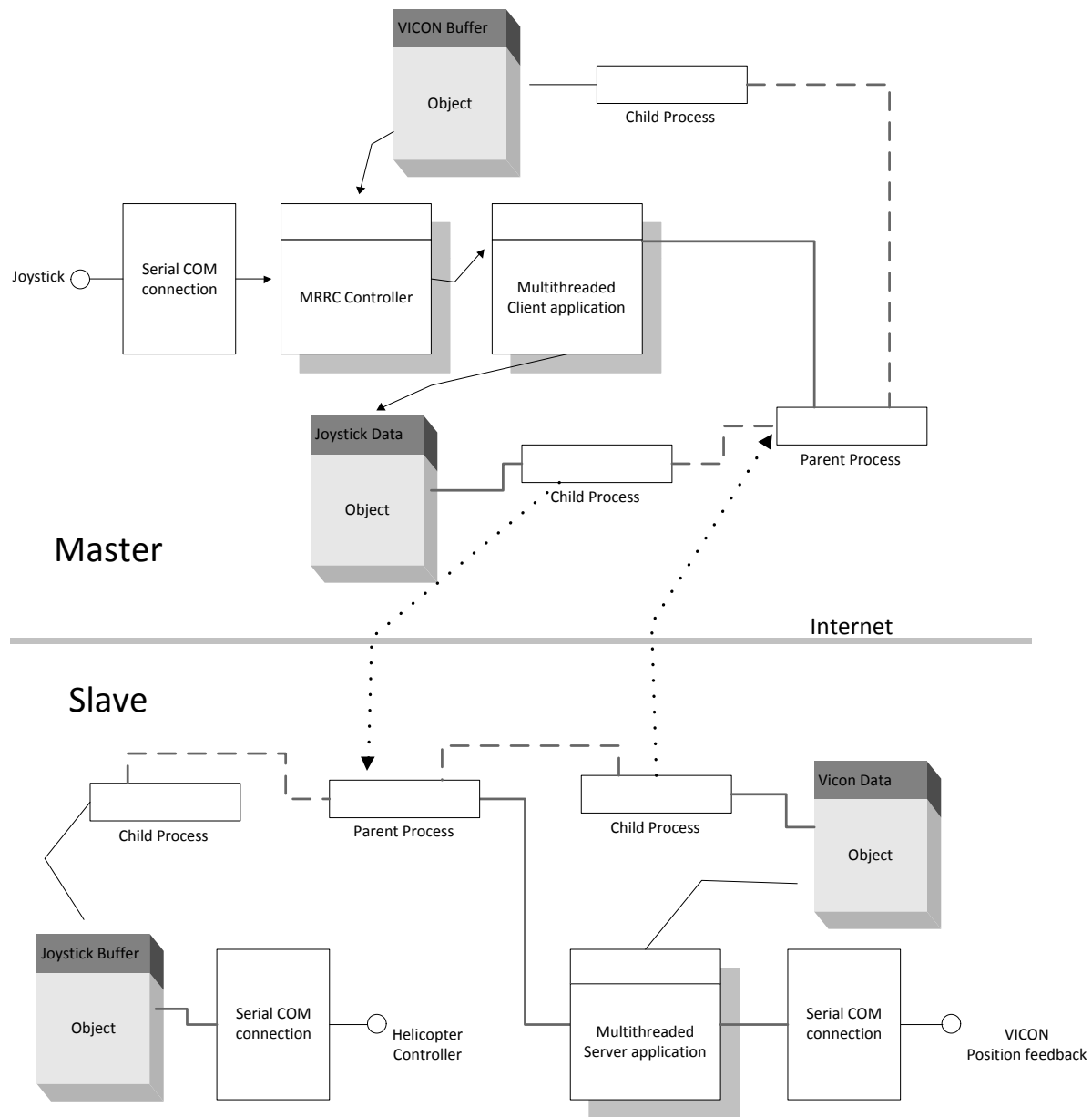


FIGURE 6.3 MULTI-THREADED ARCHITECTURE

Subsequent to the implementation of the multithreaded client and server program comes the test phase. The CORE simulation is used once again to host the server and client programs each sitting at opposite nodes. The link between the two nodes is configured to match the parameters of the Site survey parameters. At first stage a transmission stress test is applied to ensure data loss does not occur over a long period of time. Using a lorem-ipsium generator¹⁵ written in python, random text is generated on the fly and piped to the client application to be sent off to the server. A small sample of the data sent is displayed below which is generated from command line.

¹⁵ Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC

Lorem Ipsum Stress test output:

```
sam@savage:~/boeing/TCP/Threaded> python lorem.py -n 50
lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod
tempor invidunt ut labore et dolore magna aliquyam erat sed diam voluptua at
vero eos et accusam et justo duo dolores et ea rebum stet clita kasd gubergren
no sea takimata sanctus est lorem ipsum dolor sit amet
```

The parameters passed to the python program simply indicate to generate 50 random words, they can then be piped to the transmitter to send the data over the internet to the receiver.

6.5 RESULTS

Using the latter version of the communication driver the first test was conducted for 1,000,000 lines of lorem ipsum and was transmitted to the remote site within CORE using the same site survey configurations. This generated 67.7MB of data which took 17min with a 661Kbps connection that drops 92% of the packets. Using diff the results were compared with the original and there was 0% error.

Furthermore bidirectional communication was tested using single master multiple slave mode (SMMS). This required that there be multiple clients and a single server program to capture all their commands.

The server is invoked by:

```
root@n4:/tmp/pycore.43751/n4.conf> ./server
```

The first client was invoked by sending sample data containing coordinates for A, B, and C helicopter axes.

```
