SUPPORTING REAL-TIME PDA INTERACTION WITH VIRTUAL ENVIRONMENT

by

RADHEY SHAH
B.Tech. Kakatiya University, 2002

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in Modeling and Simulation
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2004

ABSTRACT

Personal Digital Assistants (PDAs) are becoming more and more powerful with advances in technology and are expanding their applications in a variety of fields. This work explores the use of PDAs in Virtual Environments (VE). The goal is to support highly interactive bi-directional user interactions in Virtual Environments in more natural and less cumbersome ways. A proxy-based approach is adopted to support a wide-range of handheld devices and have a multi-PDA interaction with the virtual world. The architecture consists of three components in the complete system, a PDA, a desktop that acts as a proxy and Virtual Environment Software Sandbox (VESS), software developed at the Institute for Simulation and Training (IST). The purpose of the architecture is to enable issuing text and voice commands from PDA to virtual entities in VESS through the proxy. The commands are a pre-defined set of simple words such as ‘move forward’, ‘turn right’, ‘go’, and ‘stop’. These commands are matched at the proxy and sent to VESS as text in XML format. The response from VESS is received at the proxy and forwarded back to the PDA. Performance measures with respect to response time characteristics of text messages between PDA and proxy over Wi-Fi networks are conducted. The results are discussed with respect to the acceptable delays for human perception in order to have real-time interaction between a PDA and an avatar in virtual world.

Dedicated to my wonderful parents, Bhaiya (elder brother) and Bhabhi (sister-in-law), with all my love…

ACKNOWLEDGMENTS

I am grateful to my advisor Dr. Mainak Chatterjee for his continuous guidance and encouragement throughout my thesis studies. I would also like to express my gratitude and special thanks towards Mr. Brian Goldiez at the Institute for Simulation and Training who supported me throughout the duration of my thesis. He is a great person to work with and I learnt a lot from him. I am sure that the knowledge I have gained under Mr. Goldiez and Dr. Chatterjee will help me throughout my professional career. I would also like to thank Dr. Turgut for serving on my thesis committee.

Aside from academic help, I would like to thank my roommates Parikshit, Karthik and Jignesh and my friends Ashish and Shailesh for their encouragement and help.

Finally, I would like to thank my beloved parents, Madhuri and Kiran, my elder brother Shon, my sister-in-law Gauri who have always been a constant source of inspiration and encouragement throughout my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

AHD	American Heritage Dictionary
AI	Artificial Intelligence
AMIE	Annotation Management Interface
AP	Access Point
API	Application Programming Interfaces
CF	Compact Framework
CLR	Common Language Runtime
CLS	Common Language Specifications
CSTA	Computer Supported Telecommunications Applications
DOM	Document Object Model
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPT	Immersive Projection Technology
IST	Institute for Simulation and Training
JAIVE	JAva based Interface to the Virtual Environment
LAN	Local Area Network
MMUI	Multi-Machine User Interfaces
MSS	Microsoft Speech Server

MSS	Maximum Segment Size
NIC	Network Interface Card
OSI	Open System Interconnection
PDA	Personal Digital Assistant
PIE	Pocket Internet Explorer
PVHA	Personal Virtual Human Assistant
RAVES	Research in Augmented Virtual Environment Systems
RF	Radio Frequency
RTT	Round Trip Time
SALT	Speech Application Language Tags
SASDK	Speech Application Software Development Kit
SDK	Software Development Kit
SES	Speech Engine Services
SML	Semantic Markup Language
SRGS	Speech Recognition Grammar Specification
SSML	Speech Synthesis Markup Language
TCP	Transmission Control Protocol
TTS	Text-to-Speech
UCF	University of Central Florida
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VE	Virtual Environments
VESS	Virtual Environment Software Sandbox

VR	Virtual Reality
W3C	World Wide Web Consortium
WLAN	Wireless Local Area Network
WYSIWYG	What You See Is What You Get
XHTML	eXtended Hyper Text Markup Language
XML	eXtensible Markup Language

CHAPTER ONE: INTRODUCTION

1.1 PDA

The Personal Digital Assistant (PDA) is a portable computing device that can include data transmission capabilities. PDAs are one of the fastest selling consumer devices in the history. The Free Online Dictionary of Computing [3] describes PDA as "A small hand-held computer typically providing calendar, contacts, and note-taking applications but may include other applications, for example a web browser and media player. Small keyboards and pen-based input systems are most commonly used for user input." With improvements in technology, these devices make possible services such as paging, data messaging, electronic mail, facsimile, date book and other information handling capabilities like word processing, playing MP3 music files, getting news, stock quotes from internet and playing games. Some of latest PDAs also have integrated digital cameras, GPS receivers and bar-code readers.

PDA's are of interest because they are low cost, multi-modal, of increasing performance, programmable, and are becoming ubiquitous. PDAs also offer portability, social interactivity, context sensitivity, connectivity, and individuality and hence they could be useful tools in Virtual Environments (VEs). PDAs currently have limitations that must be known to ensure that they are optimally used. These limitations have been widely documented [5] and include battery life tradeoffs with software execution and data storage, potential performance issues in computational and graphics intensive settings, and programmability. PDAs also lack sufficient screen space to display complex graphics. A proxy based approach was adopted in order to minimize some of these limitations and have a multi-PDA interface for VEs in the future.

1.2 Virtual Environments

In a broad sense, an interactive computer model that simulates an actual or imaginary world is called a Virtual Environment (VE). Virtual Reality (VR) can be defined [1] as "a medium composed of interactive computer simulations that sense the participant's position and actions and replace or augment the feedback to one or more senses, giving the feeling of being mentally immersed or present in the simulation (a virtual world)". The key features of virtual reality as seen from the definition are: a virtual world, immersion, sensory feedback (responding to user input), and interactivity. The term virtual environment is often used as a synonym for both virtual reality and virtual world. More specifically, a virtual environment is an instance of a virtual world presented in an interactive medium such as virtual reality. Thus, VEs are interactive computer simulations that immerse users in an alternate, yet believable reality. People move around, look at, and manipulate graphics objects using input devices ranging from the commonly accessible keyboard and mouse to more exotic devices, such as head-mounted displays and instrumented gloves.

At the University of Central Florida faculty have been investigating the technical and human aspects of VEs for several years [19]. The research focus has involved integrating existing technologies, such as helmet-mounted displays, and creating new strategies to aid the human user in navigating and interacting with other human users in a VE. Examples have included novel pointing devices and unobtrusive placement of a compass rose in the virtual environment. A variety of input and output devices have been used by UCF, but not with a focus on how the devices can be used as replacements or adjuncts for other sensory input/output devices.

1.3 Use of PDAs in Virtual Environment

With the use of handheld devices becoming common-place, it is natural to investigate integrating these devices in VEs to take advantage of their low cost and user familiarity. This is the motivation for this thesis. The goal is to investigate and prototype a design and optimize a mechanism for efficient, highly interactive multi-modal communications between participants (real and computer generated) in VEs using handheld devices over wireless networks. This work is a part of the Research in Augmented Virtual Environment Systems (RAVES) [4] program, which is a multi-disciplined research program at the University of Central Florida exploring various technological and human factors aspects of multi-modal virtual environments. Handheld devices hold promise in several areas including off-loading the central processing system for selective tasks (e.g., visual processing of maps) and supporting wearable computing for untethered VE traversal. User familiarity with these devices may facilitate better user interaction with virtual environments.

Users employ different devices like head-mounted displays and joysticks to interact with virtual entities. A PDA can be an additional tool to interact with virtual worlds. PDAs are also potentially useful tools for users in the real world, providing improved interactivity with other humans or computer avatars. An avatar is the real-time graphical representation of an identity of the user using the virtual environment. With respect to content, PDA's are low cost and high function communication devices that can also store a variety of information for instant presentation. For instance one can envision the PDA storing maps of facilities and instructions. This type of usage is relevant to military and civilian uses of virtual environments. In training and operational systems for soldiers, for example, handheld devices are used to exchange

information (send and receive messages), share information and retrieves maps as the soldier moves. With other modalities like audio, voice recognition and GPS enabled handhelds; users could perform their actions using a PDA device on the move and in a convenient way. Users could potentially perform their actions hands free, as they could keep the PDA in their pocket and have eye-glasses attached to it to see the screen and a voice recognition device to accept their commands. Alternatively, haptic or auditory information could be conveyed via the device, thereby offloading the visual modality. PDAs can be made to vibrate or flash when certain objectives have been accomplished or when there is danger or in situations where a more conventional notification might not be appropriate. PDAs have a variety of buttons that can be used to indicate alarms or to coordinate activities between soldiers and avatar that cannot see each other.

An Internet or intranet compatible architecture suitable for using PDA's in VEs is an attractive infrastructure for proxy based study, supporting studies of interactivity performance factors, and the subsequent optimization of performance (minimizing network time). It is also intended to support user interactions in VEs in a more natural and less cumbersome way and navigation in an existing VE in real time from PDA (e.g., by providing 'you-are' here maps with auditory information).

The interface for the PDA to the VE is through the Virtual Environment Software Sandbox (VESS) [6]. VESS is a suite of libraries created at the University of Central Florida's Institute for Simulation and Training (UCF IST) [7] that are used to create VE applications. VESS provides an application base that is useful and functional using today's hardware and graphics and audio libraries, extensible to support future hardware and software libraries, and easily

portable to multiple platforms, graphics and audio systems, and application programming interfaces (API's).

This work explores how the PDA can be used in a VE in issuing commands in real-time. A set of phonetically different voice commands for navigation purposes is created which can be used for navigation. The voice commands are created to be sent over from PDA using the Microsoft .NET compact framework, Microsoft Speech Server and a new technology for speech applications called Speech Application Language Tags (SALT). The response time in terms of number of bytes sent over wireless network from PDA to proxy is calculated. The effect of disabling Nagle algorithm is discussed. Nagle algorithm is used by sockets to concatenate small messages to be sent over the network in order to avoid congestion over the network.

The research is presented in four chapters. Chapter 2 provides the literature review of the work that has been done using PDAs in VE. Chapter 3 provides the architecture used and the experimental methodology. Results are presented in chapter 4 whereas chapter 5 discusses the conclusions, limitations and future scope for this work. The terms PDA, Handheld and Pocket PC (a Windows CE device) are used interchangeably.

CHAPTER TWO: RELATED RESEARCH

An informative and forward thinking article by Mark Weiser [9] was one of the early works showing the potential of handheld devices. This work inspiring ubiquitous computing suggested that devices such as PDAs and embedded machines could facilitate moving computing to a background function needed to support ubiquitous computing. Ubiquitous computing could eventually facilitate broader use of virtual environments through miniaturization and higher performance.

When PDAs were introduced into the market, they were thought to have great potential in the computing device industry. Keefe et al. [8] summarizes the histories and findings of various initiatives of PDAs and how they played out in educational settings. Initially PDAs were considered as replacements to desktop PCs or laptops. The focus now is to use PDAs and PCs together when both are available. The use of a PDA as augmentation for PCs in controlling applications like PowerPoint or WinAmp was investigated [10] using Pebbles Project [11]. In the Pebbles architecture, client programs run on one or more PDAs, the server program runs on the PC, and a special program called PebblesPC mediates between clients and servers. User studies on Multi-Machine User Interfaces (MMUI) were performed on different applications which were developed as part of Pebbles project.

The effective use of PDAs for interacting with other heterogeneous devices is made in Magic Lounge [14]. Magic Lounge is a shared virtual meeting environment which has been designed to support meetings between physically remote people who would like to interact with each other using any one of a number of heterogeneous communication devices like PCs, PDAs, palmtops, and mobile telephones.

The idea of using a PDA as an interaction device in a VE is not new. It was first introduced with the Bamboo Project [16] in order to solve the problem of choosing the appropriate interaction techniques among 2-D and 3-D techniques. Bamboo is a multi-platform system supporting real-time, networked, virtual environments. The Bamboo project offered the user an interface which included a camera, an environment, and geometry functionalities. Each functionality was implemented as an applet. Bamboo interacted with the CAVE-like environment via the PDA's wireless serial port. A 3Com PalmPilot was used as PDA and Bamboo's java-based GUI was used for user interface.

The JAIVE [17] (Java based interface to the virtual environment) project has also developed an interaction tool which provided the user with integration of common interaction methods such as selecting colors and performing push-button operations with their Immersive Projection Technology (IPT) applications.

The Virtual Harlem [12] project is an effort to create a learning environment that can enrich students' understanding of the Harlem Renaissance by having a collaborative virtual reality tour of Harlem in which participants can travel back 80 years to see historical figures and hear speeches and music from that period. Virtual Harlem is written using a high-level VR toolkit called Yggdrasil and makes use of CAVE [15] immersive environment. PDAs are used in Virtual Harlem as the Annotation Management Interface (AMIE) to store and retrieve virtual annotations for novice users. Virtual annotations are recordings in VR where both the person's hand and head gestures, as well as their voice are captured. AMIE was developed for iPAQ Pocket PC and wireless LAN is used for connectivity between AMIE and the CAVE. Using the Pocket PC interface, the user can see all the annotations in the space, people who made the annotations, and the time and locations of all the annotations.

QuickSet [13] is a wireless, handheld, agent-based, collaborative, multimodal system for interacting with distributed application. It consists of a collection of “agents” including speech recognition, gesture recognition, natural language understanding, multimodal integration, a map-based interface, and a database, running standalone on the tablet PC or distributed over a network. The system analyses continuous speech and gesture in real time, producing the best joint semantic interpretation for multimodal commands. The multimodal interface runs on machines as small as Windows CE devices, as well as on wearable, handheld, table-sized, and wall-sized displays.

Applications of 3D virtual humans inside mobile devices are discussed at length by Gutierrez et al. [18]. A prototype of a virtual human animation engine compliant with MPEG-4 for mobile devices was developed. A personal virtual human assistant (PVHA) is constructed in form of an autonomous software agent that will look, move, listen and talk as a real person.

The work done so far has been principally for student teaching methodologies or tourist guides, where teaching aids or map-paths in a building appear on the PDA and one progresses spatially by just tapping on appropriate links. As pointed out by Gutierrez et al [18], the ultimate human-computer interface would include audio/video analysis and synthesis in combination with artificial intelligence (AI) techniques, dialog management and face/body gestures to allow an intelligent and expressive dialog with the user. Little work has been done on voice-interactions using PDAs. Mobile Reality Framework [20] makes use of speech using PDAs in its architecture. It makes use of ScanSoft RealSpeak TTS engine and the Siemens ICM Speech Recognition Engine. The Mobile reality framework runs entirely on Pocket PC and synchronizes a hybrid tracking solution to offer the user a seamless, location-dependent mobile multi-modal

interface. The specification used to implement the speech interaction management is proprietary and hence has its limitations for wide use.

The goal of this work is to prototype and to implement an architecture which can be used for interactive interactions between the participants (computer generated and real) in real-time. This thesis makes effective use of pre-defined speech commands using Speech Application Language Tags (SALT), which is an industry standard for developing speech applications and Microsoft Speech Server. The proposed architecture supports different types of PDAs and multiple modalities. It also tests the performance measurements in terms of response time over the wireless network to give an idea for feasibility of such architecture to use in real-time.

CHAPTER THREE: METHODOLOGY

The purpose of this work is to investigate methods for increasing and improving the level of interactivity between handheld devices and a VE by first decreasing latency and subsequently by allowing multiple modalities to operate simultaneously during runtime.

3.1 Proposed Architecture

The proposed architecture is shown in the Figure 1 [36]. The Pocket PC connects to a wireless router through a Wi-Fi (IEEE 802.11b) network. The access point (AP) connects to VESS through a proxy in a wired Ethernet. The architecture building blocks and functionality of each block are briefly explained as follows.

3.1.1 Pocket PC

The handheld device enables one to store, retrieve and play multimedia files, exchange text and voice messages, browse the Web, and more. The handheld device offers exchange or synchronization of information with a desktop computer, takes user input via a stylus or voice commands and outputs via audio, graphics, or text. Devices such as this have been used in VE research as simple controllers. These devices are also becoming widely used in a variety of fields. The mobile device used for this work is Compaq's iPAQ h3955 Pocket PC. In terms of computing power and equipment, these are the most powerful devices at present [18], compared to other devices having PalmOS and EPOC operating system. These kinds of PDAs are situated on the high-end systems category, featuring advanced characteristics such as 16-bit color

displays, audio reproducing/recording and wireless communication (wireless LAN, infrared, Bluetooth). The iPAQ Pocket PC platform has a growing number of software development tools and high performance graphics libraries. The details of the device are in Appendix A.