root@n2:/tmp/pycore.43751/n2.conf> cat sample1 | ./client 10.0.0.11
Socket created.....OK
Server address lookup.....OK
Connection established.....OK
Transmitting...
```

The second client was invoked by sending sample data containing coordinates for D, E, and F helicopter axes.

```
root@n3:/tmp/pycore.43751/n3.conf > cat sample2 | ./client 10.0.0.11
Socket created.....OK
Server address lookup.....OK
Connection established.....OK
Transmitting...
```

The resulting output from the server is:

```
C82.2341028803748
F48.97880441913
A66.7557111170936
G0.227874332631472
B34.5903722471121
E47.8188418440368
C86.6896021832872
F33.826766526272
A88.8618727061839
G47.2337239126318
B31.9069719336767
```

By obtaining a sample output from the output it can be seen that the client is outputting one line from each client. While client1 is sending coordinates in the form of ABC, client2 is sending coordinates in the form of DEF, however the output shows an alternative selection of the two (i.e. A_B_C + G_E_F = AGBECF). The reason for the skew in data

could be either due to several packet retransmissions due to deliberate failure which was programmed into the test environment and/or also the fact that the two programs began transmitting coordinates at separate times. The code for this version is provided in Appendix C – C TCP Multi-Threaded Communication driver Code.

6.6 ROBUSTNESS AND SECURITY

As mentioned earlier in the Vulnerabilities section, there exist several ways of eavesdropping on packets and retrieving data. Furthermore the connections being established are simply open to anyone who's willing to send the right data on the right port. If the user is able to find which port the program is transmitting on they can create their own version of the transmitter and begin sending coordinates to the remote robot and control the device (in this case helicopter). Although a user authentication can be implemented within the C program, any authentication validation being sent via the internet is sent openly and the username and password can once again be sniffed by the eavesdropper. As a result a heavier more unbreakable form of authentication and encryption is required.

Before approaching methods of encryption and authentication, there exist several network layer tools that not only provide such aids but also increase robustness by providing another level of data redundancy level by using compression techniques for transmitting data which can slightly reduce the amount of data being transmitted. However since these network layer tools provide encryption and authentication they increase the overhead of the overall transmission and are thus considered as speed attenuators. Nevertheless if the modes of operation of such tools are configured properly in an optimized manner the overhead add-on will be transparent to the overall performance.

To introduce security into the transmission, the data being transmitted will be encrypted during transmission and decrypted upon arrival by using IPsec. Internet Protocol Security (IPsec) is a protocol suite for securing Internet Protocol (IP) communications by authenticating and encrypting each IP packet of a communication session. IPsec also includes protocols for establishing mutual authentication between agents at the beginning of the session and negotiation of cryptographic keys to be used during the session. IPsec is an end-to-end security scheme operating in the Internet Layer of the Internet Protocol Suite. It can be used in protecting data flows between a pair of hosts (host-to-host), between a pair of security gateways (network-to-network), or between a security gateway and a host (network-to-host) [38]. Although other internet security systems exist, such as Secure Sockets Layer (SSL), Transport Layer Security (TLS) and Secure Shell (SSH), Boeing's high security does not allow certain port forwards and thus a more transparent method will be considered.

To increase security either Boeing's local PPTP (Point-to-Point tunneling protocol) can be used or an IPsec setup could be implemented. In this chapter since PPTP is already setup and since this project could be utilized for high security channels IPSEC will be emphasized.

The security architecture of IPsec involves three stages. The first is the Authentication Header (AH) which provides connectionless integrity for IP datagrams and provides protection against replay attacks¹⁶, The next is the Encapsulating Security Payloads (ESP) which provides confidentiality which protects the origins and destination of packets. Finally the last stage is the Security Associations (SA) which provides the bundle of algorithms and data that provide the parameters necessary to operate the AH and/or ESP operations. For better robustness and increased speed a hardware router with IPSEC capability is utilized. In the case of this current setup the same server and client computers can be utilized. So long as both systems are running a Linux architecture the Racoon (IPsec Key management daemon) package can be utilized in combination with OpenSwan (Linux based implementation of IPsec) to provide IPsec support.

Configuring the IPsec tunnel is not as simple as PPTP since all three stages must be configured properly and both systems need to have the same configurations to be in tune and to make IPsec function, however IPsec allows the user more control over the setup of the secure tunnel as well as increased security.

¹⁶ A replay attack is a form of network attack in which a valid data transmission is maliciously or fraudulently repeated or delayed. This is carried out either by the originator or by an adversary who intercepts the data and retransmits it, possibly as part of a masquerade attack by IP packet substitution (such as stream cipher attack).

The main theoretical and understanding of IPsec will not be covered within this section however a brief configuration setup procedure used to obtain a secure tunnel is discussed below. In order to setup the tunnel the parameters and configurations used come as a pair, thus both configurations are identical except the host address in which the tunnel will connect to will vary between the server and the client. On each side the following settings are in place within the OpenSwan configuration file:

Server IPsec Config

```
Mode: Tunnel
Interface: WAN
DPD interval: 60sec
Local subnet: LAN subnet
Remote subnet: 192.168.1.0/24
Remote Gateway: 192.168.20.0
Description: Connect to Boeing
Phase 1 Proposal (authentication)
-----
Negotiation: Aggressive
My Identifier: 192.168.20.0
Encryption algorithm: 3DES
Hash Algorithm: SHA1
DH Key group: 1024
Lifetime: 28800
Authentication method: PSK
Phase 2 proposal (SA/Key exchange)
-----
Encryption algorithm: 3DES
Hash Algorithm: SHA1
PFS Key group: 1024
Lifetime: 3600
```