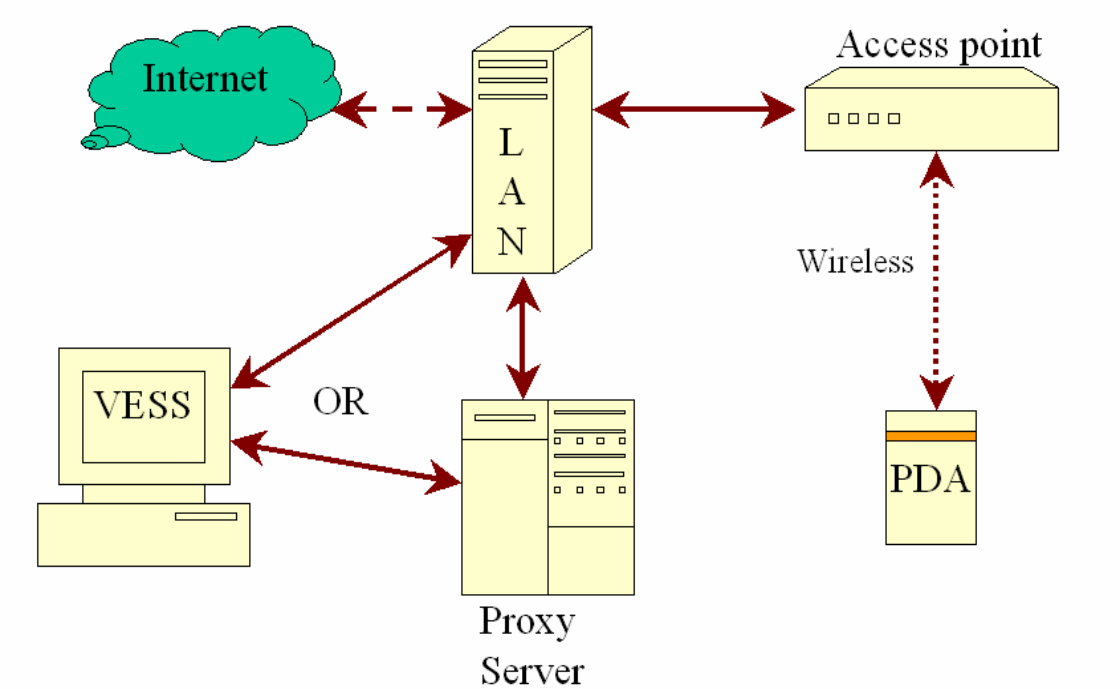


Figure 1: PDA-VESS architecture

3.1.2 Access Point

An access point (AP) is a hardware device that is a communication hub for users of a wireless device to connect to a wired LAN. They are specially configured nodes on Wireless Local Area Networks (WLANs) which act as a central transmitter and receiver of WLAN radio signals. APs are important for providing heightened wireless security and for extending the physical range of service to the users.

There are three major IEEE wireless LAN standards (802.11a, 802.11b, and 802.11g) operating on different frequency bands (2.4 for 802.11b & 802.11g and 5 GHz for 802.11a). Among these WLAN networks, 802.11b was chosen for this work as it is the most widespread version of wireless networking, which brings a theoretical throughput of 11 Mbps. A brief introduction on how 802.11b works is provided in Appendix B.

3.1.3 Proxy

The proxy is a server that handles connections on behalf of Pocket PC. It is a special software based server, which stores and manipulates data. In other words, it is a "network cache". The proxy also executes programs remotely from the handheld and provides results back to the handheld. The implemented architecture includes a proxy between the VESS system and wireless devices. The proxy has more functionality than routing and caching. For example, the proxy can re-format VESS communications for different PDA devices in real time. There are special functions added to the proxy to support voice for interacting with avatars and avatars' responses to the VE participant. The design also allows commands to other components and information to other players. Simple experiments have been prototyped using text to text and text to graphics for performance and connectivity analyses. The benefits of the proxy-based approach are outlined below:

- Flexibility for supporting multiple PDAs: A proxy based approach supports experimenting with wide array of personal communications devices. As these items grow in numbers and variety over the next few years, only interfaces will need to be changed thereby

keeping the core software intact. Thus experimentation can occur without affecting the full data content and network backbone.

- Scalability and Security: The use of a proxy that can process different types of data allows for multi-modal interaction with a VE. As modalities increase in the future with different types of handheld devices, one can add a small amount of code for each particular modality at the proxy server. The proxy also acts as a firewall between VESS and the PDA and hence does not allow unauthorized access to the virtual world.
- Support for distributed computing and caching: A proxy based approach will also be useful for using multi-PDA–VESS multimodal interactions. By making PDAs thin clients, distributed computing can be made possible where major computation is done at proxy and support for caching can be provided on it.

3.1.4 VESS

VESS [6] provides a public domain application software environment that is useful and functional using today's hardware, graphics and audio libraries; is extensible to support future hardware, graphics, audio libraries, and easily portable to multiple platforms, graphics and audio systems, and other application programming interfaces (APIs). VESS is designed to simplify and expedite the development of applications where VEs are required. It does this by providing a simple interface into the underlying graphics API and other output devices, such as haptics, while integrating support for various input devices, such as joysticks and motion tracking systems, and display devices, such as head-mounted displays and shutter glasses. Additionally, VESS provides behaviors and motion models to allow the user to manipulate his or her

viewpoint as well as control and interact with objects in the virtual environment. The user's viewpoint can be independent or attached to any transformable object in the scene. Also, VESS provides a seamless audio API that integrates directly into the VESS scene graph, giving developers the ability to easily add sound to the environment (including moving objects). Other useful routines, such as collision detection and terrain following, are also provided.

VESS provides a high-level library allowing complex virtual entities (avatars), complete with geometry and motion/articulation models, to be generated with a few simple lines of code. This is useful for dynamic networked virtual environments, which may involve many users and/or computer-generated forces at once. VESS provides the developer with the ability to handle avatars at a high level and leave the details of movement, articulations, and behaviors to the system.

VESS is also designed for easy portability. Its multi-layered architecture allows the developer to focus on the details of the application, without worrying about the specifics of the graphics API or hardware interfaces. Thus, applications built using the VESS libraries will be easily portable to any other platform. Currently, VESS runs on IRIX and Linux platforms using the SGI Open Performer API. Other platforms and API's will be supported in the future.

VESS is currently being enhanced by researchers at UCF IST to support experimentation in sequencing and latency in multi-modal virtual environments. The global multi-modal objective is to allow researchers to adjust sequencing of inputs and outputs to optimize information flow to the human user. The need for modal shifts should be sensed and switched seamlessly during runtime. This work complements this global objective by providing a handheld device to receive and transmit multi-modal information and architecture to minimize transmission and processing time to support experimentation.

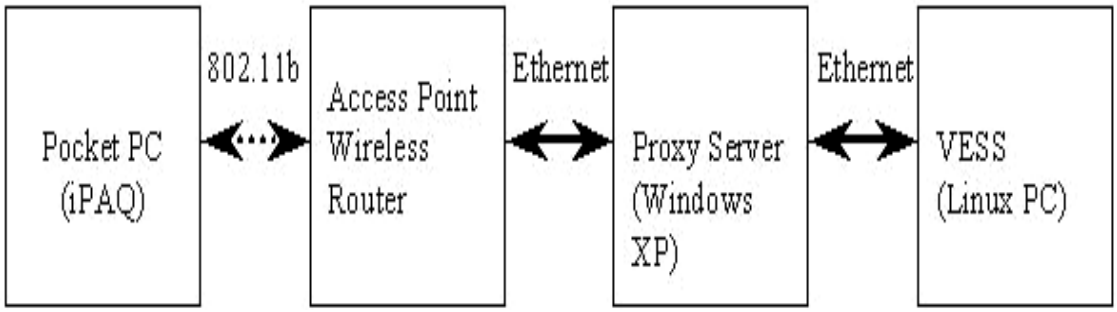


Figure 2: PDA-VESS Interface Block Diagram

Figure 2 shows the iPAQ Pocket PC as a user interface, a Windows based machine as a proxy and a Linux machine hosting VESS. This single threaded network is for controlled study to baseline network performance for text messages and images and to improve performance from that baseline.

3.2 Implementation

The implementation issue was a choice of technology to be used. Two approaches for implementation were considered:

1. Use of a Stand-alone client program: Use voice recording software on the PDA, record the commands, save them as .wav or any other recording format and send the audio file to proxy. At the proxy, run voice-to-text software to convert the commands into text, match the converted text with the pre-defined dictionary of words and if there is match, send the command to VESS.

2. Use of web-based approach: Run a web server with speech recognition and synthesis capability at the proxy. Issue commands from Pocket Internet Explorer (PIE) on the PDA. The speech engine at the proxy will convert this speech into text. The text is matched with the dictionary of pre-defined words at the proxy. If there is match between the text recognized from spoken word and the dictionary word, then that text is sent to VESS and the acknowledgement is received.

The second option is chosen, as it is an integrated approach and it makes PDA a thin client. As handheld devices grow in future with different platforms, the first approach would require them to build a specialized application for each of these devices. Microsoft .NET technology is adopted by taking advantage of its portability and excellent support for speech related applications for handheld devices. Speech Application Language Tags (SALT) is preferred over VoiceXML [26] as it supports wider variety of devices and is designed for multimodal and telephonic applications. VoiceXML was originally designed for Interactive Voice response (IVR) applications and is well suited for telephonic applications.

3.2.1 Microsoft .NET

Microsoft .NET (formerly Next Generation Windows Services) is a set of software technologies for connecting information, people, systems, and devices [21]. This new generation of technology is based on Web services—small building-block applications that can connect to each other as well as to other larger applications over the Internet. Essentially .NET represents an integrated approach to software development, deployment, and usage. Hence there is no need for developers to build separate applications for a mobile platform, a Pocket PC platform, or a

desktop platform. Each of these platforms is seen as a greater part of the whole with users able to seamlessly transfer data between the platforms.

The .NET Framework is the infrastructure for the Microsoft .NET Platform. The .NET Framework is a common environment for building, deploying, and running Web Services and Web Applications.

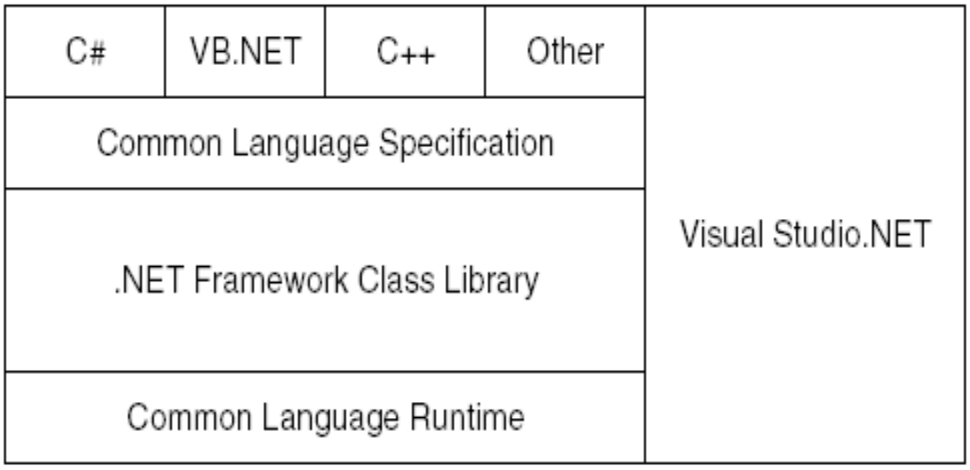


Figure 3: Overall block diagram of .NET framework

The .NET Framework contains Common Language Runtime (CLR), .NET framework class library and Common Language Specifications (CLS), .NET languages and Visual Studio as shown in Figure 3 [22]. CLR is the platform's execution engine. The code written for .NET runs under CLR's control. The .NET Framework class library consists of large number of classes for common functionalities, which can be used by all .NET languages. The CLS provides rules for multiple languages that .NET supports.

The Visual Studio.NET is a common development environment for the new .NET Framework. This What You See Is What You Get (WYSIWYG) tool provides a feature rich

application execution environment, simplified development and easy integration between a number of different development languages.

The Microsoft .NET Compact Framework [23] (CF) is a scaled down version of Microsoft .NET specifically designed for small form factor devices, such as Pocket PC. The .NET CF greatly simplifies the process of creating and deploying applications to mobile and embedded devices while also taking full advantage of the capabilities of the device. The .NET CF enables the execution of secure, downloadable applications on devices such as PDAs, mobile phones, and set-top boxes. .NET CF and C# are used for application development for this work.

3.2.2 Speech Application Concepts and Standards

The speech application uses a set of voice commands to define grammar. The three important concepts to build a speech application are dialogues, prompts and grammar. A dialogue is a composition of questions, answers, statements, and digressions. It is the conversation between the system and the user. This is the presentation logic of an application. Prompts are what the system says to the user to ask a question or provide status to the user. In a Graphical User Interface (GUI) application, the equivalent for prompts is labels and message boxes. Grammar is used to define and constrain the user input that the system recognizes. Additionally, the grammar provides a way to associate multiple phrases with a single semantic meaning. For example, the phrases “Help” and “what can I say” would be mapped to a single meaning of “Help.” Thus, grammar and prompts deal specifically with the input and output of a speech application while dialogue weaves the two together in the most natural manner possible.

These concepts are developed using industry standards. The four open standards for speech application development are:

1. Speech Application Language Tags (SALT) is the core API for spoken interaction with a user. It provides the core constructs of prompts (questions), listens (answers), and related APIs. This specification is being driven by the SALT Forum [24]
2. Speech Recognition Grammar Specification (SRGS) provides a way to define the phrases and phrase combinations that an application recognizes from a user – generally referred to as just "grammar." This specification is driven by the World Wide Web Consortium (W3C) [29]
3. Speech Synthesis Markup Language (SSML) is the text-to-speech specification being driven through the W3C. It defines output (prompts) in the application. The specification is available at [30]
4. ECMAScript (ECMA-262) is commonly seen in its implemented forms as JScript, JScript.NET, and JavaScript. JScript and JScript.NET (depending on the client) are used throughout the Speech Application Software Development Kit (SASDK). The specification for ECMAScript is available at [31].

3.2.3 Speech Application Language Tags (SALT)

The focus of this work is to have voice interaction from PDAs to VE. As speech recognition systems are becoming more practical, they provide an excellent opportunity for natural communication with computer systems. This is especially true with VEs, where the goal is to provide the most natural form of interface possible. The ultimate speech recognition system

would understand context and use it to interpret speech, and it would be able to process a steady stream of speech from any speaker. Speech Application Language Tags (SALT) [24] is an effort to achieve this goal. SALT, an open industry standard, is a speech interface markup language. It consists of a small set of XML elements, with associated attributes and Document Object Model (DOM) object properties, events and methods, which apply a speech interface to web pages. It provides the core constructs of prompts (questions), listens (answers), and related APIs. SALT tags are a lightweight set of extensions to existing markup languages; in particular HTML, XHTML and XML that enable multimodal and telephony access to information, applications and Web services from PCs, telephones, tablet PCs and wireless PDAs. Multimodal access enables users to interact with an application in a variety of ways: input with speech, a keyboard, keypad, mouse and/or stylus; and output as synthesized speech, audio, plain text, motion video and/or graphics. Each of these modes could be used independently or concurrently.

The Microsoft Speech Server [25] contains a complete solution for developing, testing, deploying, and managing telephony (speech only) and multimodal (speech/visual) applications. Specifically, the product contains the following:

- Microsoft Speech Server
- Microsoft Speech Application SDK

The Microsoft Speech Server (MSS) contains all of the server components for deploying telephony and multimodal applications. MSS runs on Windows Server 2003 and performs speech recognition and speech synthesis for telephone, cell phone and Pocket PC devices. The Microsoft Speech Application Software Development Kit (SASDK) addresses the needs of the speech application developer with APIs, controls, and tools that extend Visual Studio .NET into the

speech domain. SASDK includes the client-side speech add-ins, Speech Add-in for Microsoft Internet Explorer and Speech Add-in for Microsoft Pocket Internet Explorer (PIE) which incorporates the means for desktop PCs, Tablet PCs, and Pocket PC devices to understand speech tags embedded in HTML pages as defined by the SALT specification. The SDK also contains a desktop version of the new Microsoft speech recognition engine, a test-level version of the Microsoft Text-to-Speech (TTS) engine, and tools necessary for building and testing speech applications. Fig. 4 shows a high-level view of a deployed speech solution with Pocket PC which is a multimodal client, the speech server components, and a Web server hosting a speech application.