Client IPsec Config

```
Mode: Tunnel
Interface: WAN
DPD interval: 60sec
Local subnet: LAN subnet
Remote subnet: 192.168.1.0/24
Remote Gateway: 192.168.20.0
Description: Connect to Champaign
Phase 1 Proposal (authentication)
-----
Negotiation: Aggressive
My Identifier: 192.168.20.0
Encryption algorithm: 3DES
Hash Algorithm: SHA1
DH Key group: 1024
Lifetime: 28800
Authentication method: PSK
Phase 2 proposal (SA/Key exchange)
-----
Encryption algorithm: 3DES
Hash Algorithm: SHA1
PFS Key group: 1024
Lifetime: 3600
```

The mode of operation is chosen as tunnel rather than transport mode to help gear the network for host-to-host communication and allow NAT-T (or network address translation traversal which allows port forwarding of routing packets to function properly). The interface being used to connect to the outbound network is WAN (Wide area network) rather than LAN (local area network) since the remote host is not within the local area of the connecting host. DPD interval simply is the time interval to detect Dead Peer Detection interval, which is the time it takes to full reestablish connection. The subnet being used to connect hosts will be the local subnet since the client/server is located within the local subnet of the IPsec server, while the remote gateway points to the address of the remote host being connected to. Aggressive mode of negotiation ensures faster recovery which guarantees that the VPN tunnel rebuilds itself quickly and will not time out an application if the tunnel was down when the resource on the other end was requested. The time out mentioned within the negotiation mode is also preset to 28800 seconds which the default setup time of IPsec tunnels. Both the authentication stage and key exchange stage use military strength 1024bit to encrypt the channel. The encryption algorithm being used is 3DES¹⁷ which is now the world de facto standard. While the hash algorithm uses SHA1¹⁸ for doing checksum on the encryption hashes. While the authentication is done by PSK (pre-shared key) meaning both client and server have a preset key that both match which authenticates the encrypted communication session. Again for the key exchange 3DES algorithm is used to protect the key from being decrypted.

¹⁷ In cryptography, Triple DES (3DES) is the common name for the Triple Data Encryption Algorithm (TDEA or Triple DEA) block cipher, which applies the Data Encryption Standard (DES) cipher algorithm three times to each data block. Because of the availability of increasing computational power, the key size of the original DES cipher was becoming subject to brute force attacks; Triple DES was designed to provide a relatively simple method of increasing the key size of DES to protect against such attacks, without designing a completely new block cipher algorithm.

¹⁸ SHA-1 is a cryptographic hash function designed by the National Security Agency and published by the NIST as a U.S. Federal Information Processing Standard. SHA stands for Secure Hash Algorithm

CONCLUSION

From site survey (Chapter 3) we can obtain an overall picture of the link establishing the connection between the two nodes. This picture can define the network behavior, namely the delay, bandwidth, and consistency of delay. The round trip network delay induced within the link connecting the two nodes is measured to be 191.9ms which falls much under the controller stability threshold of 500ms. Following the controller design and the stability of the delay requirement a protocol can be selected to best suit the application. In this case since data is transmitted as a stream of information (command coordinates and feedback) the TCP protocol can best address the requirements to guarantee no packet loss (Section 5.2).

In terms of pragmatic design the chosen architecture is that of a multithreaded communication system that can simultaneously handle bidirectional communication of command coordinates and feedback information and consult the data in real-time with the model reference robust control system (Section 6.4). To further guarantee the stability of the overall system network delay compensation using play-back buffer (Section 4.3) can be utilized to set a constant upper bound on the network delay to any value T such that $191.9ms < T < 500ms$. As T approaches the upper bound of 500ms the performance is reduced however better system stability is observed.

From the simulation and test results it becomes evident that by using the above structure, a stable communication environment can be constructed to allow bidirectional communication between a human controller and a remote slave robot.