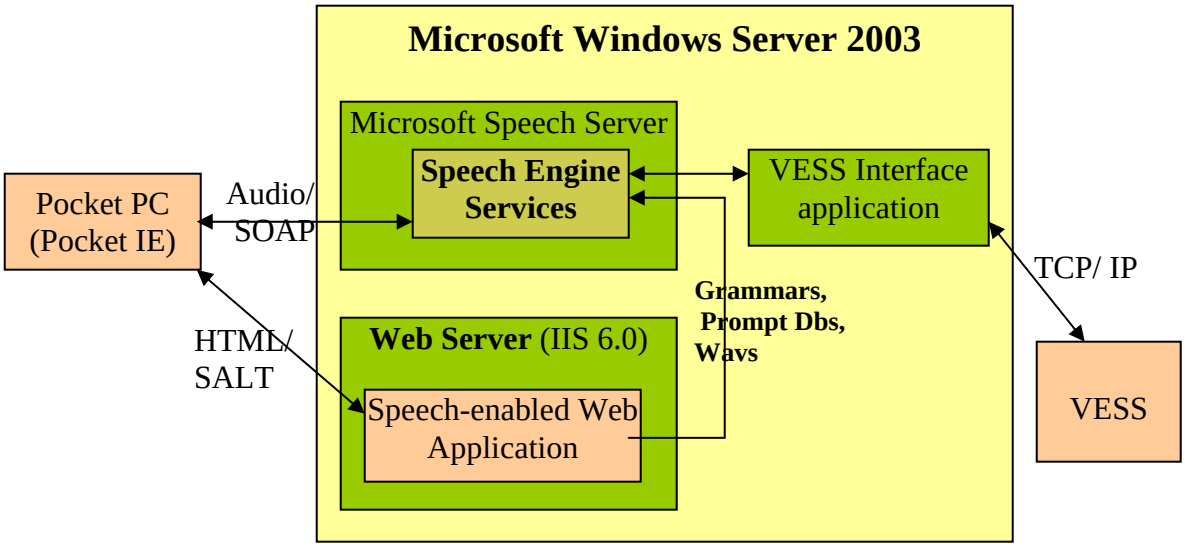


Figure 4: Implementing Architecture

Figure 4 is a detailed implementation diagram of Figure 1. The web browser, that is, pocket Internet Explorer (PIE) on the PDA provides user interaction, where the user can make a

request to the proxy server for a particular task such as getting a map, playing an audio clip etc. PIE has a speech add-in provided into SASDK. The Speech Add-in for PIE is a SALT interpreter that supports speech-enabled Web applications deployed over 802.11 wireless networks. The Speech Add-in for PIE supports all Basic Speech Controls and Dialog Speech Controls delivered in the SASDK, using MSS to perform all the actual speech processing. It also supports a Pocket PC hardware button event that developers can bind to listen and prompt elements. On a Pocket PC, the user enters a Uniform Resource Locator (URL) in Pocket Internet Explorer, which opens an .aspx file on the Web Server. The Web Server on the proxy hosts the .aspx page and sends HTML, SALT and script to the Pocket PC. The Pocket PC sends a compressed representation of the audio and a pointer to the recognition grammar to the Speech Server that performs recognition and returns results to the Pocket PC. The VESS interface program on the server takes the matched command and forwards that request to VESS through TCP/IP sockets for further action. VESS performs the necessary action corresponding to that instruction and acknowledges the request by sending appropriate information back to the Pocket PC via the proxy server. The requests and acknowledgements are cached at the Speech Engine Services (SES) proxy for future use. The acknowledgement to PDA can be sent as a text, an image or audio to provide server-side speech recognition and speech playback services for multimodal and telephony clients using the MSS. A multimodal client on a Pocket PC device accesses SES directly for both speech recognition and speech playback.

The commands in text format are sent via a predefined XML format to VESS through a socket connection over wired Ethernet.

3.2.4 Voice Commands

For the controlled study, few frequently used movement strings, which are phonetically different, were used as voice commands. Being phonetically different, there are fewer chances of confusing similar sounding phrases, e.g. “kiss this guy” instead of “kiss the sky.” Each movement string is composed of a movement word and a direction word. Movement words and direction words are listed in Table 1 and Table 2 along with their usual pronunciations and frequencies of use. Frequencies are taken from the British National Corpus (BNC) database and word frequency [27]. The frequency is the number of occurrences in the whole 100 million word collection in the BNC database. The pronunciations are taken from American Heritage Dictionary (AHD) [28].

Table 1
Movement words (can be combined with direction words)

Movement Word	**AHD Pronunciation Key**	**Frequency**
Turn	turn, turn	45487
Go	gow, gō	249540
Move	moov, mo͞ov	37836
Stop	stâp, stop	25066
Halt	holt, hɔ ːlt	1483

Table 2
Direction words

Movement Word	AHD Pronunciation Key	Frequency
Forward	'forwurd, fôr'wə̈rd	12582
Back	Bak, bă̈k	75494
Up	up,ŭ̈p	195426
Left	left, lĕ̈ft	11343
Right	rIt, rī̈t	40460

The voice commands built from these words are:

- Halt
- Stop
- Turn Left
- Turn Right
- Go Forward
- Go Back
- Go Up
- Go Down
- Move Forward
- Move Back
- Move Up
- Move Down

3.2.5 Working of Speech Application

The two major components of a speech application are a client (Web browser) and a Web server. In addition, a component generically referred to as Speech Services comes into play. The Web server stores an ASP.NET application, built with Web Forms and Microsoft ASP.NET Speech Controls. The speech controls render speech-specific markup to the client rather than the HTML rendered by other types of Web controls. Grammar files (.GRXML files) and prompt databases (containing the recorded prompts) are also stored on the Web server. The HTTP request lifecycle in a typical Web application is the same in a speech application. However, during the rendering phase, what is rendered to the client is a document containing SALT, SSML, SRGS, and CSTA tags in addition to the HTML and JScript that a typical Web application generates. When the client receives this document, two things happen:

1. The Speech Services downloads and parses the grammar in preparation for running it. The Speech Services also downloads the prompt database (if specified) for playback as prompts are encountered.
2. The SALT client and Web browser invoke the <listen> and <prompt> elements as specified in the script. Note that rather than executing elements linearly as a standard HTML client would, elements are executed in their order of invocation as indicated by calling their start() methods.

The Speech Services start listening for input from the user when a <listen> element is invoked. Once it receives the audio (an utterance), it compares its analysis of the audio stream to what is stored in the grammar, looking for a matching pattern. If the recognizer finds a match, a special type of XML document is returned to the client. The document contains markup called

Semantic Markup Language (SML) and is used by the client as the interpretation of what the user said – this is effectively what the grammar, or at least its recognized parts, is transformed into. The client then uses this document to determine what to do next (execute a prompt or listen element), and the cycle repeats itself until the application is done collecting data and the session ends. Figure 5 shows the grammar file created for this application using Visual Studio.NET.

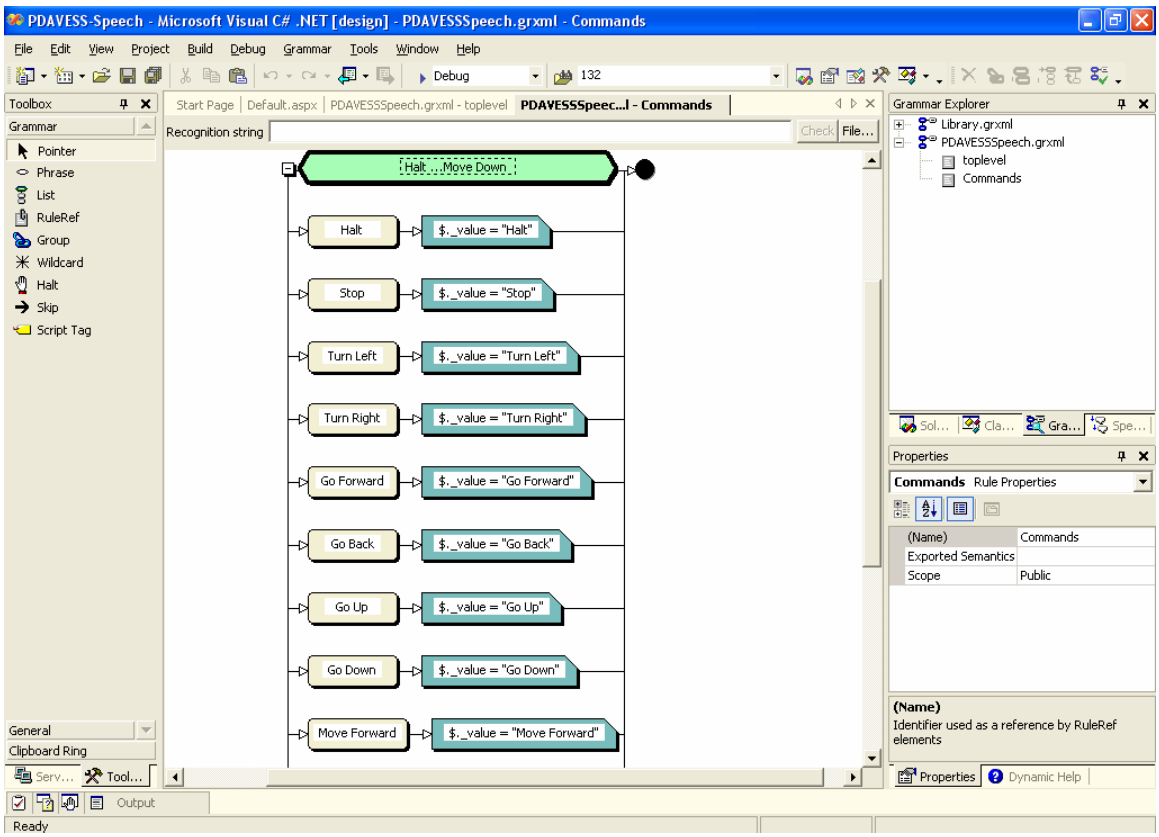


Figure 5: Grammar file in Visual Studio.NET 2003

The architecture described in chapter three requires Microsoft Speech Server with Speech Engine Services (SES) in it. At the time of experiments, this product was in beta testing and was not available commercially. Hence a speech application was created for the Pocket PC and was

tested on the desktop machine, since the SASDK contains desktop version of speech engine. The same application can be ported and used for Pocket PC with the availability of SES.

One advantage that the SALT and Speech server offers is that the system does not need training of voice. It recognized voices of different people in a similar manner.

While the speech application described above enables the interaction between a human and an avatar in more natural and less cumbersome way, the performance over the wireless communication between the PDA and proxy is important in order to achieve real-time interaction. There are two major transport protocols used to send data, a reliable Transmission Control Protocol (TCP) and unreliable User Datagram Protocol (UDP). The UDP provides best effort service in the sense that it doesn't make sure that packets did reach destination nor does it retransmit them. Transmission Control Protocol on the other hand makes sure that packets reached destination by using acknowledgements and retransmissions. TCP is used as it is reliable and it scores over UDP for multimedia applications in terms of fair share of bandwidth [34].

3.3 Transmission Control Protocol and Sockets

TCP [32] is a transport layer network protocol that offers a reliable, connection-oriented, byte-stream service. It is a full-duplex protocol, which means that each TCP connection supports a pair of byte-streams, one flowing in each direction. TCP supports flow control, which prevents the sender from overrunning the buffer capacity of the receiver. In addition, TCP implements congestion control, which prevents the sender from injecting too much traffic into the network. TCP adds connection information to the data packet. This allows programs to create an end-to-end connection between two network devices, providing a consistent path for data to travel. TCP

guarantees the data will be reliably delivered to the destination device or that the sender will receive an indication that a network error occurred. Because of this feature, TCP is called a connection-oriented protocol. Each TCP connection, or session, includes a certain amount of packet overhead related to establishing the connection between the two devices. Once the connection is established, data can be sent between the devices without the application having to check for lost or out-of-place data. TCP is an end-to-end protocol. That is, TCP turns a host-to-host packet delivery service, provided by IP, into a process-to-process communication channel. The Internet Protocol (IP) is the inter-networking protocol that TCP usually relies upon. IP is a network layer protocol in the 7-layer Open System Interconnection (OSI) model.

In order to communicate with an application on a remote device, the following information is needed:

1. The remote device's IP address
2. The TCP port assigned to the remote application

For a TCP connection to be established, the remote device must accept incoming packets on the assigned port. Because there could be many applications running on a device that use TCP, the device must allocate specific port numbers to specific applications. This tells the client which port to use for a particular application and tells the host which application an incoming packet should be forwarded to. Figure 6 shows how clients and servers use TCP ports to channel data between applications.

In Figure 6, network device A is running two server applications, waiting for incoming packets from remote devices. One application is assigned TCP port 8000 on the device and the other is assigned port 9000. Network device B is a client that wants to connect to the applications on the server. For a device to send a packet to a remote device, it must obtain a free TCP port

from the operating system, which remains open for the duration of the session. The client TCP port number is usually not important and can be assigned to any available port on the device. The client forwards the packet from an available port on Device B to the application TCP ports on Device A.

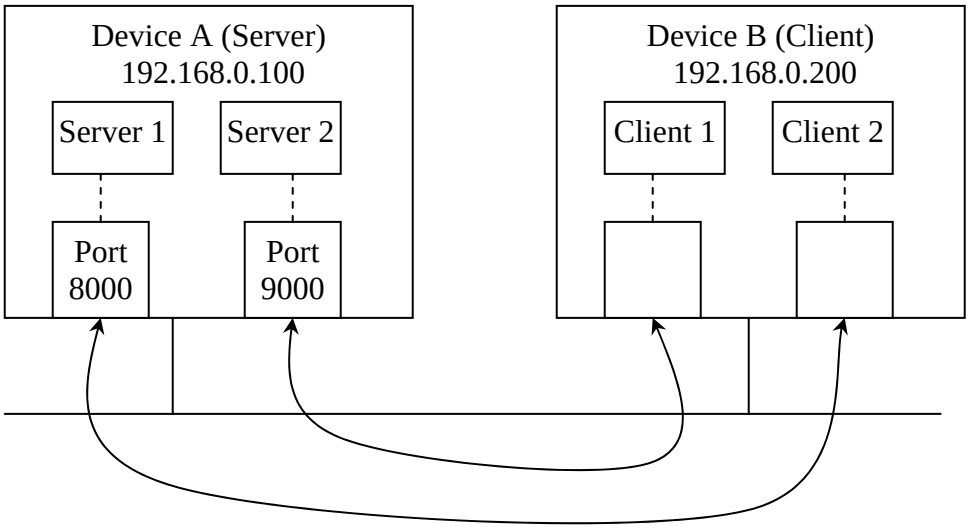


Figure 6: Simple TCP Connection

The combination of an IP address and a port number defines an IP endpoint. A TCP session is defined as the combination of a local IP endpoint and a remote IP endpoint. Only one session can have both these properties the same time. A single network application can use the same local IP endpoint, but each remote connection must have either a separate IP address or remote port number.

TCP maintains packet reliability by using sequence number and acknowledgement fields in its header. It uses connection states to determine the status of the connection between devices.

A specific handshaking protocol is used to establish these connections and to monitor the status of the connection during the session. The TCP session has three phases:

- Opening handshake
- Session communication
- Closing handshake

Figure 7 shows graphically the three states during TCP session.

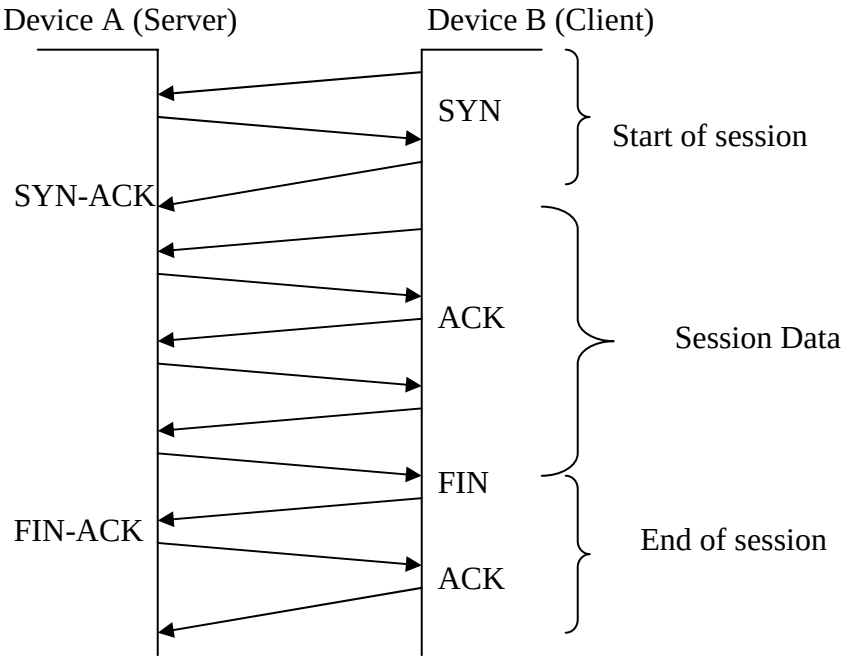


Figure 7: Simple TCP Connection

The opening handshake is often called the three-way handshake and requires three steps to establish a connection through synchronize (SYN) and acknowledgement (ACK) flags.