REFERENCES

- [1] W. Zhang, M. S. (2001). Stability of networked control systems. *IEEE control systems magazine* (49), 84-99.
- [2] Williams, T. M. (1995). Pioneering Work in the Field of Computer Process Control. *IEEE Annals of the History of Computing*, 17.
- [3] J. P. Hespanha, P. N. (2007). A survey of recent results in networked control. *proc IEEE*, 95 (1), 138-162.
- [4] Antsaklis, J. B. (2007). Control and communication challenges in networked realtime. *Proc. IEEE*, 95 (1), 9-28.
- [5] M. S. Branicky, S. M. (2000, June). Stability of networked control systems.
- [6] E. J. Rodríguez-Seda, P. O.-M. (2010). Passivity-Based Model Reference Robust Control for a Class of Nonlinear Systems with Input and State Measurement Delays. *American Control Conference*, (pp. 6585-6592). Baltimore.
- [7] Niculescu, S. I. (2001). Delay Effects on Stability: A Robust Control Approach. In M. T. Morari, *Lecture Notes in Control and Information Sciences* (Vol. 269). London.
- [8] C. Abdallah, P. D.-R. (1993, June). Delayed positive feedback can stabilize oscillatory. 3106-3107.
- [9] Davie, L. L. (2000). *Computer Networks*. Morgan Kaufmann.
- [10] Internet Engineering task group. (n.d.). Retrieved March 15, 2011, from The Internet engineering task force (IETF).
- [11] Huston, G. (2009, May 20). Retrieved from IPv4 Address Report, daily generated.
- [12] Curran, J. (2009). *Notice of Internet Protocol version 4 (IPv4) Address Depletio*. Chantilly: American registry for internet numbers.
- [13] Rajiv Ramaswami, K. N. (2009). *Optical Networks: A Practical Perspective*. Morgan Kaufmann.
- [14] Lawrence, D. A. (1993). Stability and transparency in bilateral teleoperation. 9 (5), 624-637.
- [15] Lehman, B. (1994). Stability of chemical reactions in a CSTR with delayed recycle stream. 3521-3522.
- [16] Ansary, J. S. (1972). Stability of differential-difference equations representing heat exchanger and certain other systems. 17 (1), 193-198.
- [17] T. Matiakis, S. H. (2006). Independent-of-delay stability of nonlinear networked. (pp. 2801-2806). Minneapolis: Proc. IEEE Am.
- [18] S. Hirche, T. a. (2009). A distributed controller approach for delay-independent stability of networked control systems. 45 (8), 1828-1836.
- [19] N. Chopra, M. W. (2007). Delay-independent stability for interconnected nonlinear systems with finite L2 gain. (pp. 2847-2852). Proc. IEEE Conf. Decision Control.
- [20] Chopra, N. (2008, Dec). Passivity results for interconnected systems with time delay. 4620-4625.
- [21] G. Niemeyer, J. J. (1991). Stable adaptive teleoperation. 16 (1), 152-162.

- [22] Lyon, G. F. (2009). *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Nmap Project.
- [23] Muir, J. A. (2006). Internet Geolocation and Evasion.
- [24] Heckmann, O. M. (2006). *he Competitive Internet Service Provider: Network Architecture, Interconnection, Traffic Engineering and Network Design*. Wiley-Interscience.
- [25] Forouzan, B. A. (2007). *Data Communications And Networking*. Boston: McGraw-Hill.
- [26] Surhone, L. M. (2007). *Maximum Transmission Unit*. BETASCRIP.T.
- [27] Ferrell, W. R. (1965). Remote manipulation with transmission delay. 6 (1), 24-32.
- [28] C. Secchi, S. S. (2003). Digital passive geometric telemanipulation. (pp. 3290-3295). IEEE.
- [29] J. J. Troy, C. A. (2007). Closed-loop motion capture feedback control. AIAA Infotech@Aerospace.
- [30] Liberatore, V. (2006). Integrated play-back, sensing, and networked control. IEEE InfoCom.
- [31] Graham Alldredge, M. S. (2008). Play-Back Buffers in Networked Control Systems: Evaluation and design.
- [32] Peterson, L. (2003). *Computer Networks*. Morgan Kauffmann.
- [33] RFC 791 - Section 2.1. (n.d.). Retrieved from <http://tools.ietf.org/html/rfc793>
- [34] Bellovin, S. M. (1989). *Security problems in the TCP/IP protocol suite*. Retrieved from <http://portal.acm.org/citation.cfm?id=378444.378449>
- [35] Bellovin, S. M. (1989). RFC 1948. Retrieved from Defending Against Sequence Number Attacks: <http://tools.ietf.org/html/rfc1948>
- [36] J.F. Kurose, K. R. *Computer Networking, 5th ed*. Boston, MA: Pearson Education.
- [37] J. Ahrenholz, C. D. (2008). CORE: A real-time network emulator. *IEEE MILCOM Conference*.
- [38] S. Kent, R. A. (1998). Encapsulating Security Payload (ESP). *Internet Engineering Task Force*.
- [39] Lawrence, D. A. (1993). Stability and transparency in bilateral teleoperation. 9 (5), 624-637.