1. The originating host (client) sends a SYN flag to indicate the start of a session.

2. The receiving host sends a both a SYN flag and an ACK flag in the same packet to indicate it accepts the start of the session.
3. The originating host sends an ACK flag to indicate the session is open and ready for packets.

After the session is established, the ACK flag is set on packets, indicating that the device is acknowledging the receipt of a packet with a particular sequence number. To close the session, closing handshake is done using FIN flag.

1. The host initiating the close sends a FIN flag.
2. The remote host sends both an ACK flag and a FIN flag in the same packet to indicate it accepts the end of the session.
3. The initiating host sends an ACK flag to officially close the session.

The SYN flag indicates starts of session. The ACK flags indicates acknowledgement. The FIN flag indicates close of session. The phases of the TCP session are associated with connection state names. Each connection state indicates the session's current position in the handshaking sequence. The connection states apply equally to clients as well as servers. Both devices in the TCP session follow the same TCP states.

To ensure the integrity of data, TCP keeps all sent data in a local buffer until positive acknowledgement of reception is received from the remote device. Similarly, when receiving data from the network, TCP keeps a local buffer of received data to ensure that all of the pieces are received in order before passing the data to the application program.

From the programming point of view, an application does not directly access the network interface device to send and receive packets. Instead, an intermediary file descriptor is created to handle the programming interface to the network. The special file descriptors used to reference

network connections are called sockets. A socket defines specific communication domain, a specific communication type and a specific protocol. In our case, the communication domain is Internet, communication type is stream and specific protocol is TCP.

After the socket is created, it is bound either to a specific network address and port on the system or to a remote network address and port. Once the socket is bound, it can be used to send and receive data from network. Figure 8 shows how this process works.

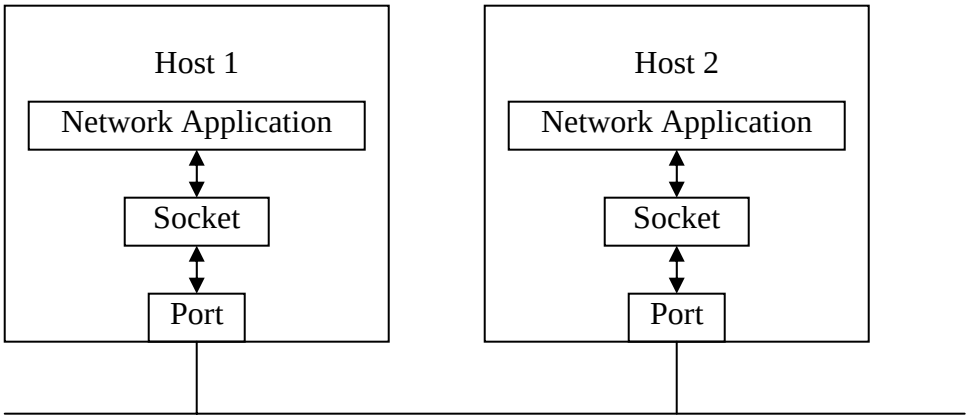


Figure 8: The Socket Interface

For the experimental setup, a socket program was created using Microsoft .NET Compact Framework on Pocket PC and a socket program was created on the proxy using .NET Framework. C# programming language was used for the network application programming [33]. The performance of the PDA-Proxy was measured based on response time characteristics. For this study, text data of various sizes was sent over the wireless network and the response time was noted. Text size varied from 1 byte till 8500 bytes. The round trip time (RTT) was

calculated as the difference between the time instance at which text was sent from PDA to the time instance when the acknowledgement in the form of a string “Received” is received at the PDA from the proxy. The same text was sent 100 times, 500 milliseconds after the previous transmission. A new connection was established between the PDA and the proxy for each request. Initially the number of bytes sent from PDA was increased in steps of 100 until it reaches 1000 bytes. Then the number of bytes sent over the network then was increased in steps of 500 bytes.

The operating system for proxy is Microsoft Windows XP Professional Version 2002 Service Pack 1 and processor is Intel Pentium 4 with speed of 2.4 GHz. The proxy is connected to VESS via wired Ethernet (100BaseT) at the Institute for Simulation and Training (IST). The experiment was performed by making use of two networks, first using the 802.11b wireless router in IST and another using the wireless router at UCF. This was to see if there were differences in the readings as UCF network is more congested than IST. When connected to the IST network, the PDA gets a class ‘C’ private address (192.168.0.100), while when connected to the UCF network; the PDA gets a class ‘A’ private address (10.171.33.35). The effect of Nagle Algorithm [35] on RTT is also studied.

CHAPTER FOUR: FINDINGS

4.1 Nagle Algorithm

The Nagle Algorithm is used by sockets to automatically concatenate a number of small buffer messages; this process (called *nagling*) increases the efficiency of a network application system by decreasing the number of packets that must be sent over the network. The algorithm applies when a TCP sender is deciding whether to transmit a packet of data over a connection. If it has only a "small" amount of data to send, then the Nagle algorithm permits sending the packet only if all previously transmitted data has been acknowledged by the TCP receiver. In this situation, "small" is defined as less data than the TCP Maximum Segment Size (MSS) for the connection, the largest amount of data that can be sent in one datagram. If more small segments are generated while awaiting the acknowledgement (ACK) for the first one, then these segments are coalesced into one larger segment. Any full-sized segment is always transmitted immediately, assuming there is a sufficient receive window available. The Nagle algorithm is effective in reducing the number of packets sent by interactive applications, such as Telnet, especially over slow links. The Nagle algorithm delays transmission of these short messages in the hope that more messages will become available soon, thereby avoiding packet congestion.

In the current speech application, the priority is to have real-time interaction and the voice messages in the form of commands are quite small. The Nagle algorithm will restrict sending such small packets increasing delay for sending the voice commands. Also, if the big packet is lost, the risk is much more in terms of packet loss. This is especially true for packets in wireless networks, which follow multipath. Each packet may follow some particular path and the

risk of packet loss is evenly spread if more small packets are sent. With the use of TCP, reliability can be added to each small packet. Large packet meant more delay and hence reliability can be increased by sending small packets. Hence experiments are performed to see the effect of Nagle algorithm on the round-trip time of the requests sent over Wi-Fi network. The Nagle algorithm can be disabled by setting TCP_NODELAY option to true.

4.2 Results

From the readings, the Average RTT and minimum RTT were plotted. Figure 9 shows the Average RTT in milliseconds for the number of bytes sent from the PDA by making use of the UCF wireless router. The Nagle algorithm is enabled.

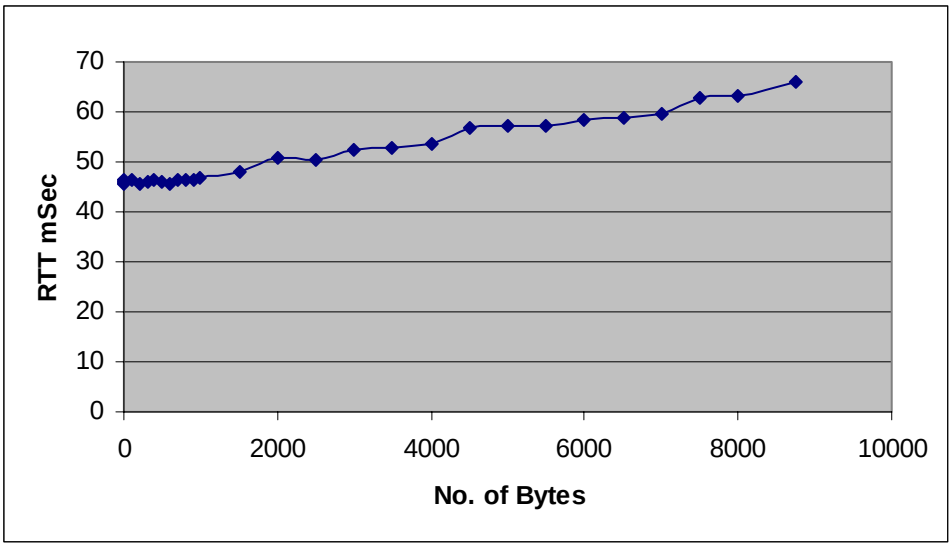


Figure 9: Average RTT using UCF wireless router – Nagle ON

Figure 10 shows the Minimum RTT in milliseconds for the number of bytes sent from the PDA by making use of the UCF wireless router. The Nagle algorithm is enabled.

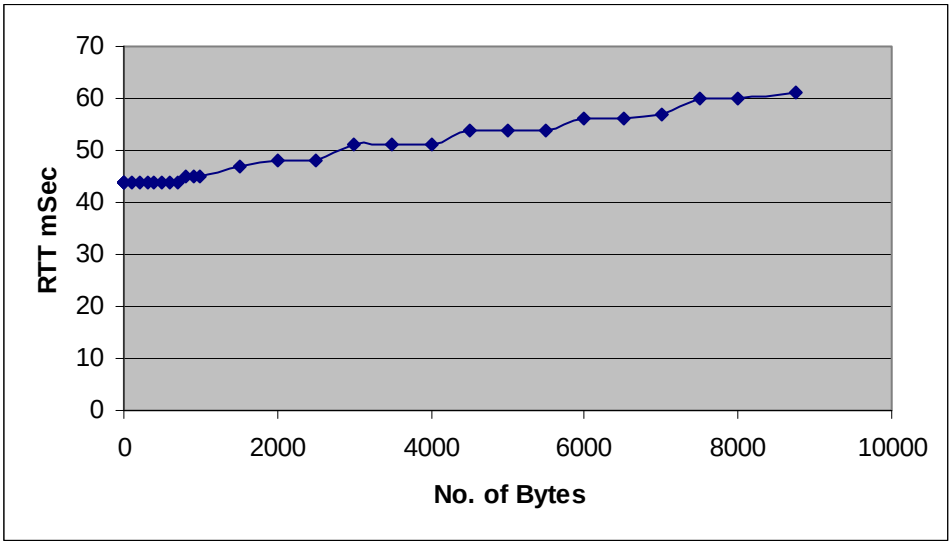


Figure 10: Minimum RTT using UCF wireless router – Nagle ON

Figure 11 shows the Average RTT in milliseconds for the number of bytes sent from the PDA by making use of the IST wireless router with Nagle Algorithm enabled.

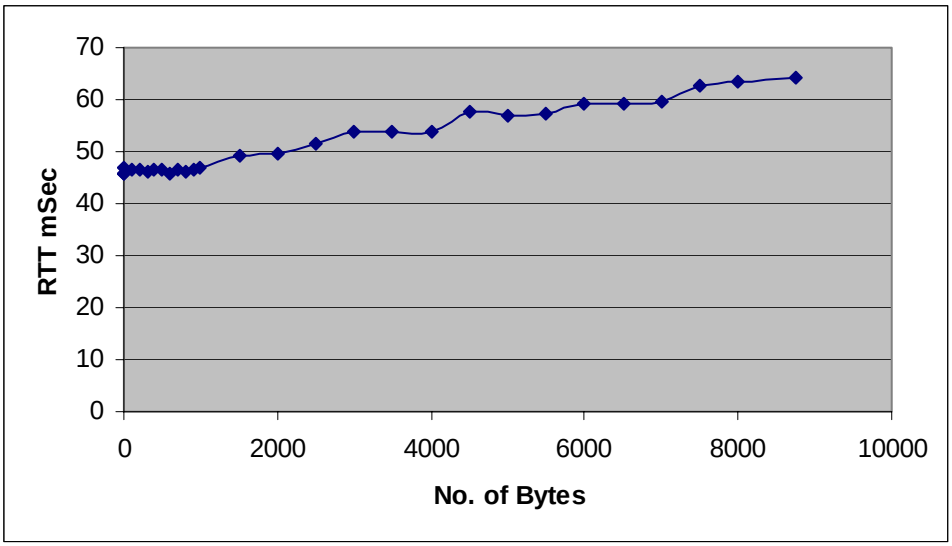


Figure 11: Average RTT using IST wireless router – Nagle ON

Figure 12 shows the Minimum RTT in milliseconds for the number of bytes sent from the PDA by making use of the IST wireless router with Nagle Algorithm enabled.

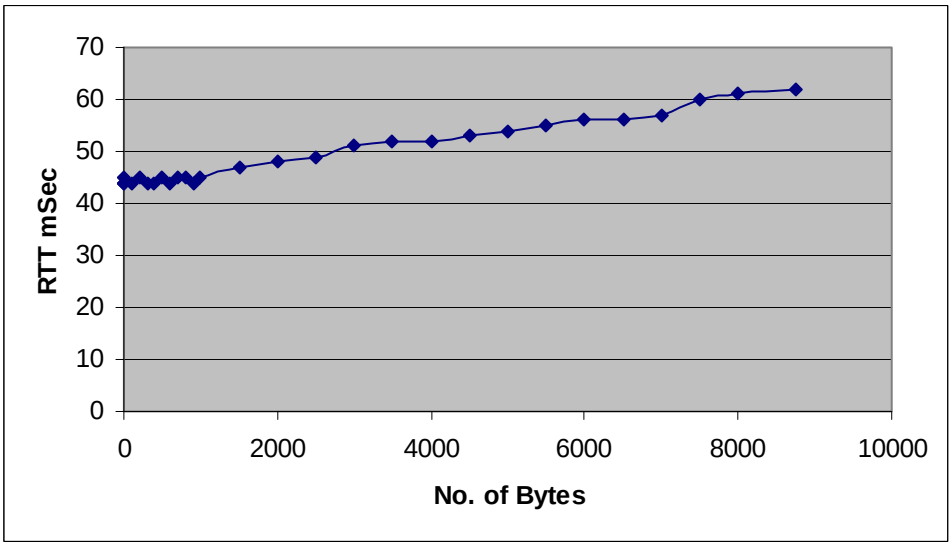


Figure 12: Minimum RTT using IST wireless router – Nagle ON

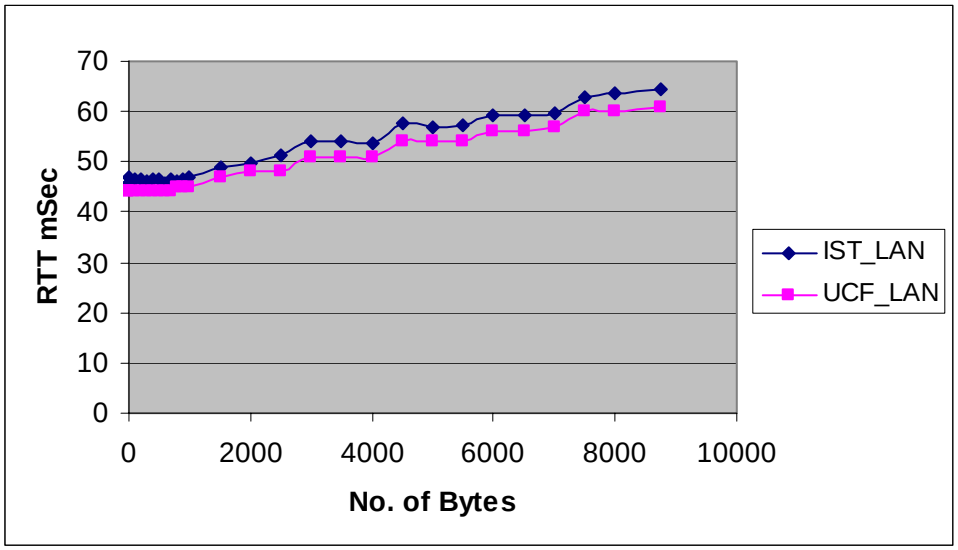


Figure 13: Comparison of Average RTTs with wireless routers at UCF and IST – Nagle ON

Figure 13 shows the comparison of Average RTTs in milliseconds for the number of bytes sent from the PDA by making use of the UCF wireless router and IST wireless router with Nagle Algorithm enabled.

Figures 9 to 12 shows that with increase in size of data, the RTT increases almost linearly with the size of data. Figure 13 shows that the RTT is almost same when comparing the IST and the UCF network.

Figure 14 shows the Average RTT in milliseconds for the number of bytes sent from the PDA by making use of the UCF wireless router with Nagle Algorithm disabled.

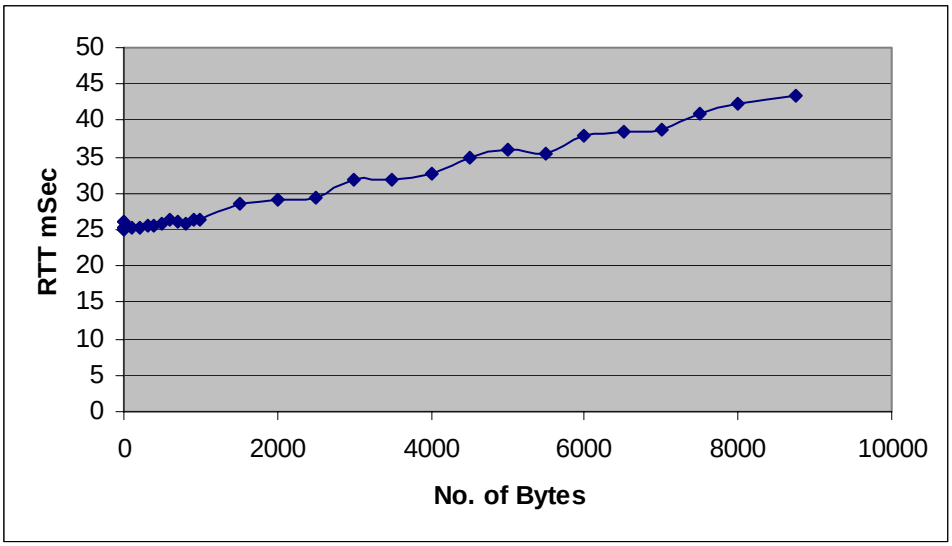


Figure 14: Average RTT using UCF wireless router – Nagle OFF

Figure 15 shows the Minimum RTT in milliseconds for the number of bytes sent from the PDA by making use of the UCF wireless router with Nagle Algorithm disabled.

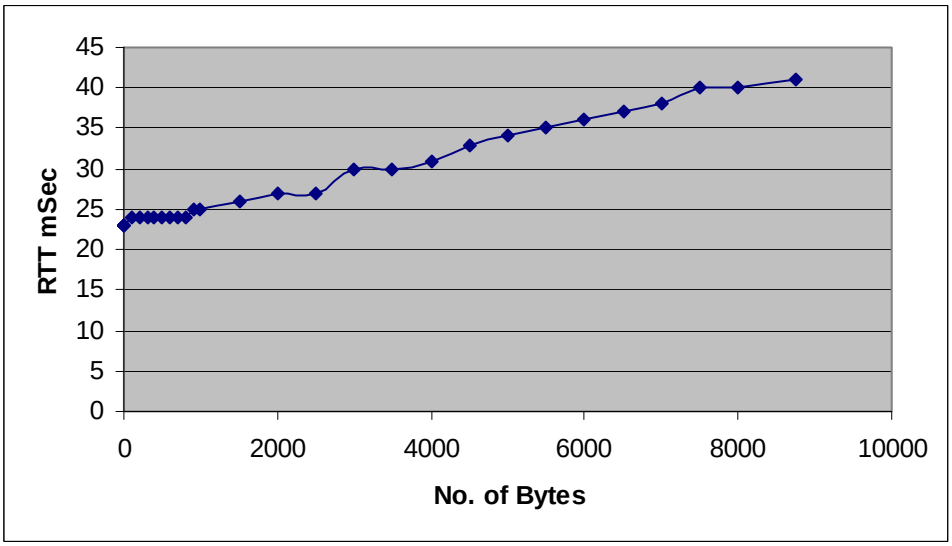


Figure 15: Minimum RTT using UCF wireless router – Nagle OFF

Figure 16 shows the Average RTT in milliseconds for the number of bytes sent from the PDA by making use of the IST wireless router with Nagle Algorithm disabled.

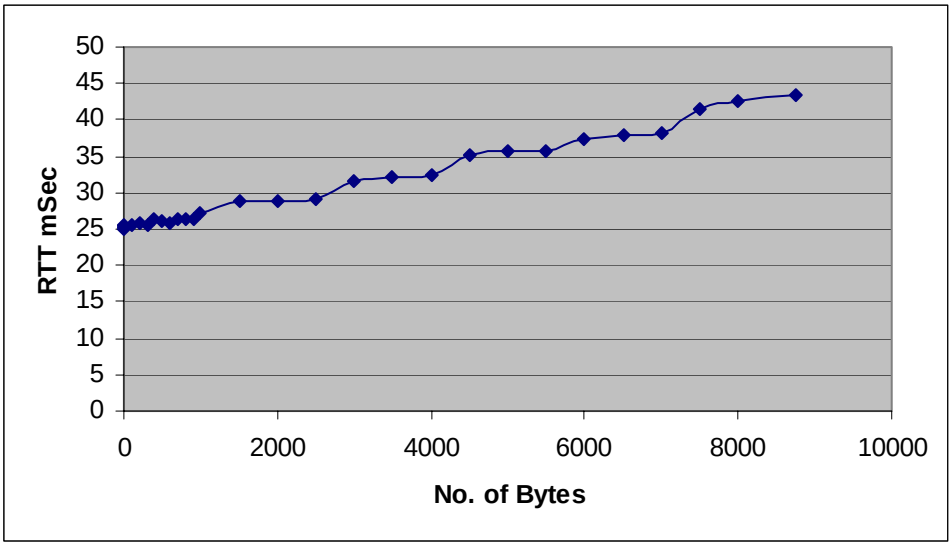


Figure 16: Average RTT using IST wireless router – Nagle OFF

Figure 17 shows the Minimum RTT in milliseconds for the number of bytes sent from the PDA by making use of the IST wireless router with Nagle Algorithm disabled.

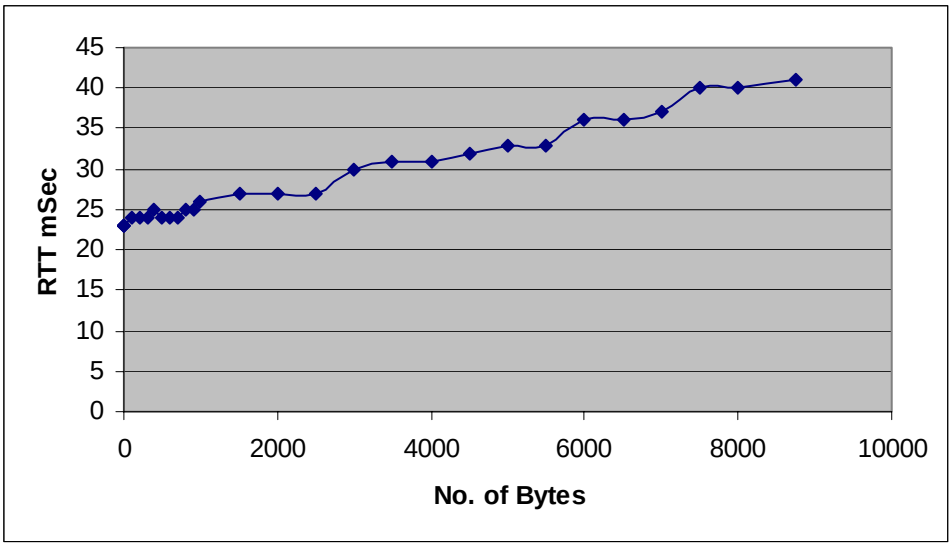


Figure 17: Minimum RTT using UCF wireless router – Nagle OFF

Figure 18 shows the comparison of Average RTTs in milliseconds for the number of bytes sent from the PDA by making use of the UCF wireless router and IST wireless router with Nagle algorithm disabled.

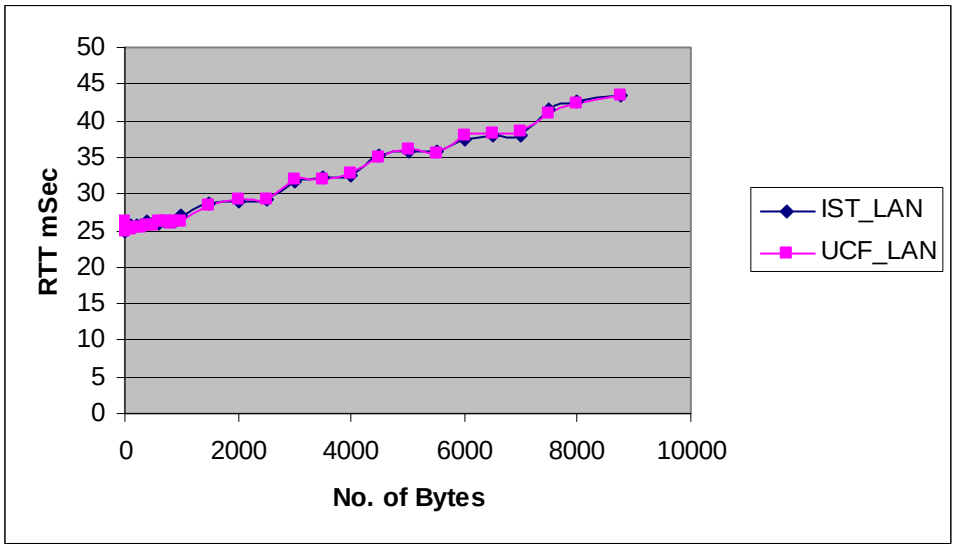


Figure 18: Comparison of Average RTT using UCF & IST wireless router – Nagle OFF

Figures 14 to 17 show similar results as that of 9 to 12 with lower RTT for Nagle algorithm disabled. Figure 18 shows that the results are almost identical for both the IST and the UCF networks.

Figure 19 shows the comparison of Average RTTs in milliseconds for the number of bytes sent from the PDA by making use of the UCF wireless router with Nagle algorithm enabled and disabled.

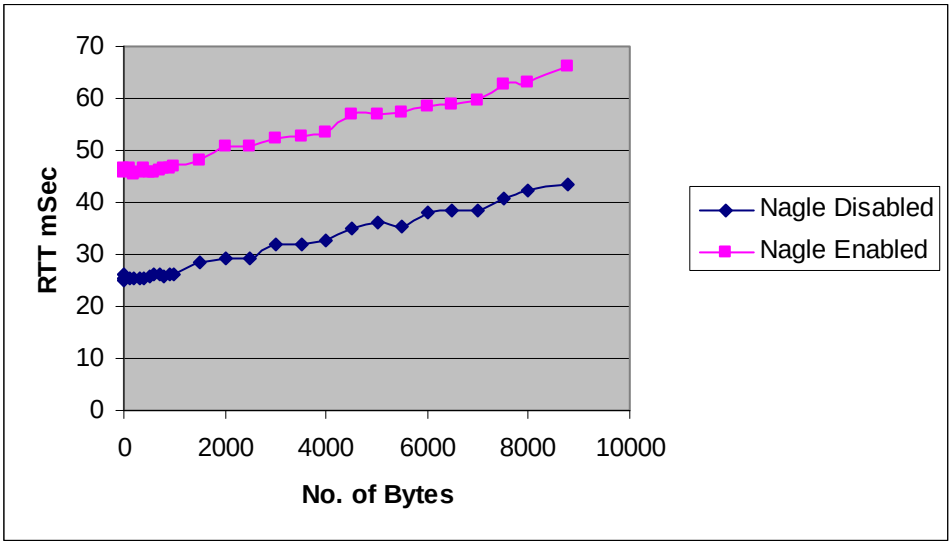


Figure 19: Comparison of Average RTT using UCF wireless router for Nagle ON and OFF

Figure 20 shows the comparison of Minimum RTTs in milliseconds for the number of bytes sent from the PDA by making use of the UCF wireless router with Nagle algorithm enabled and disabled.

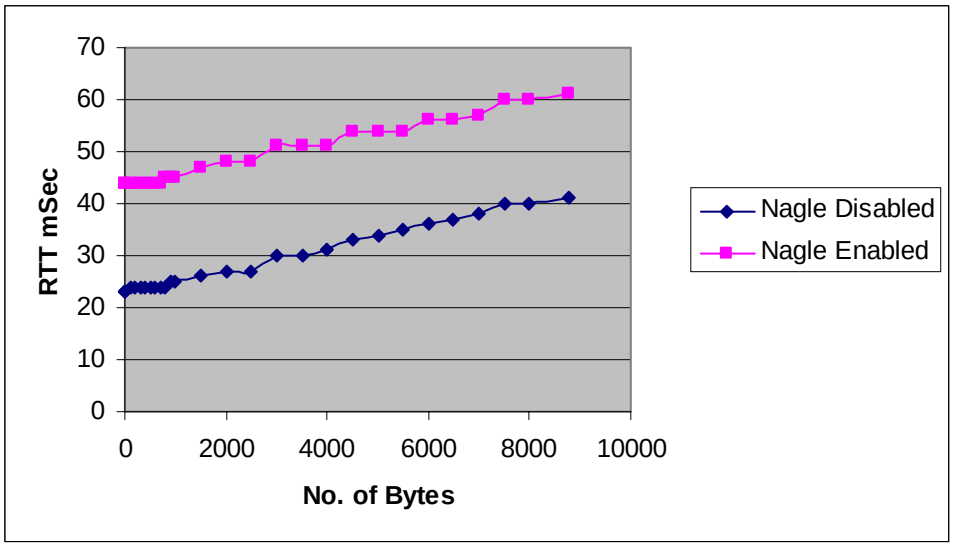


Figure 20: Comparison of Minimum RTT using UCF wireless router for Nagle ON and OFF

Figure 21 shows the comparison of Average RTTs in milliseconds for the number of bytes sent from the PDA by making use of the IST wireless router with Nagle algorithm enabled and disabled.

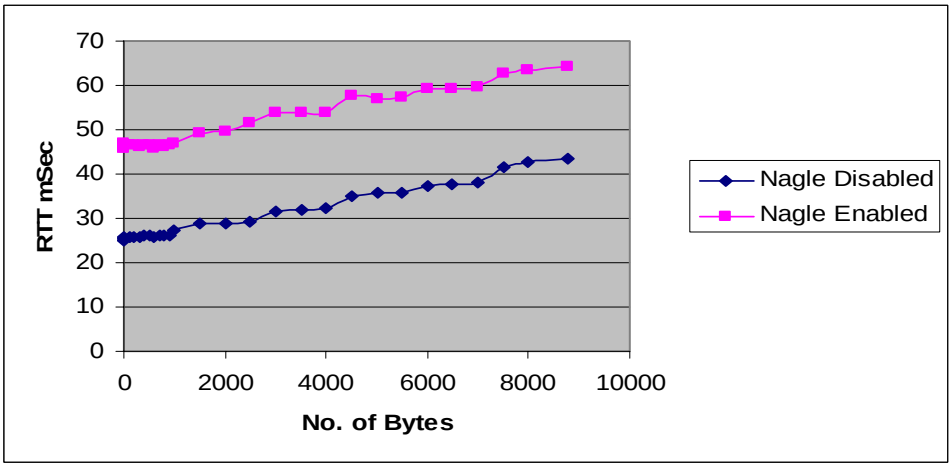


Figure 21: Comparison of Average RTT using IST wireless router for Nagle ON and OFF

Figure 22 shows the comparison of Minimum RTTs in milliseconds for the number of bytes sent from PDA by making use of IST wireless router with Nagle algorithm enabled and disabled.

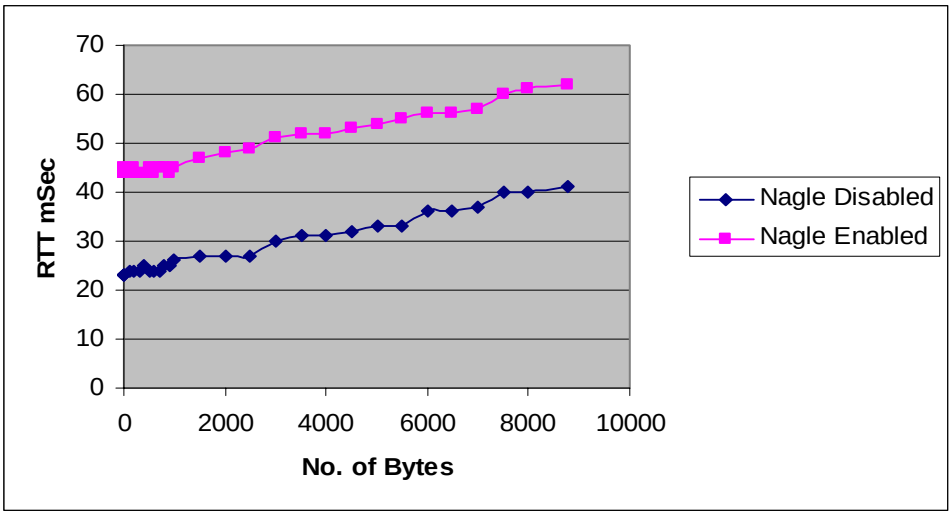


Figure 22: Comparison of Minimum RTT using IST wireless router for Nagle ON and OFF

4.3 Discussions

Figures 19, 20, 21 and 22 show that disabling Nagle algorithm gives a better performance in terms of reduced latency.