APPENDIX A – JAVA UDP COMMUNICATION DRIVER CODE V1.0

```
/*
 * File:    senderprog.c
 * Author:  Sam Naghshineh
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

/* the port users will be connecting to */

#define MYPORT 4950

int main(int argc, char *argv[ ]) {

    int sockfd;
    char data;

    /* connector's address information */

    struct sockaddr_in their_addr;
    struct hostent *he;
    int numbytes;

    /*
    if (argc != 3) {
        fprintf(stderr, "Client-Usage: %s <hostname> <message>\n", argv[0]);
        exit(1);
    }*/
    /* get the host info */

    if ((he = gethostbyname(argv[1])) == NULL) {
        perror("Client-gethostbyname() error.");
        exit(1);
    } else
        printf("Client-gethostname() is OK...\n");

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("Client-socket() error.");
        exit(1);
    } else
        printf("Client-socket() sockfd is OK...\n");

    /* host byte order */

    their_addr.sin_family = AF_INET;
    /* short, network byte order */
```



```

printf("Using port: %d\n",MYPORT);
their_addr.sin_port = htons(MYPORT);
their_addr.sin_addr = *((struct in_addr *) he->h_addr);

/* zero the rest of the struct */

memset(&(their_addr.sin_zero), '\0', 8);
while ((data = getchar()) != EOF) {

    if ((numbytes += sendto(sockfd, &data, 1, 0, (struct sockaddr *) & their_addr,
sizeof (struct sockaddr))) == -1) {
        perror("Client-sendto() error lol!");

        exit(1);

    } else
        printf("");
        //printf("Client-sendto() is OK...\n");
}

printf("sent %d bytes to %s\n", numbytes, inet_ntoa(their_addr.sin_addr));

if (close(sockfd) != 0)
    printf("Client-sockfd closing is failed!\n");
else
    printf("Client-sockfd successfully closed!\n");
return 0;
}

```

```

/*
 * File:   receiverprog.c
 * Author: Sam Naghshineh
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/* the port users will be connecting to */
#define MYPORT 4950
#define MAXBUFLen 500

int main(int argc, char *argv[])
{
    int sockfd;
    /* my address information */
    struct sockaddr_in my_addr;

    /* connector's address information */
    struct sockaddr_in their_addr;
    int addr_len, numbytes;
    char buf[MAXBUFLen];

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
    {
        perror("Server-socket() sockfd error.");
        exit(1);
    }
    //else
        //printf("Server-socket() sockfd is OK...\n");

    /* host byte order */

    my_addr.sin_family = AF_INET;
    /* short, network byte order */
    my_addr.sin_port = htons(MYPORT);
    /* automatically fill with my IP */
    my_addr.sin_addr.s_addr = INADDR_ANY;
    /* zero the rest of the struct */

    memset(&(my_addr.sin_zero), '\0', 8);

    if (bind(sockfd, (struct sockaddr *) & my_addr, sizeof (struct sockaddr)) == -1)
    {
        perror("Server-bind() error.");
        exit(1);
    }
    //else
        //printf("Server-bind() is OK...\n");
    addr_len = sizeof (struct sockaddr);
    while (1) {

```

```

        if ((numbytes = recvfrom(sockfd, buf, MAXBUFLen - 1, 0, (struct sockaddr *) &
their_addr, &addr_len)) == -1)
        {
            perror("Server-recvfrom() error.");
            /*If something wrong, just exit lol...*/

            exit(1);
        }
        else
        {

            //printf("Server-Waiting and listening...\n");

            //printf("Server-recvfrom() is OK...\n");

        }

        //printf("Server-Got packet from %s\n", inet_ntoa(their_addr.sin_addr));

        //printf("Server-Packet is %d bytes long\n", numbytes);

        buf[numbytes] = '\0';

        printf("%s", buf);

    }

    if (close(sockfd) != 0)

        printf("Server-sockfd closing failed!\n");

    else

        printf("Server-sockfd successfully closed!\n");

    return 0;
}

```

APPENDIX B – C TCP COMMUNICATION DRIVER CODE V2.0

```
/*
 * File:    server.c
 * Author:  Sam Naghshineh
 *
 */

#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>

int main(int argc, char**argv)
{
    int listenfd,connfd,n;
    struct sockaddr_in servaddr,cliaddr;
    socklen_t clilen;
    pid_t      childpid;
    char mesg[1000];

    listenfd=socket(AF_INET,SOCK_STREAM,0);

    bzero(&servaddr,sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
    servaddr.sin_port=htons(32000);
    bind(listenfd,(struct sockaddr *)&servaddr,sizeof(servaddr));

    listen(listenfd,1024);

    for(;;)
    {
        clilen=sizeof(cliaddr);
        connfd = accept(listenfd,(struct sockaddr *)&cliaddr,&clilen);

        if ((childpid = fork()) == 0)
        {
            close (listenfd);

            while (1)
            {
                n = recvfrom(connfd,mesg,1000,0,(struct sockaddr *)&cliaddr,&clilen);
                //sendto(connfd,mesg,n,0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
                printf("-----\n");
                mesg[n] = 0;
                //printf("Received the following:\n");
                printf("%s",mesg);

                //printf("-----\n");
            }
        }
        close(connfd);
    }
}
```

```

/*
 * File:    client.c
 * Author:  Sam Naghshineh
 *
 */

#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>

int main(int argc, char**argv)
{
    int sockfd,n;
    struct sockaddr_in servaddr,cliaddr;
    char sendline[1000];
    char recvline[1000];

    if (argc != 2)
    {
        printf("usage:  client <IP address>\n");
        exit(1);
    }

    sockfd=socket(AF_INET,SOCK_STREAM,0);

    bzero(&servaddr,sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr=inet_addr(argv[1]);
    servaddr.sin_port=htons(32000);

    connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr));

    while (fgets(sendline, 10000,stdin) != NULL)
    {
        sendto(sockfd,sendline,strlen(sendline),0,(struct
* )&servaddr,sizeof(servaddr));
        //n=recvfrom(sockfd,recvline,10000,0,NULL,NULL);
        //recvline[n]=0;
        //fputs(recvline,stdout);
    }
}