A similar work [36] shows the difference between the RTTs with Nagle enabled and disabled is more compared to the results obtained in this study. One reason for this difference could be that a different version of the operating system (Pocket PC 2002) was used in that work and also a server-based technology (Java servlets) was used. In this latter case, all the processing was done at the proxy and only results were sent back to the PDA. In the current application, the

code is native to the PDA and it runs on it. It also has newer version of Windows Mobile 2003 operating system.

Interactivity can be thought of as needed to support a 140 milliseconds response time [35]. This time represents a nominal measure of the time needed for a human to sense, perceive, and act on a stimulus. In order to have a highly interactive interaction for computationally intensive applications like that between users and avatars, it is necessary to provide interactivity within the human response time to keep the round trip time over wireless networks as small as possible. The RTT over wired Ethernet from the proxy to VESS is found to be 15 milliseconds. Since the RTT from the proxy to VESS over wired Ethernet is small and almost constant, a smaller RTT over a wireless network can make it possible for more interactive and computational tasks to be carried out in real-time. There will be processing time at the proxy and at VESS and then the acknowledgement would be sent back to PDA. Smaller RTT will compensate for these processing times.

Usability of the PDA for VE applications was also briefly and informally calculated using Nielsen's [37] heuristics as a guide. The outcome of this evaluation suggested that, as a minimum, scale and orientation are features that need to be included in the PDA if it is to be used to guide user navigation. Orientation refers to the user's angular position in the virtual environment. Scale refers to some means of quickly determining the distance between objects, as well as the extent of natural and man-made features. Additional consideration needs to be given to the use of alternative modalities to convey information via this device, such as a vibration to indicate when one is moving away from the target location (e.g., when trying to find a specific room in a burning building) or auditory cues to indicate a specific state, such as arrival of new

data. With additional modalities, consideration will need to be given to ensure consistency and smooth transition between modes.

The architecture has been implemented to study different utilities of PDA's in Virtual Environments. One particular usability pilot study that has been performed at UCF IST involved the use of a PDA to control movement and show positioning of users in a virtual environment using an interactive map. This pilot study used an interactive map to place users in a virtual world where they were allowed to freely explore and move around. The user can control their movements using the interactive map by touching a desired position on the map, which was presented on the PDA screen. When a position was selected on the map, the position point was sent to the VESS system and the location of the user was updated within the perceptible delay. The pilot study indicated that it was very important that the cycle of getting input from the user and updating the output to the user be completed within the perceptible delay. The previously explained architecture proved to be effective in performing this task.

There were several lessons learnt during this pilot study. The first drawback observed was the small screen space of PDAs to give sufficient information about the map. Although the virtual environment in use was fairly small, it was difficult to present enough detail on the limited sized PDA screen to allow users to easily determine their current location on the map and to input the desired destination onto the PDA. A possible future study that has arisen because of this drawback is to evaluate the effect of screen size of input/output devices on the usability of such devices to interact with virtual environments.

CHAPTER FIVE: CONCLUSION

This work investigated how PDAs can be used to interact with Virtual Environments in real-time and in a more natural way. Text messages and speech were used to study the interactivity. The architecture presented gave satisfactory results for usability pilot studies. The proxy based approach can increase the scalability and interoperability with other virtual environments. Making use of a speech application and other services like specially made goggles to display the PDA screen would allow hands free usage of the PDA.

The results show that by disabling Nagle algorithm, average response time is reduced over wireless network. It is suggested to disable Nagle's algorithm to take advantage of small packet size of multimedia (audio and video) packets [34]. Researchers have presented evidence [38] [39] that the Nagle algorithm should be disabled in order to reduce the latency as observed by the client and to protect against unforeseen interactions between TCP and HTTP with persistent connections. The results obtained have confirmed these findings.

This reduction in RTT can be used by high performance applications such as speech recognition, voice to text conversion, and gesture movement to enable natural response between the users in VE. Additionally the PDA might be useful in other VE roles, such as supporting so-called 'on-board' tracking calculations thereby minimizing or eliminating the need for line of sight in VE tracking applications. The reduced RTT also makes the messages to be sent with minimal delay to improve interactivity, especially when there are a large number of users and messages. Performance is found to depend on the type of application, its transmitting frequencies and size of message.

Since the Microsoft Speech Server was not available at the time of study, the performance for integrated speech application could not be studied. The pilot tests of speech recognition using desktop engine showed good results for speech recognition for different speakers. With the availability of SES component of Microsoft Speech Server, same application can be used to test speech applications using PDA.

APPENDIX A: COMPAQ'S iPAQ POCKET PC

Figure 23 shows Compaq iPAQ Pocket PC H3955

Figure 23: Compaq iPAQ Pocket PC H3955

Following are specifications for Compaq iPAQ Pocket PC H3955:

About Pocket PC: Microsoft® Pocket PC version 4.20.1081 (Build 13100)

Processor: Intel® PXA250

Identity:

Asset Tag #: 4G27KVL1S0PW

Serial #: 4G27KVL1S0PW

Memory:

System RAM Size: 64 MB

System ROM Size: 32 MB

ROM Is Flash: Yes

Memory Technology: SDRAM

Flash Manufacturer: INTEL

Flash Chip Type: 28F128

Flash Block Size: 128 KB

Version:

Product Revision Level: 2.5

ROM Date: 06/27/03

ROM Version: 3.00.08 ENG

OS Version: Windows CE4.20

Display:

Panel ID: Z

Display Size: 3.78 in

Display Type: LCD Display

Display Screen: Transreflective color TFT

Color Depth: 16-bit (64K colors), 0.24-dot pitch

Display Horizontal Pixels: 240 pixels

Display Vertical Pixels: 320 pixels

System:

Manufacturer: Compaq Computer Corp.

Product ID: Pocket PC

Model ID: Compaq iPAQ H3900

Processor Type: Intel® PXA250

Processor Revision: B1

Processor Speed: 400 MHz

Language: ENGLISH

Country ID: U.S.A

Bluetooth:

Module Type:

Radio Revision: 1.0

Radio Present: N

Communications:

Slot Types Provided: SD Memory Card

Wireless Connectivity: IrDA

Interface Provided: 1 x USB, 1 x infrared - IrDA, 1 x headphones - output - mini-phone stereo 3.5 mm , 1 x serial - RS-232

Expansion Slot: In use for wireless LAN Network interface card (NIC).

Battery:

Average Battery Life: 14 Hours

Battery Technology: Lithium Polymer

Backlight:

Multi-level brightness adjustment, light sensor for automatic adjustment of brightness level.

Audio:

Speaker, 3.5 mm stereo headphone jack

Indicators:

3 mode alarm notification: flashing green LED, tone, pop-up message. Charge active: flashing/solid amber LED, Bluetooth active: flashing/sound blue LED

Table 3
Physical Specifications

	Compaq iPAQ Pocket PC H3900	
	US	**Metric**
Height	5.28 in	134.0 mm
Width	3.30 – 3.03 in tapering	84.0 – 77.0 mm tapering
Depth	0.63 in	15.9 mm
Weight	6.49 oz	184 g

Table 4
Operating Environment

		US	**Metric**
Temperature	Operating	32^0 to 104^0F	0^0 to 40^0C
	Nonoperating	-22^0 to 140^0F	-30^0 to 60^0C
Relative Humidity	Operating	10 to 90%	10 to 90%
	Nonoperating	10 to 90%	10 to 90%
Maximum altitude	Operating	0-15,000 ft	14.7 to 10.1 psia
	Nonoperating	0-15,000 ft	14.7 to 10.1 psia

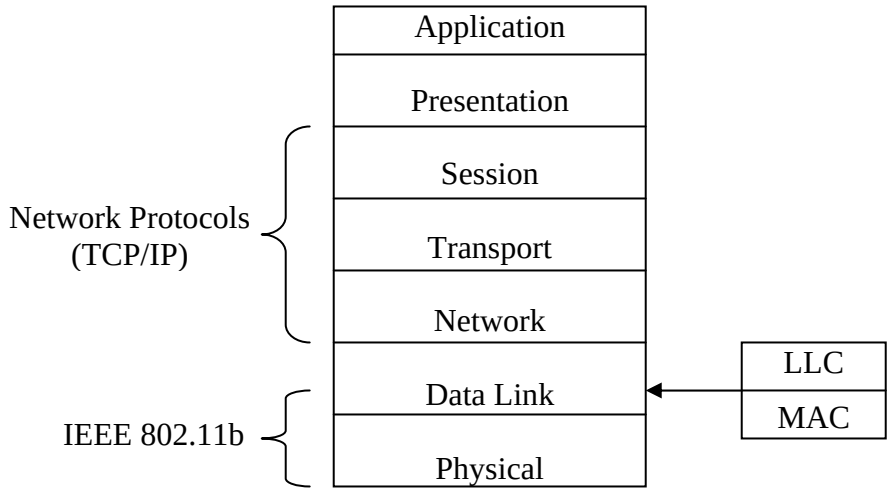
More details: http://h18000.www1.hp.com/products/quickspecs/11346_na/11346_na.HTML

APPENDIX B: Wi-Fi (IEEE 802.11b)

802.11, or IEEE 802.11, is a type of radio technology used for wireless local area networks (WLANs). It is a standard that has been developed by the IEEE (Institute of Electrical and Electronic Engineers), http://standards.ieee.org. Wi-Fi , 802.11, is composed of several standards operating in different radio frequencies: 802.11b is a standard for wireless LANs operating in the 2.4 GHz spectrum with a bandwidth of 11 Mbps; 802.11a is a different standard for wireless LANs, and pertains to systems operating in the 5 GHz frequency range with a bandwidth of 54 Mbps. Another standard, 802.11g, is for WLANS operating in the 2.4 GHz frequency but with a bandwidth of 54 Mbps.

The 802 subgroup (of the IEEE) develops standards for local and wide area networks with the 802.11 section reviewing and creating standards for wireless local area networks. 802.11b is International standard for wireless networking that operates in the 2.4 GHz frequency range (2.4 GHz to 2.4835 GHz) and provides a throughput of up to 11 Mbps. The 802.11b wireless LAN standard specifies the lowest layer of the OSI network model (physical) and a part of the next higher layer (data link). In addition, the standard specifies the use of the 802.2 protocol for the logical link control portion of the data link layer. The OSI network model is shown in Figure 24.

The difference between wireless LANs and wired networks such as Ethernet is the transmission medium. Whereas Ethernet sends electrical signals through wires, wireless LANs send radio frequency (RF) energy through the air. Wireless devices are equipped with a special network interface card (NIC) with one or more antennae, a radio transceiver, and circuitry to convert between the analog radio signals and the digital pulses used by computers.



LLC: Logical Link Control (Sub-layer)

MAC: Media Access Control (Sub-layer)

Figure 24: OSI network Model

Radio waves broadcast on a given frequency can be picked up by any receiver within range tuned to that same frequency. Effective or usable range depends on signal power, distance, and interference from intervening objects or other signals. Information is carried by modulating the radio waves.

Wireless LAN topologies / Operation Modes

IEEE 802.11b defines two pieces of equipment, a wireless station, which is usually a PC, a PDA or a Laptop with a wireless network interface card (NIC), and an Access Point (AP),which acts as a bridge between the wireless stations and Distribution System (DS) or wired networks. There are two operation modes in IEEE 802.11b viz. Infrastructure Mode and Ac Hoc Mode.

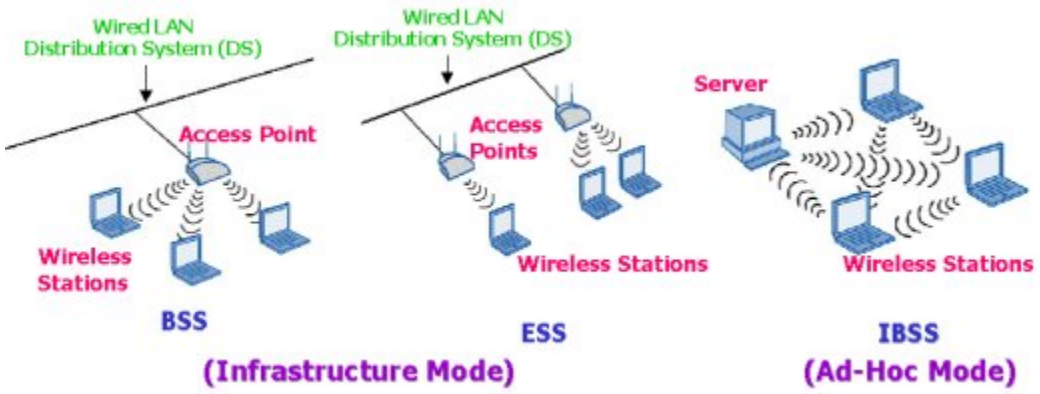


Figure 25: Wireless LAN topologies

1. **Infrastructure Mode**

   Infrastructure Mode consists of at least one Access Point connected to the Distribution System.

   - **Basis Service Set (BSS)**

     An Access Point provides a local bridge function for the BSS. All wireless stations communicate with the Access Point and no longer communicate directly. All frames are relayed between wireless stations by the Access Point.

   - **Extended Service Set (ESS)**

     An Extended Service Set is a set of infrastructure BSS's, where the Access Points communicate amongst themselves to forward traffic from one BSS to another to facilitate movement of wireless stations between BSS's.

2. **Ad Hoc Mode**

    - **Independent Basic Service Set (IBSS) or Peer to Peer**

      The wireless stations communicate directly with each other. Every station may not be able to communicate with every other station due to the range limitations. There are no Access Points in an IBSS. Therefore, all stations need to be within the range of each other and they communicate directly.

More Information:

http://grouper.ieee.org/groups/802/11/index.html

APPENDIX C: C#/SALT CODE FOR SPEECH APPLICATION

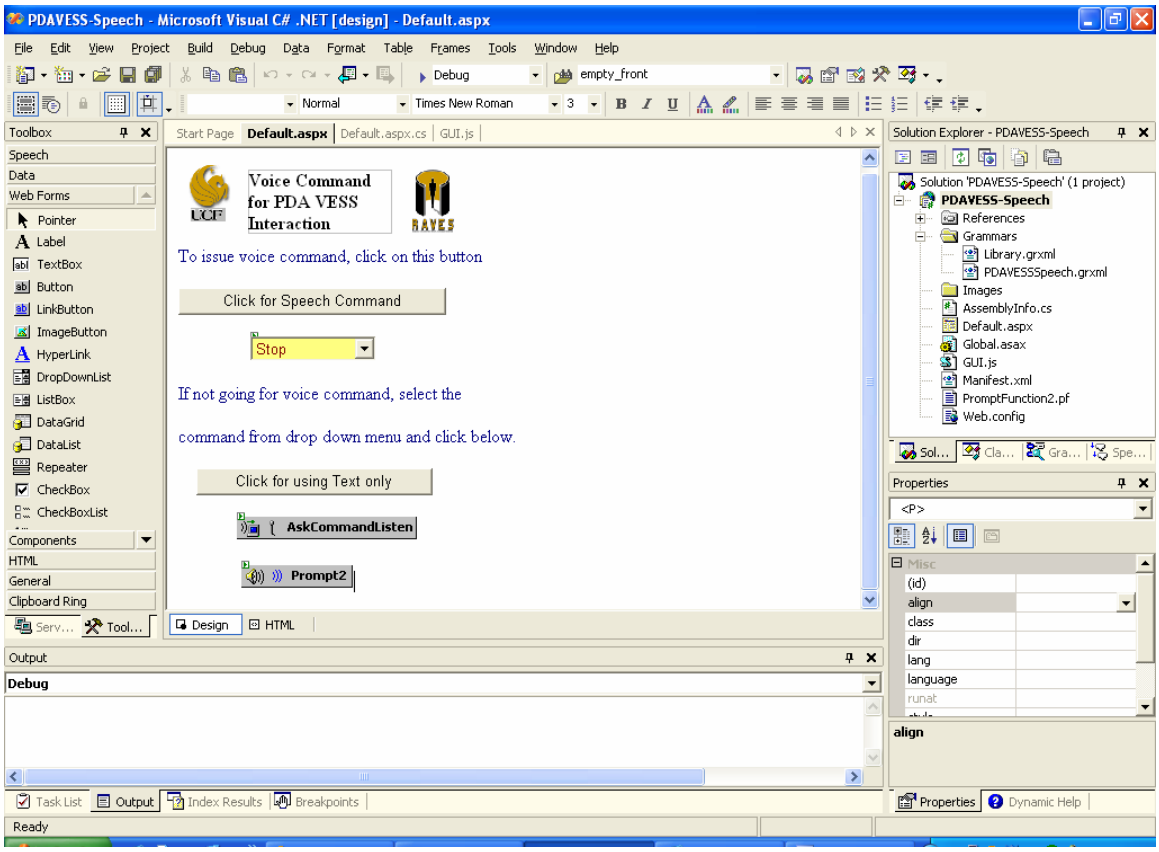


Figure 26: Graphical interface for Speech application

Default.aspx Code behind

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Configuration;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.Xml;
using Microsoft.Speech.Web.UI;
```

```
namespace PDAVESS_Speech
{
	public class Default: System.Web.UI.Page
	{
		protected System.Web.UI.HtmlControls.HtmlImage VoiceCommands;
		protected System.Web.UI.LiteralControl literalControl1;
		protected System.Web.UI.LiteralControl literalControl2;
		protected System.Web.UI.LiteralControl literalControl3;
		protected System.Web.UI.WebControls.DropDownList CommandList;
		protected Microsoft.Speech.Web.UI.Listen AskCommandListen;
		protected Microsoft.Speech.Web.UI.Prompt Prompt1;
		protected Microsoft.Speech.Web.UI.Prompt Prompt2;
		protected System.Web.UI.LiteralControl literalControl4;
		protected Microsoft.Speech.Web.UI.Value value2;
		protected System.Web.UI.LiteralControl literalControl5;
		protected System.Web.UI.LiteralControl literalControl6;
		protected Microsoft.Speech.Web.UI.Value value3;
		protected System.Web.UI.LiteralControl literalControl7;
		protected Microsoft.Speech.Web.UI.Value value4;
		protected Microsoft.Speech.Web.UI.Value value5;
		protected Microsoft.Speech.Web.UI.Value value1;

		private void Page_Load(object sender, System.EventArgs e)
		{
			// Put user code to initialize the page here
			if(!Page.IsPostBack)
			{
				PopulatePullDownControls();
			}
			if(ConfigurationSettings.AppSettings["SpeechServer"]!=null)
			{
				String
speechServer=ConfigurationSettings.AppSettings["SpeechServer"].ToString();
				Param param = new Param();
				param.Name="server";
				param.Value=speechServer;
				AskCommandListen.Params.Add(param);
			}
		}

	//Web Form Designer Generated Code

		private void PopulatePullDownControls()
		{
```

```
                CommandList.Items.Add(new
System.Web.UI.WebControls.ListItem("",""));
                CommandList.Items.Add(new
System.Web.UI.WebControls.ListItem("Halt","Halt"));
                CommandList.Items.Add(new
System.Web.UI.WebControls.ListItem("Stop","Stop"));
                CommandList.Items.Add(new
System.Web.UI.WebControls.ListItem("Turn Left","Turn Left"));
                CommandList.Items.Add(new
System.Web.UI.WebControls.ListItem("Turn Right","Turn Right"));
                CommandList.Items.Add(new System.Web.UI.WebControls.ListItem("Go
Forward","Go Forward"));
                CommandList.Items.Add(new System.Web.UI.WebControls.ListItem("Go
Back","Go Back"));
                CommandList.Items.Add(new System.Web.UI.WebControls.ListItem("Go
Up","Go Up"));
                CommandList.Items.Add(new System.Web.UI.WebControls.ListItem("Go
Down","Go Down"));
                CommandList.Items.Add(new
System.Web.UI.WebControls.ListItem("Move Forward","Move Forward"));
                CommandList.Items.Add(new
System.Web.UI.WebControls.ListItem("Move Back","Move Back"));
                CommandList.Items.Add(new
System.Web.UI.WebControls.ListItem("Move Up","Move Up"));
                CommandList.Items.Add(new
System.Web.UI.WebControls.ListItem("Move Down","Move Down"));
        }

        private void InitializeComponent()
        {
                this.Prompt2.Complete += new System.EventHandler(this.Page_Load);
                this.Load += new System.EventHandler(this.Page_Load);

        }

        private void AskCommandListen_Reco(object sender, System.EventArgs e)
        {
                string recognizedText=AskCommandListen.RecoResult.InnerText;
                if(recognizedText!=null)
                {
                        CommandList.DataTextField=recognizedText;
                }
                else
                {
```

```
                                throw new InvalidOperationException("Recognized Command not
found.The recognized text was " + AskCommandListen.RecoResult.OuterXml);
                        }
                }

        }
}
```

GUI.js (Used to act on an event)

```
function SetDropDown()
{
var
theNode=event.srcElement.recoResult.selectSingleNode("PDAVESSSpeech.grxml/Commands")
;
var theResult="";
var listCommands="";

if(theNode !=null)
{
        theResult=theNode.text;
        /* Pocket IE and IE have different ways of accesing an element on the page*/
        if(typeof(window["CommandList"])!="undefined")
        {
                listCommands=window["CommandList"];
        }
        else
        {
                listCommands=document.all["CommandList"];
                document.all('CommandList').style.backgroundColor = 'red';
        }
        listCommands.value=theResult;
}
}
function DropDownLabelOnClick()
{
        AskCommandListen.start();
}
```

```
function noRecognition()
{
	alert("Recognition Failed... Try Again");
}

function timeOut()
{
	alert("Its timeOut...Try Again");
}

function sendToVess()
{
	var result=theNode.text;
}

function error()
{
	alert("Error in Recognition..");
}
```

APPENDIX D: C# CODE FOR TCP/IP NETWORK COMMUNICATION BETWEEN POCKET PC, PROXY AND VESS

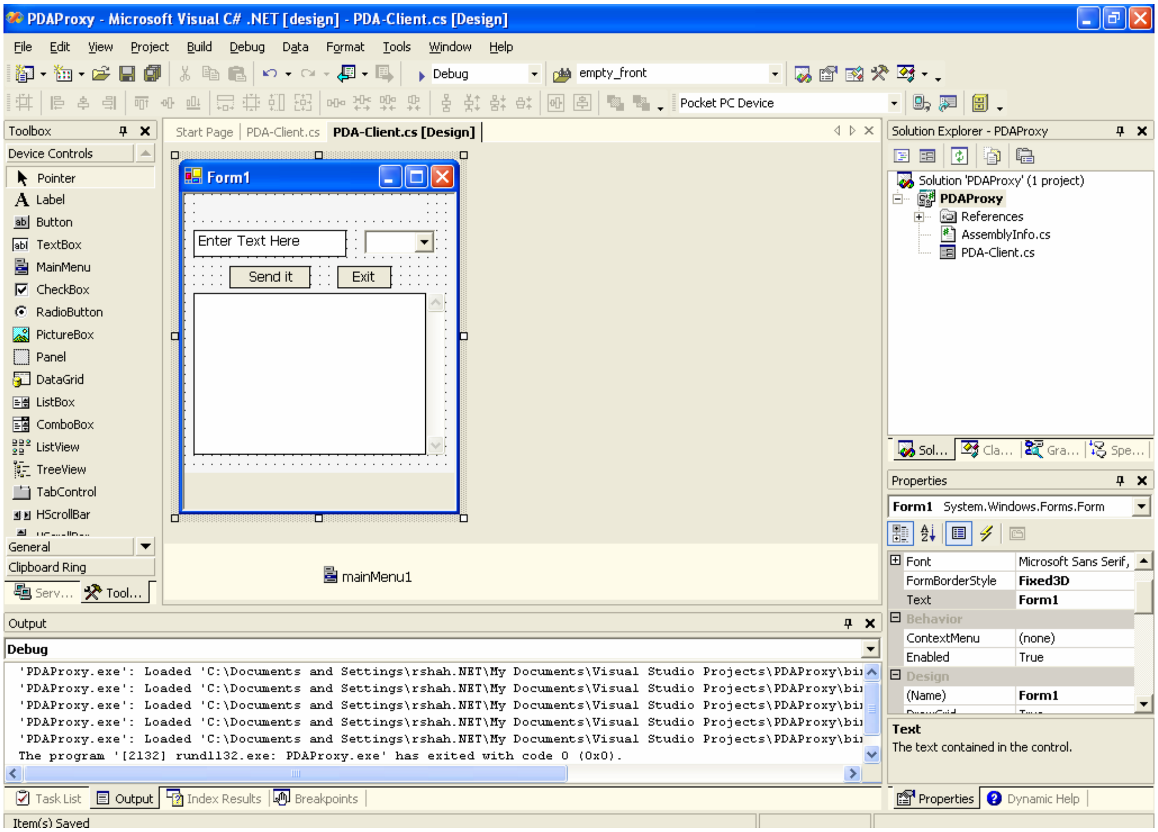


Figure 27: PDA-Client.cs [Design View]

PDA-Client.cs Codebehind: (This runs on Pocket PC)

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Net;
using System.Text;
using System.Net.Sockets;
using System.Threading;

namespace PDAProxy
{
        /// <summary>
```

```
	/// Summary description for Form1.
	/// </summary>
	public class Form1 : System.Windows.Forms.Form
	{
		//private Socket mySocket;

		private System.Windows.Forms.Button button1;
		private System.Windows.Forms.TextBox textBox1;
		private System.Windows.Forms.Label label1;
		private System.Windows.Forms.StatusBar statusBar1;
		private System.Windows.Forms.TextBox textBox2;
		private System.Windows.Forms.Button button2;
		public System.Windows.Forms.ComboBox Nagle;
		private System.Windows.Forms.MainMenu mainMenu1;
		static string fileLocation="\\myFile.txt";

		public Form1()
		{
			// Required for Windows Form Designer support
			InitializeComponent();
			// TODO: Add any constructor code after InitializeComponent call
		}
		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		protected override void Dispose( bool disposing )
		{

				base.Dispose( disposing );
		}

		#region Windows Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		private void InitializeComponent()
		{
			System.Resources.ResourceManager resources = new
System.Resources.ResourceManager(typeof(Form1));
			this.button1 = new System.Windows.Forms.Button();
			this.textBox1 = new System.Windows.Forms.TextBox();
			this.label1 = new System.Windows.Forms.Label();
			this.statusBar1 = new System.Windows.Forms.StatusBar();
			this.textBox2 = new System.Windows.Forms.TextBox();
```

```
this.button2 = new System.Windows.Forms.Button();
this.Nagle = new System.Windows.Forms.ComboBox();
this.mainMenu1 = new System.Windows.Forms.MainMenu();
//
// button1
//
this.button1.Location = new System.Drawing.Point(40, 64);
this.button1.Text = "Send it";
this.button1.Click += new System.EventHandler(this.button1_Click);
//
// textBox1
//
this.textBox1.Location = new System.Drawing.Point(8, 32);
this.textBox1.Multiline = true;
this.textBox1.Size = new System.Drawing.Size(136, 24);
this.textBox1.Text = "Enter Text Here";
//
// label1
//
this.label1.Location = new System.Drawing.Point(16, 8);
this.label1.Size = new System.Drawing.Size(200, 16);
//
// statusBar1
//
this.statusBar1.Location = new System.Drawing.Point(0, 248);
this.statusBar1.Size = new System.Drawing.Size(240, 32);
//
// textBox2
//
this.textBox2.Location = new System.Drawing.Point(8, 88);
this.textBox2.Multiline = true;
this.textBox2.ScrollBars = System.Windows.Forms.ScrollBars.Vertical;
this.textBox2.Size = new System.Drawing.Size(224, 144);
this.textBox2.Text = "";
//
// button2
//
this.button2.Location = new System.Drawing.Point(136, 64);
this.button2.Size = new System.Drawing.Size(48, 20);
this.button2.Text = "Exit";
this.button2.Click += new System.EventHandler(this.button2_Click);
//
// Nagle
//
this.Nagle.DisplayMember = "Delay";
```

```
                this.Nagle.Items.Add("NoDelay");
                this.Nagle.Items.Add("Delay");
                this.Nagle.Location = new System.Drawing.Point(160, 32);
                this.Nagle.Size = new System.Drawing.Size(64, 22);
                this.Nagle.Visible = true;
                this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.Fixed3D;
                //
                // Form1
                //
                this.BackColor = System.Drawing.Color.WhiteSmoke;
                this.ClientSize = new System.Drawing.Size(240, 280);
                this.Controls.Add(this.Nagle);
                this.Controls.Add(this.button2);
                this.Controls.Add(this.textBox2);
                this.Controls.Add(this.statusBar1);
                this.Controls.Add(this.label1);
                this.Controls.Add(this.textBox1);
                this.Controls.Add(this.button1);
                this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
                this.Menu = this.mainMenu1;
                this.Text = "Form1";

        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>

        static void Main()
        {
                Application.Run(new Form1());

        }

        public void connect()
        {

                try
                {
                        // The default SendBuffer size is 16384 and receiveBuffer size is
32768
                                int dwStart = 0;
                                int dwStop = 0;
```

```
                string destIP="132.170.190.142";
                int destPort=8001;
                int Port=8002;
                TcpClient tcpclnt;
                statusBar1.Text=String.Format("");
            try
            {
                // Getting DNS info
                IPHostEntry
ihe=Dns.GetHostByName(Dns.GetHostName());
                // Getting local IP Address
                IPAddress localIPAd=ihe.AddressList[0];
                // Assigning sending port
              IPEndPoint ipep=new IPEndPoint(localIPAd,Port);
                label1.Text=localIPAd.ToString()+ ":"+Port.ToString();
                textBox2.Text= String.Format("Client IP:{0}:{1}
",localIPAd.ToString(),Port.ToString());
                tcpclnt = new TcpClient(ipep);
                if(Nagle.Text=="NoDelay")
                {
                        //Disabling Nagle Algorithm
                        tcpclnt.NoDelay=true;
                }
                else
                {
                        tcpclnt.NoDelay=false;
                }
                textBox2.Text+= String.Format("\r\n Nagle Algorithm is:
{0} ",tcpclnt.NoDelay);
                //IPAddress IPAd=IPAddress.Parse("132.170.190.125");
                //server IP address
                IPAddress IPAd=IPAddress.Parse(destIP);
                statusBar1.Text=Convert.ToString("Trying to
Connect"+IPAd+"::"+destPort+"...");
                textBox2.Text+= String.Format("\r\nServer IP: {0} : {1} ",
destIP.ToString(),destPort.ToString());
                // Connect to server
                tcpclnt.Connect(IPAd,destPort);
                statusBar1.Text="Connected";
                //connected=true;
                }
                catch(SocketException se)
                {
```

```
                                        DialogResult dr=MessageBox.Show( se.Message
+" Unable to connect to server.. ",
se.ErrorCode.ToString(),MessageBoxButtons.AbortRetryIgnore,MessageBoxIcon.None,Messag
eBoxDefaultButton.Button1);
                                        if(dr==DialogResult.Abort)
                                        {
                                                Application.Exit();
                                        }
                                        if(dr==DialogResult.Retry)
                                        {
                                                connect();
                                        }
                                        if(dr==DialogResult.Cancel)
                                        {
                                                Application.Exit();
                                        }
                                        return;
                                }
                        // Create network stream that we will use to send and receive data.
                        NetworkStream stm = tcpclnt.GetStream();
        //                      if(stm.CanWrite)
                        for(int i=0;i<100;i++)
                        {
                        try
                        {
                                // Assigning the text entered in textbox
                                string str;
                                str=textBox1.Text;
//                              int sendBuffer=tcpclnt.SendBufferSize;
//                              textBox2.Text+=String.Format("\r\nSending
BufferSize:{0}",sendBuffer);
//                              // Getting the text in Textbox in byte format
                                byte[] sendingBytes=Encoding.ASCII.GetBytes(str);
                                statusBar1.Text="Transmitting.....";
                                // Start timing
                                dwStart=System.Environment.TickCount;
                                // Send the bytes using networkStream
                                stm.Write(sendingBytes,0,sendingBytes.Length);
                                stm.Flush();
                        //      textBox2.Text+=String.Format("\r\nStart Time
is:{0}",DateTime.Now.Millisecond);
                                textBox2.Text+=String.Format(" \r\nNumber of bytes
sent:{0}", sendingBytes.Length);
                                //textBox2.Text+=String.Format(" \r\nNumber of bytes
sent:{0}", str.Length);
```

```
                        }

                        catch (Exception noTx)
                        {
                                MessageBox.Show(" Can not send the information.. " +
noTx.Message);
                                return;
                        }

//                      if(stm.CanRead)

                        try
                        {
                                //byte[] b=new byte[tcpclnt.ReceiveBufferSize];

                                //Get the value of TCP receiveBuffer
                                //int receiveBuffer=tcpclnt.ReceiveBufferSize;
                                //textBox2.Text +=String.Format("\r\nReceive Buffer Size:
{0}",receiveBuffer);
                                // Set the value of block of bytes to be read from TCP
receiveBuffer
                                        byte[] received=new byte[10];
                                // Read the response using networkSteram
                                stm.Read(received,0,received.Length);

                                // stop timing
                                dwStop = System.Environment.TickCount;
                                int responseTime=dwStop-dwStart;
                                //textBox2.Text +=String.Format("\r\n The response is
received at: {0}", DateTime.Now.Millisecond);
                                string
instring=Encoding.ASCII.GetString(received,0,received.Length);
                                textBox2.Text +=String.Format(" \r\n {0}",instring);
                                textBox2.Text+=String.Format(" \r\n The bytes received
are:{0} ",instring.Length);
                                textBox2.Text +=String.Format("\r\nThe response time
is:{0} milliseconds ",responseTime);
                                writeToFile(responseTime);
                                statusBar1.Text="Successfully Received Ack.....";
                        }

                        catch (Exception noAck)
                        {
                                MessageBox.Show(" Can not read the acknowledgemtnt.."
+ noAck.Message);
```

```
                        return;
                    }
                        //Thread.Sleep(500);
                    }

                    stm.Close();
                    tcpclnt.Close();
                    //MessageBox.Show("Clsoing Connection");
                }
                catch (Exception ee)
                {
                    MessageBox.Show(" Error..... "+ ee.Message);
                    return;
                }
            }

            static void writeToFile(float a)
            {
                try
                {

                //      string
file=fileLocation.Insert(fileLocation.Length,count.ToString()+".txt");
                    StreamWriter sw;
                    sw=File.AppendText(fileLocation);
                    sw.WriteLine(a);
                    sw.Close();

                }
                catch(Exception noWr)
                {
                    MessageBox.Show(" Unable to write to file.. "+ noWr.Message);
                    return;
                }
            }

            private void button1_Click(object sender, System.EventArgs e)
            {
                try
                {
                    StreamWriter cf;
                    cf=File.CreateText("\\myFile.txt");
                    cf.Close();
                }
                catch(Exception fileError)
```

```
                {
                        MessageBox.Show("Error creating File... " + fileError.Message);
                        return;
                }
                try
                {
//                      for(int i=0;i<100;i++)
//                      {
                                connect();
                                // Giving some time gap between sending messages
//                              Thread.Sleep(500);
//                      }
                        StreamWriter sw;
                        sw=File.AppendText(fileLocation);
                        sw.WriteLine(textBox1.Text.ToString());
                        sw.WriteLine(textBox1.Text.Length);
                        sw.Close();

                }
                catch(Exception noCon)
                {
                        MessageBox.Show("Error: " + noCon.Message);
                        return;
                }
        }

        private void button2_Click(object sender, System.EventArgs e)
        {
                Application.Exit();
        }

    }
}
```