```

APPENDIX C – C TCP MULTI-THREADED COMMUNICATION DRIVER CODE V2.1

```
/*
 * File:    server.c
 * Author:  Sam Naghshineh
 *
 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/signal.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <strings.h>

#define SERVER_TCP_PORT 3000    /* default connection port */
#define BUFLLEN        256    /* buffer length */

int echod(int);
void reaper(int);

int main(int argc, char **argv)
{
    int    sd, new_sd, client_len, port;
    struct sockaddr_in server, client;

    switch(argc){
    case 1:
        port = SERVER_TCP_PORT;
        break;
    case 2:
        port = atoi(argv[1]);
        break;
    default:
        fprintf(stderr, "Usage: %d [port]\n", argv[0]);
        exit(1);
    }

    /* Create a stream socket */
    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        fprintf(stderr, "Can't creat a socket\n");
        exit(1);
    }

    /* Bind an address to the socket */
    bzero((char *)&server, sizeof(struct sockaddr_in));
    server.sin_family = AF_INET;
    server.sin_port = htons(port);
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    if (bind(sd, (struct sockaddr *)&server, sizeof(server)) == -1){
        fprintf(stderr, "Can't bind name to socket\n");
        exit(1);
    }

    /* queue up to 5 connect requests */
    listen(sd, 5);
```

```

(void) signal(SIGCHLD, reaper);

while(1) {
    client_len = sizeof(client);
    new_sd = accept(sd, (struct sockaddr *)&client, &client_len);
    if(new_sd < 0){
        fprintf(stderr, "Can't accept client \n");
        exit(1);
    }
    switch (fork()){
        case 0: /* child */
            (void) close(sd);
            exit(echod(new_sd));
        default: /* parent */
            (void) close(new_sd);
            break;
        case -1:
            fprintf(stderr, "fork: error\n");
    }
}

/* echod program */
int echod(int sd)
{
    char *bp, buf[BUFLEN];
    int n, bytes_to_read, end;
    end = 1;

    while(n = read(sd, buf, BUFLen)){
        write(sd, end, n);
        write(1, buf, n);
    }

    close(sd);

    return(0);
}

/* reaper */
void reaper(int sig)
{
    int status;
    while(wait3(&status, WNOHANG, (struct rusage *)0) >= 0);
}

```

```

/*
 * File:    client.c
 * Author:  Sam Naghshineh
 *
 */
#include <stdio.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <strings.h>

#define SERVER_TCP_PORT 3000    /* default connection port */
#define BUFLLEN        256    /* buffer length */

int main(int argc, char **argv)
{
    int    n, i, bytes_to_read;
    int    sd, port;
    struct hostent      *hp;
    struct sockaddr_in server;
    char    *host, *bp, rbuf[BUFLLEN], sbuf[BUFLLEN];

    switch(argc){
    case 2:
        host = argv[1];
        port = SERVER_TCP_PORT;
        break;
    case 3:
        host = argv[1];
        port = atoi(argv[2]);
        break;
    default:
        fprintf(stderr, "Usage: %s host [port]\n", argv[0]);
        exit(1);
    }

    /* Creating a stream socket */
    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        fprintf(stderr, "Can't create a socket\n");
        exit(1);
    }

    else{
        printf("Socket created.....OK\n");
    }

    bzero((char *)&server, sizeof(struct sockaddr_in));
    server.sin_family = AF_INET;
    server.sin_port = htons(port);
    if (hp = gethostbyname(host)){
        bcopy(hp->h_addr, (char *)&server.sin_addr, hp->h_length);
        printf("Server address lookup.....OK\n");
    }
    else if ( inet_aton(host, (struct in_addr *) &server.sin_addr) ){
        fprintf(stderr, "Can't get server's address\n");
        exit(1);
    }

    /* Connecting to the server */

```



```

if (connect(sd, (struct sockaddr *)&server, sizeof(server)) == -1){
    fprintf(stderr, "Can't connect \n");
    exit(1);
}
else
    printf("Connection established.....OK\n");

printf("Transmit: \n");
while(n=read(0, sbuf, BUFLLEN)){ /* get user message */
    write(sd, sbuf, n);          /* send it out */
    /*printf("Receive: \n");
    bp = rbuf;
    bytes_to_read = n;
    while ((i = read(sd, bp, bytes_to_read)) > 0){
        bp += i;
        bytes_to_read -=i;
    }
    write(1, rbuf, n);
    */
    //printf("Transmit: \n");
}

close(sd);
return(0);
}

```