server.cs: (This runs on the server)

```
// The default ReceiveBuffer size is 8192 and SendBuffer size is 8192

using System;
using System.Data;
using System.IO;
using System.Net;
using System.Text;
using System.Net.Sockets;



public class server_p
{
        public static void Main()
        {
         try
          {
                string hostName;
                //string IP="132.170.190.125";
                int port =8001;
                int InByte=8820;
                string instring="";
                int buffer=16384;
                int recv;
                int count=0;

                 //Get the Hostname of the server
                 hostName =Dns.GetHostName();
                 //IPAddress ipAd = IPAddress.Parse(IP.ToString());
                 // Get the IP address from the addresslist
                 IPAddress ipAd = Dns.Resolve(hostName).AddressList[0];


                 Console.WriteLine("The server IP address is {0} and it is running at port
{1}...",ipAd.ToString(),port);

                 Console.WriteLine("The input block is {0} bytes",InByte);

                 // Declaring TCPlistener
                 TcpListener proxyServer;
                 // Declaring TCP Client
                 TcpClient client;
```

```
proxyServer=new TcpListener(ipAd,port);
proxyServer.Start();
Console.WriteLine("Waiting for a connection.....");
client = proxyServer.AcceptTcpClient();
// Disabling Nagle's algorithm
// client.NoDelay=true;

Console.WriteLine(" Accepted connection ...");

// Console.WriteLine("The local End point is  :" + proxyServer.LocalEndpoint );

//Setting the size of receiveBuffer
client.ReceiveBufferSize=buffer;

// Getting the size of receivebuffer
int receiveBuffer=client.ReceiveBufferSize;
Console.WriteLine("The Receive Buffer Size is :{0}",receiveBuffer);
// Defining the size of block to get the data from TCP receiveBuffer
byte[] inData=new byte[InByte];
ASCIIEncoding asen=new ASCIIEncoding();
// Declairing the NetworkStream object
NetworkStream ns =client.GetStream();

while(true)
 {
        //string connected=" Connected to the Server";
        //data=asen.GetBytes(connected);
        //ns.Write(data,0,data.Length);

        //if(ns.CanRead)

        //byte[] inData=new byte[client.ReceiveBufferSize];

        try
        {
                //Receiving data using networkStream
                recv=ns.Read(inData,0,inData.Length);
                instring=Encoding.ASCII.GetString(inData);
                Console.WriteLine("Received "+instring +" from client with "
+instring.Length +" bytes: iteration="+count++ );
        }

        catch (Exception noRead)
        {
```

```
                                Console.WriteLine("Networkstream Unable to read data" +
noRead.Message);
                                return;
                        }

                        byte[] OutData=Encoding.ASCII.GetBytes("Received");

                        try
                        {
                                int sendBuffer=client.SendBufferSize;
                                Console.WriteLine("The Send Buffer Size is: {0}",sendBuffer);
                                 client.NoDelay=true;
                                //Sending the acknowledgement using networkStream
                                  ns.Write(OutData,0,OutData.Length);
                                  ns.Flush();
                        }

                        catch(Exception noAck)
                        {
                                Console.WriteLine("Networkstream Unable to send
acknowledgement... " + noAck.Message);
                                return;
                        }

                } // End While

                ns.Close();
                client.Close();
                proxyServer.Stop();

        }       // End Try

        catch(Exception e)
        {
                Console.WriteLine(" Error...."+ e.StackTrace);
        }

  } // End Main
}  // End Class
```

Proxy_VESS.cs: (This runs on the proxy to connect to VESS)

```
/* This function connects to VESS and receives acknowledgment. The RTT is
recorded at C:\myFile.txt*/

using System;
using System.Data;
using System.IO;
using System.Net;
using System.Text;
using System.Net.Sockets;


public class proxy_vess
{
      public static void Main()
      {
            try
            {
                  StreamWriter cf;
                  cf=File.CreateText("C:\\myFile.txt");
                  cf.Close();
            }
            catch(Exception fileError)
            {
                  Console.WriteLine("Error creating File... " +
fileError.Message);
                  return;
            }
            bool connected;
            int dwStart = 0;
            int dwStop = 0;

            string destIP="132.170.190.134";
            int destPort=8001;
            int Port=8004;

            TcpClient tcpclnt;
            NetworkStream stm;
            // Getting DNS info
            IPHostEntry       ihe=Dns.GetHostByName(Dns.GetHostName());
            // Getting local IP Address
            IPAddress   localIPAd=ihe.AddressList[0];
            // Assigning sending port
            IPEndPoint ipep=new     IPEndPoint(localIPAd,Port);
            tcpclnt = new TcpClient(ipep);
                        //Disabling Nagle Algorithm
            tcpclnt.NoDelay=true;

            IPAddress IPAd=IPAddress.Parse(destIP);
            try
            {
                  tcpclnt.Connect(IPAd,destPort);
            }
```

```
                catch(SocketException se)
                {
                      Console.WriteLine("Could not connect to VESS " +
se.Message);
                      connected=false;

                }
            stm = tcpclnt.GetStream();
            String instring="Please review UCF's thesis and dissertation
formatting requirements, as found in the Thesis and Dissertation Manual at
www.graduate.ucf.edu > Current Students > Forms and Files, prior to using
this template. This template is pre-formatted using MS Word styles and will
not function well if improperly formatted text is used. Please read the
tutorial "Using Microsoft Word to Format Your Document" at
www.graduate.ucf.edu > Current Students > Forms and Files prior to using this
template. Sample style settings for headings, subheadings, body text and
captions can be found in the tutorial as well. In order for bookmarks to be
automatically created in the PDF, styles must be used in the document, and
the Table of Contents, List of Tables and List of Figures must be generated
using Word's Insert > Index and Tables function.Do not include a copy of the
Thesis and Dissertation Approval Page in the manuscript; this should be a
separate file.    If removing this page causes difficulty with page numbers,
you may leave it in the Word file but delete it from the PDF.";
            for(int i=0;i<10;i++)
            {
                try
                {
                      // Assigning the text entered in textbox
                      byte[]
sendingBytes=Encoding.ASCII.GetBytes(instring);
                      dwStart=System.Environment.TickCount;
                      stm.Write(sendingBytes,0,sendingBytes.Length);
                      stm.Flush();
                }
                catch (Exception noTx)
                {
                      Console.WriteLine(" Can not send the information.. "
+ noTx.Message);
                      connected=false;

                }
                try
                {

                      byte[] received=new byte[10];
                      // Read the response using networkSteram
                      stm.Read(received,0,received.Length);
                      // stop timing
                      dwStop = System.Environment.TickCount;
                      int responseTime=dwStop-dwStart;
                      string
receivedString=Encoding.ASCII.GetString(received,0,received.Length);
                      Console.WriteLine(receivedString);
                      writeToFile(responseTime);
                      connected=true;
```

```
                  }
                  catch (Exception noAck)
                  {
                        Console.WriteLine(" Can not read the
acknowledgemtnt.." + noAck.Message);
                        connected=false;

                  }
               }
                  stm.Close();
                  tcpclnt.Close();
      }

      static void writeToFile(float a)
      {
            try
            {

            //    string
file=fileLocation.Insert(fileLocation.Length,count.ToString()+".txt");
                  String fileLocation="C:\\myFile.txt";
                  StreamWriter sw;
                  sw=File.AppendText(fileLocation);
                  sw.WriteLine(a);
                  sw.Close();
            }
            catch(Exception noWr)
            {
                  Console.WriteLine(" Unable to write to file.. "+
noWr.Message);
            }
      }

}
```

LIST OF REFERENCES

[1] Sherman W.R., Craig A.B., "Understanding Virtual Reality" (2003), Morgan Kaufmann Publishers

[2] Card S.K, Moran T.P., Newell A., "The Psychology of Human-Computer Interaction" (1983), Lawrence Erlbaum Associates, Publishers

[3] The Free On-Line Dictionary of Computing http://foldoc.doc.ic.ac.uk/foldoc/index.html

[4] Reeves L., Goldiez B., Kingdon K., Stanney K. "Raves: Developing a Testbed for Advancing Augmented and Virtual Environment Systems" MP 08, 23rd Annual Army Science Conference

[5] Satyanarayanan, M., "Pervasive Computing: Vision and Challenges", Personal Communications, IEEE, Volume: 8 Issue: 4, Aug. 2001 Page(s): 10 –17

[6] Daly, J., Kline, B., Martin, G.A., "VESS: coordinating graphics, audio, and user interaction in virtual reality applications" Virtual Reality, 2002. Proceedings. IEEE.

[7] Institute for Simulation and Training, http://www.ist.ucf.edu

[8] Keefe, D., Farag, P., & Zucker, A. (2003). *Annotated Bibliography of Ubiquitous Computing Evaluations*. Retrieved, from http://www.ubiqcomputing.org/Reference.pdf

[9] Weiser, M. "The Computer for the 21st Century", Scientific American, September, 1991.

[10] Myers B. "*Using Handhelds and PCs Together*", Communications of the ACM, November 2000, Volume 44, Number 11.

[11] Pebbles, http://www.cs.cmu.edu/~pebbles

[12] Park, K., Leigh, J., Johnson, A., Carter, B., Brody, J., Sosnoski, J., "*Distance Learning Classroom Using Virtual Harlem*", Virtual Systems and Multimedia, 2001. Proceedings.

Seventh International Conference on, (VSMM 2001), Berkeley CA, Oct 25-27, 2001, pp. 489-498.

[13] Cohen, P.; McGee, D.; Oviatt, S.; Wu, L.; Clow, J.; King, R.; Julier, S.; Rosenblum, L.; "*Multimodal interaction for 2D and 3D environments [virtual reality]*", Computer Graphics and Applications, IEEE , Volume: 19 Issue: 4

[14] Masoodian, M.; Luz, S.;, "*Heterogeneous client-server architecture for a virtual meeting environment*", Proceedings of 8th Euromicro Workshop on Parallel and Distributed Processing, 2000., 19-21 Jan. 2000, pp. 67 - 74.

[15] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, John C. Hart, "*The CAVE: audio visual experience automatic virtual environment*" June 1992, Communications of the ACM, Volume 35 Issue 6 pp. 64-72

[16] Watsen K., Darken R. P., Capps M.V., *"A handheld computer as an interaction device to a virtual environment"*. In Proceedings of the 3rd International Immersive Projection Technology Workshop, 1999. http://watsen.net/Bamboo/papers/iptw99.pdf

[17] Hill, L., Cruz-Neira, C., *"Palmtop Interaction Methods for the Immersive Projection Technology Virtual Reality Systems"*. 4th Immersive Projection Technology Workshop. June 2000, Ames, Iowa

[18] M. Gutierrez, F. Vexo, D. Thalmann, *"Controlling Virtual Humans Using PDAs"*. The 9th International Conference on Multi-Media Modeling (MMM'03), January 7-10, 2003, Taiwan

[19] Stanney, K.M., Kingdon, K., Graeber, D., & Kennedy, R.S. (2002). *"Human performance in immersive virtual environments: Effects of duration, user control, and scene complexity*" Human Performance, 15(4), 339-366.

[20] Goose S., Wanning H., Schneider G., *"Mobile Reality: A PDA-Based Multimodal Framework Synchronizing a Hybrid Tracking Solution with 3D Graphics and Location-Sensitive Interaction"*, Proceedings of the 4th international conference on Ubiquitous Computing, Göteborg, Sweden, 2002 Pp: 33 - 47

[21] Microsoft Corporation, http://www.microsoft.com/net/basics/

[22] Oberg R.J., "*Introduction to C# Using .NET*", Prentice Hall PTR, Upper Saddle River, NJ 07458, 2002.

[23] Wigley A., Wheelwright S., "*Microsoft .NET Compact Framework*", Microsoft Press, Redmond, Washington-98052-6399, 2003.

[24] The SALT Forum, http://www.saltforum.org/

[25] Microsoft Speech, http://www.microsoft.com/speech/

[26] VoiceXML Forum, http://www.voicexml.org/

[27] BNC database and word frequency lists, http://www.itri.bton.ac.uk/~Adam.Kilgarriff/bnc-readme.html

[28] American Heritage Dictionary, http://dictionary.reference.com/help/ahd4/pronkey.html

[29] Speech Recognition Grammar Specification (SRGS) http://www.w3.org/TR/speech-grammar/

[30] Speech Synthesis Markup Language (SSML) http://www.w3.org/TR/speech-synthesis/

[31] ECMAScript (ECMA-262) http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

[32] Tanenbaum, A., "*Computer Networks, 3rd edition*", Prentice Hall, 1996

[33] Blum R., "*C# Network Programming*", SYBEX, 2003

[34] Negi A., Bose B., "*Impact of Nagle Algorithm and Jitter Buffer in Multimedia Applications*", 5th World Wireless Congress, May 25-28, 2004, San Francisco.

[35] Card S. K., Moran T. P., Newell A., "*The Psychology of Human-Computer Interaction*", Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

[36] Goldiez B., Shah R., Srinivasan R., Stanney K., Jones D., "PDAs in Virtual Environments", Proc. of the 10th International Conference on Cybernetics and Information Technologies, Systems and Applications (CITSA-2004), July 2004, Orlando, Florida.

[37] Nielsen, J., "*Usability Engineering*", Academic Press, Boston, 1993.

[38] Heidemann J., "*Performance interactions between P-HTTP and TCP implementations,*" *ACM Comput. Commun. Rev.*, vol. 27, pp. 65–73, Apr.1997.

[39] Nielsen H. F., Gettys J., Baird-Smith A., Prud'hommeaux E., Lie H.W., Lilley C., "Network performance effects of HTTP/1.1, CSS1, and PNG," presented at the ACM SIGCOMM Symp. Commun. Architectures Protocols, Cannes, France, Sept. 1997.